Húszéves az ELTE Eötvös József Collegium Informatikai Műhelye

$$\Omega = (\lambda x.xx)(\lambda x.xx)$$

Egy λ-kifejezés, mely β-redukcióval önmagát reprodukálja

HÚSZÉVES AZ
ELTE EÖTVÖS JÓZSEF COLLEGIUM
INFORMATIKAI MŰHELYE

2004 – 2024

Budapest, 2024

A kiadvány a „Szakkollégiumok tehetséggondozó programjainak támogatása” című pályázat keretében (NTP-SZKOLL-23-0031) valósult meg.

Támogatta az ELTE Informatikai Kar.

A cikkek szerzőinek anyagaiból szerkesztette: Lócsi Levente

ISBN 978-615-5897-64-1

Felelős kiadó: Dr. Horváth László,
az ELTE Eötvös József Collegium igazgatója

A nyomdai munkálatokat a CC Printing Szolgáltató Kft. végezte.
1055 Budapest, Falk Miksa utca 7. II. emelet 10/A
Felelős vezető: Könczey Áron
https://www.ccprinting.hu

Köszöntő

Az Informatikai Műhely az Eötvös Collegium büszkesége. A történelmi Collegiumot naggyá tevő szellemiségben fogant, miszerint törekednünk kell arra, hogy az egyetemünkön oktatott legkülönfélébb tudományszakokat a Ménesi úton is meghonosítsuk. A tudományos *universitas* eszményének megfelelően olyan „dögész-filosz" együttélést alakítsunk ki, amely kölcsönösen lelkesítően hat az egyetemi ifjúságra, az épületben lakó vagy oda bejáró hallgatóink közösségében. A műhely és tagsága az elmúlt két évtizedben minden tekintetben betöltötte hivatását. Az alapító műhelyvezetőnek, Csörnyei Zoltán Tanár úrnak, majd az őt követő tanáregyéniségeknek, Kozsik Tamásnak és Lócsi Leventének köszönhetően nemcsak a Kar kiemelkedő színvonalú tudományos háttérműhelyévé vált – Cicero szavait idézve: ex officina *tamquam ex equo Troiano meri principes exierunt* –, hanem a Collegium „önbeporzásában" is folyamatosan szerepet vállal. A tudományos műhelykurzusok mellett ugyanis rendszeresen biztosít informatikai alapképzést a nyitott bölcsészhallgatóknak. Az informatikusok hatása azonban áthatja az épület lakószintjeit is, és itt nem csupán a rendszergazdai tevékenységekre gondolok, hanem a mindennapos, a szobákban és a közösségi tereken megélt collegista beszélgetésekre, ahol megannyi érdekesség szóba kerül. Biztos vagyok abban is, hogy e hatás nem egyoldalú, és örömmel érzékelem, hogy informatikusaink nem szakjuk barbárjai – a szó eredeti, görög értelmében értem ezt, tudniillik maguk közt, maguknak, a külvilág számára érthetetlen nyelven mormolnak dolgaikról –, hanem a Collegiumnak köszönhetően is nyitott szellemű fiatalokká válnak, akik fogékonyak a humán tudományok befogadására is.

Az Informatikai Műhely húsz évéből tizenöt esztendőn át a Collegiumot igazgatóként irányítottam, és bizony a műhely felállítását is annak idején magam kezdeményeztem *(quorum pars magna fui)*. Láthattam, megélhettem és támogathattam, hogy a műhely saját „nevelése" veszi át a vezetés feladatát. Lócsi Levente Tanár úr személyében a történelmi

Collegium számára ismeretlen tudományterületen is immár megvalósult a kezdeti törekvés, az intézményünk falai között nevelkedett tehetség „forgathatja vissza” szellemiségét az újabb nemzedékbe. Ez a fejlemény önmagában mutatja e tudományos közösség erejét.

Köszönöm a műhely vezetőinek, tanárainak és tagságának áldozatos és elkötelezett munkáját! Gratulálok e kiváló kötethez, amely a tudományos munkák mellett olyan könnyedebb áttekintéseket és portrékat is tartalmaz, amelyek a laikus olvasó számára is hiteles és rokonszenves képet rajzolnak az informatikus mindennapokról.

Horváth László
igazgató

Tartalomjegyzék

Tanulmányok 85

Húszéves az Eötvös József Collegium Informatikai Műhelye

Kozma László

ELTE Informatikai Kar**

kozma@inf.elte.hu, drkozmalaszlo48@gmail.com

Húsz évvel ezelőtt az ELTE Informatikai Kar első dékánjaként abban a megtisztelő feladatban volt részem, hogy bábáskodhattam az Eötvös József Collegium Informatikai Műhelyének létrehozásában is. Az elismerés azonban a Collegium akkori vezetését és Csörnyei Zoltán tanár urat, a frissen megalakult Informatikai Kar docensét illeti. Csörnyei tanár úr az Informatikai Műhely első vezetőjeként hatalmas munkát végzett, hogy – új színt hozva a tudományszakok közötti együttműködésbe – az Informatikai Műhely zökkenőmentesen illeszkedjen a Collegium évszázadnál régebbi hagyományaihoz, amelynek egyik fontos eleme Horváth László igazgató úr szavaival élve az „interdiszciplinaritás".

Az Informatikai Műhely nem minden előzmény nélkül jött létre. Az ELTE Természettudományi Karán 1972-ben megindult a programozó matematikusok képzése. A számítógépek szédítő gyorsasággal hódították meg a világunkat. A hatalmas csarnokokban elhelyezhető elektroncsöves gépeket pár évtized alatt leváltották a személyi számítógépek és a szinte minden elektronikai eszközbe integrált informatika. A nagy számítógépes hálózatok világméretű kiépülésével az internet hatalmas távlatokat nyitott az információáramlás számára. Mára az egyes tudományterületek kivétel nélkül a számítógépek felhasználóivá váltak. Nem véletlen, hogy a Csörnyei tanár úr által szerkesztett színvonalas

** Dékán: 2003–2012.

kiadványban [1], amely a tízéves ELTE Eötvös József Collegium Informatikai Műhelye előtt tiszteleg, számos olyan tanulmányt találunk, amely más tudományok és az informatika határterületén folytatott kutatások eredményeit mutatja be. Jó példák erre például a következők.

Lócsi Levente, aki programtervező matematikus hallgatóként az Informatikai Műhely négy alapító tagjának egyike volt, tanulmányában [1.7] tömör összefoglalását adja doktori kutatómunkájának a racionális függvényrendszerek alkalmazási lehetőségeinek vizsgálatáról a jelfeldolgozás területén, különös tekintettel az EKG-görbék analízisére. Az orvosok számára fontos eredménye az, hogy egy igen elegáns leírásmódját adja meg az EKG-görbéknek, biztató numerikus eredményekkel alátámasztva. A tanulmányában érintett témakörökben további kutatási irányok nyílnak meg mind a matematikusok, mind pedig a mérnökök számára, amelyek további segítséget nyújthatnak az orvosoknak a megbízható diagnózis felállításához.

Kovács Péter, az Eötvös József Collegium diákjaként tanulmányában [1.6] a Collatz-sejtés matematikai és informatikai megközelítésének elemzését végezte el. Lothar Collatz német matematikus 1937-ben fogalmazta meg sejtését, melyre a cikk közléséig nem volt ismert bizonyítás, de számítógépes algoritmusok általában több évi futtatásával részeredmények születtek, teljes körű eredményt azonban nem hoztak.

Nádor István, a Collegium diákja cikkében [1.8] arról értekezik, hogy az önszerveződő kritikus rendszerek az idegtudományban komoly szereppel bírnak, hiszen az agy mint rendkívül komplex rendszer, számtalan módon mutatott kritikusságra utaló jeleket. A szerző tanulmányában az alapfogalmak tisztázása után a szakirodalom alapján ismertet egy neurális modellt, a Critical Oscillations modellt.

Csörnyei Zoltán műhelyvezetői feladatai mellett az ELTE Informatikai Karán, valamint Erdélyben, a Babeș–Bolyai Tudományegyetemen is oktatott. Nagyon jó kapcsolatokat épített ki a hazai és az erdélyi társegyetemek szakkollégiumaival. Az évente megrendezett Eötvös Konferenciák Informatikai Szekcióiban rendszeresen szerepeltek vendégelőadók a társegyetemekről. Az ünnepi kiadványban is több tanulmányt ők jegyeznek.

Bodó Zalán, a Babeș–Bolyai Tudományegyetem, Matematika és Informatika Kar Farkas Gyula Szakkollégiumának tanáraként „Gépi tanulás gráfokkal” című tanulmányában [1.3] a gépi tanulásban is hasz-

nálatos három, gráfokat használó algoritmust mutat be. A *klaszterezés* felügyelet nélküli tanulást jelent, azaz nincsenek tanulási példák, melyek megmondanák, hogy adott bemenetre mi legyen a kimenet. Az *osztályozás* felügyelt tanulást jelent, vagyis a rendszer (adat, címke) tanulási példákon keresztül tanulja meg, hogy adott bemenetre (adat) mi legyen a kimenet (címke). Ebben az esetben elterjedt osztályozási módszer a Laplace-típusú regularizált legkisebb négyzetek módszere. A *címkepropagálás* a félig-felügyelt tanulás egy tipikus példája. Ez utóbbi két esetben is a megoldások kulcsa a Laplace-mátrix. Ezért ezt a speciális mátrixot sokszor a gráf alapú tanuló módszerek egyik központi fogalmaként definiálják.

Kovács Lehel István, a Sapientia Erdélyi Magyar Tudományegyetem Kiss Elemér Szakkollégiuma képviselőjeként tanulmányában [1.5] arról értekezik, hogy az ember által készített mesterséges objektumok könnyűszerrel modellezhetők fotorealisztikusan számítógépen, hiszen nem egyet már eleve számítógépen terveztek meg. A nagy kérdés a természet alkotta tájak, élőlények, kövek, sziklák stb. modellezése. Ebben segítenek a fraktálok. A cikkben a szerző rendre bemutatja a felhők, fák, bokrok, vízfelületek, hegyes tájak, domborzatok generálását, modellezését.

Grósz Tamás, a szegedi Eötvös Loránd Kollégium hallgatója és Tóth László, az MTA-SZTE Mesterséges Intelligencia Kutatócsoport munkatársa tanulmányában [1.4] bemutat négy új tanítási módszert a mély neuronhálókhoz, majd a mély neuronhálókra épülő akusztikus modelleket beszédfelismerési kísérletekben értékelik ki.

Az Eötvös József Collegium diákjai hagyományosan mindig bekapcsolódtak az életterük szépítésébe, használhatóbbá tételébe. Lócsi Levente beszámolója szerint [1.1] a műszaki beállítottságú hallgatók már a kétezres évek elején önállóan oldották meg a Collegium számítógépes hálózatának és rendszerének üzemeltetését. A saját fejlesztésű különböző honlapok nagyon gyorsan megjelentek a Collegiumban. Ezek egyszerre szolgálták a Collegiumon belüli kommunikációt, valamint a Collegium megjelenését és vizuális reprezentációját a külvilág, a világháló felé. Az évek során több honlap is megjelent, különféle funkciókat ellátva, az egy-két oldalból álló weblapoktól egészen a komplett portálokig. Például az Aktuális portál a közérdekű hírek nyilvánosságra hozásának, a Forum Collegii portálnak pedig közösségformáló szerepe

volt azzal, hogy viták folytatásának színtereként üzemelt [1.1]. Az internet gyors elterjedése kikényszerítette a fizikai hálózat korszerűsítését is. Erről számol be Cséri Tamás, az Eötvös Collegium diákjaként az „Egyszer volt, hol nem volt, volt egyszer az Üvegszál" című írásában [1.2]. A szerző nagyon érzékletesen, időnként a humort sem mellőzve írja le, hogy milyen lépésekben haladva, ezer akadályt leküzdve, jutnak el az ADSL-vonalra épülő internettől az üvegszálas internet ünnepélyes átadásáig 2012. február 28-án. A legtöbb munkafázist a hallgatók maguk végezték el, miközben az ELTE és a Collegium vezetői intenzíven keresték a forrásokat a továbblépésekhez. Az Informatikai Kar olyan számítógépekkel segítette a Collegiumot, amelyeket az oktatásban már nem tudtak használni a gyors elévülésük miatt.

Engedjék meg, hogy a Collegium többi műhelye közül egyet említsek meg, amelyben személyesen érintve van a családom. Judit lányom bölcsész hallgatóként 2001-től volt a Collegium tagja. Magyar–német szakon folytatta tanulmányait. Rajta keresztül egy diák szemével is betekintést kaphattam a Collegiumban folyó komoly szakmai munkába. Judit sokirányú érdeklődésű, így került kapcsolatba a csillagászattal. Olvasva a csillagászati ismeretterjesztő cikkeket feltűnt neki, hogy a csillagászati objektumok elnevezései körül enyhén szólva némi káosz uralkodik. Komolyabban kezdte beleásni magát a témakörbe nyelvészeti szempontból. Kiderült, hogy olyan súlyú témát talált, amely PhD-témának is megfelel. Keményen dolgozva 2014-ben sikeresen megvédte disszertációját [2], amelynek témája a tulajdonnevek helyesírása a csillagászati és az űrtani szaknyelvben. A Collegium mellett inspirációt a témához a Kerekerdő Egyesületben nyaranta végzett önkéntes munka során szerezte, ahol általános iskolai tanulóknak tartott előadásokat autodidakta módon – sokszor szakértő hiányában, szükségből – növényekről és csillagászatból.

Az elmúlt húsz év alatt életünkben, így az Eötvös Collegium életében is sok változás történt. Sokan nyugdíjba vonultunk, így Csörnyei tanár úr is, aki után az Informatikai Műhely vezetését Kozsik Tamás, az IK docense vette át. Ő töretlenül folytatta a megkezdett munkát. 2023-tól Kozsik Tamás az Informatikai Kar dékáni feladatait látja el, a Collegium Informatikai Műhelyének élére pedig Lócsi Levente került, akinek személyisége, oktatási tevékenysége és kutatói teljesítménye biztosítékot jelent arra, hogy az Eötvös József Collegium Informatikai

Műhelye tovább halad a jól bevált úton. Az újabb tíz év alatt az informatikai alkalmazások sokszínűsége tovább gyarapodott, hiszen ezen idő alatt a hardver fejlődése töretlen volt, a szoftver esetében pedig például a mesterséges intelligencia és az adatbányászat területein elért újabb tudományos eredmények új alkalmazások sokaságának nyitottak teret. Biztos vagyok benne, hogy ezek jótékony hatással voltak és lesznek az Informatikai Műhely tevékenységére az interdiszciplinaritás tekintetében is. Ehhez kívánok sok sikert, további szép eredményeket az Eötvös József Collegium és benne az Informatikai Műhely minden tagjának.

Hivatkozások

[1] Csörnyei Zoltán, *Tízéves az ELTE Eötvös József Collegium Informatikai Műhelye (2004–2014)*, Felelős kiadó: Horváth László, az ELTE Eötvös József Collegium igazgatója, 2014.

[1.1] Lócsi Levente, Az infrastruktúra fejlődése, pp. 24–34.

[1.2] Cséri Tamás, Egyszer volt, hol nem volt, volt egyszer az Üvegszál, pp. 35–48.

[1.3] Bodó Zalán, Gépi tanulás gráfokkal, pp. 61–78.

[1.4] Grósz Tamás, Tóth László, Mély neuronhálók a magyar nyelvű beszédfelismerésben, pp. 139–154.

[1.5] Kovács Lehel István, Fotorealisztikus 3D grafika: számítógéppel generált tájak, pp. 208–221.

[1.6] Kovács Péter, A Collatz-sejtés matematikai és informatikai megközelítéseinek elemzése, pp. 235–247.

[1.7] Lócsi Levente, Racionális függvényrendszerek és alkalmazásuk a jelfeldolgozás területén, pp. 286–304.

[1.8] Nádor István, Önszervező kritikus rendszerek az idegtudományban, pp. 320–352.

[2] Kozma Judit, *Tulajdonnevek helyesírása a csillagászati és az űrtani szaknyelvben,* Doktori disszertáció, ELTE BTK Mai Magyar Nyelvtani Tanszék, Budapest, 2013.

Visszatekintés

Csörnyei Zoltán*

Eötvös József Collegium**

csz@inf.elte.hu

Először ennek a kis írásnak a *Visszaemlékezés* címet akartam adni, de gondolom, hogy ezt az anyagot elsősorban matematikus lelkületű informatikusok fogják olvasni, ők lehet hogy arra fognak gondolni, hogy az *emlékezés* egy múltbeli esemény felidézése a jelenben, és a *vissza* igekötő ezt az irányt éppen megfordítja, a *vissza-emlékezés* így éppen a *jövőbe látást* jelenti.

De félre a tréfával, visszatekintve az Informatikai Műhely első tíz évére, ennek a korszaknak egyik fő jellemzője egy görög betű, a lambda volt. A „tízéves könyv" címlapján is ez szerepel, ezt kaptam emlékül üvegbe vésve a tízéves ünnepségen, és például az egyik collegista is lambda-matricákat hozott ajándékba egy konferenciáról. Hogyan került ez a betű a Collegium Informatikai Műhelyébe? Erről szeretnék írni.

A történet visszanyúlik az 1990-es évek elejére. Akkor fejeztem be a több éve tartó müncheni munkát, és hazatérve azt a feladatot kaptam, hogy a *Fordítóprogamok* tantárgyat tanítsam. Első anyagként Varga László, Fülöp Zoltán és Hunyadvári László könyveit, jegyzeteit használtam, amelyek a fordítás klasszikus algoritmusait írták le. A következő müncheni út – most már kirándulás – alkalmával elmentem a Müncheni Műszaki Egyetem könyvesboltjába, ahol döbbenten láttam, hogy a könyvespolcon majdnem egy métert foglaltak el a különböző programnyelveken írt programok fordításával foglalkozó szakkönyvek, az

* ELTE Informatikai Kar

** Az Informatikai Műhely alapító vezetője: 2004–2015.

assemblerektől kezdve az akkor még újnak számító LALR(1) algoritmusú compilerekig. Ezek közül válogatva, és itthon ezeket feldolgozva alakult ki a Fordítóprogramok tantárgy anyaga [1].

(Egy megjegyzés: nemrégen egy gyógytornász jött hozzánk, egy fiatal hölgy. Amikor először bejött, azzal kezdte, hogy „el-a-el-er-egy", nem tudja, hogy ez mit jelent, de amikor a bátyja megtudta, hogy hová jön, ezeket a betűket neki meg kellett tanulnia. Ugyanis a bátyja az ELTE-re járt 2000 környékén, és én ilyeneket tanítottam neki.)

A funkcionális programnyelvek oktatása mindig is nagy hangsúlyt kapott a tanszékünkön, talán éppen a hollandiai, nijmegeni Radboud Egyetemmel fennálló jó kapcsolatunk következtében, ahol a nagy sikerű Clean funkcionális programnyelvet és annak implementációját dolgozták ki. Az imperatív programnyelvek fordítási algoritmusainak feldolgozása után felmerült az a kérdés, hogy hogyan kell lefordítani, hogyan lehet a számítógépen futtatni egy funkcionális nyelven megírt programot? Ennek a témakörnek a feldolgozása lett a következő izgalmas feladat.

Itt jelent meg először a *lambda*. Ugyanis a legelső sikeres funkcionális programnyelv, a Prolog fordítása úgy történt, hogy a programozó által írt programot átalakították a lambda-kalkulus egy lambda-kifejezésévé, majd ezt a kifejezést redukálták a redukciós algoritmusokkal addig, amíg csak lehetett, és a kapott kifejezés lett a program futásának az eredménye [3].

A lambda-kalkulus [2] megismerésével egy új világ tárul ki az informatikusok előtt. A kalkulus nem is „mai dolog", kidolgozásához kapcsolódó néhány – talán legismertebb – név a múlt századból: Alonso Church, Haskell B. Curry, Kurt Gödel, Stephen C. Kleene, Robin Milner, Alan M. Turing.

A lambda-kalkulus gyönyörűsége például az, hogy ő is egy funkcionális programnyelvnek tekinthető, leírható vele az aritmetika, leírhatók adatstruktúrák, típusok, rekurzív függvények – ráadásul rekurzió nélkül –, a paraméterátadás, a Turing-gép is, és még tovább sorolhatnám. Sőt erről az egyszerű nyelvről az is bizonyítható, hogy Turing-teljes.

Ahogyan mondani szoktam, „minden leírható lambda-kalkulussal, amit még nem írtak le, lehet, hogy éppen most írják". A típus is csak *illúzió*, mindennek egy vagy akár több lambda-kifejezés feleltethető meg, a típusértékek, konstruktorok, típusműveletek mindegyike egy egyszerű lambda-kifejezés.

A lambda-kifejezésekhez közvetlenül is hozzárendelhetünk típusokat, így kapjuk meg a típusos lambda-kalkulust [4]. A típusos lambda-kalkulushoz kapcsolódó *típusrendszerek* használata egy viszonylag új területe a funkcionális programozásnak. A közelmúltban néhány típusrendszer különösen nagy szerepet kapott a programozásban, például a *lineáris típusrendszerek* a sokmagos programozásban, a *függő típusrendszerek* az új programozási nyelvekben.

Ezeknek a rendszereknek lényege a Robin Milner által megfogalmazott állítás, „a jól típusozott program nem fut rosszul", az ilyen program nem akad el, helyesen működik, azaz megcsinálja azt, amit a programozó leírt neki. Szerencsére a funkcionális programok típushelyessége viszonylag könnyen bizonyítható.

Befejezésül még egy kedvenc mondatom. „Figyeljék meg, az Önök unokái az Önök háta mögött majd összesúgnak: *Képzeld, a Nagymama vagy a Nagypapa tudott rossz programot írni!*" Remélem, hogy jól látom a jövőt.

A lambda-kalkulussal, típusokkal foglalkozó előadások először a Collegium tanrendjében jelentek meg, és itt a kezdetek óta folyamatosan jelen vannak, sőt részben a lágymányosi épületbe is átkerültek.

Köszönöm, hogy nem csökkent a lelkesedés a *lambda* iránt, és néhány collegistából a *lambda* lelkes tanára lett. Az elmélet fejlődése természetesen nem állt meg, és örömmel látom, fiatal tanáraink tovább folytatják, az új eredmények ismertetésével is, a húsz évvel ezelőtt megkezdett munkát.

Hivatkozások

[1] Csörnyei Zoltán, *Fordítóprogramok*, Typotex, 2006.

[2] Csörnyei Zoltán, *Lambda-kalkulus, a funkcionális programozás alapjai*, Typotex, 2007.

[3] Csörnyei Zoltán, *Funkcionális programnyelvek implementációja*, ELTE IK, Kari Digitális Könyvtár, 2010.

[4] Csörnyei Zoltán, *Bevezetés a típusrendszerek elméletébe*, ELTE Eötvös Kiadó, 2012.

Az Informatikai Műhely második évtizede (2014–2024)

Kozsik Tamás*

Eötvös József Collegium**

kto@inf.elte.hu

Az Eötvös Collegium Informatikai Műhelye 2004-ben alakult meg. A 2000-es évek eleje a hazai informatika oktatás történetében fordulópontot jelentett, ekkor váltak önállóvá a nagy tudományegyetemek informatikai karai, köztük 2003-ban az ELTE Informatikai Kara. Az oktatásban bekövetkezett szervezeti változások jól tükrözték az informatikai tudományok és általában az informatika társadalmi jelentőségének robbanásszerű növekedését, ezzel párhuzamosan pedig természetszerűleg merült fel az igény, hogy a legtehetségesebb hallgatók fejlődését segítő támogató szervezetek is létrejöjjenek. Ezek sorában kiemelkedő helyet foglal el az Eötvös Collegium Informatikai Műhelye.

A Műhely alapító vezetője *Csörnyei Zoltán* tanár úr volt, aki 2015-ben adta át a stafétát *Kozsik Tamásnak,* tőle pedig *Lócsi Levente* vette át a vezetői tisztséget 2022-ben. A Műhely tiszteletbeli elnöke *Lovász László* akadémikus professzor.

A Műhely napjainkban is hűen őrzi az EJC nagy múltú hagyományait és az alapítók elkötelezettségét aziránt, hogy a XXI. századi informatikus képzésben a legújabb elméleti és gyakorlati eredményeket ismertesse meg a jövő informatikusaival. Az Informatikai Műhely a tudós, önálló kutatásaikkal is kitűnő, hivatásukat szerető és szívvel-lélekkel végző informatikusok képzését tekinti feladatának.

A nagyszerű eredmények mögött a kis létszámú szemináriumi keretekben folyó oktatás, az Informatikai Kar kiváló oktatói gárdája,

* ELTE Informatikai Kar, dékán: 2022–

** Az Informatikai Műhely vezetője: 2015–2022.

valamint neves külsős előadók állnak, de persze a Műhely sikerének garanciáját a tehetséges és lelkes hallgatók jelentik.

A Műhely tevékenységének középpontjában a legintenzívebben fejlődő informatikatudományi területek állnak. Akár az üzleti, akár az akadémiai szféra felé indulnak el pályafutásuk során, a Műhely tagjai versenyképes tudással felvértezve, kutatási területeket integráló szemlélettel, a szakma iránti mély elkötelezettséggel lépnek ki a Collegium kapuján.

20 éves működése során a Műhely nagyszerű eredményeket tudhat magáénak, köszönhetően a tagok kiemelkedő teljesítményének az oktatás, kutatás, versenyzés, tudománynépszerűsítés területein. A hallgatók számára a Műhely tehetségük fejlesztésének komplex lehetőségeit kínálja, extrakurrikuláris tevékenységként belépnek a kutatás-fejlesztés világába, gyakran ipari partnerekkel együttműködve, jelentős számban és eredményekkel vesznek részt a tudományos diákköri munkában, de megismerkednek az oktatói munka mindennapjaival is, hiszen az Informatikai Karon rendszeresen órákat is tartanak.

A Műhely közeli kapcsolatot ápol a társ-szakkollégiummal, az ELTE Bolyai Kollégiumával, együttműködési megállapodásunk van továbbá a kolozsvári Babeș-Bolyai Tudományegyetem Farkas Gyula Szakkollégiumával, a Kolozsvári Magyar Egyetemi Intézettel, a Sapientia Erdélyi Magyar Tudományegyetem Kiss Elemér Szakkollégiumával, a Szegedi Tudományegyetem Eötvös Loránd Szakkollégiumával, valamint a Dunaújvárosi Főiskola Kerpely Antal Computers Szakkollégiumával.

A Műhely rendszeresen megjelenik standdal a Kutatók Éjszakáján – az utóbbi években általában a „Játékok és fejtörők programozási nyelvekről” cím alatt – és az Informatikai Kar nyílt napjain.

Az évente megrendezésre kerülő Eötvös Konferenciának mindig része az Informatikai Szekció is, ahol rendszerint 4–5 műhelytag előadását hallgathatja meg az érdeklődő közönség.

A műhelytagok rendszeresen eredményesen szerepelnek hazai és nemzetközi programozási versenyeken.

Kurzusok

Az Informatikai Műhely egyes óráinak anyagait egymásra építi, és a BSc–MSc képzés mindegyik évfolyamának ajánl megfelelő órákat. A Műhely tagjai kötelesek továbbá a collegiumi nyelvórákat látogatni,

valamint a Műhely szabályzatában megfogalmazott módon nyelvvizsgákat letenni. A kutatószeminárium keretében felsőbb éves collegisták két-három részes előadássorozatokat tartanak érdekes és újszerű, az egyetemen kevéssé érintett témákban, illetve a tagok saját projektjüket, kutatásukat mutatják be egymásnak. Sok collegista vett részt az Informatikai Kar K+F+I projektjeiben, ahol a legfrissebb ipari kihívásokkal is megismerkedhettek az EIT Digital innovációs projektjeiben, az Ericsson Szoftvertechnológiai Laborban és számos más projektben.

2017-től vezettük be a Programozási nyelvek kutatószemináriumot, ahol a műhelytagok saját érdeklődési területükhöz, esetleges aktív kutatásaikhoz kapcsolódó előadásokat tartanak, szakirodalmat dolgoznak fel. 2023-ra a kurzus témái már szétfeszítették az eredeti kereteket, ezért Informatikatudományi kutatószeminárium néven, kiszélesített tematikával fut tovább. A Műhely oktatási programját, illeszkedve a Collegium tagságának igényeihez, tovább szélesítettük a Programozási alapismeretek mindenkinek kurzussal, amely professzionális eszközöket ad minden collegista kezébe saját kutatási adatainak informatikai feldolgozásához.

Az Informatikai Műhely tagjai szemeszterenként változó kurzuskínálatból választhatnak. Az 1. táblázat mutatja be az előző évtizedben meghirdetett kurzusokat, tükrözve a széleskörű ismeretanyagot és magas színvonalú oktatási tevékenységet.

Tudományos diákköri tevékenység

A kari TDK-konferenciákon 2015 és 2023 között 24 műhelytag 23 dolgozattal szerepelt. Az Informatikai Műhely tagjai 3 ízben hozták el a Morgan Stanley különdíját. 7 első helyezést, 8 második helyezést, 3 harmadik helyezést, valamint 1 különdíjat nyertek el. Az OTDK-ra kijutott dolgozatok sikeressége az alábbiak szerint alakult.

A 2015. évi XXXII. OTDK-n:

- *Englert Péter:* Efficient maximum common subgraph search on molecular graphs (témavezető: Tichler Krisztián) – 2. helyezés;
- *Horváth Gábor:* Függvények modellezése globális statikus analízis céljából (témavezető: Pataki Norbert) – 2. helyezés.

Tárgy neve	Oktatók
A mesterséges intelligencia társadalmi kihívásai	Bárd Imre
Agda	Kaposi Ambrus Páli Gábor János
Funkcionális programozás 2. (Erlang)	Tóth Melinda
Kvantumszámítás-technika	Zimborás Zoltán
Haladó fordítóprogramok	Horváth Gábor
Haladó Haskell	Kovács András Páli Gábor János Podlovics Péter
Mobil rendszerek elmélete I.	Csörnyei Zoltán
Mobil rendszerek elmélete II.	Csörnyei Zoltán
Modern elméletek az informatikában I.	Csörnyei Zoltán Luksa Norbert
Modern elméletek az informatikában II.	Csörnyei Zoltán
Nevezetes algoritmusok és C++ STL	Lócsi Levente
Nyílt forráskódú szoftverek fejlesztése haladóknak	Luksa Norbert Katkó Dominik Csimma Viktor
Programfejlesztés Scalában	Kozsik Tamás
Programozási nyelvek kutatószeminárium	Kozsik Tamás
Típuselmélet	Kaposi Ambrus Csimma Viktor
Webes és mobil programozás F#-ban	Gansperger István

1. táblázat. Az Informatikai Műhely meghirdetett órái

A 2017. évi XXXIII. OTDK-n:

- *Ambrus Tamás,* Borbély Zsófia: Modelltranszformálás szoftverminőségi mutatók alapján (témavezető: Tóth Melinda) – 1. helyezés;
- *Leitereg András:* Masszívan párhuzamos architektúrák generatív programozása (témavezető: Berényi Dániel) – különdíj.

A 2019. évi XXXIV. OTDK-n:

- *Fonyó Viktória:* Pszeudovéletlen bitsorozatok konstrukcióinak gyakorlati elemzése (témavezető: Tóth Viktória);
- *Komáromi Mátyás:* Gview: Efficient graph visualisation for RefactorErl (témavezető: Tóth Melinda);
- *Leitereg András:* Masszívan párhuzamos architektúrák generatív programozása (témavezető: Berényi Dániel);
- *Luksa Norbert:* Haskell programok párhuzamosítása refaktorálással (témavezető: Kozsik Tamás);
- *Nagy Gergely,* Mészáros Áron Attila: Towards green computing in Erlang (témavezetők: Tóth Melinda, Bozó István) – különdíj;
- *Szécsi Péter:* Improved Loop Modeling in Symbolic Execution for C Family Languages (témavezetők: Porkoláb Zoltán, Horváth Gábor).

A 2021. évi XXXV. OTDK-n:

- *Kocsis Ábel:* Programok sebezhetőségének felismerése statikus kódelemzéssel (témavezetők: Tóth Melinda, Gera Zoltán);
- *Komáromi Mátyás:* Visualising software dependencies to support code comprehension (témavezetők: Tóth Melinda, Bozó István) – 3. helyezés;
- *Luksa Norbert:* Az egyszerű típuselmélet algebrai reprezentációi (témavezető: Kaposi Ambrus) – 3. helyezés;
- *Nagy Gergely:* Robust quaternion Zernike moments and their applications in color image analysis and recognition (témavezető: Németh Zsolt) – 1. helyezés;

- *Nagy Gergely,* Mészáros Áron Attila: GreenErl – Measuring the energy consumption of Erlang programs and energy conscious refactorings (témavezetők: Tóth Melinda, Bozó István) – 3. helyezés;
- *Poór Boldizsár:* Refactorings towards clean functional code (témavezetők: Tóth Melinda, Bozó István) – különdíj;
- *Varga Balázs:* Transforming concurrent Erlang programs to introduce distribution (témavezetők: Tóth Melinda, Bozó István) – különdíj.

A 2023. évi XXXVI. OTDK-n:

- *Busa Máté:* Kvaternió értékű polár-Fourier momentumok algoritmusai digitális képfeldolgozáshoz (témavezető: Németh Zsolt);
- *Szalay Gergő,* Poór Máté Bálint: Neurális kódvisszafejtés másoló mechanizmussal (témavezetők: Gregorics Tibor, Pintér Balázs) – 2. helyezés;
- *Tompos Anna:* Geometriai szemlélet fejlesztése befogadó-és gyűjtőiskolában társasjátékozással matematika órán (témavezető: Szabó Csaba).

A tudományos diákköri tevékenység intenzívebbé válása az utóbbi években azt eredményezte, hogy a műhelytagok egyre többet publikálnak tudományos konferenciákon is.

Kitüntetések, elismerések

Az Informatikai Műhely tagjainak munkáját továbbra is jól jelzi, hogy tagjaink 50 Köztársasági Ösztöndíjat, avagy 2017-től érvényes nevén **Nemzeti Felsőoktatási Ösztöndíjat** kaptak az eltelt újabb évtizedben (2. táblázat).

A kutatásokhoz sokszor elnyerték tagjaink az **Új Nemzeti Kiválóság Program** (ÚNKP) támogatását is.

Név	14	15	16	17	18	19	20	21	22	23
Busa Máté	.	.	.	.	.	.	.	■	.	.
Czirkos Angéla	.	.	.	.	.	■	■	.	.	.
Csertán András	.	.	.	.	.	.	.	■	■	■
Csimma Viktor	.	.	.	.	.	.	.	.	.	■
Englert Péter	■	.	.	.	.	.	.	.	.	.
Fonyó Viktória	.	■	.	.	■	.	.	.	.	.
Hermann Gábor	■	■	.	.	.	.	.	.	.	.
Horcsin Bálint	.	.	.	.	.	.	.	.	■	■
Horváth Gábor	■	■	.	.	.	.	.	.	.	.
Katkó Dominik	.	.	.	.	.	.	.	■	.	.
Kocsis Ábel	.	.	.	.	.	■	.	.	.	.
Komáromi Mátyás	.	.	.	.	■	.	.	.	.	.
Leitereg András	.	■	■	■	.	.	.	.	.	.
Luksa Norbert	.	.	.	■	■	■	.	.	.	.
Manninger Mátyás	■	.	.	.	.	.	.	.	.	.
Nagy Gergely	.	.	.	.	■	■	■	.	.	.
Nagy Vendel	.	■	■	■	■	.	.	.	.	.
Noszály Áron	.	.	.	.	.	.	.	■	■	.
Pitlik Mátyás	.	.	.	.	■	.	.	.	.	.
Poór Boldizsár	.	.	.	.	.	■	■	.	.	.
Szécsi Péter	.	■	■	■	■	.	.	.	.	.
Szente Péter	.	.	.	.	.	.	.	.	■	■
Szilveszter Máté	.	.	.	.	.	.	.	.	.	■
Tőkés Anna	.	■	■	.	.	.	.	.	.	.
Varga Balázs	.	.	.	.	.	■	■	■	.	.

2. táblázat. Köztársasági, illetve Nemzeti Felsőoktatási Ösztöndíj

Rendszerint minden év májusában kerül megrendezésre az Informatikai Kar ünnepe, a Neumann-nap, amikor a **Kar Kiváló Hallgatója** címet is kihirdetik a Kar legjobb tanulmányi és kutatási eredményeket, illetve közösségi aktivitást felmutató hallgatói között. Az elmúlt tíz évben összesen 30 alkalommal vehették át műhelytagok a kitüntető oklevelet (3. táblázat).

Név	14	15	16	17	18	19	20	21	22	23
Balassi Márton	■	.	.	.	.	.	.	.	.	.
Csertán András	.	.	.	.	.	.	.	■	.	.
Englert Péter	■	.	.	.	.	.	.	.	.	.
Fonyó Viktória	.	.	■	.	.	.	.	.	.	.
Gévay Gábor	.	■	■	.	.	.	.	.	.	.
Horcsin Bálint	.	.	.	.	.	.	.	.	.	■
Horváth Gábor	■	.	■	.	.	.	.	.	.	.
Katkó Dominik	.	.	.	.	.	.	.	■	.	.
Kocsis Ábel	.	.	.	.	.	■	.	.	.	.
Komáromi Mátyás	.	.	.	.	■	■	.	.	.	.
Kruppai Gábor	.	.	.	.	.	.	■	.	.	.
Leitereg András	.	■	.	■	.	.	.	.	.	.
Lövei Péter	.	.	.	.	■	.	.	.	.	.
Luksa Norbert	.	.	.	.	.	■	.	.	.	.
Manninger Mátyás	■	.	.	.	.	.	.	.	.	.
Nagy Gergely	.	.	.	.	■	■	.	■	.	.
Poór Boldizsár	.	.	.	.	.	.	■	■	.	.
Szalay Gergő	.	.	.	.	.	.	.	.	■	■
Szécsi Péter	.	.	.	■	.	.	.	.	.	.
Varga Balázs	.	.	.	.	.	■	■	■	.	.

3. táblázat. A Kar Kiváló Hallgatója

Az Eötvös Collegium kurátora az igazgató személyi javaslatai alapján a kuratórium támogatásával **Eötvös Collegiumért Emlékérmet** adományozott az Informatikai Műhely 5 tagjának, így az első évtized két műhelyünkhöz köthető emlékérmével együtt immár hétre lehetünk büszkék.

2014. szeptember 25-én Eötvös Collegiumért Emlékérmet kapott *Cséri Tamás* „távozó collegista" kiemelkedő tanulmányi eredményeiért és közösségi tevékenységéért.

2015. szeptember 3-án *Csörnyei Zoltán* tanár úr, az Informatikai Műhely alapító vezetője vehette át az emlékérmet több mint egy évtizedes műhelyvezetői tevékenységéért, a Collegium informatikus hallgatóinak oktatásában vállalt áldozatos munkájáért, az Eötvös Collegium és az Informatikai Kar kapcsolatának erősítéséért, ápolásáért.

2021. szeptember 1-jén *Luksa Norbert* kapta meg ezt a kitüntetést „távozó collegista" kategóriában. 2018 óta a méltatások elérhetők a Collegium honlapján, ezért azokat idézzük.

> Luksa Norbertet 2015-ben vette fel tagjai közé a Collegium, majd lassanként az Informatikai Műhely meghatározó személyiségévé vált. Rendszergazdaként, informatikai szolgáltatások létrehozójaként és működtetőjeként a tevékenysége összeforrt a Collegium mindennapi életével. Kiegyensúlyozott, nyugodt megfontolásai a collegisták körében gyorsan elismerést és tiszteletet vívtak ki számára, hosszú időn át valóban rátermett elnöke volt a hallgatói önkormányzatnak. Ebben a vezető szerepben egyensúlyt tudott teremteni filosz és dögész érdekek és értékek között is, amelyben bizonyosan annak is szerepe volt, hogy a bölcsész mentalitás igen közel áll hozzá. Szakmai, tanulmányi és közéleti szempontból egyaránt példaértékű collegista pályát futott be.

2022. szeptember 5-én pedig, 11 év után ismét előfordult, hogy két informatikus is egy-egy emlékérem birtokosa lett. *Katkó Dominik* mint „távozó collegista" az egyikük.

> Katkó Dominik minden idők legifjabbja azok között, akik valaha Eötvös Collegiumért Emlékéremben részesültek. Sajnos mindhiába szerettünk volna tovább várni a testületek közös lelkületével *(unanimiter)* odaítélt köszönettel, Katkó úr ugyanis peregrinusként idegen földön folytatja tanulmányait. Egyet tehetünk: biztatjuk és kérjük, tudásában, tapasztalataiban gyarapodva egykor térjen majd vissza szülőföldjére és a Collegiumba.

> Katkó Dominik 2019-től volt az Eötvös Collegium Informatikai Műhelyének tagja. Nevéhez fűződik a nagy sikerű Bölcsészinformatika kurzus bevezetése. A hallgatói választmány Kommunikációs Bizottságának elnöke volt, számos pályaválasztási és egyetemi rendezvényen vett részt, neki köszönhetünk több megújult collegiumi internetes felületet és fontos beavatkozásokat, mentéseket. Rendszergazdaként az Urán rendszer fejlesztésében is meghatározó szerepet vállalt. Mindemellett collegista társai bármikor számíthattak rá, legyen szó kirándulások, közös főzések szervezéséről, – és most idézem társai szavait –: „nagy ölelésekről, közös nevetésekről. Mindenkinek, ahogy csak tudott, segítségére volt, és részt vett a collegiumi élet valamennyi eseményén mondhatni hiánytalanul."

A másikuk pedig jómagam, amit nagy megtiszteltetésnek tartok, és őszintén köszönöm.

Közösség

A szakmai és tudományos sikerek elősegítése mellett a Collegium szerepe a közösségépítésben is rendkívüli. Számomra különösen emlékezetesek azok az alkalmak, amikor a műhelytagokkal kirándulásokon, közös ünnepléseken találkozhattam. Beszélgetős esték, Estikék… persze Infoműhelyes csaposok. Jó érzés visszaemlékezni az erdei kirándulásokra, a városnézésekre, a velencei tavi sárkányhajózásra, a kerékpártúrára a Börzsönyben, a bowlingozásra, vagy azokra a megható és vidám pillanatokra, amikor közösen ünnepeltük a karácsony közeledtét. Jó kis társaság az Infoműhely, még ha picit néha „kocka" is, még ha túl gyakran a programozásra vagy egy készülő TDK-dolgozatra terelődik is a szó – a lazulásra, a vidámságra, egymás társaságának élvezetére azért mindig kaphatók az informatikusok.

Hét éven át voltam műhelyvezető, és ez alatt a hét év alatt nagyon sokat kaptam a mindenkori műhelytagoktól. Szuper intelligens, szuper érzékeny, szuper kedves emberek, akik lelkesen viszik tovább évről évre, generációról generációra a Műhely, a Collegium hagyományait. Kívánom, hogy ez az elköteleződés a tudomány, a kultúra, a közösség és a fiatalok támogatása iránt töretlen maradjon. Vágjunk neki a következő tíz évnek! Hajrá!

1. kép. Díjazottak a Neumann-napon, 2019. tavasz

2. kép. Bográcsozás a kertben, 2019. nyár

3. kép. Bográcsozás a kertben, 2019. ősz

4. kép. Műhelytagok a góJabálon, 2022. ősz

5. kép. Csoportkép, 2023. tavasz

6. kép. Félévnyitó műhelygyűlés, 2023. ősz

7. kép. Műhelypizzázás, 2023. ősz

8. kép. Félévnyitó műhelygyűlés, 2024. tavasz

Kozsik Tamás Dékán úr az Informatikai Kar dékánhelyetteseként és a tehetséggondozásért felelős oktatójaként Csörnyei Zoltán Tanár úrtól vette át a Collegium Informatika Műhelyének vezetését. Az Informatikai Műhely puszta léte a természettudományokra szakosodott szakkollégiumok szorításában önmagában is vívmány az Eötvös Collegiumban. A Collegium fennállása óta arra törekszik, hogy a szakbarbár beszűkülés helyett a gyarapodó egyetemi karok és tudományterületek lehető legtöbbjét befogadja, és ekképpen kövesse az alapítók szándékát, az universitas meghonosítását tagságunk mindennapjaiban. Kozsik Tanár úr elődjéhez hasonlóan azonban, nem érte be ennyivel, tudniillik a Műhely puszta fenntartásával. Számos kari teendője mellett mindig szívügyének tekintette a collegiumi tehetségek gondozását, minden egyes műhelytagot féltő odafigyeléssel támogatott, tudományos és emberi előmenetelük felett jó pásztorként sáfárkodott. Az eredmények, a statisztikákba foglalt adatok önmagukért beszélnek. Tevékenységének áldásos voltát mi sem bizonyítja jobban, mint a tagság és a Collegium szeretete és hálája.

Kozsik Tamás méltatása, mely elhangzott 2022. szeptember 5-én, az Eötvös Collegiumért Emlékérem átadása alkalmával.

Programozási versenyek

Biborka Ágnes, Horcsin Bálint*

Eötvös József Collegium**

agnesbiborka111@gmail.com, horcsin@student.elte.hu

1. A versenyek

A programozási versenyek kikerülhetetlen jelenségként vannak jelen a programtervező informatikus hallgatók életében. Amikor az iskolában egy diák elkezd érdeklődést mutatni az informatika iránt, informatika tanára ezt szemfülesen kiszúrja, és a diák hamar a Nemes Tihamér versenyen találja magát, nem számít, hogy azelőtt csak Logo programokat írt, és egy „rendes" programnyelven, mondjuk Pythonban, adatokat beolvasni sem tud.

Mindenesetre, ha nem bölcsész irányból érkezik a hallgató a karra, akkor már rendelkezik több-kevesebb informatikai versenytapasztalattal, ami a teknőc mozgatástól a leghatékonyabb gráfbejárások ismeretéig terjedhet.

Minden collegistában közös, hogy szereti a kihívásokat. Természetes tehát, hogy amikor elsőéves hallgatóként progalap előadáson ülve azt látják, hogy Zsakó tanár úr lapoz egyet a diasoron, és felhívja mindenki figyelmét a Tehetségkutató egyetemi programozási versenyre, mindenki két percen belül regisztrál, és kíváncsian várja az első fordulót. A versenyen a feladatok megoldásához gyakran a programozási tételeket kell használni, de a nehezebb feladatok megoldásához ismerni kell többek között a dinamikus programozást és a gráfbejárásokat is. A versenyen

* ELTE Informatikai Kar

** Biborka Ágnes 2022–, Horcsin Bálint 2021–

kiemelkedő eredményt elért versenyzők részt vehetnek a Sapientia ECN programozási versenyen Marosvásárhelyen.

Egy másik, közkedvelt és hatalmas presztízsű verseny az International Collegiate Programming Contest, röviden ICPC. Itt a versenyzők háromfős csapatokban indulnak, és öt órájuk van arra, hogy megoldjanak tizenhárom feladatot. A feladatok változó nehézségűek, a legkönnyebbel majdnem mindegyik csapat boldogul, míg általában van egy-két feladat, amit a legjobb csapatok sem tudnak megoldani. A verseny során csak egy számítógép áll a csapat rendelkezésére, amelyen kódolni és feladatot beadni lehet, ezért a verseny papíros tervezés, csapatmunka és időbeosztás szempontjából is kihívást jelent.

Ízelítőként a versenyek stílusából, következzen a szegmensfák rövid ismertetése és egy hozzájuk kapcsolódó versenyfeladat.

2. Szegmensfák

Bináris keresőfákat számos helyen alkalmaznak, amelyeknél jellemzően hasznos, ha kiegyensúlyozottak (például AVL-fák, piros-fekete fák). Ezekkel az önkiegyensúlyozó bináris keresőfákkal megoldható, hogy a fa mélysége legalább $\lfloor \log_2(n) \rfloor$, és legfeljebb $c \cdot \log(n)$ legyen, ahol n az érintett fa csúcsainak darabszáma, c pedig az érintett implementációra jellemző konstans. Ezeken a fákon bizonyos lépések (például beszúrás, elem keresése, elem törlése, intervallumon összegzés) lépésszáma arányos a fa mélységével.

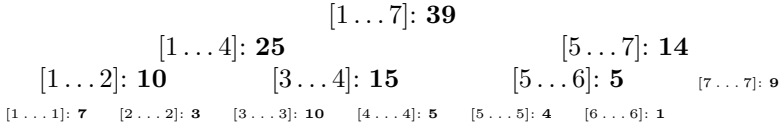
Ugyanakkor ezeket versenyeken ritkán szokták alkalmazni az implementációs nehézségek miatt. Alább egy jól ismert, gyakran alkalmazott adatszerkezetet mutatunk be, amelyek implementációja rövid, ami miatt gyakran használják programozásversenyeken.

A szegmensfák [1] célja, hogy sorban 1-től N-ig sorszámozott $x_1, x_2, \ldots, x_N$ elemek vannak, és a következő műveleteket lehet végezni:

- adatszerkezet felépítése előre megadott N darab elemből ($O(N)$ lépésben);
- elem beállítása valamilyen elemre ($O(\log(N))$ lépésben);
- adott $[a \ldots b]$ intervallumon ($a, b \in \mathbb{N}$, $1 \leq a \leq b \leq N$) az $\sum_{i=a}^{b} x_i$ összeg meghatározása ($O(\log(N))$ lépésben).

Az összegzés lehet tetszőleges művelet, amely asszociatívan végezhető az elemeken. Vagyis egész számok esetén lehet akár összeadás, vagy szorzás. Amennyiben szükséges szorzatot számolni, akkor jellemzően a szorzat egy viszonylag nagy méretű prímszámmal (pl. $10^9 + 7$) vett osztási maradékát kell eredményként megadni, amit el lehet végezni úgy, hogy minden szorzás után elvégezzük az osztási maradék meghatározását, és azzal folytatjuk a számolást.

A szegmensfa alapelve az, hogy tárolja az összeget az $[1 \dots N]$ intervallumra, és ezt követően *oszd meg és uralkodj* módszerrel „felezgeti" az intervallumot egészen addig, amíg egyelemű nem lesz az intervallum. Ezekre az intervallumokra is tárolja az összeget. Vagyis ha egy $[a \dots b]$ intervallumra (a, $b \in \mathbb{N}$, $1 \leq a < b \leq N$) tárolva van az összeg, akkor egy $k := \left\lfloor \frac{a+b}{2} \right\rfloor$ választásával tároljuk az összeget az $[a \dots k]$, valamint a $[k+1 \dots b]$ intervallumra.



1. ábra. A 7, 3, 10, 5, 4, 1 sorozathoz tartozó szegmensfa.

A teljes $[1 \dots N]$ intervallum legfeljebb $\lceil \log_2(n) \rceil$ alkalommal történő felezésével egyelemű intervallum keletkezik. Egy sorozatban történő elem módosításához így legfeljebb $\lceil \log_2(n) \rceil$ darab intervallumhoz tárolt összeget kell módosítani.

A felezgetés során létrejön egy fa szerkezet, amelyben minden intervallumhoz tartozik egy csúcs. Egy $[a \dots b]$ intervallumhoz tartozó csúcsnak (a, $b \in \mathbb{N}$, $1 \leq a \leq b \leq N$) nincs gyereke, ha $a = b$, és ha $a < b$, akkor az intervallumhoz tartozó $k := \left\lfloor \frac{a+b}{2} \right\rfloor$-val gyerekei az $[a \dots k]$, valamint a $[k+1 \dots b]$ intervallumokhoz tartozó csúcsok.

Összesen $2N - 1$ intervallum keletkezik tekintve, hogy egy olyan bináris fáról van szó, amelynek N levélcsúcsa van, és minden csúcsnak pontosan nulla vagy kettő eleme van.

Egy adott $[a \dots b]$ intervallumra a következő kóddal lehet meghatározni az összeget. Az `összeg(1,N)` meghívásával kapjuk meg az eredményt. A függvény célja, hogy megadja, hogy egy adott $[bal \dots jobb] \cap [a \dots b]$ intervallumon mennyi az elemek összege.

```
függvény összeg(bal, jobb):
    ha [bal ... jobb] ∩ [a ... b] = ∅:
        összeg := 0
    ha [bal ... jobb] ⊂ [a ... b]:
        összeg := [bal ... jobb]-hoz tartozó összeg
    különben:
        k := ⌊(a+b)/2⌋
        összeg := összeg(bal, k) + összeg(k+1, jobb)
```

Bizonyítható, hogy $O(\log(N))$ lépésben meghatározható ily módon az összeg.

Ugyanakkor sok esetben az adatszerkezet módosítása szükséges, amelyben egy adott leképezés művelettartását kell kihasználni (ezt *lazy propagation*-nek hívják). Ilyen feladat a KöMaL S. 144.-es feladata [2], amelyet a következő fejezetben részletezünk.

3. A versenyfeladat

A feladat során sorba vannak kockák helyezve, és meg kell határozni a kockák térfogatai összegének $10^9 + 7$-tel vett osztási maradékát. Egy lépés során egy $[a \dots b]$ intervallumon kell tudni megváltoztatni a kockák élének a hosszát egy adott y értékkel.

Ekkor több adatot is kell tárolni az egyes csúcsokban. Nevezetesen azt, hogy mennyi a hozzá tartozó intervallumban lévő kockák éleinek az összege, hogy mennyi a hozzá tartozó intervallumban lévő kockák élei négyzetének az összege, és hogy mennyi a hozzá tartozó intervallumban lévő kockák térfogatának az összege, valamint egy *lusta* értéket. Ez a *lusta* érték azt fejezi ki, hogy egy adott intervallumon lévő kockák éleinek a hosszát még mennyivel kell megnövelni.

Egy intervallumon történő módosításhoz meg kell növelni pontosan azokon az intervallumokon a *lusta* értéket, ahonnan az összegek számolódnának, ezen intervallumok őseit neutrálissá célszerű tenni, és őseikben frissíteni kell az értékeket (a gyerekeik értékeiből az összegek triviális módon számolhatóak). Szükséges definiálni egy olyan *push* műveletet, amely képes a *lusta* értéket neutrálissá tenni az $[a \dots b]$ intervallum számára. Ehhez a többi értéket újra kell tudni számolni, és a gyerek csúcsok *lusta* értékeit növelni a csúcs *lusta* értékével, majd az $[a \dots b]$ intervallumra *lusta* := 0-t beállítani.

Így tetszőleges tárolt intervallumon meg lehet őrizni, hogy rajta és leszármazott intervallumain mennyivel kell majd még növelni a kockák éleinek a hosszát, oly módon, hogy az ősökben ezt a módosítást már végrehajtottuk.

Az új értékek a következő módon számolhatóak (amennyiben x_{regi_i}-vel jelöljük az i-edik kocka éleinek módosítás előtti hosszát, x_{uj_i}-vel pedig az i-edik kocka éleinek módosítás utáni hosszát):

$$\sum_{i=a}^{b} x_{uj_i} = \sum_{i=a}^{b}(x_{regi_i} + lusta) = \sum_{i=a}^{b}(x_{regi_i}) + (b-a+1)\cdot lusta;$$

$$\begin{aligned}\sum_{i=a}^{b} x_{uj_i}^2 &= \sum_{i=a}^{b}(x_{regi_i} + lusta)^2 = \sum_{i=a}^{b}(x_{regi_i}^2 + 2\cdot x_{regi_i}\cdot lusta + lusta^2) = \\ &= \sum_{i=a}^{b}(x_{regi_i}^2) + \sum_{i=a}^{b}(2\cdot x_{regi_i}\cdot lusta) + \sum_{i=a}^{b} lusta^2 = \\ &= \sum_{i=a}^{b}(x_{regi_i}^2) + 2\cdot lusta\cdot\sum_{i=a}^{b}(x_{regi_i}) + (b-a+1)\cdot lusta^2;\end{aligned}$$

$$\begin{aligned}\sum_{i=a}^{b} x_{uj_i}^3 &= \sum_{i=a}^{b}(x_{regi_i} + lusta)^3 = \\ &= \sum_{i=a}^{b}(x_{regi_i}^3 + 3\cdot x_{regi_i}^2\cdot lusta + 3\cdot x_{regi_i}\cdot lusta^2 + lusta^3) = \\ &= \sum_{i=a}^{b}(x_{regi_i}^3) + \sum_{i=a}^{b}(3\cdot x_{regi_i}^2\cdot lusta) + \\ &\quad + \sum_{i=a}^{b}(3\cdot x_{regi_i}\cdot lusta^2) + \sum_{i=a}^{b} lusta^3 = \\ &= \sum_{i=a}^{b}(x_{regi_i}^3) + 3\cdot lusta\cdot\sum_{i=a}^{b}(x_{regi_i}^2) + \\ &\quad + 3\cdot lusta^2\cdot\sum_{i=a}^{b}(x_{regi_i}) + (b-a+1)\cdot lusta^3.\end{aligned}$$

Az élek új hosszáról felírt összeg, négyzetösszeg, köbösszeg kifejezhető volt a korábbi összegből, négyzetösszegből, köbösszegből, a-ból,

b-ből, valamint a *lusta* értékből konstans időben, így a *push* művelet implementálható, így tetszőleges $[a \dots b]$ intervallumon tudjuk a kockák éleinek hosszát változtatni egy adott értékkel, majd tetszőleges $[a \dots b]$ intervallumon meg tudjuk határozni a kockák térfogatainak összegét.

Hivatkozások

[1] *Algorithms for Competitive Programming,* Segment Tree.
`https://cp-algorithms.com/data_structures/segment_tree.html`

[2] Ratkó Éva, Schmieder László, Busa Máté, Csertán András, Farkas Csaba, Fodor Zsolt, László Nikolett, Lóczi Lajos, Siegler Gábor, *Középiskolai Matematikai és Fizikai Lapok,* S. 144. feladat (2020. május).
`https://www.komal.hu/feladat?a=feladat&f=S144&l=hu`

A 2023-as Nemzetközi Informatikai Diákolimpia (IOI 2023)

Noszály Áron*

Eötvös József Collegium**

noszalyaron4@gmail.com

1. Bevezető

A programozási versenyek megrendezése hosszas előkészítési munkát igényel. Ki kell találni a feladatokat, ki kell dolgozni és tesztelni őket, biztosítani kell a megfelelő technikai feltételeket (értékelőrendszer és munkaállomások technológiai része), az egyéb szervezési feladatokról nem is beszélve. Zsakó László tanár úr hívására az egyetemre érkezve rögtön be is kapcsolódhatunk ebbe a munkába a középiskolásoknak szóló országos versenyek esetén (Nemes Tihamér, OKTV II. kategória). Illetve a közelmúltban Nikházy László kezdeményezésére elindult a Kódkupa csapatverseny, aminek szervezésében is többen részt vesznek az egyetemről és a Collegiumból is.

Ugyanakkor egy nemzetközi verseny még bonyolultabb szervezést igényel. 2023-ban a világ legrangosabb középiskolásoknak szóló informatikai versenye Szegeden került megrendezésre. Ennek részleteiről szeretnék röviden írni, és bemutatom egy általam kidolgozott feladat megoldását.

* ELTE Informatikai Kar
** 2020–

2. Hogyan készült?

Egy IOI, *International Olympiad in Informatics* szervezéséhez több bizottság közös munkája szükséges (IC, HSC, ISC, HTC és ITC). Én a HSC, azaz a *Host Scientific Committee* (Hazai Tudományos Bizottság) tagja voltam[1], így ennek a munkájába nyújthatok betekintést. A feladatunk főként a kiválasztott javaslatok kidolgozása, és ezután pedig a tesztelésük volt. Vezetőnk Németh Zsolt volt, aki az ISC, azaz az *International Scientific Committee* (Nemzetközi Tudományos Bizottság) tagja is.

A HSC számára a munka 2023 januárjában kezdődött. A nemzetközi közösség által beérkező feladatjavaslatokból ki kellett válogatnunk bizonyos szempontok alapján azokat, amelyek érdemesek további vizsgálatra, majd a Winter Meetingen, ami február végén volt, az ISC felé prezentálni. Az ő feladatuk itt főként az volt, hogy a két versenynap feladatsorát, illetve a tartalék feladatokat is kiválasszák. Ebben mi is közreműködtünk, de a végső szó az övék volt.

Miután lezárult a Winter Meeting, megkezdődött a feladatok kidolgozása, felosztva magunk között a munkát. Először egy angol nyelvű statementet (leírást) kellett elkészítenünk minden feladathoz, annak minden elemével: ábrák, pontos definíciók stb. Ezután kellett a megoldásokat megírnunk C++-ban, és a tesztadatokat generáló programokat Pythonban. Majd teszteltük az egymás által kidolgozott feladatokat: nincs-e olyan megoldás, ami több pontot kap mint kellene, érthető-e a szövegezés, stb.

Ez több hónapon keresztül tartott, majd augusztus 28-án, vasárnap Szegedre utaztunk, elkezdődött, és egy héten át tartott a 2023-as IOI. A versenyhelyszín a Pick Aréna volt. A feladatokkal kapcsolatos teendők mellett egyéb szervezési dolgokban is segítettünk: például pakoltunk a versenyteremben, és nyomtattuk a feladatsorokat. A feladatsorokat az egyes versenynapok előtt az ún. GA-nak (General Assembly, az országok delegációinak vezetőiből álló közgyűlés) jóvá kellett hagyni. Itt, mivel mi ismertük a feladatokat a legjobban, segédkeztünk az ISC-nek a csapatvezetők által felmerülő kérdések megválaszolásában, illetve a leírások végső finomításában. A versenynapokon pedig mind a versenyteremben, mind pedig az ún. Control Roomban felügyeltük a versenyt.

[1] Csertán András és Szente Péter mellett, valamint Horcsin Bálint a HTC tagja volt.

Én az utóbbiban voltam mindkét alkalommal, itt az értékelőrendszeren keresztül érkező kérdésekre kellett válaszolnunk, illetve a versenyzők által beküldött megoldásokat kellett monitoroznunk. Valamint az internetre streamelt élő közvetítésben is el kellett mondanunk a feladatok megoldását.

Személy szerint én nagyon élveztem az egész hetet. Bár elég fárasztó volt a sokszor éjszakába nyúló munka, de érdekes volt megtapasztalni a versenyt a másik oldalról is. Középiskolásként volt szerencsém versenyzőként részt venni több IOI-n is, de ez a munka egészen másfajta izgalmat jelentett.

3. Soccer feladat

Ez volt az első nap harmadik feladata. Egy japán ikerpár javasolta, és rögtön megtetszett nekem, amikor elolvastam a javaslatot. Ezért én prezentáltam az ISC-nek is a Winter Meetingen, és utána én is dolgoztam ki. A feladat hivatalos leírása elérhető a verseny honlapján[2]. Következzen egy rövidített leírás.

3.1. Leírás

Adott egy $M \in \{0,1\}^{N \times N}$ mátrix, amire $1 \leq N \leq 2000$. A mátrix celláinak egy K részhalmaza *szép*:

- ha minden $(i,j) \in K$-ra $M_{i,j} = 0$,
- és bármely két különböző $(a,b) \in K$ és $(c,d) \in K$ cella legfeljebb két *rúgás* távolságra van.

Ahol (a,b) és (c,d) egy *rúgás* távolságra van, ha:

- $a = c$ vagy $b = d$,
- és minden (x,y) cellára, amire $a \leq x \leq c$ és $b \leq y \leq d$, $(x,y) \in K$.

Illetőleg (a,b)-től két *rúgás* távolságra pedig nyilván a tőle egy *rúgás* távolságra lévőktől egy *rúgás* távolságra lévők vannak.

A feladat hogy meghatározzuk $|K|$ maximális értékét, amire K *szép*.

[2] A következő URL-en: `https://ioi2023.hu/tasks/`.

3.2. Részfeladatok

Az IOI-n fontos részei a feladatoknak az ún. subtaskok, azaz részfeladatok. Ennek a feladatnak is több különböző részfeladata van, melyekre külön-külön lehet pontot szerezni:

1. (6 pont) Legfeljebb egy 1-es értékű cella van;
2. (8 pont) $N \leq 3$;
3. (22 pont) $N \leq 7$;
4. (18 pont) $N \leq 30$;
5. (16 pont) $N \leq 500$;
6. (30 pont) Nincs további megkötés.

Illetve egy adott részfeladat pontszámának negyede jár arra, ha a versenyző megoldása el tudja dönteni helyesen a részfeladat összes tesztjén, hogy az összes 0-s cella által alkotott részhalmaz *szép*-e.

3.3. Megoldás

Az első részfeladatban a (legfeljebb) 4 darab L-alakú sarokban lévő *szép* részhalmaz egyike lesz a megoldás, ezek közül kiválaszthatjuk a legnagyobb elemszámút egyszerű számolásokkal.

A második részfeladatban az összes részhalmazok száma legfeljebb $4^3 = 512$, és egy részhalmazban legfeljebb $\frac{9 \cdot 8}{2}$ pár van, melyekből egyet, hogy két *rúgás* távolságra vannak-e, legegyszerűbb módon 3^2 műveletbe telik ellenőrizni. Ez belefér, így egy $\Theta(4^N \cdot N^6)$-os megoldásunk van.

A harmadik részfeladatban a kevés lehetséges *szép* részhalmaz miatt működik az, hogy visszalépéses keresést alkalmazunk, és heurisztikákkal, ha már biztosak vagyunk benne hogy az aktuális részhalmaz nem lehet *szép*, akkor visszalépünk.

A továbblépéshez kis kitérőt érdemes tennünk a eldöntési részfeladatra. Ez segít klasszifikálni a *szép* részhalmazok tulajdonságait, és így a későbbi, gyorsabb megoldások megtalálásban is. Meggondolható, hogy az alábbi feltételek szükségesek ahhoz, hogy egy részhalmaz *szép* legyen:

1. Egy sorban/oszlopban vagy egyetlen cellát se tartalmaz a részhalmaz, vagy egy intervallumban tartalmazza a cellákat.

2. Sorokat nézve, ezekből az intervallumokból bármely kettőre igaz, hogy vagy az egyik tartalmazza a másikat, vagy fordítva.

3. A soronként kiválasztott cellák száma unimodális (először monoton nő, aztán monoton csökken).

Ezek elégségessége is belátható. Így az eldöntési feladatot $\Theta(N^2)$ időben megoldhatjuk.

A negyedik részfeladat megoldására egy, az eldöntési részfeladatban megismert tulajdonságokat kihasználó, dinamikus programozási megoldást fogunk adni. Legyen $F(D, U, L, R)$ a maximális mérete annak a *szép* részhalmaznak, aminek a D-edik vagy U-adik sorában az $[L; R]$ intervallumban vannak kiválasztott cellák. Tranzícióként léphetünk eggyel feljebb az $F(D, U-1, L', R')$ állapotba, illetve eggyel lejjebb az $F(D+1, U, L', R')$ állapotba egy-egy megfelelő (azaz 1-est a megfelelő sorban nem tartalmazó) $[L', R'] \subset [L, R]$ intervallum esetén. Az állapotaink száma legrosszabb esetben N^4, és minden tranzíciónál $O(N^2)$-nyi munkát végzünk, így meggondolható, hogy a megoldásunk komplexitása $\Theta(N^6)$. Ami hatalmas javulás, hiszen ez már polinomiális N-ben.

Az ötödik részfeladatban továbbfejlesztjük az előző dinamikus programozási megoldást. A fontos gondolat az, hogy nem szükséges az összes szép részhalmazt klasszifikálnunk, hanem elég csak azokat, amik lehetnek maximális méretűek. Ezzel az ötlettel az előző megoldás tranzícióiban már rögtön elég lenne $O(N)$ részintervallumot végignéznünk az $O(N^2)$ helyett. A másik fő gondolat, hogy elég csak egy sorindexet tárolni, a (legalsó) leghosszabb intervallum soráét. Így egy $F(M, L, R)$ függvényt kell kiszámolnunk, ami a maximális méretű *szép* részhalmaz méretét adja meg, aminek az M-edik sorra „vett vetülete” az $[L; R]$ intervallum. Ez most úgy tűnhet elsőre, hogy egy $\Theta(N^4)$-es megoldás, de valójában az (M, L, R) hármasokból elég $O(N^2)$-et figyelembe vennünk. Ennek belátásához szükséges ismerni a *largest zero submatrix* probléma négyzetes idejű megoldását, ennek ismertetésétől most eltekintek.

Az utolsó részfeladatban az előző megoldási módszert tudjuk még egy észrevétellel gyorsítani. Valójában nagyon hasonló módon látszik az előző $O(N^2)$-es becsléshez, hogy azon $((M, L_1, R_1), (M, L_2, R_2))$

($L_1 \leq L_2 \leq R_2 \leq R_1$) pároknak a száma is $O(N^2)$, amiket az előző algoritmus megvizsgált. Így a potenciális intervallumokat lineáris keresés helyett binárisan keresve egy $\Theta(N^2 \log N)$ idejű megoldást kapunk, ami elég hatékony, hogy mindegyik részfeladatot megoldja.

Érdekesség, hogy létezik egy $\Theta(N^2)$-es megoldás is, amit a versenyzők fedeztek fel, bár mi is sejtettük, hogy létezik.

Sárkány és papucs

A rendszergazdaság 2013 és 2023 között

Csimma Viktor*

Eötvös József Collegium**

csimmaviktor03@gmail.com

1. Rendszergazdák

A Collegium rendszergazdasága hivatalos módon nagyjából a kilencvenes évek közepe óta létezik. [15] Megtaláltam egyszer a választmányiban Parragi Zsolt felkérését „első rendszergazdának", 2009. november 30-i dátummal, igazgatói szignóval. [1] Ez gyakorlatilag egy kis munkaköri leírásnak is tekinthető; engedjétek meg, hogy viszonylag sokat idézzek belőle:

> *„Jelen levelemmel – korábbi felkérésemet visszavonva – felkérem, hogy lássa el az Eötvös József Collegiumban az »első rendszergazda« feladatait. A Collegiumban három rendszergazdai jogkörrel rendelkező collegista tevékenykedik, akik a számítógépes hálózathoz teljes hozzáféréssel rendelkeznek. [. . .] Felkérésem a Collegium hálózatának működtetésére vonatkozik, amely magában foglalja*
>
> - *A lakó- és műhelyszobákban (beleértve az EC munkatársainak irodáit) az internethozzáférés zavartalan biztosítását.*

* ELTE Informatikai Kar
** 2021–

- *A Collegium zavartalan levelezésének biztosítását.*
- *A Collegium informatikai eszközeinek üzemeltetését és ellenőrzését.*
- *A Collegium nyomtató rendszerének működtetését.*
- *A Collegium honlapjának frissítését, karbantartását [...], amely nem terjed ki alprogramok saját erőből történő fejlesztésére.*

[...] Jelen felkérés egyszersmind garanciát jelent – amennyiben a Collegium közössége számára e felkérés tartalmán túl szoftvereket fejleszt –, hogy a rendszergazdák által fejlesztett szoftverek tulajdonjoga a rendszergazdáknál marad, ugyanakkor a Collegium azokra használati jogot kap kilencvenkilenc évre.[1] *[...] A rendszergazdák mindennapi munkájának szervezésére a Diákbizottságot kérem fel."*

Ma már nem kapunk ilyen hivatalos kinevezést, de a feladatok nagyjából ugyanezek maradtak. Aki tehát rendszergazda lett, az ezeket vállalta magára.

Az eddigi rendszergazdák (legalábbis akiket én és Levente összegyűjtöttünk) névsora; tényked̈esük hozzávetőleges idejével [2, 10, 12]:

- Weisz Csaba [15]
- Horváth Péter
- Mizera Ferenc
- Stefán István (?–2004)
- Novák Ádám (?–2004)
- Romsics Bence (2004–2007)
- Simon Győző (2004–2005)
- Czigola Gábor (2005–2007)

[1] Ez nem véletlenül szerepel itt. Volt némi bonyodalom az „új Aktuális" rendszerrel kapcsolatban, miután az akkori rendszergazdák nem tudtak megegyezni az új igazgatóval annak áráról és tulajdonjogáról. [9]

- Sztupák Sz. Zsolt (2007–2009)
- Parragi Zsolt (2007–2009)
- Ölvedi Tibor (2009–2011)
- Cséri Tamás (2010–2013)
- Hapák József (2011–2014)
- Kovács Máté (2014–2017)
- Luksa Norbert (2015–2018)[2]
- Kocsis Ábel (2018–2019)
- Molnár László (2019–2021)
- Bondici László (?)[3]
- Katkó Dominik (2020–2022)[4]
- Szajbély Sámuel (2020–2022)
- Szabó Barbara (2022–)[5]
- Csimma Viktor (2022–)
- Sulan Ádám (2022–)
- Horcsin Bálint (2023–)
- Kámán Rebeka (2023–)
- Bertalan Dániel (2023–)

[2] Az Uránnal egészen 2021-ig dolgozott.

[3] Állítása szerint ő csak „besegített” [2]; de elég sok számlán van a neve ahhoz, hogy felírjam.

[4] Már 2019-ben uránozott; a fejlesztést most is ő felügyeli.

[5] Jelenleg külföldön van; de visszacsatlakozik, ha hazajön.

2. Szerverek

A Collegiumnak már nagyon régóta vannak saját szerverei; ezeken futott a honlapok szignifikáns része, az interneteléréshez szükséges dolgok (DHCP, Radius etc.), a nyomtatás és régen a webtárhely is. Teljesen visszakövetni az elmúlt tíz év fejlődését nem tudtam, de itt van, amit találtam.

2.1. A 2015-ös állapot

A rendszergazdai Drive [8] tanúsága szerint akkoriban három szerver működött:

- DELL: a DHCP, a DNS, a tűzfalak és a levelezőszerver tartoztak ide.
- TARDIS[6]: itt honlap- és nyomtatószerverek működtek.
- TARDISNÉ: itt futott egy másodlagos DNS-kiszolgáló, egy webszerver és a *hapakj-4dstudio* (ezt valószínűleg csak Józsi tudja, micsoda).

Ezeken a gépeken futtattak *sok* virtuális szervert. A 2015. szeptemberi beszámolóban [5] szerepelt is, hogy ebből hogyan terveznek nyesni, illetve hogy nem sikerült beszerezni új gépet (pedig kértek rá 2500 Ft-os extrát; cserébe akik adtak, attól következő félévben nem kértek netreget).[7]

2.2. A Smaug

Egy évvel később, a 2016. szeptemberi beszámolóban [5] ezt találjuk:

> *„A szervereinket próbáljuk a lehető legjobb módon kihasználni, így az egyik eddig nem használt szervert felújítottuk, a*

[6] A Doctor Who-ban az idő- és utazógépet hívták így.

[7] Azt is itt említik, hogy az internetkábelek „spontán” eltűnnek, és ez nem jó. Volt olyan ötlet, hogy betétdíjjal adják ki a kábeleket; vagy csak simán hagyják az egészet, és mindenki hozzon saját kábelt. Valószínűleg végül az utóbbi megoldás győzött, de a vezeték nélküli hálózat terjedésével és gyorsulásával ennek egyébként is csökkent a szerepe.

> *következőkben erre próbálunk rendszert építeni. [...] A tervünk az, hogy egy fizikai szerveren lenne egy Ubuntu-szerver, ami a Collegium minden igényét ki tudná szolgálni. Ehhez az előző részben említett szervert már elkezdtük az igényeinknek megfelelően telepíteni és konfigurálni."*

A számlák [6] tanúsága szerint az egyik processzort Máté még 2016 augusztusában rendelte Olaszországból, 10,69 euróért. A 2017. márciusi leírásokban [7] szerepel először a név.[8]

Még valami a beszámolóból: *„Egyetlen problémás terület a nyomtatás, eddig a legvalószínűbb az, hogy egy gyengébb szerveren [...] Windows-nyomtatószerver fog üzemelni, mivel az kompatibilis a legtöbb felhasználói eszközzel."* Ez valószínűleg TARDISNÉ lett, mert 2017-ben még szerepel az IP-címe az akkori táblázatban. [7] Arról nem esik szó, hogy őt mikor kapcsolták le; mindenesetre mi már nem találkoztunk vele.

A szerverről szóló szövegrész még a 2017. februári beszámolóban is szinte ugyanaz. Miután Máté távozott, a beszámolókat nem vezették és tárolták rendszeresen; emiatt már nem tudtam tovább követni az események alakulását.

Mindenestre a Smaug jelenleg a Collegium fő (és egészen idáig egyetlen) szervere. Az aktuális konfiguráció a következő:

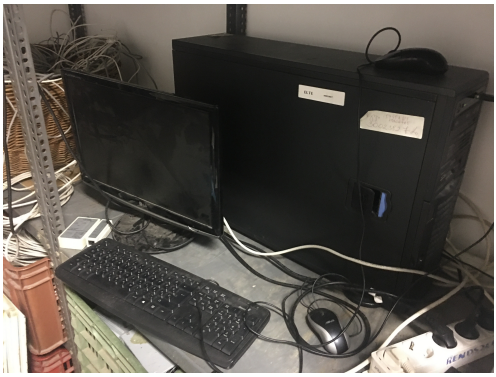
- Alaplap: Supermicro X7DWA;
- CPU: két darab Intel Xeon E5405;
- Memória: 8 GiB (2×4 egészen pontosan);
- Lemezek: 2 db 2 TB-os WDC WD20EZBX-00A (RAID 1);
- Operációs rendszer: Ubuntu 22.04.

Mivel a 2 TB-ot alig használjuk ki,[9] gondolkodunk azon, hogy egyszerűen 256 GB-os SSD-kre térünk át.

[8] Smaug Tolkien *A hobbit* című regényének főgonosz sárkánya, aki egy hegy gyomrában őrzi a törpéktől rabolt vagyonát. A 2013-as (utolsó) Forbes Fictional 15 listán a második helyen szerepelt 54,1 milliárd dollárral, Dagobert bácsi után.

[9] Régen a tárhely fontosabb volt; ugyanis minden collegistának járt egy nagy webtárhely. Akkoriban a floppyk, CD-k, DVD-k, pendrive-ok drágák voltak. [10]

Ami pedig fut rajta jelenleg: a belső honlapok (Urán, Collegium Nostrum, Collegialia, meg pár wordpresses dolog, amit át kéne rakni a Caesarra), a Radius-szerver (ez kontrollálja az internet-hozzáférést), illetve a DHCP.



Házisárkányunk természetes élőhelyén

Smaug hosszú, fekete; különös ismertetőjele, hogy világoskék a fogantyú, amivel le lehet venni az oldalát.[10] Kíváncsi lennék, vajon melyik szervert újíthatták fel.

A szerverterem valamikor 2020–2021 környékén átkerült a TMK műhelye mögül a választmányi irodába.[11] Azóta az éppen aktuális első rendszergazda is kap egy választmányi kulcsot; ez szolgál hatalmi szimbólumként.[12]

A 2022. nyári leállás

Az történt, hogy a RAID 1-be kötött merevlemezek közül az egyik elhalálozott.

[10] Ez egyébként szintén kapcsolódik a regényhez: a sárkánynak egyetlen gyenge pontja volt, egy páncélozatlan folt a hasán.

[11] Ez ma inkább raktárként, mint irodaként szolgál. Most különösen a rezsiválság idején lehordott mikrók foglalnak sok helyet, de már az én érkezésemkor sem volt irodaformája a teremnek.

[12] Nekem egyszer sikerült elvesztenem. Aztán kiderült, hogy abban a fiókban van, ahol eredetileg is kerestem; csak nem túrtam át kellően alaposan, és elbújt a sok mappa mögött.

Meg kellett nézni a szervert, és Barbara levitt magával, hátha tudok segíteni. Az Ubuntu azt mondta rajta, hogy futtassak `fsck`-t, amit megtettem, és hirtelen újra működött a gép. Nagy örömmel felmentünk.

Másnap megint rossz volt. Leültünk, és kiderült, hogy több kárt csináltam, mint amennyi hasznot, mert az `fsck` után hardveresen nem javult meg a merevlemez, csak még több információ elveszett.[13] Ültünk, ültünk; nagy nehezen kiderítettük, hogy a két winchester közül az egyik rossz, de már a másikról sem tudtunk igazán adatot kinyerni (talán pont az `fsck` miatt). Szóval újratelepítés...

Először egy darab winchestert vettünk; a másik egy raktárban talált volt. Nem értettük, hogy miért olyan lassú a gép; aztán vettünk inkább még egyet és megint újraraktuk a rendszert. Akkor már jó volt.

Gondoltuk, hogy az addigi Ubuntu-verzió (16.04!) helyett felrakjuk a 22-eset. Csak az volt a baj, hogy itt elég sok minden változott; bevezettek egy új konfigfájl-formátumot az internetnek, amivel elég sokat szenvedtünk. De csak nem akart működni.

Végül az lett a vége, hogy Samu megkérdezte Kovács Mátét, aki két kérdéssel megoldotta a problémát:

– *Milyen disztribúciót használtok?*

– *Ubuntu 22.04-et.*

– *És kikapcsoltátok az ufw-t*[14] *?*

– *Hátööö...*

Tervben volt, hogy Mátét meghívnánk korrepetálást tartani Linux-szerverekből, de aztán elhalt az ötlet.

A történet nyomai:

- Megtanultunk pár dolgot. Például:
  - Ne `fsck`-zz ész nélkül! Ne, ne, ne.
  - Ha `iptables`-t használsz tűzfalnak, kapcsold ki az `ufw`-t!
  - Kommunikálj az emberekkel! Kívülről azt látták, hogy három nap kiesést ígértünk, de lassan már egy hete nem történt semmi. Ha szóltunk volna nekik, hogy ez bonyolultabb, mint gondoltuk, és el fog húzódni, ők is jobban viselték volna.

[13] Ne `fsck`-zz, ha arra gyanakszol, hogy hardveresen tönkrement egy merevlemez!
[14] Az Ubuntu beépített tűzfala.

- Elkezdtünk Markdownban kis útmutatókat írni (erről később lesz még szó).
- Rendszergazda lettem.

Átszervezés, Docker-konténerek

Ez nem olyan régen kezdődött el. November 13-án találtunk egy csúnya kriptobányász kártevőt a szerveren, amit leszedtünk. De ennek kapcsán el kellett gondolkodni, hogyan lehetne az ilyesmit megakadályozni.

Bálint találta ki aztán, hogy Docker-konténerekkel újraszervezi a szerver szolgáltatásait. Külön konténerbe került a weboldalak frontendje és backendje, illetve az Apache-t Nginx-re cserélte. Szerepelt a Laravel-dokumentációban, hogy a korábbi Apache PHP modulja helyett PHP-FPM kiszolgálók bevezetésével szebb lesz a világ; ezt is megcsinálta. A laraveles alkalmazásoknál bevezette a gyorsítótárazást (amitől tényleg érezhetően gyorsabb). [2]

Újratelepíteni is sokkal egyszerűbb lesz majd, mert a `docker-build` gyakorlatilag telepít mindent, ami az adott részhez szükséges, és a `docker-compose` fájl tartalmazza, hogy pontosan milyen szolgáltatásoknak, milyen módon kell működnie.

Az interneteléréshez szükséges kritikus szolgáltatásokat nem tervezzük Dockerben elhelyezni, hogy a Docker-daemon esetleges problémái esetén is működőképes maradjon a collegiumi internet.

Bertalan Danival közösen készítettek egy Raspberryből egy nyomtatószervert, amihez a központi szerver képes csatlakozni, mivel a nyomtatót korábban nem csak a központi szerverről lehetett elérni. Ez egyelőre még kicsit instabil, de kicseréltük az ősi USB-kábelt, és azóta jónak tűnik.

Ezen túl egy másik Raspberryre[15] pedig telepítettek Radius- és DHCP-szervert; gyakorlatilag egy komplett tartalékgépről van szó, ami legalább az internetelérést tudja biztosítani, ha a Smaug kiesne.

Annyi teendő még mindenképpen van, hogy a pontos változásokat le kell dokumentálni, a biztonsági mentések készítését át kell gondolni, és még be kell tanulnia a többi rendszergazdának (nekem is). Ez terv szerint a télen meg fog történni.

[15] Kéne valami jó nevet adni neki.

3. Internet

Az elmúlt tíz év történetéből találtam pár fontosabb dolgot ezzel kapcsolatban. Az egyik a lakószinteken levő WiFi-hálózat újraépítése. A 2016. szeptemberi beszámolóban [5] merült fel először az ötlet, hogy a kevés erős access point helyett sok gyengébbet helyezzenek ki, a folyosók helyett a szobákba. Az év decemberéből találtam egy számlát [6], amin összesen 15 darab router szerepel kétszázezer forint értékben; nagyjából akkor építhették ki a rendszert. Vannak vele problémák, mivel ezeket otthoni felhasználásra, és nem ilyesmire tervezték: némely router időnként kiesik, megbolondul, kihúzzák.[16] De alapvetően működik.

Egy másik a Radius használata. Szintén akkoriban merült fel (bár még 2017. februárban is csak terv volt), hogy a rendszerhez ne egyetlen jelszó tartozzon, hanem az akkor készülő Urán felhasználóihoz lehessen rendelni internet-hozzáféréseket.[17] Valószínűleg az Urán elindulásával történt ez meg ténylegesen.

Időről időre felmerült az Informatikai Igazgatóság részéről, hogy a Collegiumban is az eduroam hálózat működjön, és így az ELTE-s felhasználónév-jelszó párral lehetne belépni. Mivel ennek a költségeit azonban nem vállalták, és az irányításba így a collegisták nem tudtak volna beleszólni, ez jellemzően elutasításra került. Kovács Máté 2017-es memoárjaiban [8] szerepelt a felvetés, hogy az alagsori WiFi akár működhetne az eduroam alatt: nem tűnt nagy költségnek, és így biztosítottnak látszott az oktatók hozzáférése is. Végül ez megtörtént; valószínűleg nem sokkal később.

A lakószinteken azóta is a rendszergazdák üzemeltetik és szabályozzák a vezeték nélküli hálózatot. 2022 szeptemberében felmerült, hogy pályázati pénzből ki lehetne építeni az eduroamot,[18] de ebből végül nem lett semmi.

Nemrég kérte az IIG, hogy az IP-címeket terelhessék át privát tartományokra (jelenleg az ELTE 157.181. kezdetű blokkjából kapunk tel-

[16] Már velem esett meg egyszer, hogy az üresen álló 221-es szobában állandóan kikapcsolt a router. Végül kiderült, hogy a takarítónő húzta ki mindig gondosan, mert azt hitte, egy figyelmetlen vendég hagyta ott.

[17] Ennek sok előnye van: lehet a collegiumi státusztól és a netreg befizetésétől függővé tenni az elérést; illetve elvben azt is le lehet követni, ki művel csúnya dolgokat az interneten (például ki torrentezik).

[18] Igazgató Úr ötlete volt, hogy gyorsan be lehetne szerezni az eszközöket, mielőtt a rezsiválság miatt beszüntetnék az ilyen programokat.

jesen nyilvános címeket). Ez okozni fog bonyodalmakat; valószínűleg télen vagy nyáron fogunk ezzel foglalkozni.

Néha a mi hibánkon kívül is kiesik az internet. Az egyik szélsőséges példa talán 2016 őszén volt, amikor a Bibó felújításánál véletlenül elvágták az optikai kábelt, amit csak három hét után sikerült megjavítani. Ekkor szegény collegisták mindenfélével próbálkoztak; Norbi elmondása szerint az IIG-s net lopása és a Mekibe költözés volt a legjellemzőbb; de volt, hogy a körtéren a hajléktalanok mellett találtak ingyen WiFi-t. [12]

A másik eset 2017. májusi. Ekkor Máté írt egy levelet Igazgató Úrnak [16], hogy valakik egy felújításkor a harmadik emeleti padláson darabokra vágták a kábeleket, és elvittek három switchet. Így gyakorlatilag tönkretették a WiFi-t az egész lakószinten. Jelezték a kivitelezőnek; arról nem találtam információt, hogy végül hogy oldódott meg a probléma.

Hogy az ilyen helyzetekben is tudjunk segíteni, idén tavasszal beszereztünk Barbarával és Ádámmal közösen egy 4G-modemet, amit ilyen esetekben ki lehet tenni a Társalgóba. Nagyon sok embert nem tud kellően gyorsan kiszolgálni, de vészhelyzet esetén egy éjszakai tanulásra jó lehet. Mindazonáltal a mobilinternet-hozzáférés terjedése is enyhítette a problémát: ma már a korlátlan adatforgalom sem ritkaság.

4. Honlapok

Ahogy korábban szerepelt, a honlapokat és a DNS-szervert 2015 őszén helyezték át az ELTE szervereire (ez valószínűleg a Caesart jelenti). Azóta viszont bonyolódott a szituáció.

4.1. Fő honlap

A Levente által leírt „X-es oldal” a Wayback Machine szerint 2016 nyaráig működött. [14] Akkor egy világos és narancs színekkel operáló felület váltotta fel, melynek menürendszere már nagyjából a mai honlapét követte. Az áttérést Nagy Vendel irányította. [12]

Valamikor 2019 végén vagy 2020 elején [14] a honlap átkerült az ELTE közös Loginet nevű rendszerébe (ezzel egy időben rövidült a `honlap.eotvos.elte.hu` cím egyszerűen `eotvos.elte.hu`-ra). Samu szerint ez azért volt így, mert az elavult felületet gyakran feltörték,

emiatt az IIG rendszeresen letiltotta. Az áttérés viszont azt is jelentette, hogy a kódot alapvetően már nem mi kontrolláljuk, csak a tartalmat tudjuk módosítani egy kezelőfelületen. Ennek minőségét erős kritikák érték és érik most is; nem alaptalanul... Előfordul például, hogy ha túl gyorsan ment az ember (és nem vált át grafikus módba a szerkesztőmezőnél), varázslatos módon eltűnnek a változtatások.

A másik probléma, hogy a kinézet az ELTE honlapjainak közös arculatát nagyon szigorúan és rugalmatlanul követi (például a Collegium logóját sem lehetett volna hivatalosan kihelyezni). Erre a szemfüles collegisták azt a megoldást ötölték ki, hogy minden egyes tartalom elejére beillesztenek egy kódot (`collegium_style.html`), ami *megvulkanizálja* a honlapot, hogy kellően colis kinézetű legyen. Ezt a mai napig így csináljuk. Persze mindig, amikor az ELTE arculatot vált, ronda lesz, és mindenhol újra kell írni; ennek automatizálására viszont már írtam egy Python-szkriptet.

A fenti problémák miatt időről időre felmerül, hogy újra saját gondozásba kéne venni a honlapot. Azonban meglehetősen sok tartalom került fel azóta, amiket nehéz volna lementeni, és egy új rendszerbe átvenni. Egy másik oka is van a maradásnak: a honlap egyszerűen *szép;* jól illeszkedik az egyetemi arculatba, miközben kerülőúton mégis piszkálhatjuk a kinézetét.

A karbantartás alapvetően úgy néz ki, hogy vannak emberek, akiknek van fiókjuk a kezelőfelülethez (ez eredetileg a rendszergazdákat jelentette; most már igyekszünk választmányosoknak is kérni, ha igényük van rá). A honlap szerkesztése az egyik leggyakoribb témája az igazgatósággal történő kommunikációnak; gyakran kérnek meg minket, hogy eseményeket, kiadványokat feltöltsünk.[19] A műhelyhonlapok frissítéséhez jelenleg a szakmai alelnök gyűjti be a szükséges adatokat a műhelytitkároktól, amiket aztán a rendszergazdák visznek fel.

4.2. Collegialia

A Collegialia a 125. évfordulóra készült,[20] a Collegium régi PDF-anyagait gyűjti. Megtalálhatóak itt például a *Ménesi út* és az *Ahol a maximum volt a minimum* című legendás történeti írások. Nagy része

[19] Jelenleg ezt Rebi szokta intézni.

[20] Bár az első commitok a repókon 2021-esek; valószínűleg GitHubra csak ekkor lettek feltöltve.

az itteni anyagoknak csak a Collegium belső hálózatáról érhető el; ez egyelőre az IP-címek szűrésével működik. Mindazonáltal az igazgatóság szeretné korlátozni a hozzáférést a szerzői jogi viták elkerülése érdekében; valószínűleg jelszóalapon fog ez történni.

Az eredeti munka Mészáros Lacié volt, és valójában nem is egy, hanem három komponensből áll: egy API-ból, egy nyilvános frontendből és egy admin frontendből (React- és Express-alapon). Utóbbi egyébként *Smaug* néven fut, és igazából más funkciója is volt eredetileg: lehetett a levelezőlistákat innen kezelni. Sőt valójában most is lehet; csak zavar, hogy nem lehet a „moderált" pipát bepipálni, mint a Mailman felületén.

4.3. Collegium Nostrum

Ez a Kollégium/Collegium újjászervezésének 65. évfordulójára készült; gyakorlatilag egy alumni-adatbázis. Jórészt Barbara és én írtuk (Barbara kezdte 2022 őszén, és 2023-ban, az évnyitón mutatták be); a gyűjtést pedig a töriműhelyesek vezették. A projekt most is él; van egy külön kutatószeminárium is a bővítésére, ha jól tudom, Plangár Sanyi vezetésével.

5. Az Urán (jelenlegi tervek)

Az EIR és az Urán történetét Dominik foglalja össze egy külön cikkben [3]; én inkább arra térnék ki, hogy az ő távozása óta hogy néz ki a fejlesztés.

Mivel az elmúlt időszakban először a Collegialia és a Collegium Nostrum, majd a szerver újrastrukturálása elvonta a figyelmet, ezért sajnos nem jutott annyi figyelem az Uránnak, mint amennyit megérdemelne. Jelenleg 54 nyitott issue[21] szerepel a repón; ezek közül 50 legalább 1 hónapos, és 46 nincs egy fejlesztőhöz sem rendelve (vagyis senki nem dolgozik rajta). Egy szó, mint száz; elég rosszul állunk.

Én a félévben 6 ügyön dolgoztam; főleg olyanokon, amik a Választmány kéréseihez kapcsolódtak. De jórészt csupán szinten tartottuk a rendszert, ami baj, mert így mi is kiesünk a gyakorlatból, és más sem fog tudni becsatlakozni. Ezzel mindenképpen kezdenünk kell valamit;

[21] Ez a GitHub – és általában a projektmenedzsment-eszközök – terminológiája szerint valamilyen ügy, amivel foglalkozni kell. Például egy hiba vagy egy új funkció.

különben azt kockáztatjuk, hogy az EIR sorsára jut a projekt: fejlesztők, fejlesztések híján elavulttá válik.

A jelenlegi terv az, hogy a tavaszi félévben tartunk egy kurzust az érdeklődő elsőéveseknek az Urán fejlesztéséről; olyasmit, amilyet Dominik és Norbi is tartott nekünk. Most Rebin próbálom ki az okítást; az éles kurzuson valószínűleg a Plútót fogjuk újra befogni, illetve igyekszünk lassan, strukturáltan haladni. Meglátjuk, hogy sikerül.

6. Belső kommunikáció

6.1. Levelezés

Levelezőlisták

A levelezőlistákat az ELTE üzemelteti; `Mailman` alapúak. Egy kezelőfelületet kapunk, amin fel lehet venni, törölni embereket.

A nagyobb listák:

- *eotvos-membracollegii:* Ez a régi Aktuális funkcióját követi. Nagyrészt jelenlegi collegisták a tagjai; ide mennek ki a hivatalosabb közlemények (például választmányi meghívók, szobaszemlehirdetések, gyászjelentések). Csak az igazgatóság jóváhagyása után kerülnek kiküldésre az üzenetek.

- *eotvos-epistolacollegii:* A Kommunikációs Bizottság hírlevele (gyakorlatilag egy apró havilap) számára van fenntartva.

- *eotvos-alumni*: A nevében benne van a lényege. Erre vonatkozik a legtöbb fel- és leiratkozós kérés. Ezt a listát is az igazgatóság szerkeszti.

- Az Estikének is van levelezőlistája, amire üzeneteket lehet küldeni (például tematikapályázatokat). Elvben Andris szerkeszti, de ide jóváhagyás nélkül is megjönnek a levelek.

A kevésbé fontos listák (sok már nem is nagyon él):

- *cathedraestudiosi*

- *eotvos-magistri*

- *eotvos-korus*
- *eotvos-termtudtabor-szervezo*
- *eotvos-nemec*
- *eotvos-muhelytitkar*
- *eotvos-mindenki*
- *eotvos-matfiz*
- *eotvos-infomuhely-belso*
- *eotvos-infomuhely.*

Speciális címek

Sok megosztott postafiók van speciális címmel. Ez azt jelenti, hogy ELTE-s felhasználókat hozzá lehet venni/törölni belőle, és ők akkor látják a beérkező leveleit, tudnak a nevében üzenetet küldeni. A mienk a *root@eotvos.elte.hu*; a legkorábbi levél, amit találtam, 2014. szeptemberi keltezésű. [16] Van a választmányi elnöknek és az Eötvös Konferencia szervezőinek is megosztott postafiók.

A fiókot – mint általában az ELTE-s e-mail-fiókokat – mobilon az Outlookból lehet legegyszerűbben elérni. Viszont folyamatosan problémák vannak azzal, hogy nem kapunk rendesen értesítéseket (pontosabban néha igen, néha nem). Van pár elméletünk, mi lehet a baj;[22] de valójában nem tudjuk.

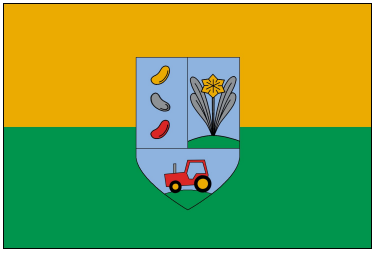
Régen lehetett bárkinek `eotvos.elte.hu` végű címet kérni, ami egy alias volt az egyetemi fiókra. Azóta elhalt a dolog; legalábbis nekem már nincs hozzáférésem a felülethez. De igazából újra lehetne életre kelteni; írtam is az IIG-nek ezzel kapcsolatban (még ha nem is a világ legfontosabb dolga ez).

[22] Az is felmerült, hogy a legutóbbi ránézés után egy ideig kapsz; de az is, hogy az energiatakarékos módtól függ.

6.2. Facebook, Discord

Annak idején volt hagyománya a Collegiumban az internetes fórumozásnak, amely eredetileg egy *Forum Collegii* nevű honlapon, majd a wiki-alapú oldal egy hasonló nevű felületén folyt. Utóbbi a `wiki.eotvos.elte.hu` és `wiki.eotvoscollegium.hu` címek alatt volt elérhető,[23] de 2015 környékén teljesen megszűnt.[24]

Miközben nézegettem a mentéseket, megtaláltam az *Eötvös MGTSZ*[25] zászlaját; úgy érzem, muszáj beillesztenem ide:



Az Eötvös MGTSZ zászlaja

Az informálisabb/beszélgetősebb kommunikáció nagyon sokáig az *Eötvös Collegium* nevű Facebook-csoportban zajlott, és zajlik részben igazából ma is. Érdemes visszatekinteni a régi bejegyzésekre; sok mindent találni az akkori fontos eseményektől ittas collegisták nemkívánatos folyosóhasználati szokásaiig. [11] 2018-ban az egykori tagokkal közös csoport is indult *Báró Eötvös József Collegium* néven.

Tudtommal a Facebook egyeduralma csak nemrég tört meg. Eredetileg a járvány alatt létesült először Discord-szerver, *Coli Digital* néven. Ez kifejezetten informális, beszélgetős volt; alapvetően azt a célt szol-

[23] Egy ideig az `eotvos.elte.hu` és az `eotvoscollegium.hu` címek is használatban voltak. Máté 2015 júliusában egyezett meg az igazgatósággal, hogy az utóbbit egy türelmi idő után lekapcsolják. [16]

[24] Az utolsó mentés a Wayback Machine-en 2015. július 6-i dátumú. [14]

[25] Levente 2011-es cikkéből [9]: *„[Az EMGTSZ] különböző tevékenységein keresztül a falusi származás és életforma nem megvetendőségét kívánta hangsúlyozni és tagjai közt a baráti kapcsolatokat szorosabbá fűzni. Egyes collegisták szerint a valaha volt legértelmesebb collegiumi szervezet, mások szerint viszont csak egy balkonláda hagyma termése köré épülő diktatórikus neokommunista társulás."*

gálta, hogy a Collegiumban kialakult barátságok valamelyest tovább tudjanak élni ilyen formában.[26] Elsősorban a 2019-es évfolyam legendás alakjait találhatjuk meg itt, különböző furcsa/vicces dolgokat (vagy éppen eddig számomra is ismeretlen collegiumtörténeti érdekességeket) posztolva. Volt egy külön csatorna az *Alszik a Coli* játéknak, ahol a szabályokat is gyűjtötték; a hangcsatornákon pedig sok azóta híres párkapcsolat kezdeti stádiumai játszódtak le. A 2021-es újranyitáskor azonban a szerver kiesett a használatból, és néhány `!cutya`-üzenetet[27] leszámítva mára gyakorlatilag teljesen elhalt. (A főnek tekinthető csatornára az utolsó üzenetet 2021. július 9-én küldték.)

Ezután egy ideig újra csak a Facebook-csoport volt használatban. A jelenleg használt *Eötvös Collegium* Discord-szerver 2022 novemberében indult; nagyrészt Sulan Ádám (részben pedig Szente Peti és Tompos Anna) szervezésével. Ezt már kifejezetten hivatalos és félhivatalos kommunikációra szánták; csatornákba szervezve a hirdetményeket, a nemhivatalos dolgokat, a bizottságokat. Akár a régi fórum helyét is átvehetné a gondolatmenetekkel, de nem igazán alakultak ki ilyen jellegű beszélgetések (leszámítva talán a 2023-as választmányi programok alatti Q&A-szekciókat).

2023 szeptemberében az akkori Választmány megpróbálta határozottabban terelni a collegistákat a Facebook helyett a technikailag és funkcióiban fejlettebb Discord felé terelni: a góJákat csak a Discord-szerverre vették fel, a Facebook-csoportba nem. Ez a lépés megosztotta a collegistákat: sokak ugyanis továbbra is kizárólag a Facebookot használták, és úgy érezték, lemaradnak fontos információkról; mások viszont úgy érezték, a Facebooknak csak hátránya van a Discordhoz képest.[28] Végül a következő Választmány leállította a kísérletet.

Azóta a két információcsatorna párhuzamosan üzemel. Egy felmérés szerint van, aki a mai napig csak a Facebookon tájékozódik; de aki használ Discordot, az általában a Facebook-csoportot is követi. Jelenleg

[26] Ugyanebben az időszakban (is [a szerk.]) Minecraftban reprodukálták a Coli épületét. Állítólag még a Rumadait is szimulálták dugattyúkkal.

[27] Ez egy robot volt, aminek ha beírtad, hogy `!cutya`, küldött egy cuki kutyás képet. Az `!aww`-ra pedig cuki macskás képet.

[28] A jellemző panaszok: nem küld értesítéseket, egyetlen csatorna van, nehezebben visszakereshető, és úgy általában nem a hatékony csoportkommunikációra tervezték. Személyes összeesküvés-elméletem fókusza, hogy a mobilappban csak a newsfeed megtekintése után lehet a csoportra átlépni (hátha meglát az ember valamit, és több időt tölt az oldalon).

ha az ember mindenkit szeretne minél hamarabb elérni, gyakorlatilag mindkét fórumon posztolnia kell a közlendőjét.

7. Nyomtatás

Már a 2000-es évek elején, az akkori *Stella* szerveren létezett egy nyilvántartás a nyomtatószámlákról, melyekre a collegisták befizethettek a rendszergazdáknál, majd erről nyomtathattak. Később a Sztupák Szilárd Zsolt és Parragi Zsolt által írt „Aktuális" kezelte ezeket; azonban a rendszer távozásukkal megszűnt.

Egy 2017-es levél szerint [16] akkoriban a Windows Intézőben fel kellett csatlakozni az `ujbela.eotvos.elte.hu` szerverre, és onnan a `bwlaser` nevű nyomtató illesztőprogramját telepíteni. Az EIR-azonosítóval kellett belépni.

Ma az Uránból lehet a nyomtatót elérni. Eredetileg közvetlenül egy statikus IP-címe volt; most egy Raspberry Pi-ra van rákötve, hogy kívülről ne lehessen hozzáférni. (A nyomtatón nincs tűzfal-funkció.)

A hardverről: 2009 óta működött egy HP LaserJet P2055dn típusú nyomtató, amely megdöbbentő módon egészen 2023 elejéig (nagyjából 14 évig!) teljesített szolgálatot. Akkor egy új nyomtatót vettünk, az előző sikerén felbuzdulva ugyanabból a típusból (még mindig vannak bontatlan példányok). Mindamellett ezt már kellett garanciáztatni; lehet, nem lesz annyira tartós.

A nyomtató üzemeltetése nehéz feladat: talán itt a legtöbb a rejtélyes hiba, az elakadás, a váratlan kérés/kérdés. Emellett a folyamatos papír- és tintaellátást is biztosítani kell.[29] Innen alakult ki a bentlakó rendszergazda sztereotípiája, aki a legváratlanabb időpontokban trappol végig a folyosón hálóöltözetben és papucsban, hogy *megint* meggyógyítsa a nyomtatót.

8. Információörökítés

Fontos kérdés, hogy a korábbi rendszergazdák meglévő ismereteiket, tapasztalataikat hogyan örökítsék tovább utódaikra. Erre a célra Kovács Máté alatt az `ejcroot@gmail.com` fiókhoz tartozó Drive-tárhely

[29] Utóbbit elfelejtettem nemrég; de hétfőn szereztem tonereket.

szolgált, mely jelen cikknek is fontos forrása. Megtalálhatók itt konfigurációs fájlok, beszámolók, leírások, netreg-nyilvántartások, útmutatók.[30] Ezt azonban Máté távozása után nem tartották karban; a legtöbb dolog utolsó módosítási dátuma 2017. március 17. Legutóbb Dominik használta a tárhelyet arra, hogy az Urán-adatbázisról biztonsági mentést készítsen rá.

Nemrég megtaláltuk a régi rendszergazdai mappát a választmányiban. Ebben 2014-től egészen 2017-ig (tehát jórészt Máté rendszergazdasága alatt) nagyon részletesen és alaposan vezették a számlákat. Itt is vannak érdekességek; az olasz és szlovák rendelésektől a Választmánynak nyújtott kölcsönön át egészen a színes szigetelőszalagokig, amikkel a különböző állapotú eszközöket jelölték.[31] De aztán ez is nagyrészt elhalt; „Bondics Laci" nevén találtunk még számlákat, illetve most a mieinket is itt gyűjtjük.

Talán általánosan érezhető, hogy Máté alatt volt a legszervezettebb, legátláthatóbb a rendszergazdai működés. (Előtte [12] és azóta is sokkal összevisszábban zajlott és zajlik.)

A 2022-es szerverleállás után merült fel az igény, hogy a nagy nehezen összeszedett tapasztalatokat olvasható formában rögzítsük, és így a későbbi rendszergazdáknak könnyebbé tegyünk egy esetleges újratelepítést. Ekkor Samu kezdett el kis cikkeket készíteni Markdownban; ezeket később kibővítettük, és ma már a rendszergazdai élet szinte minden aspektusáról van leírás.

A mostani átalakításról is kell majd írnunk részletes anyagokat; ezt a télre tervezzük.

9. Egyebek

Két egyebem van.

[30] Néha egészen vicces dolgokba is botlottam. Az EIR-t bemutató képeken például a regisztráció mintaadatai furcsák: képzeletbeli felhasználónkat Minta Aladárnak hívják, Alibabának kell szólítani, MSN-azonosítója *„ilyen van még? :D"*, Skype-neve pedig *„azt.a.rekurziv.edes.anyad"*.

Egy másik, egészen szürreális pont az *„off_nemá"* című mappa, ahol disznó collegisták mosdó- és folyosóhasználati szokásait kritizáló tartalmak szerepelnek – köztük a legendás, 2017. október 27-i Facebook-poszthoz kötődő ügy vizuális dokumentációja. [11, 12]

[31] Még garfieldos matricát is kaptak érte az Intersparban. A színkódokhoz egyébként érdemes lenne visszatérni; sokszor van, hogy valamiről nem tudjuk, működik-e.

Az egyik az Estikéhez kapcsolódik: Kocsis Ábel adta először a prémium Spotify-fiókját a zenegéphez; így történt meg a váltás a YouTube-ról. [4] (Ez most Andris neve alól működik; így akár távolról is kontrollálhatja a zenét.) Azóta a műsorlistáknak hála az estikézők kicsit finomabban tudják a többiekre erőltetni zenei ízlésüket, még ha fel is húzzák néha a saját számaikat a sorban.[32] Felmerült egy olyan gondolat, hogy előfizethetnénk YouTube Musicra és arra kéne váltani; de ezt még nem gondoltuk át teljesen.

A másik pedig a Társalgó régi funkciójára emlékeztető számítógépekről szól. Amikor én jöttem, két meglehetősen régi gép árválkodott a falak mellett. Nagyon kényelmetlen őket használni; láthatóan a rájuk erőltetett Windows 10-et nem igazán bírják. A szkenner irányítására jók,[33] de sosem láttam, hogy egynél több ember használta volna őket párhuzamosan.

Mindamellett egyszer Gondnokasszony megkérte Ádámot, hogy tegyen üzemképessé négy gépet a társalgóban, mert jön a *Minisztérium*[34] ellenőrizni. Azóta minisztériumi gépeknek gúnyoljuk szegényeket, és sokszor belőlük lopunk kábeleket.

10. Befejezés

Ez tehát, amit a Collegium elmúlt tíz évéből összegyűjtöttem. Elképzelhető, hogy valamit nem találtam, félreértettem; épp ezért szívesen fogadok korrekciókat.

Talán látszik, hogy a rendszergazdák feladata milyen sokrétű. Személy szerint rengeteget tanultam, miközben csináltam, és a Docker-konténerek kapcsán még fogok is. Igazgató Úrral és Anitával kifejezetten jó kapcsolatot sikerült kiépíteni; dögészként nem feltétlenül lett volna lehetőségem erre. És jó érzés, amikor az emberek hálásak, mert segítünk nekik a WiFi beüzemelésében, vagy a nyomtató molesztálásában.

Mindamellett öregszünk. Egészen idáig én és Bálint voltunk a legfiatalabb rendszergazdák, és mi is már a harmadik évünkben járunk.

[32] VIOLA!

[33] Érdekes; ha az én ubuntus laptopomra dugom, azonnal működik; más windowsos laptopjára még soha nem sikerült telepítenem. De a nyomtató is hasonló. Lehet, hogy a Linux hamarosan átveszi a világuralmat?

[34] *„A Minisztérium ma is erős."* [13]

Ahogy említettem, a tavaszi félévben tervezünk Urán-fejlesztő kurzust tartani, mert lassan már nem marad a Collegiumban olyan, akinek van tapasztalata a rendszerrel. Dominik vezeti még most is a fejlesztést Dániából, de ő már kizárólag szívességből teszi. A szerver újjászervezésével, a privát IP-címekre történő átállással sok munka lesz még. És felmerült, hogy a TermTudTábor és az Eötvös Konferencia honlapja átkerülhetne rendszergazdai gondozásba,[35] ehhez is kéne erőforrás.

Éppen ezért nagyon örülünk, ha csatlakoztok hozzánk! Akár kisebb feladatokban (nyomtatóverés, levelezőlisták) is lehet segíteni, de tényleg sokat lehet tanulni a bonyolultabb problémákon keresztül. Kaphatsz történeteket, ismeretségeket és sok-sok szeretetet.

Köszönetnyilvánítás

Először nyilvánvalóan a régi rendszergazdáknak szeretném megköszönni a rengeteg energiát, amit a jelenlegi infrastruktúra fejlesztésébe öltek. Nem a semmiből jutottunk ide, és bár nagyrészt elfelejtették őket, munkájuk tovább él. Némelyik számlán, gépen, elnevezésen még a kezük nyoma is felfedezhető.

De sokakhoz személyes hála is fűz. Dominik megtanított a Laravellel (és így az Uránnal) dolgozni; Samuval pedig a szervert piszkáltuk sokat közösen (a két winchestert is mi vettük).

A cikkel kapcsolatban Norbi nagyon sokat segített; ő látott rá közülünk a legjobban a 2010-es évek második felére. Dominiktól, Samutól és Bálinttól is kaptam tippeket.

Külön köszönöm Barbarának a közös munkát. A 2022–23-as tanévben első rendszergazdaként ő felügyelte a munkámat, és vigyázott a kasszára. Volt párszor, hogy éjszakáztunk a rakoncátlan routerek vagy a Collegium Nostrum utolsó pillanatos simításai miatt. Megtisztelő volt augusztusban átvenni tőle a választmányi kulcsot.

Köszönöm ezután Bálintnak a rengeteg energiát, amit a szerver mostani fejlesztésébe ölt még úgy is, hogy nem igazán tudtam segíteni neki. Tényleg sokkal jobb lesz, mikor készen leszünk; stabilabb, gyorsabb és biztonságosabb rendszer lesz a kezünkben. Meg kell említenem Berta-

[35] A járvány óta az eredeti honlapok elhaltak. Most általában az történik, hogy minden évben az éppen aktuális vezető készít egy oldalt abban a rendszerben, amihez ért (ez jellemzően a Google Sites-ot jelenti).

lan Danit is, aki ebben a munkában segített neki; ígéretes utódjelöltnek látszik.

Rebinek és Ádámnak is hálás vagyok a segítségükért; Rebi különösen lelkesen szeretne tanulni és hozzátenni a dolgokhoz. Említettem is, hogy rajta fogom kipróbálni az Urán-betanítást.

Végül köszönöm az egész Coli közösségének a rengeteg türelmet és szeretetet, amit kaptam tőlük. Talán leginkább ezért érdemes csinálni.

Hivatkozások

[1] Horváth László, *Parragi Zsolt kinevezése „első rendszergazdának”,* 2009.

[2] Katkó Dominik, Szajbély Sámuel, Bondici László, Horcsin Bálint, A *sysadmin* Messenger-csoport szóbeli közlései, 2023.

[3] Katkó Dominik, Mars, Plútó és Urán az EIR rendszerből – Változatok a Collegium informatikai rendszerére, *Húszéves az ELTE Eötvös József Collegium Informatikai Műhelye,* 2024, pp. 67–83.

[4] Kocsis Ábel, *Szakmai önéletrajz,* Az Estike chillszobájának falán, 2018.

[5] Kovács Máté et al., *Rendszergazdai beszámolók,* Az `ejcroot@gmail.com` alatti Drive-tárhely, 2014–2017.

[6] Kovács Máté et al., Számlák a rendszergazdai mappában, 2014–2017.

[7] Kovács Máté et al., *ELTE EC NET,* Az `ejcroot@gmail.com` alatti Drive-tárhely, 2017.

[8] Kovács Máté et al., *For new sysadmin,* Az `ejcroot@gmail.com` alatti Drive-tárhely, 2017.

[9] Lócsi Levente, Informatika a Collegiumban (2003–2011), *ECCE Eötvös Collegium – Collegiumi Értesítő,* 2013, pp. 353–371.

[10] Lócsi Levente, Az infrastruktúra fejlődése, *Tízéves az ELTE Eötvös József Collegium Informatikai Műhelye,* 2014, pp. 24–34.

[11] Luksa Norbert, Lócsi Levente et al., *Van egy szint.* Az *Eötvös Collegium* Facebook-csoportban, 2017.

[12] Luksa Norbert, Szóbeli közlések, 2023.

[13] J. K. Rowling, *Harry Potter és a Halál ereklyéi,* Bloomsbury, 2007.

[14] The Internet Archive, *Wayback Machine,* 1996–2023.

[15] Weisz Csaba, Levelezés, 2023.

[16] A `root@eotvos.elte.hu` megosztott postafiók alatti levelezés, 2014–2023.

Mars, Plútó és Urán az EIR rendszerből

Változatok a Collegium informatikai rendszerére

Katkó Dominik*

Eötvös József Collegium**

katkodominik@gmail.com

Az Eötvös József Collegium rendszergazdáinak feladata ugyan nincs jelenleg hivatalosan definiálva, de tevékenységük talán összefoglalható abban, hogy a collegisták, a Titkárság és az ELTE igényeit kielégítve próbálnak új programokat készíteni, kísérletezni új megoldásokkal, ezzel pedig fejlesztik magukat és a Collegiumot.

Az épületben az internet kiépítése után a collegisták igényei alapesetben kimerültek annyiban, hogy használni tudják az internetet és a nyomtatót. A Titkárság számára elsősorban a levelezőlisták és a honlap karbantartása volt fontos (később a felvételi honlapé is); az ELTE Informatikai Igazgatóság számára pedig, hogy jogi okokból visszakövethető legyen, ki, mikor, melyik IP-címmel csatlakozott az internetre. A három fél kielégítésére adott lenne egy, esetleg kettő egyszerű honlap, ahol ezek a funkciók elérhetőek, ám a gyakorlatban nem alakultak ilyen egyszerűen a dolgok.

Az elmúlt évtizedben két publikus honlap, négyféle jelentkezési felület, háromféle belső információs rendszer volt használatban, ha nem számoljuk a Collegialiat a köteteknek, valamint a Collegium Nostrumot az alumni nyilvántartására.

* ELTE Informatikai Kar

Az összefoglaló elkészítésében közreműködtek: Szajbély Sámuel, Luksa Norbert.

** 2019–2022

Most a Collegium belső információs rendszerének fejlődésére koncentrálva[1] próbálom feltérképezni, milyen döntések és célok mentén, milyen eredményre jutottunk az elmúlt években.

1. Az Egységes Információs Rendszer

2009–2010 környékén Ölvedi Tibor jóvoltából jött létre az Egységes Információs Rendszer, az EIR. Főbb funkciói közé tartozott a felhasználónév és internet-hozzáférés igénylése, a szobabeosztás megtekintése, valamint az akkori Diákbizottság (ma Választmány) programjairól lehetett rajta olvasni [2].

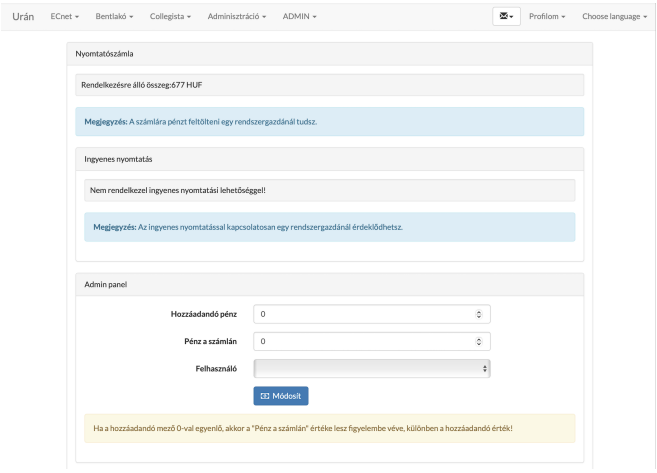
1. ábra. „Az adatokat értelemszerűen töltsétek ki.” – Regisztrálási útmutató az EIR-hez

Az EIR egy ASP.NET alapú oldal volt, és egy Active Directory-hoz volt kötve, ahol a collegisták IP címei, nyomtatószámlái és hasonlók voltak vezetve. A rendszer (Tibor szavaival élve) „tökéletesen működött” évekig, de rendszeres frissítések híján meg voltak számlálva a napjai.

[1] A publikus honlapokról, a Collegialiaról és a Collegium Nostrumról Csimma Viktor írt részletesebben [1].

2. Urán (1.0)

2016-ban Kovács Máté indította el az Urán („Neptun után szabadon”) fejlesztését, amikor is az EIR funkcióit PHP és *Laravel* [8] 5-ös alapokon elkezdte újraírni. A projektbe Luksa Norbert hamar becsatlakozott. Az Urán a GitHubra is felkerült „Eötvös Collegium Tanulmányi Rendszer” névvel [5], így a forráskód nyilvánosan elérhető volt, valamint a GitHub különböző adminisztratív és projektmenedzsment eszközeit is ki tudták használni a fejlesztés során. A felhasználói felülethez a népszerű *Bootstrap* keretrendszer 3-as verzióját [6] használták.



2. ábra. Az Urán felhasználói felülete

Több új funkció is létrejött, többek között az Uránban tudták kezelni a rendszergazdák a levelezőlistákat, a Diákbizottság fel tudta tölteni a jegyzőkönyveket, valamint feladatokat tudott kiosztani és azokat követni egy feladatközponton keresztül, a Titkárság számára pedig fel tudták tölteni a collegisták a nyelvvizsgáikat. Ezek még kezdetleges modulok voltak, és nem voltak kitapasztalva, széleskörűen használva (bár hozzá kell tenni, hogy a nyelvvizsgák feltöltésén kívül ezek a funkciók az Urán jelenlegi verziójában implementálva sincsenek még).

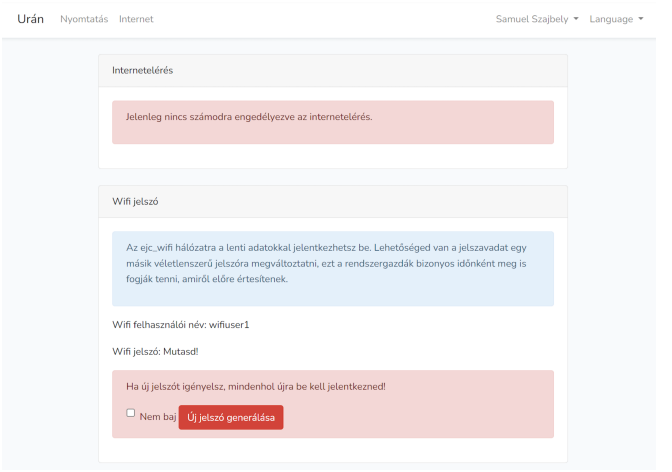
2019-re azonban egyértelművé vált, hogy az alaprendszert nehéz bővíteni, a felépítése nem optimális a sok, különböző típusú funkció tá-

mogatására, és a *Laravel* is fejlődött közben eleget ahhoz, hogy ismét felmerülhessen az újraírás gondolata.

3. A Mars projekt

3.1. Urán 2.0

2019. szeptember 29-én Luksa Norbert indította el az újabb projektet Mars kódnéven [3]. A *Laravel* 6-os verziójának köszönhetően többek között az autentikáció és felhasználókezelés mint beépített funkciók elérhetővé váltak, így nem kellett azokat kézzel megírni, de általánosságban is, a több *Laravel*es tapasztalattal egy jobban átgondolt rendszert tudtak elkezdeni felépíteni. Eleinte az előző rendszer másolata épült meg, hasonló funkciókkal, de ez hamar változott, és ez lett a ma aktív Urán-verzió alapja is. A fejlesztés alatt álló rendszerre hivatkoztunk Marsként (erre később külön, narancssárga színtéma és külön logó is született, ld. 6. ábra), míg a kiadott, éles változat megmaradt Uránként.



3. ábra. Az Urán 2.0 megjelenésekor. 2020. február 19.

Az első kiadás Norbi mellett főként Nagy Vendelnek és Bohony Eriknek köszönhető, de ekkor már igyekeztek minél inkább bevonni a fejlesz-

tésbe a Collegium Informatikai Műhelyének tagjait is, így – szorgalmas elsőévesként – én is ekkor csatlakoztam be és készítettem el a bejelentkezési oldal magyar fordítását. Ekkortájt került fel a kezdőoldalra az új verziót dicsőítő *„jobb, gyorsabb, briliáns, elengedhetetlen, modern, open source"* kulcsszavak is. A főbb újdonságok közé tartozott, hogy a nyomtatás esetén meg lehetett adni a másolatok számát, és hogy latin nyelven is elérhető lett az oldal.

A fejlesztést olyan komolyan vettük, hogy hackathonokat szerveztünk, és havonta adtunk ki frissítéseket (legalábbis terveztünk), folyamatosan beszámolva a munkánkról a collegiumi levelezőlistán. Persze a hétköznapi életben csak annyi látszott, hogy a 3. emeleti fiúszárny végén a kanapékon ülve magyarázunk egymásnak, bölcsész collegisták csodáló (olykor lenéző) tekinteteivel párosulva.

3.2. Urán 3.0

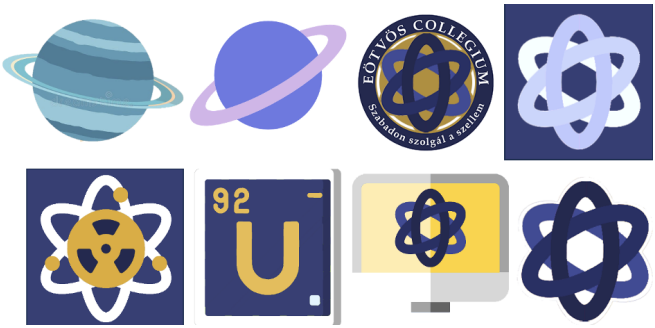
A nagy tanácskozásokban arra jutottunk, hogy nem vagyunk elégedettek az oldal kinézetével. 2020 nyarán fogtam tehát magam, kidobtam az addig használt *Bootstrap* alapú kinézetet (ahelyett, hogy testreszabtuk volna), és újraírtam az oldal felhasználói felületét a Google Material Design 2 [9] alapelveire építve. Így készült el a jelenlegi kinézet a Collegium logójából is ismert kék és narancssárga színekkel, új Urán logóval, a bal oldali navigációs menüvel, és a kártyás elrendezéssel. Bár a funkciókban nem történt sok változás, felhasználói szemszögből nézve úgy gondoltuk, hogy kiadjuk ezt egy új főverzióként. Így született meg a 3.0-s verzió 2020 őszén.

Nagyratörő tervek

A fejlesztésben ketten maradtunk Norbival, de ez nem szegte kedvünket, rengeteg tervünk volt. Mindent digitalizálni akartunk, a teremfoglalástól elkezdve a választmányos pénzek beszedéséig. Viszont a collegisták számára az Urán továbbra is csak az az oldal maradt, ahova nyomtatni jár az ember, vagy ha nem működik az internete.

Az új design nem segített, és ekkor jöttek az olyan ötletek, hogy **a választható nyelveket** a teljes ALFONSÓ[2] sorra kibővítjük. Renge-

[2] A Collegiumban tanulható nyelvek: Angol, Latin, Francia, Olasz, Német, Spanyol, Ógörög.

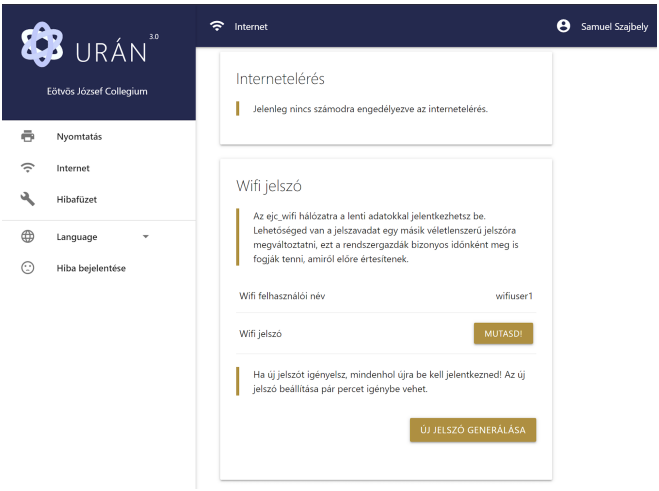


4. ábra. Inspirációk és ötletek az Urán logójára, jobb szélén a jelenlegi logóval, sötét és fehér háttéren

teget dolgoztunk egy olyan rendszeren, ahol a collegisták fordításokat tudnak ajánlani, a nyelvtanárok azokat elfogadni, és így, közös erővel elérjük, hogy ógörögül tudjon nyomtatni hajnalban az Eötvös collegista. A 3.2-es verzióval ez elérhetővé is vált, de a menüpont azóta is ott áll, pár kósza javaslat híján, üresen.

Elkészült (a rendszergazdai mellett) a **választmányos kassza** első változata is. Az elképzelés szerint a Gazdasági Bizottság tagjai a rendszerben rögzítik a befizetett összeget, és egy automata e-mail váltja ki a papíron aláírt bizonylatot. A befizetők internet-hozzáférése és a műhelyek egyenlege pedig automatikusan frissül a befizetések szerint, ezzel csökkentve a gazdasági elnök és a rendszergazdák munkáját is. A tranzakciók és statisztikák nyilvánosan elérhetővé váltak a collegisták számára, ezzel növelve az átláthatóságot. A kezdetleges oldal azonban az akkori gazdasági elnök, Tóth Jenő Excel-táblájával nem ért fel, akármennyire is próbáltam meggyőzni az ellenkezőjéről. Nem segítette a helyzetet a hirtelen megnövekedő adminisztratív teher, mivel gyakran a befizetőket (főleg a bejáró collegistákat, akiknek nem volt feltétlenül szükséges, hogy regisztráljanak az Uránba előtte) utólag kellett levadászni és beregisztráltatni a rendszerbe, hogy el lehessen könyvelni a tranzakciókat. Ez viszont legalább valamelyest segítette a regisztrált felhasználók számának növelését.

Elkészült a **digitális hibafüzet** is, mely mai napig párhuzamosan működik a portán lévő füzetecskével. A modul elkészülte után majd egy éven keresztül a rendszergazdák kaptak mindenről egy e-mailt, majd

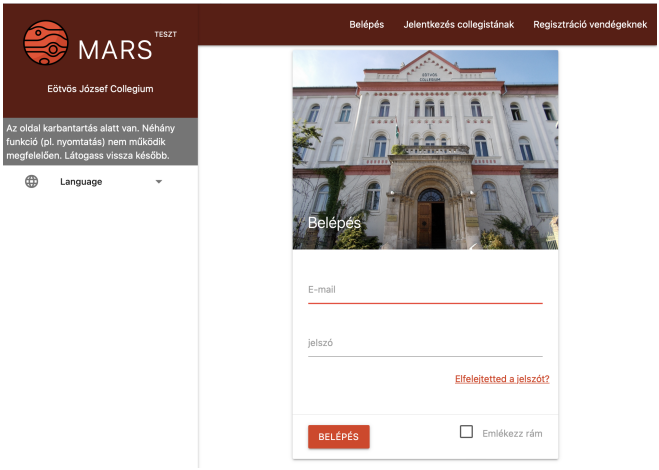


5. ábra. Urán 3.0 verzió. 2020. június 27.

rendszeresen hasra ütésre lezárták azokat a hibajelentéseket, amiket késznek gondoltak az ágyukból ülve, mivel egyszerűen nem volt megoldható, hogy a karbantartó frissítse az egyes hibajelentések státuszát. Egy ilyen e-mail után hirtelen felindulásból kitöröltem az ide vonatkozó részeket, így most a rendszergazdák (és mások is) nyugodtan nézhetik a hibajelentések sokasodó listáját. (A gondnokasszony értesítései megmaradtak, ő pedig megnyugtatott minket már párszor, hogy minden e-mailt adminisztrál az oldalról azóta is.)

A Collegiumban a Kommunikációs Bizottság elnökeként is tevékenykedtem, melynek során elkészítettem a Választmány hírlevele, az **Epistola Collegii** számára egy szerkesztői felületet. Itt minden collegista tud híreket feltölteni egy űrlap segítségével, és a szerkesztő (azaz akkor én) a megadott határidők és egyéb adatok alapján tud időzíteni és elküldeni egy hírlevelet. A funkció azóta is használatban van, bár kétlem, hogy a szerkesztőn kívül bárki önakaratából töltött volna fel oda bármit.

Sikernek könyvelhető el azonban a **Mr. és Miss Eötvös szavazatok** vezetése. Az Urán nemcsak a szavazatszámlálást tette pofonegyszerűvé az automatikus összesítéssel, de elérhetővé tette, hogy az egyéni



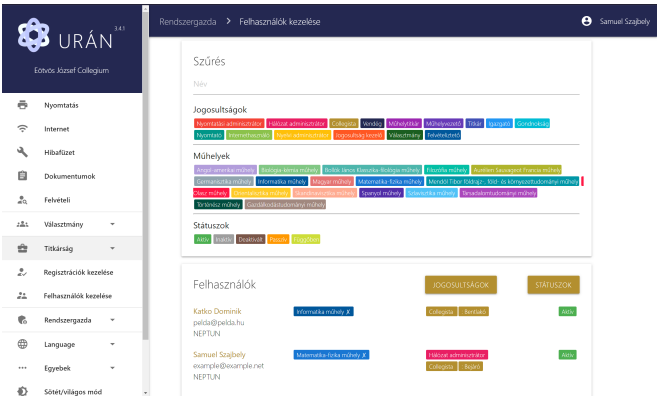
6. ábra. A fejlesztés alatt használt Mars téma

kategóriákra ne csak a kategória kitalálója tudjon szavazni (összefogás/összeesküvés nélkül).

A **titkársági modul** is fejlődésnek indult. Lefejlesztettük, hogy tagsági igazolást és egyéb dokumentumokat tudjanak igényelni a collegisták, valamint elkezdtük rögzíteni a collegisták státuszát és egyéb adatait, hogy egy rendes adatbázist tudjunk építeni a tagságról. Azonban ismételten belefutottunk abba a problémába, hogy nincs mindenki regisztrálva az Uránba, és általánosságban nem éri meg használni a rendszert, és karbantartani az adatokat, mert nincs hozzáadott értéke, nem mellesleg nem is volt intuitív és könnyen kezelhető.

A következtetések levonása után határoztuk el, hogy az akkor egy külön oldalon működő[3] collegiumi **felvételi rendszert** átírjuk a Marsba, így felmenő rendszerben egy idő után minden collegista a rendszerbe kerül. Szó volt egyébként arról is, hogy szabályzatba foglaljuk a regisztrációt, de ez egy időigényes folyamat, amire nem tudtunk támaszkodni (ettől függetlenül a jövőben valószínűleg visszatérünk rá).

[3] Az e-mailen keresztüli jelentkezés után Luksa Norbert 2017 nyarán készítette el az első felvételi honlapot, ami egy egyszerű php és html alapú űrlap volt. Ehelyett szintén készült 2020-ban Molnár László által egy új honlap, majd 2021-ben Szajbély Sámuel írt egy scriptet, amivel az ott regisztrált adatokat a felvett collegisták esetében áttettük az Urán rendszerébe.



7. ábra. Urán 3.4-es verzió. 2020. október 18.

A felvételi modul elkészítése azonban az egyik legkomplexebb feladat az egész oldalon. A teljes felvételiztető rendszer még a második év tapasztalatai alapján is finomhangolásra szorult. Ezt jól demonstrálja, hogy jelenleg 7 különböző jogosultságot kell kezelni, hogy ki melyik felvételiző adataihoz fér hozzá, és ki tudja az egyes felvételizők különböző státuszait állítani. A gyakorlatban pedig olyan problémák kerültek elő utólag, minthogy mi történik, ha az egyik műhely behívna valakit, de egy másik műhely nem; hogyan felvételiznek újra bejáró collegisták bentlakó státuszra; mi történik ugyanebben az esetben, ha az illető választmányi tag is, így látja az „ellenfeleit”; hogyan és ki kezelje az utólagos lemondást, adatmódosítást vagy dokumentumfeltöltést; hogyan érjük el, hogy az összes műhelyvezető tanár regisztráljon és használni tudja a funkciókat; és mindezzel együtt az adatvédelmi tájékoztatót is át kellett gondolni és átírni.

2022-ben, lényegében az utolsó pillanatokban készült el a modul, és akkor még a rendszergazdák kezeltek mindent. Abban az évben összesen 51 e-mailt kaptunk a felvételivel kapcsolatban jelentkezőktől, tanároktól, a Titkárságtól vagy műhelytanároktól, technikai vagy adminisztratív okokból. Ezek után adtunk minél több jogkört és lehetőséget a műhelytitkároknak, műhelyvezetőknek és a Titkárságnak, hogy mindent tudjanak ők intézni, ahol nem kell hozzányúlni a kódbázishoz.

A felvételi felülettel az Uránban a collegisták nyilvántartása tehát

megoldódni látszott, de ez a modul nem oldotta meg az adatok karbantartásának problémáját.

Ekkorra már Luksa Norbi teljes állású munkát vállalt, így nem volt ideje tovább segíteni a fejlesztésben, viszont Szajbély Sámuel fizikus collegista kitanulta a Marsot, először a sötét mód, majd a szobabeosztás implementálásával (mely az Urán 1.0 után ekkor vált ismét elérhetővé), és beszállt a titkársági és felvételi modul fejlesztésébe is.

Az adatok és szemeszterhez kötött státuszok (aktív/passzív, bejáró/bentlakó/vendég) frissítésére a megoldást a **szemeszter végi kérdőív** jelentette, aminek elődje egy, a választmányi elnök által összeállított Google form volt. A szokásos kérdések megválaszolása mellett (közösségi és kulturális tevékenység, tanulmányi eredmények, stb.) a collegistáknak frissíteniük kell a státuszukat a következő szemeszterre.

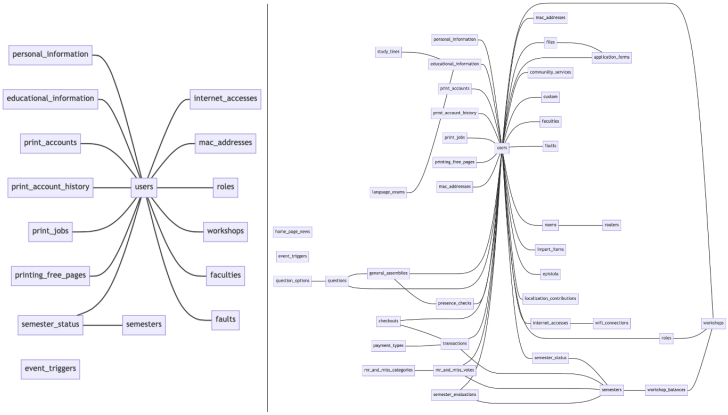
Ehhez kapcsolódóan megjöttek az első, részben vagy egészben a collegisták által kért funkciók is. Elkészült egy oldal a nyelvvizsgák feltöltésére, a közösségi tevékenységek igazolására, valamint a Közgyűlésnél a szavazásra és a jelenléti ívre is. Mindezt összekombinálva pedig az ide tartozó adatokat automatikusan kitölti a rendszer a szemeszter végi kérdőívben, jelezve, hogy a Collegium követelményeit teljesítették-e a collegisták.

A Titkárság, a Tanári Kar, és a Választmány számára kiexportálhatóak lettek a collegisták különböző adatai (a jogosultságok figyelembe vételével) Excel-táblába, így mostanra sikerült elérni az egyik legfőbb eredeti célunkat: hogy létrehozzunk egy **naprakész adatbázist** a collegisták adataival, az exportálással pedig mindenki úgy tudja ezt felhasználni, ahogy az neki kellemes.

Túl nagyra nőtt megoldások

Az évek során itt is felmerült a probléma, mint a Mars elődeinél, hogy a projekt túlnő saját magán. Ez a legjobban talán az adatbázis szerkezetének fejlődésén látszik. A 8. ábrán látható gráfon jól látszik, hogy az előző fejezetben tárgyalt funkciók milyen mértékben emelték meg a rendszer komplexitását. Ez magától értetődő, a probléma viszont ott kezdődött, hogy folytattuk azt a konvenciót, hogy mindent a *users* táblához[4] kötünk, és így alakult ki a jobb oldalon látható állapot,

[4] A *users* tábla tartalmazza a felhasználók legalapvetőbb adatait: a név, e-mail cím, titkosított jelszó, és pár egyéb metaadat.



8. ábra. Az Urán 3.0 (2020) és 3.18 (2023) adatbázis-szerkezete gráfban ábrázolva, ahol minden él a táblák közötti bármilyen közvetlen kapcsolatot jelöl

ahol az ehhez a táblához kötődő fájl egy több mint ezer soros monstrum, ami olyan függvényeket is tartalmaz, amik számolják, hogy kell-e fizetnie az adott felhasználónak KKT-t, vagy éppen hány ingyenes oldalt nyomtatott ki eddig összesen. Ez az ún. monolitikus architektúra, ami nagyon egyszerűen használható és előnyös kis rendszerek esetén, de nehéz fenntartani egy akkora rendszerben, amekkorára a Mars nőtt. Ha hiba van, vagy fejleszteni szeretnénk valamit a nyomtatásnál, nagyon nehéz megtalálni a hiba pontos forrását, főleg, ha nem ismerjük a rendszer működését, a teljes képet.

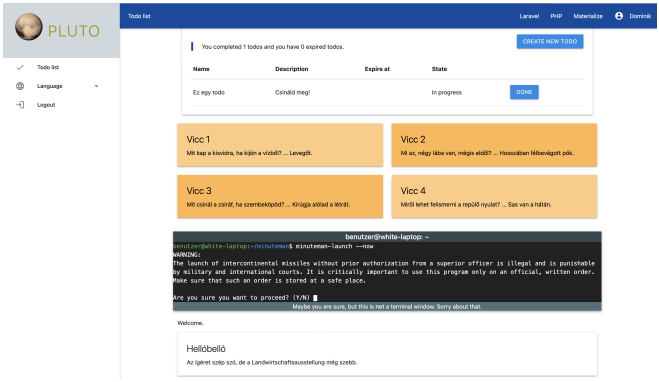
A problémát fokozta, hogy az új funkciók gyors fejlesztése háttérbe szorította a tesztelést, így gyakran egy hiba kijavítása újabb hibákat szült, amelyeket rendes tesztesetek híján nem vettünk időben észre.

A bővülő, szépülő Urán mögött meghúzódó rendszer tehát egyre csúnyább lett, és ez nem segített abban, hogy új fejlesztőket tudjunk bevonzani segítségnek. Több próbálkozásunk is volt, hogy az Informatikai Műhely tagjainak kitanítsuk a Marsot, többek között kurzusokat is szerveztünk, de a kezdeti lelkesedés szinte minden esetben kudarcba fulladt. Egy egyszerű funkció lefejlesztése is inkább telt több napnyi szenvedéssel, mint sikerélménnyel. Ez részben a fejlesztői környezet felépítésének is köszönhető volt, mivel egy ilyen, lassan ipari méretű

projektnek a futtatásához szükséges követelménye összemérhetetlen az egyetem mintapéldáihoz képest.

3.3. A Plútó lenne a megoldás?

Így született meg a Plútó [4], ami a Mars kistestvéreként szolgált. Ugyanabban a környezetben, a Mars lecsupaszított imitálásaként a Plútó egy jó tanulási lehetőséget adott az érdeklődő collegistáknak, ahol egy féléves tárgy keretében közösen fejlesztettünk le nulláról kezdve egyszerű megoldásokat, szabad kezet hagyva az egyéni kívánságoknak. Így született meg egy modul To-Do listáknak, vagy egy másik a személyes pénzügyek nyilvántartására.

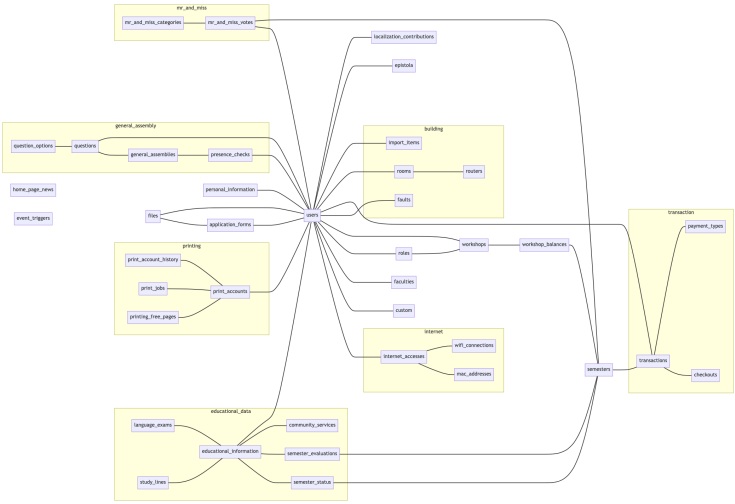


9. ábra. Képernyőkép a Plútóról

Ez egy jó kezdetnek bizonyult, de a Mars rendszerszintű problémáit nem oldotta meg, és mivel rengeteg energiát öltünk az Urán új funkcióiba, fel sem merült az, hogy újból újraírjuk az egészet, okosabban, szebben, jobban.

3.4. Úton a 4.0 felé

Az egyetlen megoldás az volt, hogy felülvizsgáljuk a teljes rendszert, a régi, elavult megoldásokat felújítjuk, és átstrukturáljuk az egészet, miközben új dokumentációt és teszteket írunk. Ezt a „refaktorálási folyamatot” 2022 elején kezdtem el főként én, és azóta is dolgozok rajta,



10. ábra. Az új, modularizált szerkezet reprezentációja

egyedül a felvételi rendszer élvezett prioritást ellene a nyáron. Ez egy nagyon tanulságos és időigényes folyamat, emlékszem, először 2022 végére terveztem vele kész lenni, majd fokozatosan toltuk a határidőt, mindeközben pedig adtunk hozzá több munkát és új célokat is. Jelenleg, összehasonlítva a 8. ábrával, a 10. ábrán látható állapot az a cél, amire már büszke tudok lenni, és kijavítva érzem az elmúlt évek hibáit.

Az újabb internetes megoldások fejlődésével megoldódni látszik a fejlesztői környezet problémája is, mivel ma már a felhőben is lehet fejleszteni, ahol előre fel lehet telepíteni a szükséges eszközöket, és a számítási erőforrásokkal sincs gond.

Az elmúlt egy évben ugyan tehát nem történt sok újdonság a felhasználók számára, a háttérben viszont rengeteg munka történik, hogy a Mars fejlesztése ugyanolyan élmény legyen az új, leendő fejlesztők számára, mint amilyen az nekem volt.

Ennek a munkának a megünneplésére tervezzük kiadni a 4.0-s főverziót, és hogy a felhasználók is kapjanak valami újat, újabb (kisebb) változásokat tervezünk a felhasználói felület kinézetében, hogy követsük a trendeket.

A belső megújulásnak persze nem szabad leállnia, a távolabbi jövőben felmerülhet a mikroszolgáltatások fogalma, egy fejlettebb, automatizált frissítési folyamat (CI/CD), vagy egy erre a célra elkülönített tesztszerver bekonfigurálása, hogy minél inkább közelebb kerüljünk az iparban használt megoldásokhoz.

4. A fejlesztők

A Mars fejlesztése kétségkívül a rendszergazdák legnagyobb projektje, és mára már fejlesztése túl is lépett a rendszergazdákon, funkciói pedig a rendszergazdai feladatkörökön.

Jelenleg **33 ezer sorból** áll a kódbázis, eddig összesen **21 fejlesztővel**, akik között rendszergazdák és más collegisták mellett fellelhetőek egy-egy kisebb funkció erejéig számunkra ismeretlen emberek is, főként a nyílt forráskódú fejlesztést népszerűsítő Hacktoberfestnek [7] köszönhetően. A fejlesztők sora, a *commitok* száma szerint, ami leegyszerűsítve az új funkciókat és frissítéseket jelenti:

- kdmnk – Katkó Dominik – 277 commit
- (kovacsur10 – Kovács Máté – 214 commit az Urán 1.0-ban)
- luksan47 – Luksa Norbert – 211 commit
- machiato32 – Szajbély Sámuel – 46 commit
- viktorcsimma – Csimma Viktor – 19 commit

10 commit alatt:

- erko185 – Bohony Erik
- bgeree
- vendi95 – Nagy Vendel
- mraron – Noszály Áron
- kgerg2 – Kovács Gergely
- uno2001 – Pőcze Barnabás
- gungvri – Ungvári Gábor
- steakhal – Benics Balázs
- domonkos2001 – Nagy Domonkos Máté
- LukaSK351
- thebalu – Varga Balázs

- TheBarbaraIsTaken – Szabó Barbara Noémi
- csandras05 – Csertán András
- Kenzso – Kenessei Zsombor
- Trigary – Sárközi Gergely
- htetwunsan
- dapagyi – Apagyi Dávid

Bár nem mértük, de talán kijelenthető, hogy 2019 szeptembere óta **összesen 2–3 ezer munkaórát** fektettünk a fejlesztésbe, ami több, mint egy ember teljes évi munkája teljes állású programozóként.

5. Mi jön még?

Miközben én befejezem a refaktorálást, jelenleg Csimma Viktorék foglalkoznak az aktuális kérésekkel és új, egyszerűbb funkciókkal, valamint újabb kurzust indítunk a műhely új tagjainak is. Új funkciókat én nem tervezek már írni, de a fejlesztést felügyelni fogom még egy darabig, biztosítva a jelenlegi kódminőséget, amiért annyit küzdöttünk és küzdünk.

A jelenlegi nagy kérdés pedig, hogy sikerül-e új fejlesztőket szerezni, mivel tennivaló az lenne még. A főbb tervek között szerepel a tanterem- és mosógépfoglaló modul lefejlesztése, az e-mail listák menedzselése, a jogosultságok rendszerének finomhangolása, egy online tárhely létrehozása a Google Drive-val integrálva, majdnem az összes jelenlegi modul további fejlesztése, és műhelyoldalak létrehozása, amiknek adatai a nyilvános Eötvös honlapra is exportálhatóak lennének.

Kicsit nagyobb távlatokban pedig felmerült a kérdés a teljes rendszer általánosítására, mivel irigykedve megkerestek már minket más szakkollégiumokból is. Jelenleg a rendszer (még) túlságosan is az EC igényeire van szabva, kezdve az egyéni státuszrendszerünktől a választmányi tagokhoz kötődő jogosultságokig. Én nem látom, hogy lenne valaki, aki ezzel foglalkozna, de egyébként az MIT licencnek [10] köszönhetően bárki szabadon elérheti és személyre szabhatja a szoftvert, szóval ebből a távolabbi jövőben akár még lehet valami.

Persze, ahogy eddig is, minden a motiváción múlik, és a Mars egy kiváló projekt arra, hogy összekösse az ember az egyetemen tanultakat az iparban használt megoldásokkal, és valódi tapasztalatot szerezzen. Ennek előnyét Samu is és én is megtapasztaltuk az első munkahelyünkre

jelentkezésnél. Az egyetemi tanulmányok kiegészítéseként a Mars pedig a Collegium céljait is lefedi, miközben ebből a Collegium egésze közvetlenül is profitál.

Bátorítok mindenkit tehát arra, bármikor is olvassa ezt, hogy csatlakozzon be a fejlesztésbe, írjon GitHubon, és keresse meg az aktuális fejlesztőket vagy engem, mivel a Mars minden hátránya ellenére ennél jobb lehetőség nincs, hogy egy olyan projekten dolgozzon az ember, aminek eredményeit rendszeresen érzékelnénk a közvetlen környezetünkben.

Köszönetnyilvánítás

Köszönöm Norbinak a közös munkát, és hogy tizedszer is elmagyarázta nekem a Gitet, és rebase-eltük a master branch-et; Samunak a közös ötleteléseket bárhol bármikor, és a cikk létrehozása alatt a segítséget a képek keresésében, adatbázisok lementésében; valamint Viktornak, hogy segít továbbadni a tudást a Műhely új tagjainak számára.

Ezen felül többek nevében is hálás vagyok mindenkinek, aki a GitHubon aktív volt, segített a fejlesztésben, és hogy sikerült fenntartanunk egy viszonylag szép és strukturált oldalt a kódbázisnak.

A Collegium lakóinak pedig köszönjük a visszajelzéseket, ötleteket, és a türelmet.

Hivatkozások

[1] Csimma Viktor, Sárkány és papucs – A rendszergazdaság 2013 és 2023 között, *Húszéves az ELTE Eötvös József Collegium Informatikai Műhelye,* 2024, pp. 45–66.

[2] Lócsi Levente, Az infrastruktúra fejlődése, *Tízéves az ELTE Eötvös József Collegium Informatikai Műhelye,* 2014, pp. 24–34.

[3] A Mars GitHub oldala,
`https://github.com/EotvosCollegium/mars`

[4] A Plútó GitHub oldala,
`https://github.com/EotvosCollegium/pluto`

[5] Az Urán GitHub oldala,
`https://github.com/kovacsur10/uran`

[6] Bootstrap 3.4, *The world's most popular mobile-first and responsive front-end framework*,
`https://getbootstrap.com/docs/3.4/`

[7] Hacktoberfest, DigitalOcean
`https://hacktoberfest.com`

[8] Laravel, *The PHP Framework for Web Artisans,*
`https://laravel.com`

[9] Material Design 2,
`https://m2.material.io`

[10] The MIT License,
`https://opensource.org/license/mit/`

– Volna itt még valami, amit meg szeretnék beszélni veled. Ugye a tízéves kötet borítóján egy λ szerepelt, de vajon mi legyen a húszévesen? Van több ötlet is. Például lehetne π, ugye mint a kalkulus folytatása, meg az olyan matekos is kicsit. Vagy μ, mert az a következő betű a görög ábécében. Meg van még pár… Mit gondolsz? Sőt, voltaképpen, Tamás, ezt rád bíznám, mondd meg te, mi legyen! Mégis csak te voltál a műhelyvezető az utóbbi 10 év túlnyomó részében.

Dékán úr bekap még egy falatot az ebédjéből. Ma kivételesen az InfoPark „A" épületének menzáján („éttermében") eszünk. Abban a megtiszteltetésben volt ugyanis részem, hogy az előző másfél tanévben a műhelyvezető-váltás ürügyén többször, már-már rendszeresen Kozsik Tamás dékán úrral ebédelhettem.

Jól megrágja azt a falatot. Majd így szól:

– Λ. Típusabsztrakció. Felnőtt az a λ.

Tanulmányok

Álmodnak-e a Turing-gépek automatikus kódgenerálással?

Bodó Zalán*

Farkas Gyula Szakkollégium**

zalan.bodo@ubbcluj.ro

1. Bevezető

Alan Turing 1936-ban leírt egy úgynevezett *univerzális számítógépet*, ami ma *univerzális Turing-gép* néven ismert, illetve megmutatta azt is, hogy léteznek olyan problémák, melyeket egy ilyen automata nem tud megoldani [32]. Minden modern digitális számítógép a Turing által bevezetett univerzális számítógép megvalósítása, és bár a tanulmánya „pusztán" számításelméleti munka, ez feltételezhetően nagy hatással volt a későbbi brit és amerikai számítógéptervezőkre [26].

A Turing-gép egy véges számú állapottal rendelkező egyszerű automata: a gép egy végtelen szalagról szimbólumokat olvasva két irányba, balra vagy jobbra mozgathatja a fejét az éppen beolvasott szimbólumtól függően, valamint az aktuális szimbólumon egy helyettesítési/írási műveletet végezhet. Ennek az egyszerűnek tűnő automatának a kifejezőereje elképesztően nagy, minden algoritmus megvalósítható vele – erről szól a Church–Turing-tézis [22]. Ennek ellenére nem minden probléma oldható meg algoritmikusan; annak eldöntése például, hogy egy

* Babeș–Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar, Magyar Matematika és Informatika Intézet

** A Kolozsvári Magyar Egyetemi Intézet (KMEI) keretében működő Farkas Gyula Szakkollégium vezetője

program valaha is befejeződik-e, nem lehetséges, vagyis a *megállási feladat* nem eldönthető.

A Turing-gépeket szimulálni képes Turing-gépeket univerzális Turing-gépeknek nevezzük, továbbá minden olyan rendszert, amely fel van ruházva ezzel a képességgel, Turing-teljesnek vagy -ekvivalensnek nevezünk. A ma használt általános célú programozási nyelvek rendelkeznek ezzel a tulajdonsággal, ezért az ilyen nyelveken írt programok Turing- vagy univerzális Turing-gépek.[1] Tehát egy Turing-gép képes egy másik Turing-gépet szimulálni, azaz *futtatni*, azonban képes-e adott Turing-gépet szintetizálni, más szóval programot írni adott specifikáció alapján? E rövid írásban ezt a kérdést fogjuk körüljárni és megvizsgálni.

2. Az intelligenciáról

Az intelligenciát általában úgy értelmezzük, mint tanulást, új helyzetekhez való alkalmazkodást és absztrakt gondolkodást. A mesterséges intelligencia (MI) definíció szerint a természetes intelligencia utánzása, reprodukálása, ami nem feltétlenül jelent emberi intelligenciát – azonban úgy véljük, hogy az ember az általunk eddig ismert legintelligensebb életforma, ezért a mesterséges intelligenciát gyakran az emberi intelligenciával egyenértékűnek tekintjük. Nem könnyű ezt általánosan megragadni (általános/erős MI), ezért sokkal gyakoribb, hogy ehelyett konkrét feladatokra koncentrálunk (szűk/gyenge MI), mint a képfelismerés, kérdésmegválaszolás, hangfelismerés stb., melyek nekünk egyszerűnek tűnnek, azonban nem ismerjük a megvalósításukhoz szükséges „algoritmust". E szűk területek némelyikén a gépek egyértelműen felülmúlják az embereket: a rákdiagnosztikában a gépi tanulási módszerek előrejelzései adott esetben már pontosabbak az emberi patológusok előrejelzéseinél [12], de természetesen sok más példát is említhetnénk, ahol a gépek előttünk járnak.[2] Azonban teljes bizonyossággal kijelent-

[1] Valójában a végtelen memória is feltétele a Turing-teljességnek, de mi itt azt a definíciót tekintjük, hogy elégséges *elegendően nagy* memóriával rendelkeznie.

[2] Habár a gépi predikció pontossága sok esetben meghaladja az emberi pontosságot, a rendszerek robusztussága és a döntések magyarázhatósága még sok kívánnivalót hagy maga után.

hetjük, hogy emberi szintű általános mesterséges intelligencia még nem létezik.[3]

Az automatikus kódgenerálás egy mesterséges intelligencia feladat: a számítógépnek valamilyen formában megadjuk a szintetizálandó program leírását, azaz követelményeit – például logikai formulákkal vagy természetes nyelven –, majd megkérjük, állítson elő egy programot, amely ezt megoldja. Mivel jelenleg a mély mesterséges neuronhálókat használó módszerek teljesítenek a legjobban ezen a területen is – lásd később a Google AlphaCode rendszerét –, röviden szót kell ejtenünk ezekről is. A ma használt mesterséges neurális hálózatok McCulloch és Pitts [25], Hebb [17], illetve Rosenblatt [27] korszakalkotó munkáin alapulnak, melyek a természetes neuronok működését modellezik. Rosenblatt *perceptron* algoritmusának megjelenése óta a tudományos közösség folyamatosan kutatja a mesterséges neurális hálók számítási képességeit: Cybenko *univerzális approximációs tétele* [11] kimondja, hogy egy előrecsatolt háló egyetlen rejtett réteggel, véges számú neuronnal és szigmoid aktivációs függvénnyel képes bármilyen folytonos függvényt közelíteni, Hornik pedig megmutatta, hogy az univerzális approximáció kvintesszenciája nem egy bizonyos aktivációs függvény alkalmazásában, hanem a neurális hálók többrétegű architektúrájában rejlik [19]. Siegelmann és Sontag [30] bebizonyította, hogy a rekurrens neurális hálók Turing-teljesek, kifejezőerejük pedig a súlyok típusától függ: egész számú súlyokat használva bármilyen véges automatát képesek szimulálni, végtelen pontosságú racionális és valós súlyokkal az univerzális Turing-gépekkel, illetve *szuper-Turing-gépekkel* egyenértékűek [29]. Nem lekicsinylendő az idézett munkát, ez mindössze elméleti jelentőséggel bír, mivel a végtelen pontosság nem megvalósítható. Valójában belátható, hogy függetlenül az alkalmazott számítási modelltől, Turing-gépekkel legfennebb Turing-gépek szimulálhatók, nem képesek ennél erősebb automatákat modellezni. A kérdés továbbra is a következő: elegendő számítási kapacitással bír-e egy Turing-gép ahhoz, hogy kódot generáljon?

[3] És kérdéses, hogy fog-e valaha is létezni, mivel nem tudjuk, hogy mi Turing-gépek vagyunk-e, vagy legalábbis az emberi intelligencia szimulálható-e egy Turing-géppel [10], és egyelőre Turing-gépeknél erősebb gépeket nem tudunk építeni.

3. Az automatikus programgenerálásról

Az utóbbi időben az emberek egyre inkább hangot adnak azon aggodalmuknak, hogy a mesterséges intelligencia az elkövetkező években sok munkahelyet fog megszüntetni – hasonlóan az első három ipari forradalomhoz [35]. Ez a folyamat már el is kezdődött, a negyedik ipari forradalom kibontakozásával viszont nemcsak bizonyos munkahelyek tűnnek el, hanem új, újszerű készségeket igénylő új munkahelyek jönnek létre [13].

A programozók, szoftverfejlesztők sem kivételek a fenti probléma alól, a programozói munka egyes fázisait már automatizálták – gondoljunk például a különböző keretrendszerekre, melyek testreszabható kódot generálnak, de ugyanígy létezik már intelligens kódkiegészítés is egyes integrált fejlesztőkörnyezetekben.[4] Az utóbbi néhány évben a kutatók a teljes mértékű automatizált programszintézisre is elkezdtek összpontosítani [1, 4, 5, 16, 23, 31, 36]. Az AlphaCode, a Google DeepMind rendszere például nagy *transzformer* alapú nyelvi modelleket használ arra, hogy természetes nyelvű feladatleírások és példaként szolgáló bemeneti-kimeneti párosok alapján programkódot generáljon [23]. Modelljük 10 szimulált Codeforces[5] programozási versenyen átlagosan a feladatok 25%-át megoldva a mezőny első 54,3%-ban végzett, és az 1238-as becsült rangot érte el, ami nagyobb, mint a versenyeken induló felhasználók 72%-ának rangja. Az 2023-as év végén megjelent AlphaCode 2 jóval túlszárnyalja elődje képességeit: a versenyzők 85%-ánál jobb eredményt elérve a feladatok 43%-át oldja meg sikeresen [1].

Logikai szempontból a programszintézis egy másodrendű logikai egzisztenciális formula megoldása,

$$\exists P_1 P_2 \dots P_m \; Q_1 x_1 \; Q_2 x_2 \dots Q_n x_n \; \sigma(P_1, P_2, \dots, P_m, x_1, x_2, \dots, x_n)$$

ahol P_i, $i \in \{1, 2, \dots, m\}$ függvényeket, Q_j kvantorokat, x_j, $j \in \{1, 2, \dots, n\}$ pedig alaptermeket jelöl [14]. A programszintézist tehát egy másodrendű logikai kielégíthetőségi problémára redukáltuk: az egzisztenciális kvantorral kvantált változók a keresett program függvényei, a formula pedig azon követelményeket írja le, melyeknek a prog-

[4] *Type-matching completion,* IntelliJ IDEA 2023.2 documentation, `https://www.jetbrains.com/help/idea/auto-completing-code.html#smart_type_matching_completion`

[5] `https://codeforces.com/`

ram meg kell feleljen. Mivel a másodrendű logika eldönthetetlen, azaz két tetszőleges logikai formula kielégíthetősége eldönthetetlen, ezért a programszintézis is eldönthetetlen [16].[6] A szakirodalomban találunk olyan munkákat, melyek a másodrendű logika egy eldönthető töredékével dolgoznak, azonban ez meglehetősen korlátozza a szintetizálható programok halmazát [15, 24].

$$S \to 0S1S \mid 1S0S \mid \varepsilon \qquad \text{(a)}$$

$$\begin{aligned} S &\to 0AS \mid 1BS \mid \varepsilon \\ A &\to 0AA \mid 1 \\ B &\to 1BB \mid 0 \end{aligned} \qquad \text{(b)}$$

1. ábra. A $\{w \in \{0,1\}^* \mid n_0(w) = n_1(w)\}$ nyelv leírására szolgáló két környezetfüggetlen grammatika: (a) többértelmű, (b) egyértelmű grammatika

A programszintézis eldönthetőségének kérdése megközelíthető a Turing-gépek ekvivalenciájának vizsgálata irányából is. A magasszintű programozási nyelvek alapjait képező *generatív grammatikákat* Noam Chomsky valójában a természetes nyelvek modellezésére vezette be 1965-ben [9], és ezek nélkül az informatika ma valószínűsíthetően nem tartana ott, ahol tart. Chomsky javasolt egy hierarchiát is, ami a formális nyelvek/generatív nyelvtanok bizonyos komplexitási osztályait írja le, s melyben felfelé haladva egyre bonyolultabb nyelveket vagyunk képesek alkotni – minden ilyen osztályhoz hozzárendelhető egy ezzel egyenértékű automataosztály. A legegyszerűbb a reguláris nyelvek osztálya, melyek a véges automatákkal ekvivalensek – ezen a szinten minden „egyszerű", el lehet dönteni például, mégpedig polinomiális idő alatt, hogy két ilyen automata ekvivalens-e egymással vagy sem. Egy osztállyal feljebb lépve a kontextusfüggetlen nyelvek osztályába, ez megszűnik igaznak lenni, a veremautomaták ekvivalenciája már eldönthetetlen. Az ekvivalencia eldönthetetlensége megmarad a kontextusfüggő nyelvek, vagy ennek megfelelően a lineárisan korlátos automaták, és a rekurzívan felsorolható nyelvek vagy Turing-gépek szintjén is [18]. Az 1. ábrán látható két, (a) és (b) környezetfüggetlen grammatika ekvivalenciájának vizsgálatához tehát nem létezik algoritmus, azaz egyenértékűségük nem eldönthető.

[6] Valójában már az elsőrendű logika is eldönthetetlen.

Shepard azt mondja, hogy a hasonlóság alapvető a tanuláshoz, minden egyénben van egy „belső hasonlósági metrika a lehetséges helyzetek között", és az általánosítás a helyzetek, események, tárgyak stb. közötti hasonlóságon alapszik [28]. Egy gépi tanuló rendszerben, például egy mesterséges neurális hálóban is lennie kell egy ilyen – explicit vagy implicit – hasonlósági mértéknek, mivel elvárjuk ezektől, hogy hasonló bemenetekre hasonló kimenetet produkáljanak. Így az ekvivalenciának – vagy az ekvivalencia szintjének, azaz a hasonlóságnak – az automatikus programszintézisben (is) központi fogalomnak kell lennie, és mivel az ekvivalencia nem eldönthető, „pontos" hasonlósági metrikát sem lehet találni. A kérdés az, hogy mennyire kell hűnek lennie ennek a metrikának? A statisztikai tanuláselmélet perspektívájából nézve minden csak közelítés – véges darabszámú példák által tanulunk, azaz közelítünk. Hasonlóképpen, közelítéssel azt is meg tudjuk vizsgálni, hogy egy program valaha is befejeződik-e vagy sem, például ha sok különböző bemenetre megfigyeljük a viselkedését. Tulajdonképpen a tesztesetek megírásával és végrehajtásával ugyanezt tesszük komplexebb szoftverünk validálásakor: sok-sok unit tesztet írunk, hogy lefedjük velük a lehetséges programutakat, megfigyeljük a program viselkedését és eldöntjük, hogy az helyesen működik-e. Minél több tesztet írunk, és ezekkel minél több megvalósítható programutat fedünk le, továbbá a programunk minden teszten átmegy, annál nagyobb valószínűséggel jelenthetjük ki, hogy a program helyesen működik.

Ez akkor azt jelentené, hogy statisztikai módszerekkel megfelelő módon megközelíthetők lennének az eredmények? Sajnos nem, azaz nem tudjuk. Egy viszonylag új elméleti munkában a szerzők megmutatták, hogy léteznek olyan gépi tanulási feladatok, melyekről a matematika standard axiómái segítségével sem azt nem tudjuk bebizonyítani, hogy – PAC (*probably approximately correct*) értelemben [34] – megtanulhatók, sem ennek ellenkezőjét [6].

4. Zárszóként

Habár az utóbbi években egymást érik az újabbnál újabb jelentős eredmények a mesterséges intelligencia és gépi tanulás területén, ez nem feltétlenül jelenti az emberi szintű általános mesterséges intelligencia közelgő eljövetelét. Tekintsük például nagy nyelvi modelleket, mint a GPT-3.5 vagy GPT-4 [7], melyek a ChatGPT-t is vezérlik, de ugyan-

úgy említhetnénk a LLaMA [31], Gemini [3] stb. rendszereket is. Egyes kutatók szerint a természetesnyelv-megértés úgynevezett MI-teljes feladat, ami azt jelenti, hogy ha az intelligenciát kiszámíthatónak tekintjük, akkor ezen feladatok bonyolultsága megegyezik az általános MI feladatának bonyolultságával [37]. Mások viszont úgy vélekednek, hogy a nyelvnek megvannak a maga korlátai, az emberi intelligencia nagy része nem nyelvi, és egy olyan MI rendszer, amely csupán a nyelvre összpontosít, sohasem fogja megközelíteni az emberi intelligenciát [8]. Mindazonáltal nem szabad figyelmen kívül hagyni az olyan tényeket sem, hogy a mai nagy nyelvi modellek képesek jobb, meggyőzőbb propagandaszöveget generálni, mint az emberek [38].

Egy másik kérdés, amely manapság foglalkoztatja a kutatókat, a robusztusság és az értelmezhetőség kérdése [2] – ebből a szempontból sok, a pontosság tekintetében jól teljesítő rendszer leszerepel, vagyis a predikció nem marad stabil a bemenet kismértékű megváltoztatása esetén, illetve a rendszer nem nyújt megfelelő módon értelmezhető döntéseket. Mindaddig, amíg sok rendszer kijátszhatóan és/vagy fekete dobozként működik, nem fogják helyettesíteni az embert magas kockázatú helyzetekben.

Holott e rövid írásban az automatikus kódgenerálás vagy programszintézis kérdését jártuk körbe, hasonlóképpen felmerül(het) az emberi szintű mesterséges intelligencia elérésének kérdése. Egy jól ismert módszer az emberi intelligencia mérésére a Turing által javasolt próba [33]: az *imitációs játék* vagy *Turing-teszt* akkor sikeres, ha egy ötperces (billentyűzetalapú) beszélgetés során a tesztalanyok legalább 30%-a tévesen embernek ítéli meg gépi beszélgetőpartnerét. Habár korábban is voltak kiemelkedő eredmények ez irányban, egy a közelmúltban elvégzett kísérletsorozat eredményei alapján a nagy nyelvi modellekkel működő csevegőrendszereknek sikerült az esetek 32%-ában megtéveszteniük a résztvevőket [20], egy másik kísérletben azonban az 1960-as években épített ELIZA megelőzte az egyik modern nagy nyelvi modellt [21]. Ezek az eredmények megkérdőjelezhetők, de hasonlóképpen a Turing-teszt validitása is: valóban képes mérni az emberi szintű intelligenciát? Függetlenül attól, hogy a fenti kérdésekre mi a helyes válasz, hogy elérhető-e valaha is az emberi szintű mesterséges intelligencia, az új (MI) algoritmusok fejlesztése, illetve a létezők tökéletesítése kulcsfontossággal bír, hogy a költséges, komplex, biztonságkritikus folyamatainkat egyre optimálisabbá, megbízhatóbbá tegyük.

Hivatkozások

[1] AlphaCode 2. Technical report, Google DeepMind, 2023.

[2] D. Alvarez Melis, T. Jaakkola, Towards robust interpretability with self-explaining neural networks, *Advances in Neural Information Processing Systems,* 31, 2018.

[3] R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, et al., Gemini: a family of highly capable multimodal models, arXiv preprint arXiv:2312.11805, 2023.

[4] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, et al., Program synthesis with large language models, arXiv preprint arXiv:2108.07732, 2021.

[5] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, D. Tarlow, Deepcoder: Learning to write programs, arXiv preprint arXiv:1611.01989, 2016.

[6] S. Ben-David, P. Hrubeš, S. Moran, A. Shpilka, A. Yehudayoff, Learnability can be undecidable, *Nature Machine Intelligence*, 1(1) (2018), pp. 44–48.

[7] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, *Advances in Neural Information Processing Systems,* 33 (2020), pp. 1877–1901.

[8] J. Browning, Y. LeCun, AI And The Limits Of Language, Noēma, August 2022.

[9] N. Chomsky, *Aspects of the theory of syntax,* MIT Press, 1965.

[10] B. J. Copeland, Computation, L. Floridi, editor, *The Blackwell guide to the philosophy of computing and information,* John Wiley & Sons, 2004.

[11] G. Cybenko, Approximation by superpositions of a sigmoidal function, *Mathematics of Control, Signals and Systems,* 2(4) (1989), pp. 303–314.

[12] S. Dabeer, M. M. Khan, S. Islam, Cancer diagnosis in histopathological image: CNN based approach, *Informatics in Medicine Unlocked,* 16 (2019), 100231.

[13] Data Science in the New Economy. A new race for talent in the Fourth Industrial Revolution. World Economic Forum, Centre for the New Economy and Society, 2019.

[14] C. David, D. Kroening, Program synthesis: challenges and opportunities, *Phil. Trans. R. Soc. A,* 375(2104) (2017), 20150403.

[15] C. David, D. Kroening, M. Lewis. Using program synthesis for program analysis, *Logic for Programming, Artificial Intelligence, and Reasoning: 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24–28, 2015, Proceedings 20,* Springer, 2015, pp. 483–498.

[16] S. Gulwani, O. Polozov, R. Singh, Program synthesis, *Foundations and Trends® in Programming Languages,* 4(1–2) (2017), pp. 1–119.

[17] D. O. Hebb., *The organization of behavior,* Wiley, New York, 1949.

[18] J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to automata theory, languages, and computation,* Pearson, 2014.

[19] K. Hornik, Approximation capabilities of multilayer feedforward networks, *Neural Networks,* 4(2) (1991), pp. 251–257.

[20] D. Jannai, A. Meron, B. Lenz, Y. Levine, Y. Shoham, Human or Not? A Gamified Approach to the Turing Test, arXiv preprint arXiv:2305.20010, 2023.

[21] C. Jones, B. Bergen, Does gpt-4 pass the turing test? arXiv preprint arXiv:2310.20216, 2023.

[22] S. C. Kleene, *Introduction to metamathematics,* University series in higher mathematics, Van Nostrand, New York, 1952.

[23] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, et al., Competition-level code generation with AlphaCode, *Science,* 378(6624) (2022), pp. 1092–1097.

[24] P. Madhusudan, U. Mathur, S. Saha, M. Viswanathan, A decidable fragment of second order logic with applications to synthesis, arXiv preprint arXiv:1712.05513, 2017.

[25] W. S. McCulloch, W. Pitts. A logical calculus of the ideas immanent in nervous activity, *The Bulletin of Mathematical Biophysics,* 5(4) (1943), pp. 115–133.

[26] B. Randell, On Alan Turing and the origins of digital computers, In B. Meltzer and D. Michie, editors, *Machine intelligence 7,* Edinburgh University Press, Edinburgh, UK, 1972, pp. 3–20.

[27] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, *Psychological Review,* 65(6) (1958), pp. 386–408.

[28] R. N. Shepard, Toward a universal law of generalization for psychological science, *Science,* 237(4820) (1987), pp. 1317–1323.

[29] H. T. Siegelmann, Turing on super-Turing and adaptivity, *Progress in biophysics and molecular biology,* 113(1) (2013), pp. 117–126.

[30] H. T. Siegelmann, E. D. Sontag, On the computational power of neural nets, *Proceedings of the Fifth Annual Workshop on Computational Learning Theory,* ACM, 1992, pp. 440–449.

[31] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al., Llama: Open and efficient foundation language models, arXiv preprint arXiv:2302.13971, 2023.

[32] A. M. Turing., On computable numbers, with an application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society,* s2-42(1) (1937), pp. 230–265.

[33] A. M. Turing., Computing machinery and intelligence, *Mind,* 59(236) (1950), pp. 433–460.

[34] L. G. Valiant, A theory of the learnable, *Communications of the ACM*, 27(11) (1984), pp. 1134–1142.

[35] C. Vallance, AI could replace equivalent of 300 million jobs – report, BBC News, March 2023.

[36] M. Vechev, E. Yahav, et al., Programming with „big code", *Foundations and Trends® in Programming Languages,* 3(4) (2016), pp. 231–284.

[37] R. V. Yampolskiy, Turing test as a defining feature of AI-completeness, *Artificial Intelligence, Evolutionary Computing and Metaheuristics,* Springer, 2013, pp. 3–17.

[38] R. Zellers, A. Holtzman, H. Rashkin, Y. Bisk, A. Farhadi, F. Roesner, Y. Choi, Defending against neural fake news, *Advances in Neural Information Processing Systems,* 32 (2019), pp. 9054–9065.

Kvaternió értékű polár–Fourier-momentumok algoritmusai digitális képfeldolgozáshoz[‡]

Busa Máté[*]

Eötvös József Collegium[**]

busamate.2000@gmail.com

Témavezető: Németh Zsolt
ELTE IK Numerikus Analízis Tanszék

1. Bevezetés

A momentumalapú képfeldolgozás már régóta kutatott terület az informatikában. A momentumok kiszámolásának motivációja az, hogy fontos információkat nyerjünk ki egy képről, amit aztán használhatunk például tömörítésre, vízjelezésre [11, 12] vagy objektumfelismerésre [1].

Kezdetben ezek a módszerek főleg szürkeárnyalatos képeknél voltak használatosak, ahol a képet egy valós értékű függvényként lehet reprezentálni. Színes képek kezelésére az adott lehetőséget, hogyha külön kezelték az RGB színcsatornákat, egymástól függetlenül.

Az utóbbi években megjelent egy új technika a színes képek reprezentálására: valós helyett kvaternió [6] értékű függvénnyel reprezentáljuk a képet. Ennek a reprezentálási módnak az előnye, hogy egységesen kezeli a színcsatornákat, így számos képfeldolgozási eljárás az emberi szem számára természetesebb eredményt ad.

[‡] A szerző 2023. évi OTDK-ra benyújtott dolgozatának rövidített változata.

[*] ELTE Informatikai Kar

[**] 2018–2021

2. Elért eredményeink

Kutatásunk során [2, 10] többféle kvaterniós polár–Fourier-típusú momentumot vizsgáltunk (mely egy aktívan kutatott terület a képfeldolgozás területén [3, 5, 7, 8, 13]), különös tekintettel azok hatékony kiszámítási módjaira:

- az ismert kvaterniós Legendre–Fourier-momentumok (QLFM) [7] kiszámolására bevezettünk egy új módszert, és ezt összehasonlítottuk más ismert kiszámolási algoritmusokkal;
- definiáltuk az új, kvaterniós Csebisev–Fourier-momentumokat (QCFM) és több kiszámítási módszert rájuk, melyek eltérő feltételek esetén is hatékonyan alkalmazhatók;
- implementáltuk a kvaterniós RadialHarmonic–Fourier-momentumokat (QRHFM) [8] többféle kiszámolási módszerrel, és ezeket összehasonlítottuk az előző momentumokkal.

A 3–4. szekciókban definiáljuk az új Csebisev–Fourier-momentumokat az őket meghatározó ortogonális függvényrendszeren keresztül, továbbá ismertetjük alkalmazásukat színes képek reprezentálására.

Az 5. szekcióban kidolgoztunk egy saját, interpolációs újramintavételezésen és mátrixaritmetikán alapuló, az ismerteknél jóval gyorsabb momentumszámítási és rekonstrukciós eljárást, ami egységesen alkalmazható a fent említett momentumtípusokra. A Csebisev–Fourier-momentumoknál továbbá egy gyors Fourier-transzformáción (FFT-n) alapuló, nagyobb mérettartományban még hatékonyabb momentum kiszámítási és rekonstrukciós eljárást készítettünk.

A kutatásunk során továbbá:

- Implementáltuk a polár–Fourier-momentumok numerikus kiszámítására használt főbb, a szakirodalomból ismert módszereket. A Legendre–Fourier-momentumok kiszámítására szolgáló Accurate módszert általánosítottuk a többi momentumtípusra. Végül egy új, a korábbiaknál időhatékonyabb módszert adtunk az Accurate pontrendszeren alapuló interpolációs képrekonstrukcióra.
- Numerikusan összehasonlítottuk a különböző polár–Fourier-momentumtípusokat és számítási algoritmusaikat.

- Definiáltunk a Csebisev–Fourier-momentummódszerhez eltolás, forgatás és skálázás invariánsokat, majd teszteltük ezek robusztusságát a többi polár–Fourier-momentummódszer invariánsaihoz képest.

- Egy lehetséges alkalmazásként az invariánsokat felhasználva objektumfelismerési algoritmust implementáltunk. A felismerés pontosságát széles körben teszteltük, és összehasonlítottuk különböző polár–Fourier-momentummódszerekkel;

- További alkalmazásként bitsorozatokat kódoló, szabad szemmel nem látható vízjeleket helyeztünk el képekre, melyek a képek transzformálása, más formátumba való exportálása, valamint zaj hozzáadása után is kis veszteséggel visszanyerhetőek.

3. Csebisev–Fourier-függvényrendszer

Az egységkörlemezen ortogonális Legendre–Fourier- (LF) [7], RadialHarmonic–Fourier- (RHF) [8] és az általunk definiált, eddig nem használatos Csebisev–Fourier-függvényrendszerek konstrukciója hasonló: egy $[0, 1]$ intervallumon (sugármenti) ortogonális függvényrendszer, valamint a $[0, 2\pi]$ intervallumon (szögelfordulási) ortogonális trigonometrikus függvényrendszer direkt szorzataként állnak elő. Az ilyen jellegű szorzatrendszerek jól használhatóak olyan képfeldolgozási feladatokban, amelyek képek geometriai transzformációihoz kötődnek.

Az egységkörlemezen a vizsgált függvényrendszerekkel természetesebb számolni polárkoordinátákkal, így azokat használunk.

Mindhárom, az egységkörlemezen ortogonális függvényrendszerhez szükségünk van az $E_q(\phi) = e^{\mathrm{i}q\phi}$ (komplex) trigonometrikus polinomokra, amik a $[0, 2\pi]$ intervallumon ortogonálisak:

$$\int_0^{2\pi} E_q(\phi) E_{q'}^*(\phi)\,\mathrm{d}\phi = 2\pi\delta_{qq'}, \tag{1}$$

ahol * a komplex (később a kvaterniós) konjugáltat jelöli, δ pedig a Kronecker-delta.

A továbbikban definiáljuk az új, Csebisev-polinomokon alapuló, az egységkörlemezen ortogonális függvényrendszert.

Jelölje T_p a p-ed fokú, elsőfajú Csebisev-polinomot, melynek függvényértékeit a

$$T_p(x) = \cos(p \cdot \arccos(x)) \quad (p \in \mathbb{N})$$

képlettel számolhatjuk.

Ez a függvényrendszer a $[-1, 1]$ intervallumon ortogonális a $\frac{1}{\sqrt{1-x^2}}$ súlyfüggvény mellett. Az ortogonalitást a következő integrálformula fejezi ki:

$$\int_{-1}^{1} T_p(x)T_{p'}(x)\frac{\mathrm{d}x}{\sqrt{1-x^2}} = \begin{cases} 0, & \text{ha } p \neq p', \\ \pi, & \text{ha } p = p' = 0, \\ \pi/2, & \text{ha } p = p' \neq 0. \end{cases}$$

Számunkra azonban $[-1, 1]$ helyett a polársugarak mentén, azaz a $[0, 1]$ intervallumon van szükség az ortogonalitásra, ezért az alábbi, eltolt Csebisev polinomokat használjuk:

$$\bar{T}_p(r) = T_p(2r^2 - 1).$$

Ekkor az eltolt Csebisev polinomok értéke is könnyen számolható az eredeti polinomokból:

$$\bar{T}_p(r) = T_p(2r^2 - 1) = \cos(p \cdot \arccos(2r^2 - 1)).$$

Az eltolt Csebisev polinomokra már teljesül az alábbi, számunkra hasznos ortogonalitás, ezúttal $\frac{1}{\sqrt{r^2-r^4}}$ súlyfüggvény mellett (ami könnyen levezethető a $x = 2r^2 - 1$ helyettesítéssel):

$$\int_0^1 \bar{T}_p(r)\bar{T}_{p'}(r)\frac{r\,\mathrm{d}r}{\sqrt{r^2-r^4}} = \int_0^1 \bar{T}_p(r)\bar{T}_{p'}(r)\frac{\mathrm{d}r}{\sqrt{1-r^2}}$$
$$= \frac{1}{2}\int_{-1}^{1} T_p(x)T_{p'}(x)\frac{\mathrm{d}x}{\sqrt{1-x^2}} = \begin{cases} 0, & \text{ha } p \neq p', \\ \pi/2, & \text{ha } p = p' = 0, \\ \pi/4, & \text{ha } p = p' \neq 0. \end{cases} \tag{2}$$

Az előzőek alapján elkészíthetjük az egységkörlemezen ortogonális

$$C_{pq}(r, \phi) = \bar{T}_p(r)E_q(\phi)$$

(Csebisev–Fourier) függvényrendszert:

$$\begin{aligned}\langle C_{pq}, C_{p'q'}\rangle &= \int_0^{2\pi}\int_0^1 C_{pq}(r,\phi)C^*_{p'q'}(r,\phi)\frac{\mathrm{d}r}{\sqrt{1-r^2}}\,\mathrm{d}\phi \\ &= \int_0^1 \bar{T}_p(r)\bar{T}_{p'}(r)\frac{\mathrm{d}r}{\sqrt{1-r^2}}\cdot\int_0^{2\pi} E_q(\phi)E^*_{q'}(\phi)\mathrm{d}\phi \\ &= \begin{cases} 0, & \text{ha } p\neq p' \text{ vagy } q\neq q', \\ \pi^2, & \text{ha } p=p'=0,\ q=q', \\ \pi^2/2, & \text{ha } p=p'\neq 0,\ q=q'. \end{cases}\end{aligned} \tag{3}$$

4. Képek ábrázolása Csebisev–Fourier-momentumokkal

Láthattuk, hogy az előző függvényrendszer az egységkörlemezen értelmezett, ezért valamilyen módon le kell képeznünk a feldolgozandó képünket az egységkörlemezre.

Feltehetjük, hogy a képünk négyzet alakú, ami $M \times M$ pixelből áll.

A következőkben bemutatunk két módot az egységkörlemezre való áttérésre.

Részleges ábrázolás

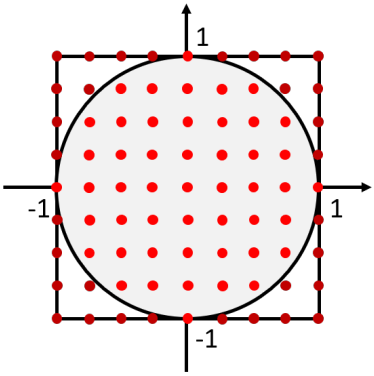
A képünk bal alsó sarkának koordinátája legyen $(-1,-1)$, a jobb felsőé pedig $(1,1)$. Ekkor a képnek csak az egységkörlemezbe eső pontjaival foglalkozunk. Ez a fajta ábrázolási mód nagyobb pontosságot eredményez, ezért jól használható képelemzési feladatokban (például objektumfelismerésnél vagy vízjelezésnél).

Az 1. ábrán látható az egységkör és a kép elhelyezkedése. A piros pontok a kép pixeleit jelölik.

A kép pixeleinek koordinátái Descartes-féle koordináta-rendszerben felírva $(x_i, y_j) = \left(\frac{2i}{M-1}-1, \frac{2j}{M-1}-1\right)$ alakúak, ahol $0 \le i,j < M$.

Teljes ábrázolás

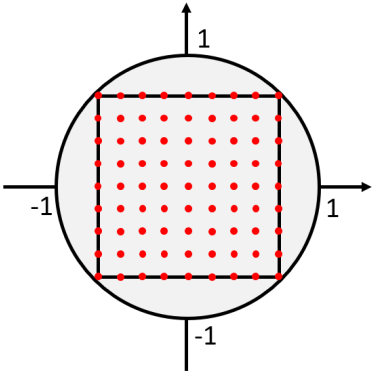
A képünk bal alsó sarkának koordinátája legyen $\left(-\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}\right)$, a jobb felsőé pedig $\left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right)$. Ekkor a kép teljes egészében az egységkör-



1. ábra. Kép részleges lefedése

lemezbe esik. Ez a fajta ábrázolási mód használatos, hogyha szükséges a teljes kép megőrzése (például tömörítés vagy átméretezési feladatok esetében).

A 2. ábrán látható az egységkör és a kép elhelyezkedése. A piros pontok a kép pixeleit jelölik.



2. ábra. Kép teljes lefedése

A kép pixeleinek koordinátái Descartes-féle koordináta-rendszerben $(X_i, Y_j) = \left(\frac{\sqrt{2}i}{M-1} - \frac{\sqrt{2}}{2}, \frac{\sqrt{2}j}{M-1} - \frac{\sqrt{2}}{2}\right)$, ahol $0 \leq i, j < M$.

Érdemes megjegyezni, hogy az egységkörlemezbe eső, de a képen kívül levő részeket jellemzően nem feketével töltjük ki, hanem sugárirányban kiterjesztjük a kép szélén levő színeket.

4.1. Szürkeárnyalatos kép ábrázolása CF momentumokkal

Szürkeárnyalatos képeket $f : [0,1] \times [0,2\pi) \to \mathbb{R}$ valós függvényként reprezentálhatunk az egységkörlemezen, ahol $f(r,\phi) \in \mathbb{R}$ jelöli a kép (r,ϕ) polárpontjában vett értékét.

A továbbiakban tegyük fel, hogy az f függvény négyzetesen integrálható az egységkörlemezen. Látható, hogy a direktszorzatos konstrukció következtében a négyzetesen integrálható függvények körében a korábban definiált függvényrendszer teljes ortogonális rendszer.

A Csebisev–Fourier-függvényrendszer ortogonalitását és teljességét kihasználva a kép egy pontja felírható a következő módon:

$$f(r,\phi) = \sum_{p=0}^{\infty} \sum_{q=-\infty}^{\infty} M_{pq}^{(C)} C_{pq}(r,\phi),$$

ahol $M_{pq}^{(C)} \in \mathbb{C}$ komplex értékek a Csebisev–Fourier-momentumok: egy tetszőleges $M_{pq}^{(C)}$ momentum az f-nek C_{pq}-val vett skalárszorzata az egységkörlemezen (felszorozva megfelelő konstansokkal, az ortogonalitás képletéből adódóan):

$$M_{pq}^{(C)}(f) = M_{pq}^{(C)} =$$
$$= \begin{cases} \frac{1}{\pi^2} \langle f, C_{0q} \rangle = \frac{1}{\pi^2} \int_0^{2\pi} \int_0^1 f(r,\phi) C_{0q}^*(r,\phi) \frac{\mathrm{d}r}{\sqrt{1-r^2}}\, \mathrm{d}\phi, & \text{ha } p = 0, \\ \frac{2}{\pi^2} \langle f, C_{pq} \rangle = \frac{2}{\pi^2} \int_0^{2\pi} \int_0^1 f(r,\phi) C_{pq}^*(r,\phi) \frac{\mathrm{d}r}{\sqrt{1-r^2}}\, \mathrm{d}\phi, & \text{ha } p \neq 0. \end{cases} \quad (4)$$

Természetesen a gyakorlatban az összes momentum kiszámolására nincs lehetőség, csak valamilyen előre rögzített p_0, q_0 indexekig tesszük ezt meg. Ezek felhasználásával a képnek is csak egy megközelítő értéke adható meg:

$$f(r,\phi) \approx f_{p_0 q_0}^{(C)}(r,\phi) = \sum_{p=0}^{p_0-1} \sum_{q=-q_0+1}^{q_0-1} M_{pq}^{(C)} C_{pq}(r,\phi). \quad (5)$$

4.2. Színes kép ábrázolása CF momentumokkal

Láthatjuk, hogy az előzőekben leírt módszer csak valós értékű f függvény reprezentálására alkalmas. Ebből adódóan, ha egy színes képet akarnánk így feldolgozni, akkor szét kellene bontanunk a képet 3 színcsatornára, és külön foglalkozni velük. Ekkor azonban elvesznek a színkomponensek közti esetleges összefüggések.

Tisztán képzetes kvaterniókkal [6] azonban könnyen tudunk egy $\hat{f}(r,\phi)$ színes pixelt reprezentálni:

$$\hat{f}(r,\phi) = f_R(r,\phi)\cdot \mathrm{i} + f_G(r,\phi)\cdot \mathrm{j} + f_B(r,\phi)\cdot \mathrm{k},$$

ahol f_R, f_G, f_B az adott színcsatornákat reprezentálják, és $\mathrm{i},\mathrm{j},\mathrm{k}$ a kvaterniós képzetes egységek.

Komplex helyett definiálhatunk kvaternió értékű polár–Fourier-függvényrendszereket, illetve momentumokat, hogyha az eddig használt $E_q(\phi)$ függvényeket lecseréljük a kvaterniós megfelelőjükre:

$$\hat{E}_q(\phi) = e^{\mu q \phi}, \quad \text{ahol} \quad \mu = \frac{1}{\sqrt{3}}\cdot(\mathrm{i}+\mathrm{j}+\mathrm{k})$$

a tisztán képzetes kvaterniós egység.

A kvaternió értékű Csebisev–Fourier-függvényrendszer például a következő alakban írható fel:

$$\hat{C}_{pq}(r,\phi) = \bar{T}_p(r)\hat{E}_q(\phi).$$

Ekkor a színes kép egy pontja az alábbi módon határozható meg:

$$\begin{aligned}\hat{f}(r,\phi) &= \sum_{p=0}^{\infty}\sum_{q=-\infty}^{\infty} \hat{M}_{pq}^{(C)}\hat{C}_{pq}(r,\phi) \approx \hat{f}_{p_0 q_0}^{(C)}(r,\phi)\\ &= \sum_{p=0}^{p_0-1}\sum_{q=-q_0+1}^{q_0-1} \hat{M}_{pq}^{(C)}\hat{C}_{pq}(r,\phi),\end{aligned} \tag{6}$$

ahol $\hat{M}_{pq}^{(C)} \in \mathbb{H}$ kvaterniós értékek a jobb oldali Csebisev–Fourier-momentumok (a kommutativitás hiánya miatt beszélhetünk bal és jobb oldali kvaterniós momentumokról).

Egy tetszőleges jobb oldali kvaterniós Csebisev–Fourier-momentum a következőféleképpen írható fel:

$$\hat{M}_{pq}^{(C)} = \begin{cases} \frac{1}{\pi^2}\left\langle \hat{f}, \hat{C}_{0q} \right\rangle = \frac{1}{\pi^2}\int_0^{2\pi}\int_0^1 \hat{f}(r,\phi)\hat{C}_{0q}^*(r,\phi)\frac{\mathrm{d}r}{\sqrt{1-r^2}}\,\mathrm{d}\phi, \text{ ha } p=0, \\ \frac{2}{\pi^2}\left\langle \hat{f}, \hat{C}_{pq} \right\rangle = \frac{2}{\pi^2}\int_0^{2\pi}\int_0^1 \hat{f}(r,\phi)\hat{C}_{pq}^*(r,\phi)\frac{\mathrm{d}r}{\sqrt{1-r^2}}\,\mathrm{d}\phi, \text{ ha } p\neq 0. \end{cases}$$

A kvaterniós Csebisev–Fourier-momentumokhoz (QCFM) hasonló módon definiálhatók a kvaterniós Legendre–Fourier- (QLFM) és RadialHarmonic–Fourier-momentumok (QRHFM).

4.3. Áttérés kvaternió értékű CF momentumokra

A következőkben megmutatjuk, hogy egyszerűen áttérhetünk komplexről kvaterniós Csebisev–Fourier-momentumokra. Ez hasonlóan megtehető a másik két polár–Fourier-momentum típusnál is.

Jelölje $M_{pq}^{(C)}(f_R), M_{pq}^{(C)}(f_G), M_{pq}^{(C)}(f_B)$ értékek az adott színcsatornához tartozó komplex Csebisev–Fourier-momentumokat.

Ekkor a jobb oldali kvaterniós Csebisev–Fourier-momentumok felírhatóak

$$\hat{M}_{pq}^{(C)} = \alpha_{pq} + \beta_{pq}i + \gamma_{pq}j + \delta_{pq}k$$

alakban, ahol

$$\alpha_{pq} = -\frac{1}{\sqrt{3}} \cdot \Im(M_{pq}^{(C)}(f_R) + M_{pq}(f_G) + M_{pq}(f_B)),$$

$$\beta_{pq} = \Re(M_{pq}(f_R)) + \frac{1}{\sqrt{3}} \cdot \Im(M_{pq}^{(C)}(f_G) - M_{pq}(f_B)),$$

$$\gamma_{pq} = \Re(M_{pq}(f_G)) + \frac{1}{\sqrt{3}} \cdot \Im(M_{pq}^{(C)}(f_B) - M_{pq}(f_R)),$$

$$\delta_{pq} = \Re(M_{pq}(f_B)) + \frac{1}{\sqrt{3}} \cdot \Im(M_{pq}^{(C)}(f_R) - M_{pq}(f_G)).$$

Ezek a formulák a momentumok definíciójából vezethetők le, a formulát a kvaterniós trigonometrikus tag szerint kifejtve.

Hasonló számolással a kvaterniós Csebisev–Fourier-momentumokat felhasználva megkaphatjuk mindhárom színcsatorna értékeit egy adott (r, ϕ) pontban a következő módon:

$$f_R(r,\phi) = \Re(\tilde{\beta}_{pq}) + \frac{1}{\sqrt{3}} \cdot \Im(\tilde{\alpha}_{pq} + \tilde{\gamma}_{pq} - \tilde{\delta}_{pq}),$$

$$f_G(r,\phi) = \Re(\tilde{\gamma}_{pq}) + \frac{1}{\sqrt{3}} \cdot \Im(\tilde{\alpha}_{pq} + \tilde{\delta}_{pq} - \tilde{\beta}_{pq}),$$

$$f_B(r,\phi) = \Re(\tilde{\delta}_{pq}) + \frac{1}{\sqrt{3}} \cdot \Im(\tilde{\alpha}_{pq} + \tilde{\beta}_{pq} - \tilde{\gamma}_{pq}),$$

ahol

$$\tilde{\alpha}_{pq} = \sum_{p=0}^{\infty} \sum_{q=-\infty}^{\infty} \alpha_{pq} C_{pq}(r,\phi) \approx \sum_{p=0}^{p_0-1} \sum_{q=-q_0+1}^{q_0-1} \alpha_{pq} C_{pq}(r,\phi),$$

$$\tilde{\beta}_{pq} = \sum_{p=0}^{\infty} \sum_{q=-\infty}^{\infty} \beta_{pq} C_{pq}(r,\phi) \approx \sum_{p=0}^{p_0-1} \sum_{q=-q_0+1}^{q_0-1} \beta_{pq} C_{pq}(r,\phi),$$

$$\tilde{\gamma}_{pq} = \sum_{p=0}^{\infty} \sum_{q=-\infty}^{\infty} \gamma_{pq} C_{pq}(r,\phi) \approx \sum_{p=0}^{p_0-1} \sum_{q=-q_0+1}^{q_0-1} \gamma_{pq} C_{pq}(r,\phi),$$

$$\tilde{\delta}_{pq} = \sum_{p=0}^{\infty} \sum_{q=-\infty}^{\infty} \delta_{pq} C_{pq}(r,\phi) \approx \sum_{p=0}^{p_0-1} \sum_{q=-q_0+1}^{q_0-1} \delta_{pq} C_{pq}(r,\phi).$$

Látjuk tehát, hogy a továbbiakban elég a komplex értékű esetet tárgyalnunk; onnan könnyedén áttérhetünk kvaterniós esetre.

5. Diszkrét módszer

Természetes módon használhatjuk a rekonstrukciós képleteket (például Csebisev–Fourier esetben az (5) képletet) közvetlenül a kép pixelpontjaiban. Jelenleg polár–Fourier-momentumoknál és más hasonló jellegű momentumtípusoknál szinte kizárólag ez a rekonstrukciós módszer van használatban.

Ekkor egy pixelpontbeli érték kiszámítása (a függvényrendszer értékeinek előszámítása mellett) $O(p_0 q_0)$ futási idővel történik, ezért az összes pixel meghatározásához $O(p_0 q_0 M^2)$ időre van szükség.

Egyik fő célkitűzésünk volt időhatékonyabb eljárások keresése rekonstrukcióra, melyek megőrzik a rekonstrukció pontosságát is. Kidolgoztunk több módszert is, melyek jelentősen gyorsabbak az eddig használt módszernél, és nem vesztenek jelentősen a pontosságból sem.

Ebben a szekcióban bemutatunk egy integrál diszkretizáción alapuló új módszert az általunk definiált Csebisev–Fourier-momentumok kiszámolására (melyet kiterjesztettünk a Legendre–Fourier-momentumok kezelésére is). Jelenleg egyetlen hasonló módszerről tudunk a szakirodalomban [8] (melyet az összehasonlítás végett szintén implementáltunk), de ott a fő motiváció a trigonometrikus rendszerre való visszavezetés, és nem a diszkrét ortogonalitás elérése volt. Az ily módon konstruált RadialHarmonic–Fourier-rendszer azonban számos, elméleti és gyakorlati szempontból előnytelen tulajdonsággal rendelkezik. A mi megközelítésünk ezzel szemben széles körben alkalmazható, általános módszer a momentumszámítás és a rekonstrukció hatékony kezelésére.

A módszerhez szükségünk van előre rögzített $N_R, N_A \in \mathbb{N}^+$ értékekre, amik a mintavételezési pontrendszer nagyságát fogják meghatározni sugár és szögelfordulási irány szerint. Általánosságban elmondható, hogy a Diszkrét módszer mindhárom polár–Fourier-momentumtípusnál $O(p_0 q_0 N_A + p_0 N_R N_A + M^2)$ futásidővel határozza meg a momentumokat.

A módszerünk a momentumértékek számolását ortogonális mátrixokkal való szorzásokra vezeti vissza (ami numerikusan stabil), így hatékonyan implementálható, és rendkívül könnyen párhuzamosítható.

Bemutatunk továbbá a Csebisev–Fourier-momentumok esetében egy gyors Fourier-transzformáción (FFT) alapuló, aszimptotikusan még gyorsabb momentumszámítási módot.

Megadunk továbbá egy új rekonstrukciós eljárást, ami használható mindegyik polár–Fourier-momentummódszernél, és az eddig használt eljárás $O(p_0 q_0 M^2)$ futási ideje helyett $O(p_0 q_0 N_A + p_0 N_R N_A + M^2)$ időben fut. Látni fogjuk, hogy ez szintén tovább gyorsítható Csebisev–Fourier és RadialHarmonic–Fourier esetben FFT-vel.

Pontrendszer

A Diszkrét módszer által használt pontrendszer polár–Fourier-momentumtípusonként eltérő, de általánosságban felírható a következő alakban polárkoordinátákkal:

$$\{\, (r_i, \phi_j) \mid i < N_R; j < N_A; i, j \in \mathbb{N} \,\}$$

ahol r_i-k az adott momentumtípus integráldiszkretizációjához szükséges alappontok, $\phi_j = \frac{j2\pi}{N_A}$ pedig a szögtartomány egyenletes felosztása.

Mintavételezzük az f képünket az (r_i, ϕ_j) polárpontokban. A mintavételezésre itt bicubic [15] interpolációt használunk, aminek a lényege, hogy először origótól való távolság szerint, aztán polárszög szerint alkalmazunk cubic (köbös) interpolációt. Ilyen módon megkaphatjuk az $f(r_i, \phi_j)$ függvényértékek egy megközelítő értékét.

5.1. CF momentumok meghatározása

A momentumok meghatározásához szükséges (Csebisev–Fourier esetben (4)) integrált diszkretizációval kezeljük.

Látni fogjuk, hogy egy tetszőleges típusú polár–Fourier M_{pq} momentum $O(N_R N_A)$ komplexitással számolható, ezért az összes momentum kiszámolásához $O(p_0 q_0 N_R N_A)$ időre lenne szükség. Szerencsére kihasználva a Diszkrét alappontok körkörös, illetve sugármenti elhelyezkedését, ez csökkenthető $O(p_0 q_0 N_A + p_0 N_R N_A)$-re. A Legendre–Fourier-momentummódszer kivételével pedig még gyorsabb eljárást tudunk megadni.

Jelöljük T_{N_R} gyökeit $x_i^{(C)}$-vel. Könnyen látható, hogy

$$x_i^{(C)} = \cos\left(\frac{(i+0.5)\pi}{N_R}\right),$$

és pontosan N_R darab gyök van ($i < N_R$, $i \in \mathbb{N}$). Jelöljük $\bar{T}_{N_R}$ gyökeit $r_i^{(C)}$-vel, ekkor $r_i^{(C)} = \sqrt{\frac{x_i^{(C)}+1}{2}}$.

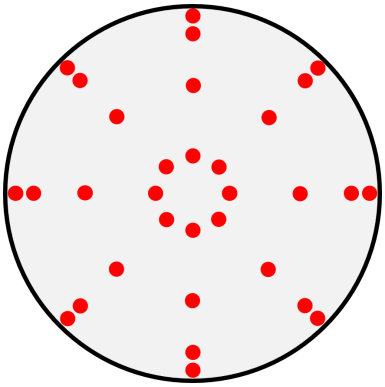
Ekkor a pontrendszerünk legyen tehát a következő alakú:

$$\left\{ (r_i^{(C)}, \phi_j) = \left(\sqrt{\frac{x_i^{(C)}+1}{2}}, \frac{j2\pi}{N_A} \right) \mid i < N_R; j < N_A; i, j \in \mathbb{N} \right\}.$$

A 3. ábrán látható ezen pontok elhelyezkedése.

Szükségünk lesz a későbbiekben az E_q polinomok, illetve a $\bar{T}_p$ polinomok diszkrét ortogonalitásához. Lásd például [4] alapján

$$\sum_{j=0}^{N_A-1} E_{q'}(\phi_j) E_q^*(\phi_j) = \frac{\delta_{qq'}}{N_A}, \tag{7}$$



3. ábra. Csebisev–Fourier Diszkrét pontrendszer ($N_R = 4, N_A = 8$)

továbbá [9] alapján

$$\sum_{i=0}^{N_R-1} \bar{T}_{p'}(r_i^{(C)})\,\bar{T}_p(r_i^{(C)}) = \sum_{i=0}^{N_R-1} T_{p'}(x_i^{(C)})\,T_p(x_i^{(C)}) = \begin{cases} 0, & \text{ha } p \neq p', \\ N_R, & \text{ha } p = p' = 0, \\ N_R/2, & \text{ha } p = p' \neq 0. \end{cases} \tag{8}$$

A momentumok meghatározásához induljunk ki $p \neq 0$ esetből:

$$\begin{aligned} M_{pq}^{(C)} \approx & \frac{2}{\pi^2} \left\langle f_{p_0 q_0}^{(C)}, C_{pq} \right\rangle = \frac{2}{\pi^2} \int_0^{2\pi} \int_0^1 f_{p_0 q_0}^{(C)}(r,\phi) C_{pq}^*(r,\phi) \frac{\mathrm{d}r}{\sqrt{1-r^2}}\,\mathrm{d}\phi \\ = & \frac{2}{\pi^2} \int_0^{2\pi} \int_0^1 \left(\sum_{p'=0}^{p_0-1} \sum_{q'=-q_0+1}^{q_0-1} M_{p'q'}^{(C)} C_{p'q'}(r,\phi) \right) C_{pq}^*(r,\phi) \frac{\mathrm{d}r}{\sqrt{1-r^2}}\,\mathrm{d}\phi \\ = & \frac{2}{\pi^2} \sum_{p'=0}^{p_0-1} \sum_{q'=-q_0+1}^{q_0-1} \left(M_{p'q'}^{(C)} \int_0^{2\pi} \int_0^1 C_{p'q'}(r,\phi) C_{pq}^*(r,\phi) \frac{\mathrm{d}r}{\sqrt{1-r^2}}\,\mathrm{d}\phi \right) \\ = & \frac{2}{\pi^2} \sum_{p'=0}^{p_0-1} \sum_{q'=-q_0+1}^{q_0-1} \left(M_{p'q'}^{(C)} \int_0^1 \bar{T}_{p'}(r) \bar{T}_p(r) \frac{\mathrm{d}r}{\sqrt{1-r^2}} \cdot \int_0^{2\pi} E_{q'}(\phi) E_q^*(\phi)\,\mathrm{d}\phi \right). \end{aligned} \tag{9}$$

Innen (1) és (2), valamint (7) és (8) ortogonalitásokat felhasználva:

$$
\begin{aligned}
M_{pq}^{(C)} \approx & \frac{2}{\pi^2} \sum_{p'=0}^{p_0-1} \sum_{q'=-q_0+1}^{q_0-1} \left(M_{p'q'}^{(C)} \int_0^1 \bar{T}_{p'}(r) \bar{T}_p(r) \frac{\mathrm{d}r}{\sqrt{1-r^2}} \cdot \int_0^{2\pi} E_{q'}(\phi) E_q^*(\phi)\, \mathrm{d}\phi \right) \\
= & \frac{2}{\pi^2} \sum_{p'=0}^{p_0-1} \sum_{q'=-q_0+1}^{q_0-1} \left(M_{p'q'}^{(C)} \frac{\pi}{2N_R} \sum_{i=0}^{N_R-1} \bar{T}_{p'}(r_i^{(C)}) \bar{T}_p(r_i^{(C)}) \cdot \frac{2\pi}{N_A} \sum_{j=0}^{N_A-1} E_{q'}(\phi_j) E_q^*(\phi_j) \right) \\
= & \frac{1}{2N_R N_A} \sum_{i=0}^{N_R-1} \sum_{j=0}^{N_A-1} \left(\left(\sum_{p'=0}^{p_0-1} \sum_{q'=-q_0+1}^{q_0-1} M_{p'q'}^{(C)} \bar{T}_{p'}(r_i^{(C)}) E_{q'}(\phi_j) \right) \bar{T}_p(r_i^{(C)}) E_q^*(\phi_j) \right) \\
= & \frac{1}{2N_R N_A} \sum_{i=0}^{N_R-1} \sum_{j=0}^{N_A-1} \left(\left(\sum_{p'=0}^{p_0-1} \sum_{q'=-q_0+1}^{q_0-1} M_{p'q'}^{(C)} C_{p'q'}(r_i^{(C)}, \phi_j) \right) C_{pq}^*(r_i^{(C)}, \phi_j) \right) \\
= & \frac{1}{2N_R N_A} \sum_{i=0}^{N_R-1} \sum_{j=0}^{N_A-1} f_{p_0 q_0}^{(C)}(r_i^{(C)}, \phi_j) C_{pq}^*(r_i^{(C)}, \phi_j).
\end{aligned}
\tag{10}
$$

Tehát az általunk bevezetett pontrendszer segítségével a következőképpen határozhatjuk meg a momentumokat:

$$
M_{pq}^{(C)} \approx \begin{cases} \frac{1}{4N_R N_A} \sum_{i=0}^{N_R-1} \sum_{j=0}^{N_A-1} f(r_i^{(C)}, \phi_j) C_{0q}^*(r_i^{(C)}, \phi_j), & \text{ha } p = 0, \\ \frac{1}{2N_R N_A} \sum_{i=0}^{N_R-1} \sum_{j=0}^{N_A-1} f(r_i^{(C)}, \phi_j) C_{pq}^*(r_i^{(C)}, \phi_j), & \text{ha } p \neq 0. \end{cases}
\tag{11}
$$

5.2. Momentumszámítás gyorsítása I.

Az előbbiekben láthattuk, hogy egy momentum kiszámolása a kettős szumma miatt $O(N_R N_A)$ időbe telik, így az összes momentum kiszámolása $O(p_0 q_0 N_R N_A)$ ideig tartana.

Megmutatjuk, hogy ez hogyan gyorsítható a Csebisev–Fourier-momentumoknál (a többi polár–Fourier-momentumtípusnál ez hasonlóan megtehető).

(11) képletből kiindulva ($p \neq 0$ esetet tekintve):

$$
\begin{aligned}
M_{pq}^{(C)} \approx & \frac{1}{2N_R N_A} \sum_{i=0}^{N_R-1} \sum_{j=0}^{N_A-1} f(r_i^{(C)}, \phi_j) \bar{T}_p(r_i^{(C)}) E_q^*(\phi_j) \\
= & \frac{1}{2N_R N_A} \sum_{j=0}^{N_A-1} \left(E_q^*(\phi_j) \sum_{i=0}^{N_R-1} f(r_i^{(C)}, \phi_j) \bar{T}_p(r_i^{(C)}) \right).
\end{aligned}
$$

Láthatjuk, hogy a belső szumma értéke nem függ q-tól, az összes lehetséges értékének kiszámítása $O(p_0 N_R N_A)$ ideig tart. Ha ezeket az értékeket már ismerjük, akkor egy momentum kiszámolása csak $O(N_A)$ ideig tart a külső szumma miatt. Tehát az összes momentum kiszámítása (a belső szummák előkalkulációjával együtt) $O(p_0 q_0 N_A + p_0 N_R N_A)$ ideig tart.

Megjegyzendő, hogy a szummák sorrendjének megcserélésével egy hasonló módszer adható, melynek futási ideje $O(p_0 q_0 N_R + q_0 N_R N_A)$.

5.3. Momentumszámítás gyorsítása II.

Lehetőségünk van a Csebisev–Fourier-momentumok kiszámolását FFT-vel még jobban gyorsítani. Ez hasonlóan megtehető a RadialHarmonic–Fourier-momentumokkal is [8] alapján.

Rögzítsük a mintavételezési nagyságokat p_0, q_0 értékek alapján: $N_R := p_0, N_A := 2q_0 - 1$. Ekkor a (11) képletből adódik, hogy:

$$\begin{aligned} M_{pq}^{(C)} \approx & \frac{1}{2p_0(2q_0-1)} \sum_{j=0}^{2q_0-2} \left(E_q^*(\phi_j) \sum_{i=0}^{p_0-1} f(r_i^{(C)}, \phi_j) \bar{T}_p(r_i^{(C)}) \right) \\ = & \frac{1}{2p_0(2q_0-1)} \sum_{j=0}^{2q_0-2} \left(E_q^*(\phi_j) \sum_{i=0}^{p_0-1} f(r_i^{(C)}, \phi_j) T_p(x_i^{(C)}) \right). \end{aligned}$$

Ekkor észrevehetjük, hogy $x_i^{(C)} = \cos\left(\frac{(i+0.5)\pi}{N_R}\right)$ miatt [16] alapján a belső szumma felírható egy II-es típusú DCT-ként (Discrete Cosine Transform, FFT speciális esete). Emiatt az összes lehetséges belső szumma kiszámítása nem $O(p_0 N_R N_A) = O(p_0^2 q_0)$, hanem csupán $O(p_0 \log(p_0) q_0)$ ideig tart.

A külső szumma pedig $\phi_j = \frac{j2\pi}{N_A}$ miatt [16] alapján egy FFT-ként írható fel. Emiatt ha a belső szummák értékét már ismerjük, az összes momentum kiszámítása $O(p_0 q_0 N_A) = O(p_0 q_0^2)$ helyett $O(p_0 q_0 \log(q_0))$ időben elvégezhető.

Tehát az összes momentum kiszámolása a belső szummák előkalkulációjával együtt $O(p_0 q_0 (\log(p_0) + \log(q_0)))$ ideig tart, ami jelentősen gyorsabb mint az eddig használt módszerek.

5.4. Rekonstrukció gyorsítása I.

Ebben a szekcióban bemutatunk egy új, gyorsabb rekonstrukciót a Diszkrét pontrendszer felhasználásával. Az eddig használatos rekonstrukciós módszerekkel ellentétben most nem az eredeti pixelpontokban, hanem a Diszkrét pontrendszerben állítjuk elő az f színértékeket.

A következőkben a Csebisev–Fourier-momentummódszeren fogjuk demonstrálni az eljárást, de ez könnyedén átültethető a többi polár–Fourier-momentummódszerre is. Az (5) rekonstrukciós képletet felhasználva a Diszkrét pontrendszeren:

$$\begin{aligned} f(r_i^{(C)}, \phi_j) &\approx \sum_{p=0}^{p_0-1} \sum_{q=-q_0+1}^{q_0-1} M_{pq}^{(C)} C_{pq}(r_i^{(C)}, \phi_j) \\ &= \sum_{p=0}^{p_0-1} \left(\bar{T}_p(r_i^{(C)}) \sum_{q=-q_0+1}^{q_0-1} M_{pq}^{(C)} E_q(\phi_j) \right). \end{aligned} \tag{12}$$

Láthatjuk, hogy a belső szumma értéke nem függ i-től, az összes lehetséges értékének kiszámítása $O(p_0 q_0 N_A)$ ideig tart. Ha ezeket az értékeket már ismerjük, akkor egy Diszkrét pontrendszerbeli színérték kiszámolása csak $O(p_0)$ ideig tart a külső szumma miatt. Tehát az összes f érték kiszámolása (a belső szummák előkalkulációjával együtt) $O(p_0 q_0 N_A + p_0 N_R N_A)$ ideig tart.

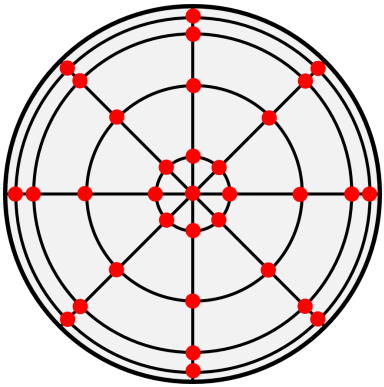
Osszuk fel a pontrendszerünket polártartományokra a 4. ábrán látható módon.

Az eredeti pixelpontokban megkaphatjuk f egy megközelítő értékét, ha a megfelelő polártartományban bilineárisan interpolálunk [14].

A rekonstrukció futási ideje $O(p_0 q_0 N_A + p_0 N_R N_A + M^2)$ az interpolációkkal együtt.

Megjegyzendő, hogy a szummák sorrendjének megcserélésével egy hasonló módszer adható, $O(p_0 q_0 N_R + q_0 N_R N_A + M^2)$ futási idővel.

Megemlítendő továbbá, hogy Legendre–Fourier esetben annak a meghatározására, hogy egy kép pixelpontját melyik polártartományban kell interpolálni csak $O(\log(N_R))$ futásidejű algoritmust adtunk, így itt a rekonstrukció futási ideje $O(p_0 q_0 N_A + p_0 N_R N_A + M^2 \log(N_R))$, vagy $O(p_0 q_0 N_R + q_0 N_R N_A + M^2 \log(N_R))$.



4. ábra. Csebisev–Fourier Diszkrét pontrendszer tartományokra osztása

5.5. Rekonstrukció gyorsítása II.

Lehetőségünk van Csebisev–Fourier esetben a rekonstrukciót FFT-vel még jobban gyorsítani. Ez hasonlóan megtehető a RadialHarmonic–Fourier esetben is.

Rögzítsük a mintavételezési nagyságokat p_0, q_0 értékek alapján: $N_R := p_0, N_A := 2q_0 - 1$. Ekkor a (12) képletből adódik, hogy:

$$\begin{aligned} M_{pq}^{(C)} &\approx \sum_{p=0}^{p_0-1} \left(\bar{T}_p(r_i^{(C)}) \sum_{q=-q_0+1}^{q_0-1} M_{pq}^{(C)} E_q(\phi_j) \right) \\ &= \sum_{p=0}^{p_0-1} \left(T_p(x_i^{(C)}) \sum_{q=-q_0+1}^{q_0-1} M_{pq}^{(C)} E_q(\phi_j) \right). \end{aligned} \tag{13}$$

Ekkor észrevehetjük, hogy $\phi_j = \frac{j2\pi}{N_A}$ miatt [16] alapján a belső szumma felírható egy IFFT-ként (Inverse Fast Fourier Transform). Emiatt az összes lehetséges belső szumma kiszámítása $O(p_0 q_0 N_A) = O(p_0 q_0^2)$ helyett pusztán $O(p_0 q_0 \log(q_0))$ ideig tart.

A külső szumma pedig $x_i^{(C)} = \cos\left(\frac{(i+0.5)\pi}{N_R}\right)$ miatt [16] alapján egy III-as típusú DCT-ként írható fel. Emiatt ha a belső szummák értékét már ismerjük, az összes f színérték kiszámítása $O(p_0 N_R N_A) = O(p_0^2 q_0)$ helyett $O(p_0 \log(p_0) q_0)$ időben elvégezhető.

Tehát az összes pixelérték kiszámolása a belső szummák előkalkulációjával és interpolációval együtt $O(p_0 q_0(\log(p_0)+\log(q_0))+M^2)$ ideig tart. Ez a módszer aszimptotikusan gyorsabb az eddig mutatottaknál, mely nagy felbontású képeknél, nagy momentumindexek esetén mutatkozik meg. Kisebb méretű bemenet esetén azt láthatjuk, hogy hatékonyabbak az FFT-t nem használó megoldások.

6. Összegzés

Az elért eredményeink többrétűek. Kutatásuk egyik fő eredménye a közelmúltban bevezetett, polár–Fourier típusú Legendre–Fourier-függvényrendszer momentumaival történő számítások hatékony algoritmusainak megalkotása volt: sikerült az eddig használt módszereknél pontosabb és gyorsabb eljárást kidolgoznunk, a diszkrét integráláson alapuló Diszkrét módszert.

Következő főbb eredményként definiáltunk egy új momentumtípust, a Csebisev–Fourier-momentumokat, melyekre az eddig ismert számítási módszerek mellett a saját Diszkrét módszerünk is hatékonyan alkalmazható. A kapott momentumok a Legendre–Fourier-momentumokhoz geometriailag hasonlóak, azonban értékeik a Diszkrét módszerrel történő számolás esetén aszimptotikusan alacsonyabb komplexitással is megkaphatóak, gyors Fourier-transzformáció (FFT) alkalmazásával.

Harmadik fő eredményként az Accurate és a Diszkrét módszerhez is kidolgoztunk egy új, a korábban használtaknál gyorsabb rekonstrukciós eljárást, mely minimális rekonstrukciós hibanövekedés mellett jelentősen csökkenti a futásidőt. Ezt a módszert még gyorsabbá tettük a Csebisev–Fourier típus esetében FFT használatával.

A Csebisev–Fourier momentumok megalkotásának másik motivációja egy közelmúltban publikált harmadik függvényrendszer, a Radial-Harmonic–Fourier szerinti momentumok előnytelen tulajdonságainak kimutatása állt: ezek a momentumok a Diszkrét módszerünkhöz hasonlóan számolhatók FFT segítségével, viszont a függvények origóbeli szingularitásai sejtésünk szerint rosszabb eredményeket okoznak a pontosság szempontjából a gyakorlati alkalmazásokban. Ezért eredményeinket erre a momentumtípusra is levezettük, hogy összehasonlításokat végezhessünk mindhárom momentumtípus között, mind az Accurate mind a Diszkrét számítási módszerek alkalmazása mellett.

Mindhárom momentumtípus számítási algoritmusait implementáltuk, és gyakorlati tesztek során összehasonlítottuk őket. Tapasztalataink alapján a számítási stratégiák közül a mi Diszkrét módszerünk adja a legpontosabb eredményt, és ezt a leggyorsabban teszi. A függvényrendszerek vonatkozásában a Csebisev–Fourier-momentumainkkal kapott rekonstrukció közel megegyező eredményt ad a Legendre–Fourier-momentumokat használó rekonstrukcióval, azonban nagyobb méretű bemenet esetén az FFT-n alapuló számolások jelentősen gyorsabbá teszik. Ugyanakkor a RadialHarmonic–Fourier-rendszerrel összehasonlítva a Csebisev–Fourier-momentumokat a számítási időeredmények hasonlóak, viszont pontosságban (különösen a gyakorlati alkalmazásokban kritikus alacsony indexű momentumok vonatkozásában) a mi függvényrendszerünk kimutathatóan jobban teljesített.

Definiáltunk a Csebisev–Fourier-momentummódszerhez eltolás, forgatás, skálázás és RST invariánsokat. Ezen invariánsok robusztusságát vizsgáltuk különböző transzformációkon átesett, zajosított képek esetén, majd a kapott eredményeket összehasonlítottuk a többi polár–Fourier-momentummódszer invariánsai által adott eredményekkel. Méréseink alapján nagyságrendileg azonos módon teljesítettek a különböző polár–Fourier-momentummódszerek, ezért érdemes vizsgálni a Csebisev–Fourier-invariánsok használhatóságát különböző alkalmazások esetén is.

Az invariánsok egy lehetséges alkalmazásaként objektumfelismerési algoritmust implementáltunk. A tesztelés során a cél egy transzformáción, zajhatáson átesett objektum eredeti (nem transzformált) képének azonosítása volt. Méréseink alapján a Csebisev–Fourier-momentummódszer hasonló eredményeket ad, mint a többi polár–Fourier-momentummódszer.

Egy további alkalmazásként szabad szemmel nem látható, rejtett vízjeleket helyeztünk el képeken, melyek ellenállnak a zajhatásokkal szemben, és az invarianciáknak hála a transzformációkkal szemben is. Vizsgáltuk a különböző polár–Fourier-momentummódszerek által generált vízjelek láthatóságát a vízjelezett képen, valamint a robusztusságokat különböző transzformációk, zajhatás esetén. Méréseink során azt tapasztaltuk, hogy a Csebisev–Fourier-momentummódszer hasonló eredményeket produkál a Legendre–Fourier-módszerhez.

Szempontok konkrét feladathoz illő módszer kiválasztásához

Ha csak alacsony rendű momentumok számolására van szükség, alacsony felbontású képeken, akkor érdemes olyan módszert választani, ami minél pontosabb eredményt ad, kevésbé foglalkozva a futásidővel. Erre a legalkalmasabb lehet az általunk kidolgozott Diszkrét módszer Legendre–Fourier- vagy Csebisev–Fourier-momentumtípusokkal, eredeti pixelpontokban történő rekonstrukcióval.

Amennyiben nagy felbontású képekkel dolgozunk, magasabb rendű momentumok mellett, akkor a legjobb eredményt szintén az általunk kidolgozott Diszkrét módszer adja Legendre–Fourier- vagy Csebisev–Fourier-momentumtípusokkal, az általunk kidolgozott interpolációs rekonstrukcióval. Még gyorsabb eredményt kaphatunk, hogyha Csebisev–Fourier esetben FFT-t használunk (ez nagy felbontású képek esetén mindig megéri, hisz nem jár pontosság csökkenéssel). RadialHarmonic–Fourier-momentumtípus is adhat gyors eredményt FFT-vel, de láthattuk, hogy hátrányos tulajdonságai miatt érdemesebb Csebisev–Fourier-momentumtípust használni.

Objektumfelismerés és vízjelezés céljából a módszerek nagyságrendileg hasonló eredményt adnak, de a RadialHarmonic–Fourier origóbeli szingularitása miatt inkább a másik két momentummódszer használatát javasoljuk.

Hivatkozások

[1] S. Belkasim, M. Shridhar, M. Ahmadi, Pattern recognition with moment invariants: A comparative study and new results, *Pattern Recognition*, 24(12) (1991), pp. 1117–1138.
`https://www.sciencedirect.com/science/article/pii/003132039190140Z`

[2] M. Busa, Kvaternió értékű polár–Fourier momentumok algoritmusai digitális képfeldolgozáshoz *TDK dolgozat*, Eötvös Loránd Tudományegyetem, 2022.

[3] B. Chen, H. Shu, H. Zhang, G. Chen, C. Toumoulin, J. Dillenseger, L. Luo, Quaternion zernike moments and their invariants for color image analysis and object recognition, *Signal Processing*, 92(2)

(2012), pp. 308–318, 2012.
https://www.sciencedirect.com/science/article/pii/S0165168411002520

[4] C. Gasquet, R. Ryan, P. Witomski, *Fourier Analysis and Applications: Filtering, Numerical Computation, Wavelets*, Ser. Texts in Applied Mathematics, Springer, New York, 2014.
https://books.google.hu/books?id=ZZkWswEACAAJ

[5] L.-Q. Guo M. Zhu, Quaternion Fourier–Mellin moments for color images, *Pattern Recogn.*, 44(2) (2011), pp. 187–195.
https://doi.org/10.1016/j.patcog.2010.08.017

[6] W. R. Hamilton, *Elements of Quaternions.* Longmans, Green, 1866. http://eudml.org/doc/203504

[7] K. Hosny, M. Darwish, Invariant color images representation using accurate quaternion Legendre–Fourier moments, *Pattern Analysis and Applications,* 22 (2019), pp. 1105–1122.

[8] Y. Liu, S. Zhang, G. Li, H. Wang, J. Yang, Accurate quaternion radial harmonic Fourier moments for color image reconstruction and object recognition, *Pattern Analysis and Applications,* 23 (2020).

[9] J.C. Mason, D.C. Handscomb, *Chebyshev Polynomials*, Chapman & Hall/CRC, 2002.

[10] Zs. Németh, M. Busa, Quaternion harmonic Chebyshev–Fourier moments and their computation for color image representation, analysis and applications, *Signal Processing*, beadva, 2024.

[11] S. Parah, J. Sheikh, F. Ahad, N. Loan, G. Bhat, Information hiding in medical images: a robust medical image watermarking system for e-healthcare, *Multimedia Tools and Applications*, 76 (2017)

[12] H. Shojanazeri, W. A. Wan Adnan, S. M. Syed Ahmad, S. Rahimipour, Authentication of images using Zernike moment watermarking, *Multimedia Tools and Application,* 76 (2017), pp. 577–606,
https://doi.org/10.1007/s11042-015-3018-2

[13] X. Yang Wang, W. Yi Li, H. Ying Yang, P. Wang, Y. Wei Li, Quaternion polar complex exponential transform for invariant color image description, *Applied Mathematics and Computation,* 256 (2015), pp. 951–967.
https://www.sciencedirect.com/science/article/pii/S0096300315001071

[14] Barycentric coordinate system
https://en.wikipedia.org/wiki/Barycentric_coordinate_system

[15] Bicubic interpolation
https://en.wikipedia.org/wiki/Bicubic_interpolation

[16] FFTW
https://fftw.org

Gyors, pontos – bizonyíthatóan helyes?

Kalandjaim valós számokkal és az agda2hs-sel[‡]

Csimma Viktor*

Eötvös József Collegium**

csimmaviktor03@gmail.com

Témavezető: Kaposi Ambrus
ELTE IK Programozási Nyelvek és Fordítóprogramok Tanszék

1. Bevezetés

Miért érdemes elolvasni ezt a cikket? Remélhetőleg érdekes lesz; igyekszem közben érdekeseket mesélni is. Röviden összefoglalva, miről volna szó: a cikk egy **Acorn** nevű, bizonyíthatóan helyes, mégis gyors, pontos valós aritmetikai Agda-könyvtárról szól. Szóval olvass tovább, ha:

- érdekel, hogyan lehet irracionális számokat pontosan reprezentálni számológépen;
- kíváncsi vagy, hogy lehet valami egyszerre gyors és számítógéppel bizonyítottan helyes;

[‡] A szerző 2023. decemberi Kari TDK Konferenciára benyújtott angol nyelvű dolgozatának könnyedebb hangvételű, magyar nyelvű változata.

* ELTE Informatikai Kar
** 2021–

- izgalmasnak találnád, ha lehetne bizonyításokból programokat kisajtolni;
- szeretnéd látni, hogy mi a csudára jó az Agda.

A projekt elméleti alapja Robbert Krebbers és Bas Spitters 2013-as cikke [10], illetve az ezen alapuló (Coqban írt) CoRN könyvtár [15]. Azonban több szempontból is új oldalról közelíti meg a célt. Szeretném az érdeklődést azáltal felkelteni, hogy *először* készítek el egy gyakorlatban is használható könyvtárat, és a bizonyítások nagy része csak ezután kerül majd kitöltésre. Az *agda2hs* [4] fordítóval való kompatibilitás nemcsak sokkal hatékonyabbá teszi a kódot, hanem letisztultabbá, érthetőbbé is (na jó, legalábbis a Haskell-részét). Mindemellett az Agda Haskellhez hasonló, világos szintaxisa közelebb hozhatja a bizonyítási asszisztensekkel eddig nem foglalkozó szakembereket, sőt talán a nyilvánosságot is a bizonyítottan helyes valós aritmetikához.

1.1. Történeti áttekintés

A Bishop-féle konstruktív analízis

Mi a konstruktív matematika? Röviden: olyan matematika, amiben csak konstruktív egzisztenciabizonyítások vannak.

Jó, ezzel nem kerültünk sokkal közelebb... mik a konstruktív egzisztenciabizonyítások? Akkor konstruktív egy létezésbizonyítás, ha konkrétan megmutatjuk, hogyan kell előállítani egy objektumot, amiről aztán bebizonyítjuk, hogy megfelel az állításunknak [17]. Tehát nem bizonyíthatunk létezést azzal, hogy feltesszük, hogy az illető objektum nem létezik, majd ebből levezetünk egy ellentmondást. (Mindamellett *nem létezést* lehet indirekten bizonyítani.)

Miért jó ez? Mert lehetőséget ad arra, hogy az eredményeinket gyakorlatban is kiszámolhassuk. Például ha konstruktívan bizonyítjuk egy sorozat konvergenciáját, azzal egyben algoritmust is adunk a határérték kiszámítására. Lehet, hogy ezt az algoritmust mi már nem látjuk át, de a felhasznált tételeink sora összességében mégis kiad egy eljárást. És ugyanez elképzelhető deriváltakkal, integrálokkal, inverzekkel.

A konstruktivizmusról a legélesebb vita a korai huszadik században zajlott; azonban a konstruktív analízis egy alapos, átfogó kidolgozására csak jóval később került sor. *Errett Bishop* amerikai matematikus 1967-

ben publikálta *Foundations of Constructive Analysis* című könyvét[1] [3]. Bishop a klasszikus függvénytan eredményeinek hatalmas részét kidolgozta konstruktív környezetben; sok tétel viszont nem volt eredeti formájában konstruktívan igazolható. Ebben az esetben sokszor két konstruktív verziója is született a tételnek: egy olyan, ami azonos feltételek mellett kevesebbet állít; illetve egy olyan, ami erősebb feltételek mellett azonosat állít [1]. (Az előbbi megközelítésre példa Rolle tétele: a konstruktív verzió nem állítja, hogy létezik olyan x, amire a derivált 0; csak azt, hogy *bármely* $\varepsilon > 0$-*ra* van olyan x, ahol $f'(x) \leq \varepsilon$.) Általában a nevezetes függvényekre úgyis tudunk viszonylag erős feltételeket bizonyítani, ezért az utóbbi változat szokott a hasznosabb lenni.

Bishop a számtípusok felépítésekor a pozitív egészekből indul ki: szerinte ezek „God-given" számok, amiknek aritmetikája az emberi intelligencia természetéből adódik. Feltételezi hát, hogy az olvasó már tisztában van ez alapján a racionális számok aritmetikájával is. A valós számokat ezután úgy definiálja, mint racionális számok egy (x_n) sorozatát, amire igaz, hogy

$$\forall m, n \in \mathbb{N}^+ : |x_m - x_n| \leq m^{-1} + n^{-1}.$$

Két valós szám pedig definíció szerint akkor egyenlő, ha

$$\forall n \in \mathbb{N}^+ : |x_n - y_n| \leq 2n^{-1}.$$

Később a szerző bizonyítja, hogy egy ilyen sorozat n-edik tagja n^{-1} pontossággal közelíti az ábrázolt valós számot. Tehát gyakorlatilag a sorozat minden tagja egy-egy közelítés, egyre nagyobb pontossággal.

Ez a megközelítés remekül működött papíron; a matematikát lehetett erre építeni – a gépeket azonban, mint ki fog derülni, nem. Vizsgáljuk meg két valós szám összegének definícióját:

$$x + y := (x_{2n} + y_{2n})_{n=1}^{\infty}.$$

Látszik, hogy a $2n$-edik közelítést kell mindkettőből venni, hogy n^{-1}-es pontosságot kapjunk. De van ennél sokkal nagyobb probléma is. Szükségünk lesz egy racionális összeadásra, amihez általános esetben össze kell szorozni a két nevezőt; a tetszőleges hosszú egész számok pedig,

[1] A *Könyv* azóta már elérhető az Informatikai Kar könyvtárában az ÚNKP jóvoltából; pontosabban most nem, mert kivettem. De december végén visszaviszem.

ha nagyra nőnek, nagyon nem szeretik, ha összeszorozzuk őket. Az idő- és a memóriaigény is hatalmasra nő, és a számítás használhatatlanul lelassul.

Persze rá kell mutatni arra, hogy Bishop idejében még közel sem voltak olyan elterjedtek a számítógépek, mint ma. Nagyok, lassúak és drágák voltak, és csak nagy szervezetek engedhették meg őket maguknak [1]. A szerző maga is megemlíti, hogy az eleganciára nagyobb hangsúlyt fektetett, mint a hatékonyságra:

> *„The transcendence of mathematics demands that it should not be confined to computations that I can perform, or you can perform, or 100 men working 100 years with 100 digital computers can perform. Any computation that can be performed by a finite intelligence [...] is permissible. This does not mean that no value is to be placed on the efficiency of a computation. [...] Mathematics should and must concern itself with efficiency, perhaps to the detriment of elegance, but these matters will come to the fore only when realism has begun to prevail. Until then our first concern will be to put as much mathematics as possible on a realistic basis without close attention to questions of efficiency."* [3]

Mégis, már ő is gondolt arra, milyen hasznos lenne az elméletet számítógépekre átültetni. Hadd idézzem ezt a gondolatát is, a B jelű függelékből:

> *„It is clear that many of the results in this book could be programmed for a computer, by some such procedure as decribed above. In particular, it is likely that most of the results of Chaps. 2, 4, 5, 9, 10, and 11 could be presented as computer programs. [...] As written, this book is person-oriented rather than computer-oriented. It would be of great interest to have a computer-oriented version."* [3]

Tehát az ötlet nem új, csak akkor még nem volt megvalósítható. Azóta viszont az elmélet és a hardverek is sokat változtak.

Fejlemények Bishop után

Csak a számunkra legfontosabb lépcsőkről beszélnék. (Az összes lent felsorolt ember a Nijmegeni Egyetemen dolgozott, Hollandiában.)

- Az első az FTA nevű projekt. Ennek eredetileg az volt a célja, hogy az algebra alaptételét *(Fundamental Theorem of Algebra)* Coqban formalizálja. A későbbi `CReals`[2] nevű típus is itt jelent meg először. A megközelítés az volt, hogy axiomatikusan definiálták, majd bizonyították, hogy a racionális Cauchy-sorozatokkal felépített valós számok ekvivalensek ezzel. Az eredményeket 2000-ben publikálták. [5, 8]

- A C-CoRN projekt eredeti, Luis Cruz-Filipe által 2004-ben jellemzett [5] verziója célozta meg először azt, hogy a konstruktív analízis egy formalizálásából futtatható programokat készítsen. Ez már a Bishop-féle definíciót használja. Cruz-Filipe futtatott kísérleteket, de szegény 30 óra alatt tudta kiszámolni a $\sqrt{2}$-nek egy 1/12 pontosságú közelítését. Mindamellett reménykedett abban, hogy *„a nem olyan távoli jövőben egyszer majd lehet értelmes időben futtatni az előállított programot”.* [5]

- Az áttörés Russell O'Connor monadikus valós számaival következett be, 2005-ben [14]. Gyakorlatilag ő tette futtathatóvá az elméletet. Számtalan új koncepciót vezetett be:

  – Az értékkészlet már nem kifejezetten a racionális számok halmaza, hanem metrikus és ún. *prelength* terek, amik általánosabb fogalmak. Hogy megkerülje a valós számok használatát, metrika helyett egy ekvivalenciarelációt használ:

  $$B : \mathbb{Q}^+ \Rightarrow X \Rightarrow X \Rightarrow *.$$

  Ha $B_\varepsilon(x, y)$ igaz, az felel meg annak, hogy x az y ε sugarú környezetén (*ball*, innen a név) belül van. (A konkrét implementáció mindamellett továbbra is egész számpárokkal megadott racionális számokra épít, de az elmélet később fontos lesz majd.)

  – Reguláris *sorozatok* helyett már reguláris *függvényekkel* találkozunk. A valós számok ezúttal olyan $x : \mathbb{Q}^+ \Rightarrow \mathbb{Q}$ függvények, amelyekre

  $$\forall \varepsilon_1, \varepsilon_2 : B_{\varepsilon_1+\varepsilon_2}(x(\varepsilon_1), x(\varepsilon_2)).$$

[2] Magyar fordításban: müzlik. Bár valószínűleg inkább Cauchyhoz van köze.

Így már nemcsak $1/n$ alakú, hanem akármilyen pozitív racionális szám lehet a pontosság. Ezzel sok feleslegesen pontos becslést kerülünk el. (ε_1, ε_2 helyére m^{-1}-et és n^{-1}-et helyettesítve nagyjából visszakapjuk a Bishop-féle definíciót.)

– Amiről a nevét kapta a modell, az a $\mathfrak{C}$ monád, ami egy metrikus tér teljessé tételét jelenti (gyakorlatilag $\mathfrak{C}(X)$ az X-be képező reguláris függvények halmaza). A monád jelleg azért jó, mert így a racionálisokon értelmezett *egyenletesen folytonos* függvényeket felemelhetjük a valósok közé. Ez sokkal egyszerűbbé teszi a munkát, és új lehetőségeket is ad (esetszétválaszthatunk például $\leq, \geq$ stb. alapján; ami a valósokon nem lenne lehetséges, mert ott nem eldönthetőek ezek a relációk). Talán a legfontosabb koncepció a map függvény (a típusszignatúrája $(X \to Y) \Rightarrow (\mathfrak{C}(X) \to \mathfrak{C}(Y))^3$), ami egy *egyenletesen folytonos* (fontos!) racionális függvényt valósra emel:

$$\mathsf{map}(f) := \lambda x.f \circ x \circ \mu_f.$$

Itt μ_f az a függvény (*modulus of continuity*, folytonossági modulus), ami az ε-hoz a megfelelő δ-t rendeli a folytonosságnál. Ez garantálja, hogy két δ-nál nem messzebbi értékre a függvényértékek se legyenek ε-nál messzebbiek. Ezért is csak egyenletesen folytonos függvényekre működik.

– A compress függvény a függvény által visszaadott racionális értékeket egyszerűsíti, miközben az általa reprezentált valós szám nem változik. Russell így definiálja:

$$\mathsf{compress}(x) := \lambda\varepsilon.\mathsf{approx}(x(\varepsilon/2), \varepsilon/2),$$

ahol az $\mathsf{approx}(q, \varepsilon)$ a legegyszerűbb relatív prímes alakú racionális szám a $[q-\varepsilon, q+\varepsilon]$ intervallumban. (Erre a Haskellnek van egy beépített függvénye a `Data.Ratio` modulban.) A compress alapvetően olyan racionális közelítéseket keres a visszatérési értékekhez, amik egyszerűbben ábrázolhatók (kisebb a számláló és a nevező), de „elég jók” ahhoz, hogy

[3] O'Connor az egyenletesen folytonos függvényeket sima szárú nyíllal ($\to$) különbözteti meg, míg a dupla szárú nyíllal jelöl ($\Rightarrow$) bármilyen tetszőleges függvényt.

reguláris maradjon a függvény. Ez nagyban gyorsítja a számításokat; tulajdonképpen szinte minden függvény bemenetére érdemes odavágni egy compress-t.

2005 októberében O'Connor egy versenyen is indult egy Haskell-implementációval, ami több ezer tizedesjegyet tudott perceken belül kiszámolni sok problémára. Mindamellett a dobogósoktól jócskán lemaradt, de azok számítógéppel nem validált, C/C++-alapú megoldások voltak.

- Végül ami a címben is szerepelt: Robbert Krebbers és Bas Spitters 2011-ben benyújtott cikke O'Connor munkáját javította; állítólag „akár 100-szoros" javulásokkal az alapvető műveletekben [10].

  – A könyvtáruk típusosztályokon alapul (ez akkoriban a Coq-ban egy viszonylag új koncepció volt). Különböző (az FTA-ban is használt) algebrai fogalmaknak, de akár még operátoroknak is típusosztályokat hoztak létre. Ezáltal nem kellett például külön összeadás operátort írni különböző típusokra, és a függvények is általánosíthatóak lettek.

  – Pontos racionális számok helyett a reguláris függvények úgynevezett *közelítő racionálisokat* (*approximate rationals*) adnak vissza. Ez szintén egy típusosztály, és az itteni számoktól csak egy olyan osztási operátort várunk el, ami egy megközelítőleges pontossággal végzi el a műveletet. A konkrét implementáció jellemzően diadikus törtekkel dolgozik (gyakorlatilag kettes számrendszerbeli normálalakokkal). Ez hatalmas mértékben javít a hatékonyságon.

  – A compress függvényt is hatékonyabban meg lehet így írni: általánosan az approx-nak elég egy tetszőleges közelítést visszaadnia egy adott pontossággal, az eredmény pedig változhat implementációnként. Konkrétan a diadikus törteknél egyszerűen a mantissza megfelelő shiftelésével működik, amivel egy előre kiszámítható mértékben vesztünk a pontosságból, miközben a szám kezelhetőbbé válik.

Ma a C-CoRN projekt Krebbers és Spitters elméleti munkáján alapszik [15], ahogy a most bemutatandó könyvtár is.

1.2. Motiváció

A C-CoRN ma egy hatalmas könyvtár; egészen a Newton–Leibniz-tételig kiterjed, és futtatható is. Kérdezhetnénk tehát: minek még egy formalizáció (ami ráadásul jelentősen leegyszerűsített)?

- 2011-ben írták meg a hollandok a cikküket. 2020-ban viszont elkezdődött az agda2hs fejlesztése (szóval ez egy elég új dolog). Ennek az a lényege, hogy ha bizonyos szabályokkal írjuk meg az Agda-kódot (pontosabban erase-eljük azt, amit a Haskell nem támogat), akkor lehet szép Haskellre fordítani a kódunkat. És ez nemcsak szép lesz, hanem gyors is; de még azt is tudni fogjuk, mi történik a háttérben; debuggolhatunk a GHC profilozóival, ilyesmi. Nem tudok róla, hogy pontos valós aritmetikában az agda2hs-t használták volna már.

- A Coqkal szemben az Agda elsősorban programok és nem bizonyítások írására van tervezve. A mi célunk a gyakorlati használhatóság kidomborítása, amihez ez jobban illik. Ahogy később ki fogom fejteni, először egy használható, futtatható és szerethető könyvtárat szeretnék építeni, ami remélhetőleg felkelti a közösség érdeklődését annyira, hogy elkezdjenek többen foglalkozni a témával. És elég lesz ezután kitölteni a bizonyításokat (ami viszont Agdában jóval nehezebb feladat, mint Coqban).

- Krebbers és Spitters két implementációt írt: egy számítógéppel nem validáltat Haskellben és egy számítógéppel validáltat Coqban. Méréseik szerint a $\sqrt{2}$ értékének 3000 tizedesjegy pontos közelítését 0.2 másodpercbe telt Haskellben kiszámolni, míg 7.4 másodpercbe Coqban. Írnak arról, hogy valaki felvetette, hogy a Coq állítólag nem olyan szépen ábrázolja az egész számokat [10]. Na de az Agda és a mögötte levő Haskell-Integerek majd segítenek.

Fontos megemlíteni, hogy bár nagyra becsülöm Bishopot és a munkáját, elsősorban baráti jellegű a vonzalom. Jelen munka célja nem a konstruktív matematika népszerűsítése, hanem egy számítógéppel validált, megbízható és mégis gyors valós könyvtár felépítése, amire aztán egy kiszámolható analízist lehet építeni. Örülnék neki, ha újra elindulna egy diskurzus a matematika filozófiai alapjairól, de a konstruktivizmust inkább eszköznek tekintem, mint célnak. Részben azért is emelem

ki ezt, mert Bishop úgy érezte, nem veszik komolyan a munkáját [1]; ebben pedig szerepet játszhatott, hogy az öncélúnak tűnt, mintha csak hibbant matematikusok öncélú játéka volna. Szeretném megmutatni, hogy ez nem így volt, és most sem így van.

2. Korábbi munkám

2.1. A kezdetek

Most kisebbet ugrunk vissza az időben, 2022 elejéig. Kaposi Ambrus akkor tartott nekem először órát (a *Modern elméletek az informatikában I.* című tárgy utolsó iterációját, ami gyakorlatilag a mostani *Típuselmélet* kötvál anyagának nagy részét fedte). Nem tudtam azonnal, hogy ezzel fogok kutatni; bár már akkor is izgalmasnak tűnt. Aztán félév közepén említette egyszer Bishopot; valamikor pedig azt, hogy fejlesztenek az egyetemen egy új nyelvet, ami már tudna külön megadott egyenlőségi szabályokat is [9]. Ez izgalmasnak tűnt, és gondoltam, hogy abban elkezdhetném a Bishop-féle analízis formalizálását; de Ambrus azt mondta, még nem tart ott a nyelv.

Úgyhogy úgy döntöttünk, Agdában fogok dolgozni. Már beadtam az ÚNKP-t, amikor megtaláltam egy Zachary Murray nevű kanadai informatikushallgató alig két hónapos szakdolgozatát [12], ami pontosan erről szólt. Hogy ne haljon el a dolog, és ne is párhuzamosan dolgozzunk, megkérdeztem őt, hogy használhatom-e az eddigi munkáját. Használhattam.

2.2. Murray Bishop-formalizációjának folytatása

Murray a könyv második fejezetét kezdte el formalizálni; a sorozatok határértékére vonatkozó tételekig jutott (sőt még a szuprémumokkal kapcsolatban is írt bizonyításokat). A munkám során:

- bizonyítottam a Leibniz-típusú sorok konvergenciáját (és ezzel definiáltam a π-t);
- megvizsgáltam a véges hosszú sorozatok különböző reprezentációit;

- formalizáltam a folytonosság Bishop-féle definícióját (ami gyakorlatilag az egyenletes folytonosság) és a Weierstrass-tétel[4] egy konstruktív változatát.

Van egy repó [13], ahova ezeket a dolgokat feltöltöttem; ennek a *main* branchéről van szó. Erről olyan sokat nem szeretnék beszélni, mert nem tartozik igazán a témához, de ez volt az első lépés, amivel tapasztalatot szereztem.

2.3. Kísérletek a futtathatóvá tételre

Ezzel párhuzamosan szerettem volna, hogy a könyvtárban levő bizonyításokkal, amik ugye algoritmusok is, ténylegesen számolni lehessen (akkor még nem sejtettem a fent leírt problémákat). A nagy cél az e kiszámolása volt, amit ha megpróbáltam kiértékelni, nagy és hosszú számolásba kezdett az Agda, és soha nem ért a végére. Próbáltam az erasure-ökkel[5] megjelölni a tényleges számoláshoz nem szükséges részeket, hogy azok futásidőben eltűnjenek, de ez sem segített.

Aztán Kovács András egyszer felhívta a figyelmemet az agda2hs-re, amiről fent írtam, hogy milyen szép és jó. Úgyhogy a szakmai gyakom nagyrészt egy agda2hs-kompatibilis átirat elkészítésével telt, amit kifejezetten élveztem. Közben belekóstoltam magának a fordítónak a fejlesztésébe is: hibákat javítottam, és új funkciókat is írtam, amikkel kényelmesebben meg lehetett erőszakolni a nem agda2hs-kompatibilis standard könyvtárat. Valószínűleg az agda2hs-es munkám hasznosabb volt, mint maga az átirat; akit érdekel, megnézheti a pull requesteket [4].

Hát átírtam, de az e ezután sem futott le. Viszont így már használhattam a GHC profilozó eszközeit [16], amik azt mondták, hogy az összeadás művelet fut nagyon sokszor. Ez pszeudonyelven leírva valahogy így néz ki:

```
(+) :: Real -> Real -> Real
x + y = \ n -> x (2 * n) + y (2 * n)
```

[4] Angolul *extreme value theorem*.

[5] Ez az Agdában egy viszonylag új funkció. A lényege, hogy megjelölhetünk paramétereket vagy definíciókat, amiknek kizárólag a bizonyításokban van szerepük, de a számolásban nem. Ekkor a típusellenőrző kikényszeríti, hogy ne használhassuk az értéket; cserébe a futó kódba már nem kerül bele, amit megjelöltünk.

Ártatlannak tűnik, ugye? De itt van a fenti probléma azzal, hogy a nagy Integerek összeszorzása időigényes. Márpedig egy racionális összeadáshoz ez kell. Szóval ezen a ponton kiderült, hogy szegény Bishop-féle valósok ebben a formában sosem fognak futni.

Úgyhogy utánanéztem, van-e megoldás a problémára (Ambrus javasolta, hogy nézzek meg újabb reprezentációkat). És ekkor találtam meg O'Connort, Krebberst és Spitterst. Elhatároztam, hogy az ő munkájuknak egy agda2hs-kompatibilis átiratán fogok dolgozni, ami egyszerre gyors és számítógéppel validálható.

2.4. Tanulság

Igazából egy évig olyat csináltam, aminek aztán közvetlen haszna nem lett. De olyan szempontból volt értelme, hogy ezalatt tudtam az Agda, az agda2hs és más hasznos dolgok használatát igazán megtanulni. (Még az Emacsre is rászoktam.) Mindamellett ha van egy jó ötletetek, mindenképpen nézzétek meg, mi a legjobb jelenlegi megoldás, mielőtt továbbindultok.[6]

3. A könyvtár

A kód egy egy új repón [6] érhető el.

3.1. Dizájn

A valós számok ábrázolása

Itt nem találtam fel a spanyolviaszt; Krebbers és Spitters koncepcióját követtem [10].

- Definiálunk egy `AppRationals` nevű típusosztályt. Itt a szokásos racionális számokhoz képest csak egy megközelítőleges osztási operátort várunk el, egy általunk választott 2-hatvánnyi pontossággal. Illetve egy az `approx`-nak megfelelő függvényt is előírunk, amivel lehet egy szebb számmal közelíteni egy AppRationalt, szintén általunk választott pontossággal.

[6] De azért ne dobjátok el a saját gondolatotokat sem. Van egy nagyon kis esély rá, hogy a tietek jobb.

- Ezután implementáljuk az O'Connor-féle kiteljesítési monádot, és a valós számokat tetszőleges `AppRationals`-beli osztály kiteljesítéseként definiáljuk.

- A `RealTheory.Reals` fájlban szerepelnek a valós operátorok, amiket legtöbbször a monádos bind-dal és hasonló függvényekkel írunk meg.

- Aztán jönnek a nevezetes függvények.

  – Ezeket általában hatványsorokkal definiáljuk. De hatványsorokat csak akkor tudunk egyszerűen használni, ha azok Leibniz-típusú sorok[7], és nullához tartanak a tagjai (különben nagyon nehéz lenne belátni a konvergenciájukat). Akkor viszont van egy tételünk [14], ami szerint egy ε-pontos becsléshez elég addig összeadogatni a tagokat, amíg azok nagyobbak ε-nál.

  – Kiterjesztjük őket az összes racionális számra; ezt ilyesmi átírási formulákkal csináljuk:

$$e^x = \frac{1}{e^{-x}}, \qquad e^x = e^{\frac{x}{2}} \cdot e^{\frac{x}{2}}.$$

  És a szinuszra:

$$\sin x = 3 \cdot \sin \tfrac{x}{3} - 4 \cdot (\sin \tfrac{x}{3})^3$$

  Ezt addig csináljuk, amíg *jó kicsik* nem lesznek az argumentumok.[8]

  – Bebizonyítjuk róluk, hogy egyenletesen folytonosak. Ez az exponenciális függvényre például nem teljesen igaz; ilyen esetekben azt csináljuk, hogy keresünk egy intervallumot, ami magától a paramétertől függ, és erre látjuk be az egyenletes folytonosságot. Végül így már `bindC`-vel felemelhetjük őket valós-valós függvényekké.

[7] Vagyis a tagok váltakozó előjelűek, és az abszolút értékük monoton csökken.

[8] Valójában sokkal kisebbre zsugorítjuk őket, mint amennyire az elmélet miatt kéne. Krebbers és Spitters nagyobb precizitású számításokhoz 2^{-75}-es nagyságrendet ajánlanak.

- A konkrét `AppRationals`-reprezentáció, amit használni fogunk, a diadikus törtek lesznek (`Implementations.Dyadic`).

További részletek a hivatkozott cikkben [10] és a kódban találhatóak. A matematikai elmélethez valójában nem igazán tettem hozzá; csak használtam, ami már megvolt.

Típusosztályok

A két holland egyik fő újítása a típusosztályok használata volt. Úgyhogy én is ezekkel dolgoztam (tényleg egész kényelmesek![9]), de néhol eltértem az eredeti struktúrától.

- A csoportokat, félcsoportokat kihagytam, mert Agdában nehéz lett volna őket a félgyűrű definíciójához felhasználni (valahogy a `_+_`-ra, `_*_`-ra kellett volna rakni őket). Ezeket a szabályokat belesűrítettem a `SemiRing` osztályba.
- Eredetileg az operátorokhoz volt külön típusosztály, ami csak az operátort definiálta, a tulajdonságai nélkül (pl. `Plus`, `Zero`). Én ezeket kihagytam a legtöbbször, és eleve hozzáadtam a tulajdonságokat is. Így rendezettebb lett a kód. (Néha azért szükség volt erre, mikor más típusosztályok más tulajdonságokat írtak le. Ez leginkább a rendezési relációknál jött elő.)
- A CoRN-tól eltérően én először szeretnék egy gyors és praktikus könyvtárat készíteni, és csak aztán betölteni a bonyolultabb bizonyításokat *(erről hamarosan írok bővebben)*. Ezért legtöbbször csak azt formalizáltam, ami kellett; nem egy egész, szép algebrai hierarchiát akartam felépíteni.

Függőségek

Ahogy korábban írtam, az Agda standard libraryt [7] eléggé meg kellett erőszakolni, hogy agda2hs-sel használni lehessen. Én ezt el akartam most kerülni, és ezért az Agda-oldalon kizárólag az agda2hs-be épített könyvtárat használom (ami jórészt Haskell-koncepciók megfelelőit írja le). Persze így pár struktúrát és bizonyítást újra kellett írni; ehhez sokszor beleolvastam az agda-stdlib kódjába.

[9] Leszámítva, amikor összeakadnak, mert kettő is illeszkedik. De szerencsére ilyen ritkán van.

Haskell-oldalon beépített, hatékony függvényeket is használok néha kézzel írtak helyett. (Pl. van egy beépített `shift` függvény a `Data.Bits`-ben, de egészlogaritmust is találtam.[10]) És mivel újradefiniálom az összeadást, szorzást, ezért a Prelude-öt többnyire csak kvalifikálva lehet importálni (példa gyanánt `Prelude.(+)` lesz a beépített összeadás).

Először fusson...

Erről már korábban is írtam, de itt részletesebben beszélek róla. A cél egy olyan könyvtár használata, ami gyors *és* számítógéppel validálható; elsősorban viszont a gyorsaságra és gyakorlati használhatóságra helyezem a hangsúlyt. Emiatt használom olyan gyakran a `cheat` posztulátumot (amivel el lehet csalni bizonyításokat), `FOREIGN` pragmákat és hasonló gusztustalan dolgokat. Szerintem nagyobb lendületet tud nyerni a projekt, ha először az látszik róla, hogy hasznos; ezután akár már be lehetne vonni új résztvevőket a bizonyítások megírásába.[11] Persze ennek hátrányai is vannak: így ugyanúgy kell néha debuggolni a kódot, mintha Haskellben lennénk; és ha kiderül, hogy valamit, amire támaszkodtam, nem lehet bebizonyítani, egy szignifikáns részt újra kellhet írni. De még ha csak egy kis részét is sikerül bizonyítani az eredményeknek (akár csak az analízist, ami később jönni fog), már előrébb vagyunk a legtöbb programozási nyelv matematikai könyvtáraihoz képest.

Kompatibilitás az agda2hs-sel

Ez lenne talán, ami megkülönböztetné a projektet másoktól. Néha bonyolultabbá is teszi a munkát, mert minden fordítandó kódra meg kell gondolni, hogy Haskellben is értelmes-e: megfelelően kell elnevezni a függvényeket; erase-elni kell a függő típusokat; nem illeszthetünk mintát természetes számokra (szegény `suc` nem működik a bal oldalon).

Emiatt nagyobb döntéseket is kellett hozni. Például itt a Murray-könyvtár átírásával ellentétben a szigorú egyenlőtlenség erased lesz. Ebben viszont volt eredetileg egy létezésbizonyítás, és a benne lévő

[10] Utóbbihoz a Haskell sötét, kettőskeresztes oldalához kellett nyúlnom. Nem szeretném itt részletezni, mert lehet, hogy ártatlan elsőévesek is olvassák ezt a cikket; de a bátrak vegyék fel a *Funkcionális nyelvek* kötvált.

[11] Akár téged is; ha érdekel a dolog, keress meg! De nem véletlenül nem ezzel a munkával kezdtem...

számra (a *tanúra*) szükségünk lesz később; de ezt csak akkor számoljuk ki, amikor már használni kell (például a reciproknál). Ha nem eraseelnénk a `_<_` típust, le kéne írni az `Lt` osztályban, mi a tartalmának a típusa; de ezt a Haskell jelenleg nem támogatja. Ezért ahogy Krebbers és Spitters is teszi, a tanút utólag keressük meg. (Részletekért lásd a `RealTheory.Reals` fájlt.)

Van olyan is, hogy elvben értelmes lenne a kód Haskellben, de az mégsem fordul le. (A legprominensebb példa a `suc`-os mintaillesztés természetes számokra; ezt elég bonyolult lenne értelmes Haskell-kódra átírni, még ha elvben lehetséges is.) Más esetekben lefordulna, de nem lenne kellően hatékony. Ezért ilyenkor csúnya `FOREIGN`-pragmákkal megkerülöm a fordítót, és kézzel írom meg a Haskell-kódot, követve a fenti *„először fusson”*-elvet. Persze ezeket a lyukakat egyszer majd be kell tölteni.

A Prelude-del kell még sokat bajlódni, hogy ne legyen névütközés a beépített függvényekkel. Itt egy `import qualified Prelude` segít. De főleg a `FOREIGN` pragmák miatt vannak olyan importok, amik nem kerülnek be, pedig kellenének; akkor azokat is kézzel kell megírni.

3.2. Felépítés

Az alábbiakban röviden összefoglalom az egyes modulok szerepét.

- **Tools.** Néhány olyan eszköz, ami egyik lenti kategóriába sem illik, de fontos.
- **Algebra.** Algebrai struktúrák és operátoraik, mint a `SemiRing` (a `_+_` és `_*_` operátorokkal) vagy a `Field` a `recip` függvénnyel. Itt vannak a metrikus és *prelength* terek típusosztályai is.
- **Operations.** Operátorok, amik nem egy adott algebrai struktúrához tartoznak, illetve a tulajdonságaik; egy-egy típusosztályba foglalva.
- **Implementations.** Különböző számtípusok konkrét implementációi (az elején gyakran az `Agda.Builtin`-ből importálva), és a szükséges típusosztályok példányosítása hozzájuk.
- **RealTheory.** Olyan eszközök, amelyek már közvetlenül a valós számok implementációjához kapcsolódnak, mint például az

`AppRationals` osztály és a kiteljesítési monád. A `Reals` fájl is itt található, amiben a típusosztályokat példányosítjuk az absztrakt valósokra.

- **Function.** Nevezetes irracionális értékű függvények. Egyelőre egészen pontosan az exponenciális függvényről, a szinuszról és a négyzetgyökfüggvényről van szó. Itt van a „szent” e és π is.
- **HaskellInstances.** Néhány példányosítás haskelles típusosztályokhoz (`Num`, `Fractional`, `Floating`). Ezek igazából elsősorban a sebességtesztekhez kellenek, magában a könyvtárban nem használom őket.

Az egyes fájlok célját jellemzően kommentben a tetejükre írtam.

3.3. Jelenlegi állapot

Az agda2hs `57811ed` commitja Ubuntu 20.04-en sikeresen fordítja az Acorn `c06c7de` commitjában leírt verzióját. A generált Haskell-kódot pedig a GHC 9.2.8 módosítások nélkül elfogadja.

Jelenleg minden, a testeken értelmezett művelet működik, és van pár irracionális értékű függvény (exp, sin, négyzetgyök, limitált értelmezési tartományon az arctg), illetve π is.

3.4. Sebességtesztek

Megtaláltam Krebbers kifejezetten Haskellben írt implementációját [11], amiben volt egy mérőprogram is (ezt használta a cikkben). Mivel az én kódom is Haskellre fordul, gondoltam, hogy akár rögtön a hatékonynak szánt változattal versenyeztethetem.

A letöltött kód régi. Illetve 2011-es, de az már réginek számít. Szóval kicsit kellett benne turkálni, hogy működjön mostani GHC-vel. És a számítógépek is fejlődtek azóta; úgyhogy fele-harmada volt az időigény annak, mint amit ők több mint tíz éve leírtak.

Ha már itt tartunk, a konfiguráció:

- Típus: Asus UM431-DA[12]
- CPU: AMD Ryzen 7 3700U

[12] Ez egy jó laptop.

- Memória: 16 GiB
- GPU: Radeon RX Vega 10
- Háttértár: SK Hynix 256 GB NVMe-M.2 SSD

A `test.hs`-ben felsorolt eredeti tesztesetekből [11] csak azokat tartottam meg, amiknél minden szükséges függvényt leimplementáltam már (pl. logaritmusom még nincs). A futási időket másodpercekben adom meg.

Az eredmények:

	Kifejezés	Tizedesjegyek	Acorn	K&S
P01	$\sin(\sin(\sin 1))$	5 000	0.70	0.71
P02	$\sqrt{\pi}$	5 000	0.52	0.61
P03	$\sin e$	5 000	0.27	0.45
P04	$\exp(\pi \cdot \sqrt{163})$	5 000	1.02	0.53
P05	$\exp(\exp e)$	5.000	0.86	0.77
P07	$\exp 1000$	20.000	0.22	0.27
C02	$\sqrt{\frac{\exp 1}{\pi}}$	10 000	2.77	3.11
C03	$\sin((1+\exp 1)^3)$	5.000	0.25	0.43
C04	$\exp(\pi \cdot \sqrt{2011})$	5.000	1.16	0.58
C05	$\exp(\exp(\sqrt{\exp 1}))$	5.000	1.42	0.85
C07	π^{1000}	20.000	2.11	3.74

Nagyon dönteni nem lehet; a könyvtáram néha gyorsabb, néha lassabb (még nem jöttem rá, mitől függ). Viszont már az egy izgalmas dolog, hogy egy kifejezetten Haskellben írt implementációval versenyezni tud. Belegondolva, hogy ez Agdából lett generálva emberi interakció nélkül, tényleg elképzelhető, hogy az agda2hs közelebb hozza majd a bizonyítható valós aritmetikát (és úgy általában az Agdát) a gyakorlati használathoz. Talán egyszer még Paksra is Agdában írnak programot.

Köszönetnyilvánítás

A kötetben talán inkább a cikk végére illik a köszönetnyilvánítás, de így is ez a legfontosabb része.

Először is szeretném kifejezni az elismerésemet az itt hivatkozott szerzők (különösen *Errett Bishop*, *Luís Cruz-Filipe*, *Russell O'Connor*, *Robbert Krebbers* és *Bas Spitters*) munkája iránt. Egészen lenyűgöző, mennyit fejlődött ez a terület még úgy is, hogy az informatikustársadalom sem tud sokat róla. Remélhetőleg így sikerül majd kicsit jobban megismertetni a világgal.

Különösen hálás vagyok a témavezetőmnek, *Kaposi Ambrusnak*, aki nemcsak megtanította nekem az Agdát, de folyamatosan támogatta és támogatja a munkámat szakmailag és emberileg is. Tanácsokat adott; mindamellett viszont szabadságot, hogy azon dolgozzak, ami érdekel, a szakmai gyakorlatom alatt is. (Mondása, hogy a legjobb ötletek akkor születnek, amikor vitatkozol a témavezetőddel, és csak azért is más irányba mész. Remélem, igaza van ebben.) Nem felejtem el megemlíteni az Informatikai Műhely vezetőit, *Kozsik Tamást* és *Lócsi Leventét*, akik tanácsokkal, támogatással és egy kiváló atmoszférával segítették a munkámat. (Tamás hívta meg Ambrust is; így ismertem őt meg. És rajta keresztül az egész témát is.)

A következő embert szintén nem sokan ismerik, pedig lehetne: ő *Jesper Cockx*, az agda2hs vezető fejlesztője. Amikor a szakmai gyakorlatom alatt elkezdtem bedolgozni a fordító fejlesztésébe, átnézte a munkámat; útmutatást adott a konvenciókkal és praktikákkal kapcsolatban; sőt pár hibajavítást, ami nekem szükséges volt, maga elvégzett. Az agda2hs nélkül nem hiszem, hogy hasznos lett volna ez a kutatás.

De azért tudtam elkezdeni rajta dolgozni, mert már volt tapasztalatom, amit egy kész konstruktív formalizáción szereztem meg. Szeretném megköszönni *Zachary Murray* kanadai informatikushallgatónak, amiért megengedte, hogy molesztáljam a 2022-es BSc-szakdolgozati munkáját. Enélkül valószínűleg még mindig a Cauchy-sorozatok határértékének hatékony kiszámításán töprengenék, eredménytelenül.

Ha már a Collegium: az itteni barátaim nélkül jó eséllyel nem készült volna el ez a munka. Segítettek, amikor szükségem volt rá, példaképként szolgáltak, és sokszor felvidítottak egy beszélgetéssel vagy akár egy mosollyal. Szerintem ez a legjobb hely, ahol lehetek.

Külön szeretném megköszönni *Orendács Petrának*, amiért vállalta, hogy megnézi az eredeti cikk angolját, ha esetleg továbbjutnék OTDK-ra vagy publikálnék valahol. Lehet, hogy sok fejfogástól fogja megmenteni az olvasókat.

Igazából a collegiumi és egyetemi tanítványaimat is megemlítem, mert megmutatták, hogy az Agda iránt nem csak én lelkesedem. Mikor gyakorlatot tartottam nekik, mindig motiváltabb lettem.

És természetesen nem fejezhetem be úgy, hogy nem köszönöm meg a családomnak a folyamatos támogatást: nemcsak a rendelt ebéddel segítettek (különben csak konzerven élnék), hanem tanácsokkal, türelemmel és őszinte szeretettel, jóban és rosszban is.

Hivatkozások

[1] Michael Beeson, *Foundations of Constructive Analysis – a New Foreword.* Ishi Press International, New York, 2012.

[2] Mark Bickford, *Formalizing Constructive Analysis in Nuprl,* Cornell University, 2016.
`https://www.nuprl.org/MathLibrary/ConstructiveAnalysis`
(2023.09.17.)

[3] Errett Bishop, *Foundations of Constructive Analysis.* McGraw-Hill, 1967.

[4] Jesper Cockx et al., agda2hs – Compiling Agda code to readable Haskell.
`https://github.com/agda/agda2hs`

[5] Luís Cruz-Filipe, Constructive Real Analysis: a Type-Theoretical Formalization and Applications, 2004.
`https://repository.ubn.ru.nl/handle/2066/19456`
(2023.09.30.)

[6] Viktor Csimma, Acorn – An agda2hs-compatible implementation of Krebbers–Spitters reals, focusing on usability.
`https://github.com/viktorcsimma/acorn`

[7] Nils Anders Danielsson et al., The Agda standard library.
`https://github.com/agda/agda-stdlib`

[8] Herman Geuvers, Milad Niqui, Constructive Reals in Coq: Axioms and Categoricity, *TYPES 2000,* Durham, UK.
`https://www.researchgate.net/publication/221186647_Con`

structive_Reals_in_Coq_Axioms_and_Categoricity (2023.09.30.)

[9] András Kovács, Efficient Evaluation for Cubical Type Theories. *2nd Conference on Homotopy Type Therory,* 2023. https://andraskovacs.github.io/pdfs/hott23prez.pdf (2023.09.18.)

[10] Robbert Krebbers, Bas Spitters, Type classes for efficient exact real arithmetic in Coq. *Logical Methods in Computer Science,* 9(1:01) (2013).

[11] Robbert Krebbers, Jelle Herold, fewdigits – forked from r6.ca/FewDigits/. https://github.com/robbertkrebbers/fewdigits/

[12] Zachary Murray, *Constructive Analysis in the Agda Proof Assistant,* Dalhousie University, 2022. https://arxiv.org/abs/2205.08354 (2023.09.18.)

[13] Zachary Murray, Viktor Csimma, bishop – A constructive analysis library written in Agda. https://github.com/viktorcsimma/bishop

[14] Russell O'Connor, A monadic, functional implementation of real numbers, *Mathematical Structures in Computer Science,* 2007. https://www.cambridge.org/core/journals/mathematical-structures-in-computer-science/article/monadic-functional-implementation-of-real-numbers/F5891E41D4F956571395A7A22B893AC9 (2023.09.30.)

[15] Bas Spitters et al., CoRN – Coq Repository at Nijmegen. https://github.com/coq-community/corn/tree/master.

[16] The GHC Team, The Glasgow Haskell Compiler, 9.6.3. GHC User's Guide, Chapter 8. Profiling. https://downloads.haskell.org/ghc/latest/docs/users_guide/profiling.html (2023.09.30.)

[17] Constructive Mathematics, *Stanford Encyclopedia of Philosophy.* https://plato.stanford.edu/entries/mathematics-constructive (2023.09.17.)

Budapest, London, Hong Kong
Két évtized az információ tengerén

Csipek-Czigola Gábor*

Eötvös József Collegium**

gabor.czigola@gmail.com

Karrierem zenitjén szeretnék számot vetni elméleti ismereteim tükrében a szerzett tapasztalataimmal, és feltenni azokat a kérdéseket, amelyek meghatározóak lehetnek a következő évtizedekben, hogy az elkövetkező generációknak talán támpontot nyújthasson az előttük álló kihívásokhoz.

1. Bevezető

„Akinek csak kalapácsa van, az hajlamos minden problémát szögnek látni."

(Abraham Harold Maslow)

Gyermekkorom óta foglalkozom hivatásszerűen információtechnológiával, szoftverfejlesztéssel és tanácsadással. Több földrészen és iparágban is megfordultam, féltucat országban-kultúrában kifejtve értékteremtő tevékenységet. Nem szeretném a kedves olvasót a teljesség igényével untatni, se felsorolni minden programkönyvtárat, keretrendszert, platformot, domaint, folyamatot, felhőt vagy verziót.

* ELTE Informatikai Kar, Budapest, programtervező matematikus MSc, 2011
VU Amsterdam, Computer Science MSc, 2009

** 2004–2010

Ott szeretném kezdeni, hogy most is magam előtt látom, amikor kisgyermekkoromban bekötötték hozzánk a telefont. Ez az akkor még teljesen analóg eszköz (impulzus-tárcsázó POTS) alig volt jó valamire, mert (segélyhívó és hivatalok számain kívül) egyetlen egy ismerősünk rendelkezett még csak telefonszámmal.

Szeretném elismeri, hogy részem volt abban, hogy a digitális távközlés és a számítástechnika elterjedésével párhuzamosan, a szoftverfejlesztés eljuttatott minket ide 2024-ben, hogy vigyázni kell az utcán, nehogy beleütközzünk valakibe, aki éppen önmagán kívüli szuszpenzióban, teljes figyelmével egy mesterséges intelligencia által generált, a fizikai valóságtól teljesen elvonatkoztatott, *„közösségi oldalon”* végzett interakciókba merül. Milyen félelmetes általános félelemmé vált, hogy nem találjuk a telefonunkat… (Már nem attól félünk, hogy követnek minket, hanem attól, hogy nem követnek minket…)

Több kortárs mű is feldolgozza a jelenkor félelmeit (Cyberpunk, Altered Carbon, Black Mirror stb.), de én nem a félelmekről szeretnék írni, hanem elméleti tanulmányaim során felmerülő kérdéseket átültetni a jelen és jövő kontextusába.

A jövőt megjósolni olyan mint Isten ellen fogadni: végtelenféleképpen tévedhetünk, miközben csak egyetlen egy jövő jön csak el igazán. Azzal a feltételezéssel, hogy a fizikai és lelki valóságnak alárendeltje minden, ami absztrakt vagy virtuális, kérdéseket szeretnék csupán feltenni, hogy segítsem a következő generációt improvizálni, alkalmazkodni, igazodni és felülemelkedni. *(Improvise, adapt, adjust, and overcome.)*

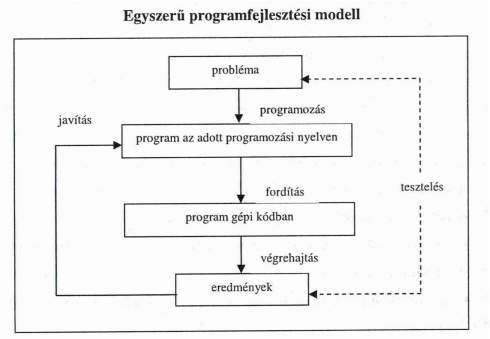
2. Kapcsolódó munka

„A tudattalan olyan, mint az iránytű, nem mondja meg, mit kell tennünk. Ha nem tudjuk olvasni az iránytűt, ha nem igazodunk el rajta, nem tud segíteni nekünk.”

(Carl Gustav Jung)

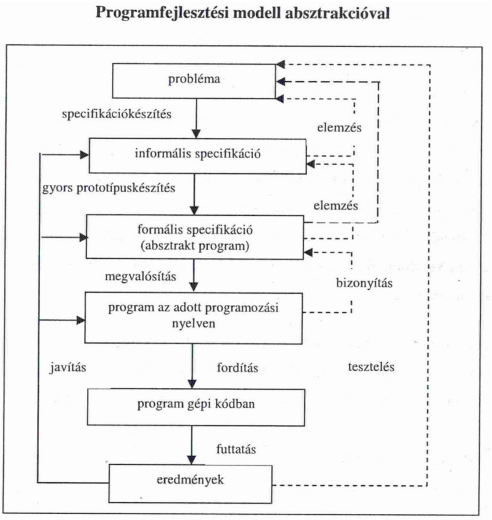
Kiindulási pontnak tekintsük *A szoftvertechnológia elméleti kérdései* c. könyvet [1], lásd 1. ábra.

Az egyszerű programfejlesztési modell továbbra is alkalmazható, azonban az egyre komplexebb rendszerekben egyre valószínűtlenebbül



1. ábra. Egyszerű programfejlesztési modell [1, 10. o.]

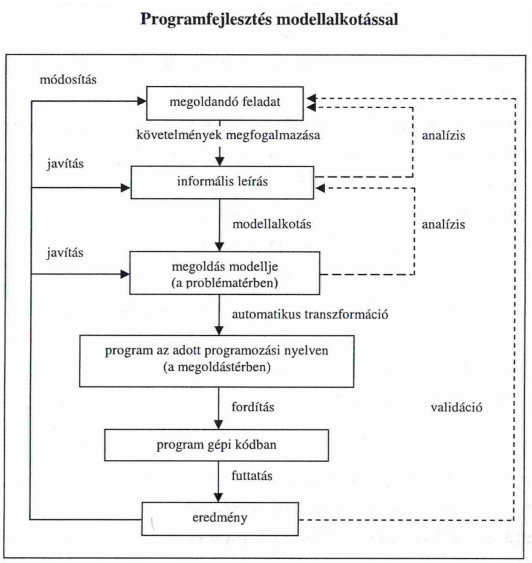
érhető el a megkívánt célállapot ezzel a módszerrel. Ez a módszer bár skálázható (több fejlesztő függetlenül több irányba indul el), a gyakorlatban nem ajánlom az alkalmazását, hiszen nincs felső korlátja, hogy mennyi ideg tart egy elfogadható eredményt megtalálni.



2. ábra. Programfejlesztési modell absztrakcióval [1, 11. o.]

Az absztrakciók jelentik a fordítóprogram megjelenése óta a megoldást, hogy a specifikációból egyenesen levezessük (és egyben validáljuk) a megoldást. A fordítás idejű kódellenőrzés, a linterek, a típusos OOP nyelvek, a unit-, e2e- és egyéb tesztek, a TDD és agilis módszertanok mind erre a koncepcióra épülnek.

Ezt a módszert univerzálisnak tekintjük, de nem szabad azt gondolni, hogy korlátlan: nem terjed ki az architekturális, nem-funkcionális követelményekre, nem sarkall új tervminták [2] felismerésére, nem ad megoldást szerkezeti, csapat vagy szervezet szintű problémákra.



3. ábra. Programfejlesztés modellalkotással [1, 14. o.]

Modellalkotással végzett programfejlesztést ajánlom a kedves Olvasó figyelmébe. Ez állandó és változó technológia mellett is lehetővé teszi az eredmény további megközelítését. A korábbi modellek minden lehetőségét is magában foglalva, és a teljesség igénye nélkül *(including, but not limited to)* olyan megoldásokat is lehetővé tesz mint: refaktoráció, újratervezés *(rearchitecture),* horizontális (pl. felhőbe) / vertikális (pl. platformváltás) / konceptuális (pl. interfészváltás) migráció, leegyszerűsítés, felbontás, kiváltás, leváltás, megnyitás stb.

Ezekre (és további) példákat találni a gyakorlatban házi feladat. Ezek alkalmazására jelenleg kevés olyan eszköz áll rendelkezésre, amely bevett, jól dokumentált és automatizált. Egyedül a refaktorációra tudok példát mondani: IDE környezetek képesek erre egy kódbázison belül. Ennél bonyolultabb, összetettebb javítási műveleteket talán a mesterséges intelligenciák fognak a kezünkbe adni. Nagy igény lenne pedig ilyen eszközökre, amik modellszintű beavatkozásra képesek.

3. A szoftverfejlesztés mennyiségi kérdései

„Erővel markunkba szorítva a homokot, az kicsúszik belőle;
de csak finoman tartva, megmarad."

(Princess Leia, Star Wars ep. IV.)

I. Amdahl törvénye [5] és a The Goal c. könyvben [3] leírtak alapján, létezik-e elméleti maximuma, hogy meddig skálázható a szoftverfejlesztés?

II. Melyik az a pont a mennyiségi szoftverfejlesztésben, ahol a minőségi vagy nem-funkcionális követelmények sérülnek? Ezt hogyan lehet mérhetővé tenni?

III. Egyre több tanulmány állítja [4], hogy a fejlesztések 45–80% nem kerül széleskörű alkalmazásra. Valóban pazarlás-e ez, és ha igen, hogyan lenne elkerülhető?

IV. A produktivitási folyamatos növekedésének követelménye meddig és hogyan tartható fenn? Elkerülhető-e akár az, hogy minőségi korlátokat eredményez, vagy, hogy fenékig tartó versennyé kerekedjen [7], felülírva önnön magával az eredeti célkitűzéseket?

4. A szoftverfejlesztés minőségi kérdései

„Quality is not an act, it is a habit."
(Aristoteles)

I. Hogyan ismerhetőek fel azok a körülmények, amelyekben a kívánt célállapot nem elérhető, és mit lehet, illetve érdemes tenni ilyen helyzetben?

II. Hogyan hidalhatóak át tartósan a nyelvi, kulturális és a távoli munkavégzésből adódó kommunikációs szakadékok? Milyen az optimális munkaszervezés annak a tükrében, hogy a szervezeti szerkezet és kommunikáció csatornák predesztinálják a szoftverarchitektúra minőségét [6]?

III. Életciklus-szingularitás tükrében (amikor a komponensek, programkönyvtárak életciklusa egyre rövidebb) hogyan tartható fenn komplex rendszerek fejlesztése?

IV. Ha a komplexitás soha nem csökken, hogyan tartható fenn komplex rendszerek életciklusa, hogyan tartható fenn az analízishez szükséges folyton növekvő kognitív szükséglet? Milyen megszorítással, illetve módszerrel tartható kordában a komplexitás?

5. A szoftverfejlesztés etikai kérdései

„A testnek érzetei vannak, a léleknek vágyai,
az értelemnek elvei."
(Marcus Aurelius Antoninus)

I. Milyen szerepe van szakembereknek egy nem fenntartható projekten? Mi az etikus viselkedés egy lehetetlen küldetés esetén?

II. Fenn tudja-e tartani magát e sorok olvasója a programfejlesztési modellen kívül? Ha fél évig nem lenne elérhető az Internet (vagy akár áram), mit tenne?

III. Milyen befolyással lehetünk a technológia további alakulására? Hogyan nevelhető fel az elkövetkező nemzedék, hogy a tudat maradjon az absztrakt állapottér ura, és ne a virtuális állapottér uralja a tudatot?

IV. Hogyan mutathatunk példát az elkövetkező generációknak készség és diszciplína terén? Mennyi az egészséges képernyőidő különböző életkorokban, és hogyan tudjuk megőrizni az emberséget bármilyen körülmények között?

6. Összefoglaló

„Navigare necesse est, vivere non est necesse."
(Gnaeus Pompeius Magnus)

E sorok írásakor a mesterséges intelligencia felemelkedését jövendölik egyre többen. Ez a kérdés még nem dőlt el, de személy szerint ezt is a modellalkotás egy speciális esetének tartom: új eszközök, új paradigmák, új módszerek állnak majd a rendelkezésünkre [8]. Úgy gondolom, semmilyen technológia nem újítja meg az alapkérdést, hogy emberként mit cselekszünk?

Biztatnám a kedves Olvasót, hogy merüljön el a fenti kérdésekben, és találja meg a saját válaszát a gyakorlatban.

Tartsuk minden absztrakciónál, modellalkotásnál szem előtt, hogy a végén mindig vissza kell térnünk az eredeti állapottérbe, a valóság talajára. Milyen hatással van a megoldásunk a tudatra, a testre, a lélekre, az emberségességre?

Hivatkozások

[1] Kozma László, Varga László, *A szoftvertechnológia elméleti kérdései,* második kiadás, ELTE Eötvös Kiadó, ISBN 9634636489, 2006.

[2] Christofer Alexander, *The Timeless Way of Building,* Oxford University Press, ISBN 9780195018240, 1975.

[3] Eliyahu M. Goldratt, Jeff Cox, *The Goal: A Process of Ongoing Improvement. Theory of Constraints,* 4th ed., North River Press, ISBN 9780884271956, 2012.

[4] The 2019 Feature Adoption Report.
`https://www.pendo.io/resources/the-2019-feature-adoption-report/`

[5] Amdahl's Law.
`https://en.wikipedia.org/wiki/Amdahl's_law`

[6] Conway's Law.
`https://en.wikipedia.org/wiki/Conway's_law`

[7] Race to the bottom.
`https://en.wikipedia.org/wiki/Race_to_the_bottom`

[8] The Rise of the Meaning Economy, 2024.
`https://www.youtube.com/watch?v=Nns4Lvs1CG8`

Adatbáziskezelés-feladatok automatikus értékelése[‡]

Horcsin Bálint[*]

Eötvös József Collegium[**]

horcsin@student.elte.hu

Témavezető: Abonyi-Tóth Andor
ELTE IK Média- és Oktatásinformatikai Tanszék

1. Bevezetés

Jelen tanulmány a 2022/2023/2 tanévben készült TDK dolgozatom kivonata. Egyes témakörök kevésbé vannak kifejtve, mint az eredeti műben, valamint egyes eredményeket sikerült javítani. A javításokat jelen dolgozat kizárólag említi, de pontos mérést nem tartalmaz velük kapcsolatban.

Szoftverfejlesztés során gyakori megoldás a programok helyességének vizsgálata éles használat előtt. Hasonló rendszereket alkalmaznak az informatikaoktatásban is, ahol a tanuló megoldását egy tesztkörnyezetben futtatják, és megvizsgálják, hogy adott bemenetre milyen kimenettel válaszol. Ilyen rendszerből sok létezik, amelyekkel általában általános célú programnyelvekre készített megoldásokat tesztelnek. Jelen kutatás célja, hogy az informatika érettségi és az OKTV

[‡] A szerző 2023. júniusi Kari TDK Konferenciára benyújtott dolgozatának rövidített, átdolgozott változata.

[*] ELTE Informatikai Kar

[**] 2021–

adatbáziskezelés-feladataira gyakorlás céljából készített megoldásokat lehessen ellenőrizni webes felületen beküldve.

Adatbáziskezelés-feladatok automatikus értékelésére már léteznek megoldások. Viszont ezek az általuk használt technológiák korlátai miatt nem adhatnak lehetőséget felhasználók és jogosultságok kezelésére vonatkozó feladatok ellenőrzésére, illetve Microsoft Access programban készített adatbázisok biztonságos ellenőrzésére szerveroldalon.

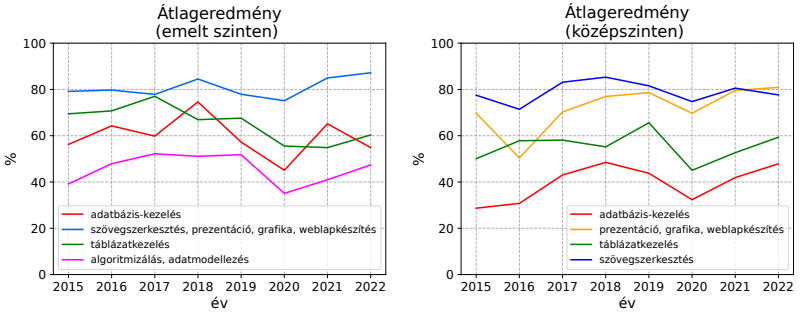
A tesztkörnyezet kialakításához virtuális gépeket használtam, mely megoldás ilyen céllal nem elterjedt. A támogatott adatbázis-kezelők a LibreOffice, a MariaDB és a Microsoft Access. Cél volt néhány korábbi érettségi feladat kitűzése mindhárom adatbázis-kezelőhöz, illetve MariaDB-hez néhány adminisztrációval kapcsolatos feladat kitűzése is.

A kutatás fontos része volt egy hatékony módszer keresése virtuális gépek létrehozására. Az általam elvégzett optimalizációk hatására egy perc alatt akár 35 megoldás is kiértékelhető. Ha csak egy megoldást kell értékelni, az kivitelezhető 6 másodperc alatt. A kutatási eredmények lehetővé tehetik a jövőben más irodai szoftverekkel készített megoldások automatizált értékelését is.

2. Igények vizsgálata

A középiskolai oktatásban rendkívül fontos a tantervben foglalt anyag elsajátítása, a felkészülés az érettségi vizsgára, valamint az Országos Középiskolai Tanulmányi Versenyre (OKTV). Így egy tanulást támogató rendszer abban az esetben tudja a legtöbb tanár és tanuló munkáját segíteni, ha az jól használható a Nemzeti alaptantervben, a középszintű és emelt szintű érettségiken és az OKTV-n szereplő anyag elsajátítására. Tovább növelheti az igényt, hogyha az adott témakörbe befektetett munka sok – érettségin vagy OKTV-n – megszerezhető pontban meg tud térülni. A terület kutatása különösen fontos lehet, hogyha jelentős a különbség a diákok eredményei között, így a gyengébben teljesítők felzárkóztatásával csökkenthetők a társadalmi különbségek.

Informatikából kifejezetten hangsúlyosak a gyakorlati ismeretek, tisztán lexikális tudás mindössze az érettségi szóbeli részén elvárt, azonban a digitális kultúra tantárgy bevezetésével a szóbeli vizsga is gyakorlatiassá válik. A lexikális ismeretek elsajátítására pedig léteznek tankönyvek, így kizárólag a gyakorlati feladatokkal foglalkozom jelen



1. ábra. Érettségi átlageredmények informatikából 2015–2022

dolgozatban. Az informatika (és a digitális kultúra) érettségiken és az informatika OKTV-n a szövegszerkesztés, a prezentáció-, a grafika-, a weblapkészítés, a táblázatkezelés, az adatbázis-kezelés és a programozás témakörével kapcsolatos feladatok fordulnak elő. Vizsgáltam, hogy melyikhez lenne célszerű tanulást támogató rendszert fejleszteni.

Az érettségi eredményeket vizsgálva arra jutottam, hogy a vizsgázó eredménye és iskolatípusa között van összefüggés, illetve a vizsgázó eredménye és a vármegyéje között is. Ugyanakkor iskolatípustól függetlenül megállapítható volt, hogy a feladatokon elért százalékos eredmények – lásd az 1. ábrán – általában a következő sorrendet követték (egyes feladattípusok csak egyik szinten szerepelnek, a felsorolásban elöl szerepelnek a magasabb pontszámúak):

1. szövegszerkesztés
2. prezentáció, grafika, weblapkészítés
3. táblázatkezelés
4. adatbázis-kezelés
5. algoritmizálás, adatmodellezés (programozás)

Jelenleg emelt szinten a legrosszabb eredményeket programozásból érnek el a tanulók, amely a 2009–2012-es időszakban is igaz volt. A relatív szórás is ezen témakörnél a legnagyobb, így a tanulók teljesítményei között jelentős különbségek vannak. Ennek oka lehet, hogy az

egyes iskolák oktatásának minősége nem egyenlő. Ennek lehet olyan oka, hogy egyes iskolákban kevésbé foglalkoznak programozással az informatikaórákon, mivel nem elvárás középszintű érettségi vizsgán. Így részben a tanulókra, szakkörökre, magántanárokra hárul a programozás elsajátításának feladata. A 2012-es elemzés alapján feltételezhető, hogy a tehetősebbek magántanárok segítségét veszik figyelembe, ami növeli a társadalmi egyenlőtlenségeket. [8]

Viszont programozáshoz új tanulást segítő rendszert több okból sem érdemes készíteni. Egyrészt, már léteznek nyilvánosan elérhető rendszerek ezen a területen (például Mester, ProgCont, kodolosuli). Másrészt, digitális kultúra tárgyból már elvárás középszinten is a programozás ismerete, és kifejezett célja volt a tanterv módosításának, hogy a tanulók jobban elsajátítsák a programozást. Így vélhetően nagyobb hangsúlyt fog kapni a programozás a digitáliskultúra-órán, ezáltal remélhetőleg az emelt szintű vizsgaeredmények is javulni fognak programozásból.

Adatbázis-kezelésnél a digitális kultúra tárgy középszintű és emelt szintű érettségije között jelentős különbség van az eltérő választható szoftverkörnyezet miatt. Középszinten továbbra is választható lesz Microsoft Access vagy LibreOffice, de emelt szinten kizárólag MariaDB lesz használható. Így mindenképpen szükség van egy olyan rendszerre, amely segíti az áttérést MariaDB-re, mivel lehet, hogy egyes iskolák tanóráin elsősorban irodai szoftverekkel fog zajlani továbbra is az oktatás. Így lehet, hogy az adatbázis-kezelés lesz digitális kultúra tárgyban az a terület, ahol jelentős különbség van középszinten és emelt szinten, ahogy jelenleg informatika tárgyban a programozásnál van.

Ez az igény hívta életre az `https://www.sqlsuli.hu/` portált, amivel MySQL-t lehet gyakorolni. [4] Viszont vannak kutatandó területek, melyről jelen dolgozat is szól.

A felsőoktatási felvételi ponthatárok vizsgálata alapján kiemelten fontos, hogy a kevésbé szerencsés tanulók is minél jobb minőségű tanulást támogató rendszerhez hozzáférjenek, hogy legyen lehetőségük a felsőoktatásba bekerülni.

2.1. Szoftvertámogatás

A rendszer által értékelt nyelvnek pontosan ugyanannak kell lennie, mint amiben a tanuló dolgozik. Nem tudhatjuk, hogy egy adott iskolában milyen módon tanítják az informatikát. Elképzelhető, hogy egyes

tanulók nehézség esetén procedurális SQL kódot fognak írni. Sőt esetenként akár nem dokumentált működését is használhatják az érintett szoftvernek.

Módszertanilag nem lenne megalapozott, hogy egy alapvetően helyes megoldást amiatt utasítson el a rendszer, mert nem képes értelmezni a leírt kódot. Így elengedhetetlen, hogy pontosan ugyanazon a nyelven ellenőrizzen, mint amivel a tanuló dolgozik.

Tesztjeim során az is kiderült, hogy az eredeti szoftvereket kell használni. Például a UCanAccess nem támogatta a `SELECT 'A';` utasítást, míg a Microsoft Access igen.

A szoftverlistán informatika tárgyból – az érettségi vizsga minden szintjén és az alkalmazói OKTV-n – Microsoft Access és LibreOffice Base (MySQL adatbázismotorral) adatbázis-kezelő program szerepelt. Digitális kultúra tárgyból középszinten javasolt a Microsoft Access és LibreOffice Base (MySQL adatbázismotorral), emelt szinten pedig kizárólag MariaDB (PHPMyAdminon keresztül) adatbázis-kezelő program használható, az alkalmazói OKTV döntőjén használható mind a Microsoft Access, mind a MariaDB.

Kérdéses a támogatni kívánt lekérdezések típusa is. Az informatika érettségi követelményeiben szerepelnek a választó, a törlő és a frissítő lekérdezések, habár jellemzően elegendő tudni választó lekérdezéseket készíteni a vizsgán. Digitális kultúra emelt szintű érettségin viszont már tágabb az elvárható SQL lekérdezések típusának köre (DQL, DML, DDL egyes típusai). Informatika OKTV-n általában már nincsenek különösebb megkötések, annak szokásaira a korábbi évek feladatsorai mutathatnak. Digitális kultúra OKTV-ről még pontos információ nem érhető el, de ha esetlegesen DCL vagy TCL lekérdezések ismeretét is számonkérnék egy versenyen, a rendszernek képesnek kell lennie, hogy onnantól kezdve lehessen az adott lekérdezéstípust is gyakorolni.[1]

Így szükséges oly módon kidolgozni a technikai hátteret, hogy az utólag kiegészíthető legyen tetszőleges lekérdezéstípus támogatására.

[1] Az egyes lekérdezéstípusok rövidítéseinek kifejtése: Data Query Language, Data Manipulation Language, Data Definition Language, Data Control Language, Transaction Control Language.

2.2. Nehézségek összefoglalása

Adatbáziskezelés-feladatok automatikus értékelésével kapcsolatban már léteznek kutatások, de jellemzően egy-egy rendszer csak egy-egy SQL dialektusban beküldött megoldást tud értékelni.

Különös nehézséget okoz, hogy az ellenőrzés során is pontosan ugyanazt a szoftvert – vagy legalább kompatibilis verziót – kell használni az értékelés során, mint amit a tanuló is használ, hogy elkerülhetőek legyenek a különböző szoftverek okozta különbségek. Ezáltal szükséges szerveroldalon futtatni a Microsoft Access szoftvert.

Szövegszerkesztés, prezentáció-, grafika-, weblapkészítés és a táblázatkezelés feladatok automatikus értékelése már komoly nehézséget jelenthet, tekintve, hogy a formátumot is valahogyan ellenőrizni kellene. Nem találtam publikációt, amely ilyen rendszerrel foglalkozna. Viszont mind a szövegszerkesztés, mind a prezentációkészítés, mind a táblázatkezelés számára szükséges Microsoft Office-t szerveroldalon futtatni. Így a Microsoft Office szerveroldalon történő biztonságos futtatásának elérése ezen rendszerek számára is szükséges, egy az adatbáziskezelés-feladatok automatikus értékelésében elért eredmények ilyen rendszerek készítését is elősegíthetik.

3. Létező rendszerek

Programozási feladatok automatikus értékelésére sok különböző megoldás létezik. Ezeket három csoportra osztottam az általuk használt technológiák alapján.

3.1. Általános célú programnyelvek

Általános célú programnyelvek automatikus értékelése jellemzően dinamikusan zajlik. A futtatáshoz szükséges környezet kialakítása általában a Linux jogosultság-kezelésével (namespace-ek, cgroupok) történik.

A dinamikus értékelés során megvizsgálják, hogy adott bemenetre a beküldött kód milyen kimenettel válaszol. A Nemes Tihamér Nemzetközi Programozási Versenyen és az Országos Középiskolai Tanulmányi Verseny programozás kategóriájának a 2. fordulójától és a Nemzetközi Informatikai Diákolimpián is ezt a módszert alkalmazzák.

Ennek a megközelítésnek előnye, hogy a tanuló kódjának értékelése teljes mértékben objektíven, gyorsan, emberi beavatkozás nélkül történik. Amennyiben a tanuló megoldása helyes, és betartja a korlátokat, akkor a rendszer elfogadja.

Ennek a módszernek ugyanakkor hátránya, hogy helytelen megoldásokat is elfogadhat a rendszer. Ez ellen a feladatsorok és tesztesetek gondos összeállításával lehet védekezni.

Bíró, Mester Az ELTE-nek a Mester rendszere egy nyilvánosan elérhető, gyakorlásra jól használható értékelő. A Bíró ennek a versenyeken és egyetemi oktatásban használt változata. Képes a futáshoz szükséges idő pontos mérésére. A rendszer zárt forráskódú, a futtatás sandboxban történik, de a sandboxolást biztosító szoftverről az adatok nem nyilvánosak. [3]

TMS Az ELTE egyetemi oktatásban használt rendszere. Nyílt forráskódú, a biztonságos futtatáshoz Docker konténereket használ.

CMS A Nemzetközi Informatikai Diákolimpián használt rendszer, nyílt forráskódú. A diákolimpiai válogatókon az ELTE-n is ezt használják. Nyílt forráskódú, a biztonságos futtatáshoz az IOI-isolate-et használja. Az IOI-isolate egy nyílt forráskódú program, amely sandboxok létrehozására alkalmas.

Progcont A Debreceni Egyetem nyilvánosan elérhető rendszere, amely használható gyakorlásra, és szerveznek rajta versenyeket, számonkéréseket. A rendszer zárt forráskódú, biztonsági intézkedései nem nyilvánosak. [6, 7]

kodolosuli Az Eszterházy Károly Egyetem nyilvánosan elérhető kifejezetten középiskolásoknak szánt rendszere. A rendszer zárt forráskódú, a futtatás sandboxban történik, de a sandboxolást biztosító szoftverről az adatok nem nyilvánosak. [1]

3.2. Adatbáziskezelés-feladatok, statikus kódelemzés

Korom Richárd (SZTE TTIK) kutatása statikus kódelemzés segítségével ellenőrzi a beküldött megoldás helyességét. Gráffá alakítja a

beküldött SQL kódot és a helyes megoldás gráfjától szerkesztési távolságot számol. Ha több helyes megoldás is meg van határozva, akkor veszi a beküldött kód és az összes helyes megoldás gráfjának szerkesztési távolságai közül a minimumot. Ezt a rendszert elsősorban egyetemi számonkérésekre tervezték. Előnye, hogy képes részpontszámokat adni. Viszont a megoldás kihasználja, hogy a tanulóktól elvárhatják azt, hogy a tanult módon írják meg a lekérdezéseket. Ennek feltétele viszont, hogy a hallgatókat erre kifejezetten fel kell készíteni. [5]

3.3. Adatbáziskezelés-feladatok, dinamikus vizsgálat

Készültek korábbi kutatások, amelyek dinamikusan, tesztesetekkel vizsgálják az adatbáziskezelés-feladatokra készült megoldásokat. A Debreceni Egyetem megoldásában és az Eszterházy Károly Egyetem rendszerében is a megoldás értékelése dinamikusan, tesztesetekkel történik. Ennek során meghatározott állapotokra megnézik, milyen kimenetet ad a beküldött megoldás, vagy milyen módon befolyásolja az adatbázis tartalmát. A módszer hasonlít az általános célú nyelveken készül megoldások tesztelésére elterjedt dinamikus vizsgálathoz. A Debreceni Egyetem SQL tesztelőjében és az Eszterházy Károly Egyetem rendszerében is a kód futtatásánál ugyanazt a MySQL adatbázist használják akkor is, ha két eltérő felhasználó küld be megoldást. Ezért fontos kérdés, hogy milyen módon futtatják az egyes lekérdezéseket biztonságosan, hogyan állítják helyre az adatbázis eredeti állapotát. [4]

3.4. A Debreceni Egyetem tesztelője

A rendszer a készítők leírása alapján `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE TABLE` és `ALTER TABLE` utasításokat képes tesztelni. Pontos információt nem tettek közzé, hogy milyen módon érik el, hogy a lekérdezések biztonságosan legyenek futtatva. Egy `SELECT` utasításnak önmagában mellékhatásmentesnek kell lennie, így ilyen esetekben az adatbázis állapota nem változik. Az `INSERT`, `UPDATE` és `DELETE` lekérdezéseket tranzakcióban futtatják, a végén így vissza tudják állítani az adatbázis állapotát. Egyes utasításokat nem lehet tranzakcióban végrehajtani. Ezeket mindenképpen az eredeti adatbázison kell lefuttatni, így az adatbázis-motor ilyenkor egy ún. implicit commitet hajt végre. Hogy milyen utasításokat lehet végrehajtani implicit commit nélkül, az

adatbázis-kezelőtől függ, de jellemzően a DQL és a DML lekérdezések nem okoznak ilyet, míg a DDL, a DCL és a TCL lekérdezések igen. MySQL és MariaDB esetén a `CREATE TABLE` és az `ALTER TABLE` utasítások implicit commitet vonzanak maguk után, így ezt követően nincs lehetőség az adatbázist egy `ROLLBACK` utasítással helyreállítani.

A `CREATE TABLE` és `ALTER TABLE` után a visszaállítás módját nem pontosan fejtették ki, de ez nem egy nehéz probléma. [4]

3.5. Az Eszterházy Károly Egyetem tesztelője

Az Eszterházy Károly Egyetem értékelőjében a `SELECT` utasításokat olyan felhasználóval futtatják, amelynek csak erre van jogosultsága. Frissítő lekérdezéseket ideiglenes táblákban futtatják, amelyhez a `CREATE TEMPORARY TABLE` jogosultság szükséges. Ezen a táblán ezt követően a felhasználó tetszőleges lekérdezést futtathat.

3.6. Létező értékelő rendszerek értékelése

Általános célú programnyelveken készült megoldások tesztelésére alkalmazott rendszerek évek óta működnek megbízhatóan, így az általuk használt értékelési és védelmi eszközöket vagy hasonlókat célszerű lehet használni egy adatbáziskezelés-feladatokat automatikusan értékelő rendszernél is. Adatbáziskezelés-feladatok automatikus értékelése statikus elemzéssel egy jó megoldás lehetne, de vizsgálatom arra jutott, hogy a saját kutatásomban a célközönség számára ez nem lenne megfelelő. Korábbi, adatbázis-kezeléssel kapcsolatos dinamikus értékelők értékelési módszerei kiválóan használhatóak, viszont azok a lekérdezések biztonságos futtatását az adatbázis-kezelőkre bízzák.

Ugyan adatbázis-szerverekben vannak védelmi intézkedések jellemzően, de ezek megbízhatósága függ az érintett adatbázis típusától, és Microsoft Access-ben nincs ilyen jellegű védelem.

Így célszerű az eddigi dinamikus értékelők értékelési módszereit használni. Emellett érdemes az általános célú nyelvekhez használt futtatási környezethez hasonlóan magát az adatbáziskezelő-motor folyamatát korlátozni, és minden esetben újat létrehozni. Ennek előnye, hogy nem függ az adatbázis típusától, hogy biztonságosan futnak-e a tesztek, és így tetszőleges – akár jogosultságkezelő DCL – lekérdezések is futtathatók biztonságosan.

4. Az ellenőrzés folyamata

Feltételezzük, hogy tetszőleges lekérdezést futtathatunk egy adatbázis-szerveren biztonságosan (vagyis úgy, hogy a rendszer a következő értékelés előtt helyreállítja az adatbázis-kezelőt).

SQL kódok dinamikus ellenőrzésére már volt eddig is megoldás. Megállapították, hogy adatbázis-kezelőkben SQL-lekérdezések futtatásával és azok eredményének vizsgálatával van lehetőség ellenőrizni, hogy egy beküldött megoldás helyes-e. Ezt az elvet én is követtem, és próbáltam az értékelési lépéseket oly módon absztrahálni, hogy az egyes feladatok kitűzése során elég legyen mindössze egyetlen adatbázis-kezelőn kitűzni a feladatot, a többin automatikusan működjön. Új adatbázis-kezelők hozzáadásakor pedig elég legyen ezeket az absztrakt lépéseket implementálni az adott adatbázis-kezelőhöz, és a korábban elkészített feladatokon ne kelljen módosítani.

Ezt a célt sikerült elérni, amelyhez a következő lépéseket kell támogatnia az adatbázis-kezelőnek:

1. a beadott adatbázisban a lekérdezések nevének és SQL-szkriptjének megszerzése;
2. egy adott tábla tartalmának cseréje adott tartalommal;
3. egy adott tábla tartalmának mentése;
4. egy adott névvel rendelkező (beadott) választó lekérdezés eredményének mentése;
5. egy adott névvel rendelkező (beadott) módosító lekérdezés végrehajtása;
6. adatbázis visszaállítása eredeti állapotára.

Választó lekérdezések ellenőrzése Amennyiben választó lekérdezést akartam tesztelni, ahhoz elég volt teszteseteket készíteni, az azokhoz tartozó táblák tartalmát cserélni, a tesztelt lekérdezés eredményét lementeni, és azt ellenőrizni utólag. Az ellenőrzést már a lementett adatokból, adatbázistól függetlenül meg tudtam csinálni.

Módosító lekérdezések ellenőrzése Amennyiben módosító lekérdezést akartam tesztelni, ahhoz elég volt a tesztesethez tartozó táblákat cserélni, a módosító lekérdezést lefuttatni, és ezután a táblák tartalmát lementeni, és később ellenőrizni. Ha az adatbázis sémája módosul, akkor vissza kell állítani az adatbázis eredeti állapotát, majd a tesztesethez tartozó betöltéseket elvégezni, és ez után folytatni tovább az értékelést.

4.1. Az absztrakció előnyei

Ez az absztrakció jelentősen egyszerűsítette a feladatok kitűzését és újabb adatbázis-kezelők támogatását, és a legtöbb érettségi és OKTV feladat az absztrakciót követően is kitűzhető maradt.

Egy feladat kitűzéséhez a teszteseteket össze kell készíteni, amiket egy megfelelő programba be kell tölteni, amely előkészíti a teszteléshez szükséges fájlokat. Az előkészítésre használt program ezen kívül megkaphatja a feladat megoldását, így képes generálni a helyes megoldásokat. A tesztelés lépéseinek egységesítése egyszerűvé teszik, hogy az egyes feladatokat több adatbázis-kezelővel is meg lehessen oldani. Ehhez nem szükséges a megfelelő SQL dialektusra átalakítani kézzel az egyes utasításokat, mivel az absztrakt lépésekből a rendszer meg tudja adni az adatbázis-kezelő típusának megfelelő utasítást. Ennek, és a sandboxolásnak köszönhetően újabb adatbázis-kezelők támogatása rendkívül egyszerű. Így, ha a jövőben változnak a közoktatás igényei, könnyen korrigálható a rendszer.

További előnye a tesztelés lépéseinek egységesítésének, hogy az egyes lépésekhez a vizualizációt elkészítő kód univerzálisan használható a feladatok között. Így a tanulók könnyebben megérthetik, hogy milyen jellegűek a tesztesetek, mit csinál a kódjuk, és mi lenne az elvárt működés.

A jövőben érdemes lehet további lépéseket is támogatni, hogy többféle lekérdezés-típust is lehessen ellenőrizni. Ez okozhatja, hogy csak bizonyos adatbázis-kezelőkkel lesz az ellenőrzés kompatibilis. Ilyenkor elég rögzíteni, hogy mely adatbázis-kezelőkre fogad megoldást az adott feladatra az értékelő. A sandboxolás miatt lehetséges jogosultságkezelő függvényeket is tesztelni elméletben, de például a Microsoft Access legújabb .accdb adatbázisaiban nincs jogosultságkezelés, így rajta ilyen feladatokat nem lehet kitűzni, de MariaDB-ben és MySQL-ben nincs akadálya.

5. Sandboxolás

Elvárás volt, hogy a felhasználónak teljes hozzáférése legyen az adatbázis-kezelőhöz, ezáltal tönkre is tudja tenni azt. Viszont ha kárt tesz benne, akkor csak a számára kijelölt adatbázis-kezelő romlik el, ami az értékelés végén megsemmisül. Így károkozásra törekvéskor csak az éppen aktuális értékelés nem sikerül, a többiben viszont nem keletkezik kár.

Ez megfelelő működés, a felhasználó saját magának okozhat kárt.

A futtató kizárólag a tesztesetek bemenetét és a tesztelés kijelölt folyamatát kapja meg. Így van lehetőség arra is, hogy sem a felhasználó, sem a programja nem fér hozzá a helyes megoldáshoz. Ez hasznos lehet, ha számonkérésekre szeretnék használni a rendszert. Az ellenőrző ezért is fut ettől eltérő kontextusban, mert neki szüksége van a helyes megoldásokra. Ezt szokás más értékelőknél is használni. A publikus és privát adatok közti szétválasztást a feladatkitűző program elvégzi.

Különös problémát jelentett, hogy Microsoft Access-ben van lehetőség makrókat írni, így tetszőleges kódot futtatni. Egy MySQL és egy MariaDB adatbázist a `DROP SCHEMA mysql;` utasítás lefuttatásával tönkre lehet tenni. Ezeket le lehet ugyan tiltani, de nehéz egyes lekérdezésekről eldönteni, hogy valahogy veszélyt jelenthetnek-e, illetve hogy kijátszható-e a védelem. Microsoft álláspontja, hogy a Microsoft Access-t nem tervezték szerveroldalon futtatni, mivel nem képes mellékhatásmentes működésre, nem is biztonságos, nem feltétlen determinisztikus, illetve grafikus felület nélkül holtpontra juthat. Így mind Microsoft Access-t, mind MariaDB-t és MySQL-t egy biztonságos környezetben (sandboxban) kell futtatni, nem célszerű az adatbázis-kezelőre bízni a biztonságos futtatást. Az értékelések között a sandboxot meg kell semmisíteni és újat kell létrehozni. Sandboxolásra gyakran a Linux jogosultságkezelő rendszerét szokás használni (namespace-ek, cgroupok), amit például az IOI-isolate kezel, amit a CMS rendszer használ, vagy a Docker kezel, amit a TMS használ.

Jelen probléma megoldásakor a kritikus rész a lekérdezések futtatása, amelyet mindenképpen sandboxban szükséges lefuttatni. Sok esetben a kommunikáció a sandboxban futó programokkal úgy történik, hogy a sandboxnak felcsatolják a bemeneti fájlt és a kimeneti fájlt. A bemeneti fájlon módosítást nem végezhet, a kimeneti fájl maximális mérete pedig rögzített. Általában tesztesetenként új sandbox van lét-

rehozva. Utóbbi azért is fontos, hogy pontos adat legyen a futási időre vonatkozóan. Az eredményt általában egy másik környezetben szokás ellenőrizni, hogy ne tudja megszerezni a beküldött kód az eredményt. Általában rossz megoldás, hogyha ugyanaz a sandbox határozza meg az eredményt, amelyen a versenyzői kód fut. Ezt az elvet én is követtem.

A lekérdezés hatékonyságával csak korlátozott mértékben foglalkoztam ebben a rendszerben, így nem volt szükséges az egyes teszteseteknek saját sandboxot készítenem, csak a különböző beküldéseknek. Mivel egy virtuális gép előállítása mindenképpen egy erőforrásigényes folyamat, így tesztelésenként csak egy virtuális gépet állítottam elő. Cél volt, hogy multiplexelés segítségével csökkentsem az új virtuális gépek létrehozásának költségét. A multiplexelés célja, hogy a különböző folyamatok az egyforma erőforrásaikat (pl. merevlemez) megosztják egymással, hogy ne kelljen több példányt fenntartani, ha van lehetőség.

Microsoft Access csak Windows rendszeren támogatott. A WINE lehetőséget adhat Windows-ra írt szoftverek Linuxon történő futtatására. Modern Microsoft Office futtatására képes volt már felhasználó, de Access és Outlook nem működött. Szerverkörnyezetre stabil megoldás készítésére nem látok módszert ami WINE-t használ. Így ténylegesen Windows-os környezetben szükséges a Microsoft Access-t futtatni.

MariaDB és MySQL futhat mind Windows-os környezetben, mind linuxos környezetben, és Dockerben is, így őket a kari TMS rendszerbe is sikerült integrálni. Annyi megkötéssel, hogy a pontokat számító program is ugyanabban a sandboxban fut, mint az adatbázis. De ezek az adatbázisokon kívülre nehéz jutni lekérdezések segítségével.

Az alapvető Windows-os sandboxolási megoldások (Windows Sandbox, Windows Container) nem voltak alkalmas a feladatra, így rendes virtuális gépeket kellett használnom.

Ilyenkor lehet készíteni másolatokat álló virtuális gépekről (pl. Vagrant, virtuális gépkezelők klónozási, snapshotolási funkciói), viszont ezek során meg kell várni a virtuális gép indulását, ami több idő lehet, mint maga a tényleges értékelés.

Futó gépek klónozására már létezett a VMware vSphere Instant Clone technológiája, amely célja pontosan az, hogy egy éppen futó virtuális gépről másolat készítsen. Klónozáskor létrejön még egy virtuális gép, amely pontosan ugyanabban az állapotban van, mint az eredeti (például memóriaállapot, merevlemez). Sem a merevlemezt, sem a memóriát nem szükséges ehhez lemásolni, hanem elegendő innentől kezdve

tárolni, hogyha valamelyik virtuális gép egy adott adatot módosít rajta (copy-on-write stratégia). [9]

A teljesítményéről készült korábbi kutatás alapján nem célszerű használni instant klónokat, hogyha adatbázisok futnak a virtuális gépen, mivel a lemez írási sebessége komoly korlátozó tényező. A szoftver emellett nem nyílt forráskódú, így más megoldást kerestem.

5.1. Saját megoldás

A saját megoldást Libvirt–QEMU–KVM szoftverekre, AMD processzorra implementáltam, de más architektúrán és virtuálisgép-kezelőkkel is működnie kell a megoldásnak.

Az egyértelmű volt, hogy az egyes virtuális gépekhez egy-egy differencing image-et kellett rendelnem, amelyek egy közös lemezképfájlból származtak le. Így elkerülhető volt, hogy értékelésenként egy 10–20 GB méretű lemezképfájlt kelljen másolnom. Másik egyértelmű megoldás volt, hogy a differencing image-eket egy nekik készített TMPFS-ben tároltam (néhány más, értékeléshez fontos fájllal együtt), amelyeket amúgy is törölnöm kellett az értékelés végén. A virtuális gép lemezre írásai mind RAM-ba mentek, így az egyes virtuális gépek írásai egymást érdemben nem befolyásolták. Ennek további előnye volt, hogy az SSD-re az értékelés közben minimális mennyiségű adatot kellett írni, így növelve élettartamát. A túlzottan nagy méretű lemezképfájl készítését is meg tudtam akadályozni, mivel a TMPFS nem engedte az előre megadott méretnél nagyobbra növeszteni a lemezt. Ha túllépné a keretét, a virtuális gép nem tud több adatot menteni a lemezre. Teszteléskor a Windows ilyen esetben nem tudott tovább működni, Ubuntu Desktop viszont ezt követően is reszponzív maradt, de kikapcsolás után nem lehetett újra beindítani. Ezt a megoldást alkalmazták már korábbi kutatásban is. [2]

A lemezre írást ezekkel a módszerekkel megoldottam, és már így is rendkívül gyorsan tudtam megoldásokat értékelni. A méréseim során bebizonyosodott, hogy a dolgozók számának (észszerű) növelésével nőtt az egységnyi idő alatt értékelhető megoldásának száma, viszont az egyes értékelések kiértékeléséhez szükséges idő is. Előbbi pozitív eredmény volt, utóbbira számítottam. Az is kiderült, hogy ennek egyik okozója az SSD-k lassú olvasási sebessége volt, ha egyszerre több virtuális gép

futott. Korábbi kutatások bemutatták, hogy ez a VMWare vSphere számára is problémát okozott hasonló terhelés mellett. [9]

Az egyes értékelésekkor sok esetben hasonló folyamatok mennek végbe a virtuális gépben, mint egy másik értékeléskor (például a Microsoft Access betöltődése), és ehhez lényegében ugyanazoknak a fájloknak a betöltésére van szükség. Így készítettem egy kis méretű lemezképfájlt, ami a nagy méretű képfájl differencing image-e, és a kis méretű lemezképfájlon eltároltam az értékeléshez általában szükséges blokkokat. Ennek készítésére lehetőséget biztosít, hogyha a differencing image-en beállítottam a copy-on-read stratégiát. A kis méretű képfájlt RAM-ban tároltam, és abból származtattam az egyes virtuális gépek lemezképfájlát.

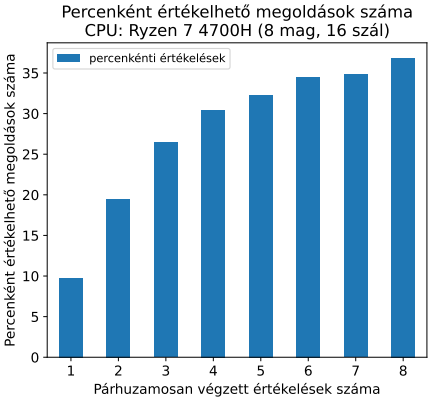
A virtuális gép processzor és memóriaállapota visszaállítható fájlból. Ekkor a különböző ellenőrzések miatt a gépkezelő nem engedélyezi az ismételt betöltést. Nevesített csővezetékbe megfelelő tartalom készítésével a probléma megoldható. Ennél gyorsabb megoldás az ellenőrzések kiiktatása. Utóbbi jelen dolgozat méréseibe nem került be.

A virtuális géppel a kapcsolatot több módon is fel lehet venni, én egy egyedi fájlrendszert és fájlrendszer-meghajtót készítettem erre a célra.

6. Összefoglalás

Vizsgáltam az értékelés sebességét oly módon, hogy hány értékelést tudok elvégezni egy személy számítógépen percenként. A tesztelés során Microsoft Accessben készült megoldásokat teszteltem, a 2022-es tavaszi idegen nyelvű emelt szintű informatika érettségi „Oscar-díjas filmek” című feladatát kitűzve. A feladat 9 tesztesetből állt, tesztesetenként 3 táblát töltött be az alkalmazás, és 8 választó lekérdezést ellenőrzött. A másik két adatbázis-kezelő még ennél is gyorsabb volt. Így a mérés jól reprezentálja a jövőbeli felhasználást.

Az alkalmazott környezet egy notebook volt AMD 7 4800H CPU-val, 32 GB RAM-mal, egy Kingston SUV400S37240G SSD-vel, Ubuntu 22.04.2 LTS (64-bit) operációs rendszerrel. A 2. ábra szemlélteti, hogy a percenként elvégezhető értékelések száma hogyan változott annak függvényében, hogy egyszerre legfeljebb hány értékelést végeztem.



2. ábra. Percenként elvégezhető értékelések száma

Sikerült kidolgoznom egy absztrakt értékelési módszert, amely alkalmas adatbáziskezelés-feladatok automatikus értékelésére, és könnyen bővíthető újabb adatbázis-kezelőkkel.

Sikerült létrehozni az első olyan automata értékelő-rendszert, amely virtuális gépek hatékony létrehozásával alakítja ki a futtatási környezetet az értékelések számára. Először alkalmaztam copy-on-read, copy-on-write stratégiát TMPFS-sel kombinálva programok gyors és hatékony betöltésére virtualizált környezetben.

Ezen előrelépésekkel sikerült bizonyítanom, hogy van lehetőség automatizált módon Microsoft Access-ben készült megoldásokat szerveroldalon értékelni. Az értékelés rendkívül hatékony módon történik, percenként akár 35 megoldást is lehet értékelni.

A kutatási eredményeim lehetővé tehetik a jövőben más irodai szoftverek automatizált értékelését is (például Excel).

6.1. További fejlesztési irányok

A jövőben célszerű lesz megvizsgálni, hogy megoldható-e a virtuális gépek hatékonyabb létrehozása a QEMU vagy a libvirt módosításával vagy cseréjével. Célszerű lenne kiegészíteni a rendszert kevésbé erőforrásigényes sandboxolási megoldásokkal olyan programokhoz, amelyek nem igényelnek egy teljes virtuális gépet.

Hatékonyabbá lehetne tenni a táblák betöltéséhez az előfeldolgozást Microsoft Access-hez. Az elért eredmények alapján célszerű megvizsgálni más irodai szoftverek automatizált értékelését is (például Excel). Célszerű lenne további feladatokat is előkészíteni a rendszerben, valamint a tesztesetek hatékonyabb összekészítéséhez megoldást készíteni.

6.2. További kutatási irányok

Mindenképpen célszerű lenne beválásvizsgálatot tartani, jelenlegi tervek szerint ez a 2023/2024/2 félévben meg is fog történni egyetemi környezetben.

Köszönetnyilvánítás

Köszönöm témavezetőmnek, Abonyi-Tóth Andor egyetemi docensnek ötleteit és tanácsait, amik nagyban hozzájárultak jelen dolgozat sikeres elkészültéhez. Köszönettel tartozom továbbá az Eötvös József Collegium Informatika Műhely tagjainak és vezetőjének, Lócsi Levente adjunktusnak, akik javaslataikkal támogatták ezen dolgozat elkészültét.

A Kulturális és Innovációs Minisztérium ÚNKP-22-6 kódszámú Új Nemzeti Kiválóság Programjának a Nemzeti Kutatási, Fejlesztési és Innovációs Alapból finanszírozott szakmai támogatásával készült.

Hivatkozások

[1] T. Balla, S. Király, A Discussion of Developing a Programming Education Portal, *Central-European Journal of New Technologies in Research, Education and Practice*, DOI:10.36427/CEJNTREP.2.2.833

[2] A. Coleșa, T. Bura, A. Pop, S. Lukács, Fast creation of short-living virtual machines using copy-on-write RAM-disks, *Proceedings of 2014 IEEE International Conference on Automation, Quality and Testing, Robotics, AQTR 2014*, DOI:10.1109/AQTR.2014.6857854

[3] Gy. Horváth, Gy. Horváth, L. Zsakó, A BÍRÓ és a MESTER – az online értékelés szerepe a programozás oktatásban, *Mérési és értékelési módszerek az oktatásban és a pedagógusképzésben*, ELTE Eötvös Kiadó, 2017, ISBN 978-963-284-910-2, pp. 89–103.

[4] S. Király, T. Balla, R. Király, Learning SQL by practicing on popular movie databases, *Central-European Journal of New Technologies in Research, Education and Practice*, DOI:10.36427/CEJNTREP.4.1.4465

[5] R. Korom, Adatbázis modellezés és SQL feladatok automatikus kiértékelése gráfelméleti alapokon, `https://www.inf.u-szeged.hu/sites/default/files/koromrichard_tdka2020o.pdf`

[6] J. Pánovics, T. Kádek, P. Biró, A ProgCont rendszer jelene és jövője, `http://konferenciak.inf.elte.hu/infodidact/InfoDidact22/Manuscripts/PJKTBP.pdf`

[7] J. Pánovics, M. Kósa, A. Orosz, R. Tóth, Új fejlesztési irányok programozási feladatok megoldáskiértékelő rendszereihez, `http://konferenciak.inf.elte.hu/infodidact/InfoDidact17/Manuscripts/KTKMOATR.pdf`

[8] G. Siegler, Érettségi vizsgatárgyak elemzése 2009–2012 tavaszi vizsgaidőszakok – Informatika, `https://www.oktatas.hu/pub_bin/dload/unios_projektek/tamop318/erettsegi_vizsgatargyak_elemzese/informatika.pdf`

[9] S. Sreekanth, Understanding Clones in VMware vSphere 7, `https://www.vmware.com/techpapers/2021/cloning-vSphere7-perf.html`

Órarendtervező a BGGYK részére[‡]

Kámán Rebeka[*]

Eötvös József Collegium[**]

kamanrebeka@student.elte.hu

Témavezető: Lócsi Levente
ELTE IK Numerikus Analízis Tanszék

Előszó

Ez az írás a szakdolgozatom kissé átdolgozott, rövidített változata. Igyekeztem csak azokat a részeket benne hagyni, amelyek bárki érdeklődő számára érdekesek lehetnek, és kihagyni a technikai részleteket. Ez többnyire azt jelenti, hogy a felhasználói dokumentációból marad a legtöbb, mert az látványosabb, a fejlesztőiből épphogy csak egy picit hagytam, hogy el lehessen képzelni, hogy milyen adatbázis-kapcsolatok vannak, illetve kiegészítettem egy utószónak nevezett résszel, amiben a köszönetnyilvánítással egyben leírom a személyesebb részét annak, hogy hogyan született meg ez a program.

Az órarendtervező oldal még fejlesztés alatt van, így lehetséges, hogy nem minden kép és leírás teljesen naprakész.

1. Bevezetés

Az ELTE Bárczi Gusztáv Gyógypedagógiai Kara azzal a megkereséssel fordult az Informatikai Karhoz, hogy segítsünk nekik az órarendszerkesztést hatékonyabbá tenni. Eddig Excelben szerkesztettek,

[‡] Programtervező informatikus BSc szakdolgozat, 2023. május.
[*] ELTE Informatikai Kar
[**] 2020–

aminek nagy hátránya, hogy ha egyszer egy tanórát beosztanak, hogy mikor, melyik teremben, melyik oktató, melyik csoportnak tartsa, akkor azt az adott terem, oktató és csoport órarendjébe is át kell másolni, ami, ha félbemarad vagy elírnak valamit, inkonzisztenciához vezethet, nem is beszélve a honlapjukra kikerülő órarendekről, ami szintén plusz formázást igényel.

A dolgukat az is nehezíti, hogy az Informatikai Karral ellentétben nem heti órarend van, hanem minden egyes nap más. A félévente 1000 körüli meghirdetett kurzus nehezen fér el a termekben, így a termek kihasználtságának maximalizálása is egy fontos szempont, illetve az is, hogy a 16 szakiránypár kötelező tárgyai ne ütközzenek (és a kötelezően választhatóakból is minél kevesebb, de ez sajnos nem mindig megoldható).

A szakdolgozatom célja egy olyan felhasználóbarát felület létrehozása a Gyógypedagógiai Kar órarendszerkesztői számára, ami segíti őket az emberi figyelmetlenségek, és az ebből adódó ütközések, inkonzisztenciák elkerülésében.

A program egy weboldal, ami tartalmaz publikusan hozzáférhető órarendeket (termekét, oktatókét és az egyes szakiránypárokhoz tartozó összes kurzusét), illetve Caesar azonosítóval bejelentkezés után csak a jogosultaknak hozzáférhető szerkesztőfelületet, ami figyelmeztet, ha az éppen beosztani kívánt kurzusalkalom ütközést okozna, majd adatbázisba menti az adatokat, így nem kell kézzel frissíteni a termek, oktatók és szakiránypárok órarendjét.

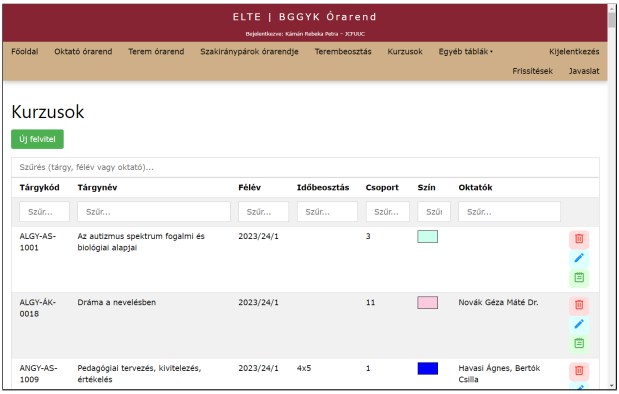
2. Felhasználói dokumentáció

2.1. Első használat

A program a `https://barczi-orarend.elte.hu/` oldalon érhető el. Az „órarend-nézegetők” bárki számára publikusan elérhetők.

A szerkesztő felület a Caesar azonosítóval való bejelentkezés után érhető el, akkor is csak a jogosultsággal rendelkezőknek. Akiknek nincs plusz jogosultságuk, azoknak a bejelentkezéssel (egyelőre) csak a fejléc változik: megjelenik a nevük és a Neptun kódjuk.

Az oldal reszponzív, így mobilon is használható (lásd *Mobil nézet* rész, 174. oldal).



1. ábra. Képernyőkép az alkalmazásról: kurzusok táblázata

2.2. Felhasználók

A felhasználókat két csoportba osztjuk jogosultság szerint:

1. **„bárki”:** A publikus órarendeket (terem, oktató, szakiránypár) bárki megtekintheti bejelentkezés nélkül is. Jellemzően ilyen a portás, a hallgatók és az oktatók.

2. ábra. Bejelentkezés nélkül elérhető menüpontok

2. **„szerkesztő”:** bizonyos felhasználóknak, akik szerkesztik az órarendet, vagy beleláthatnak a folyamatokba (pl. oktatási dékánhelyettes), bejelentkezés után elérhetővé válnak a szerkesztéshez szükséges táblázatok.

3. ábra. A szerkesztők által elérhető menüpontok

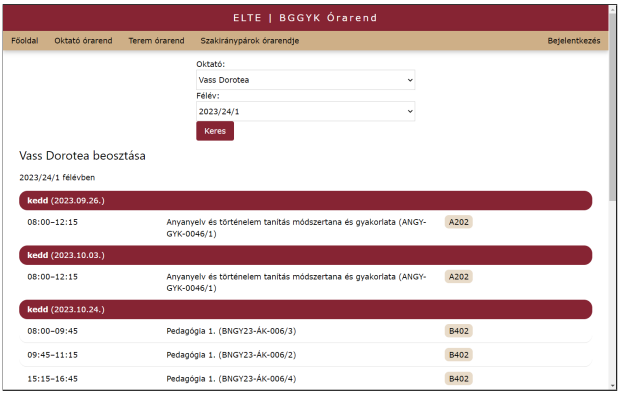
2.3. Publikus órarendek

Oktató órarend

A felületen a legördülő listákból oktató és félév kiválasztása után (mindkettő választása kötelező) megjelenik egy lista napok szerint csoportosítva (azon belül időpont szerint rendezve) az összes releváns alkalommal (időpont, tárgy neve és kurzus kódja, terem/termek), vagy az, hogy nincs ilyen.

A lista frissül a „Keres” gombra kattintva, illetve a beviteli mezők megváltoztatása után is. A változtatásra frissülés kényelmi funkció, viszont a szerver nem mindig válaszol, ezért megeshet, hogy nem történik semmi, így szükség volt egy gombra is, amit ilyenkor meg lehet nyomni.

Amennyiben oktató vagy félév nincs kiválasztva, egy üzenet jelenik meg az órarend helyén, hogy az órarend megjelenítéséhez mindkettő szükséges. A „Keres” gombra kattintás után felugró ablak is figyelmeztet, még egyértelműbbé téve ezt.



4. ábra. Egy oktató órarendje

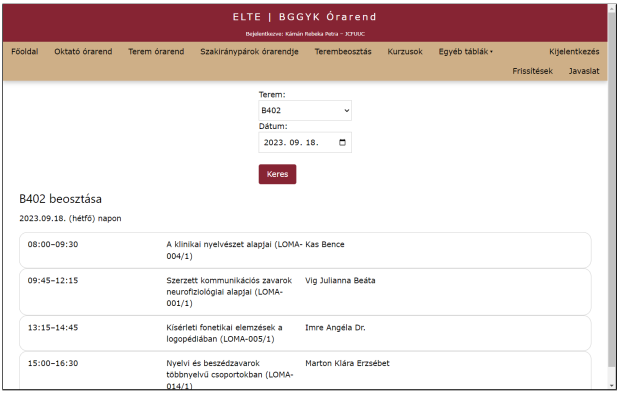
Különösen hasznos:

- Az oktatók számára, akik láthatják már az órarend készülése közben is, és ezáltal időben jelezni tudják a szerkesztőknek, ha mégsem jó nekik a beosztott időpont.

- Az oktatók láthatják a saját órarendjüket (az oldal betöltésekor automatikusan az adott napit), illetve más oktatókét is, akikkel pl. együtt tartanak egy kurzust.
- A hallgatók számára, akik meg szeretnének keresni egy oktatót személyesen.

Terem órarend

Egyszerre egy terem egy napi foglaltságát lehet megtekinteni.



5. ábra. Egy terem órarendje

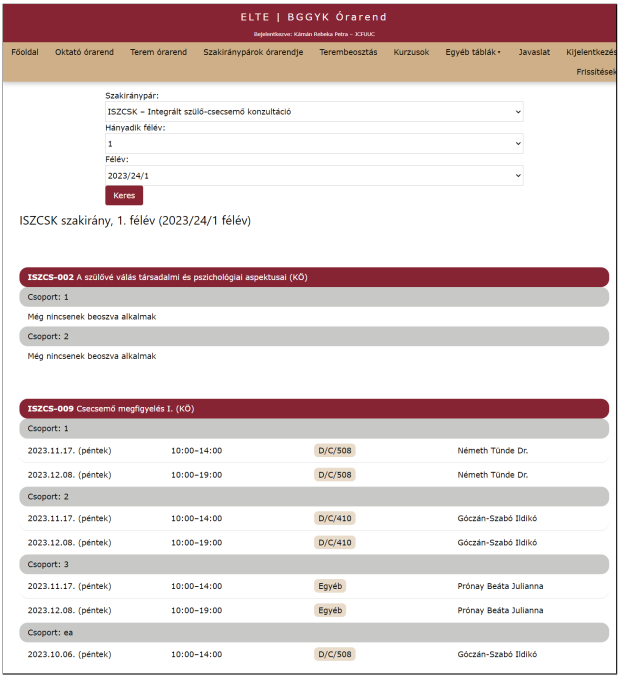
Különösen hasznos:

- A portásnak, akitől szeretnének szabad termet kérni. (Főleg szombaton, amikor az órarendszerkesztő jellemzően nem dolgozik, így nincs kit kérdeznie.)
- A hallgatóknak, akik beülnének egy üres terembe tanulni pl. egy lyukasórájukban.

Szakiránypár órarend

A lista tartalmazza az összes tárgyat, ami a kiválasztott szakiránypárt érinti, azon belül tárgyanként az adott félévre meghirdetett kurzusokat, akkor is, ha alkalmak még nincsenek hozzá beosztva. Tárgy,

azon belül kurzus (csoport) szerint van csoportosítva. A sorokban megtalálható az alkalom dátuma, ideje, helyszíne (terme) és a hozzárendelt oktatók.



6. ábra. Egy szakiránypárhoz tartozó tárgy összes kurzusának alkalmai egy félévre

Különösen hasznos:

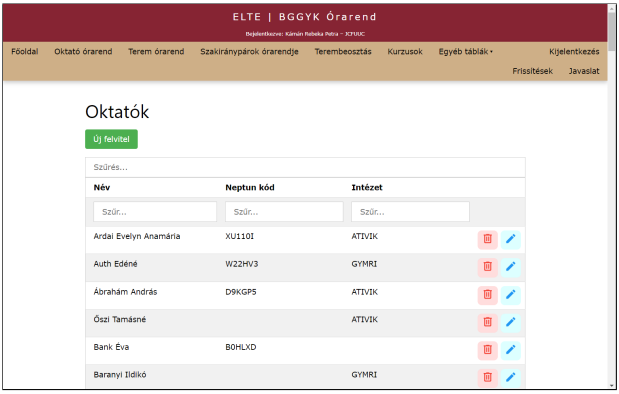
- A hallgatók számára, akik félévkezdés és kurzusfelvétel előtt szeretnék látni, hogy melyik tárgyból milyen időpontokban hirdetnek meg alkalmakat, és ezek melyik másik kurzus alkalmaival ütköznek, így segítve a kurzusok kiválasztását.
- Akár a szerkesztőknek is, hogy könnyebben lássák, hogy melyik kurzusokhoz hiányoznak még alkalmak.

2.4. Szerkesztő táblázatok

Egyszerűbb táblázatok

Olyan adatokat (pl. oktatók nevét vagy tárgyak kódját) tároljuk ezekben, amikre később keresni és szűrni fogunk, vagy kiválasztani legördülő listából, ha valamihez hozzá akarjuk rendelni.

A legtöbb ilyen táblázatot egyszer kell feltölteni, illetve félévente egy részét kiegészíteni és módosítani.



7. ábra. Oktatók táblázata

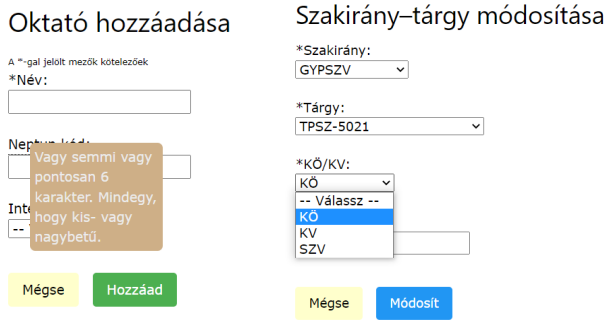
Az ilyen táblázatok oldalainak felépítése:

- A táblázat címe.
- „Új felvitel” gomb.

  Kattintásra megjelenik egy űrlap a kitöltendő adatokkal.

  A kötelező mezők jelölve vannak. Aminél nem teljesen egyértelmű, hogy mit vagy hogyan kell megadni, ott egy súgó jelenik meg. Ilyen például a kurzusoknál a csoport, ahol 0-t is meg lehet adni, ami előadást jelent és „ea”-ként jelenik meg a többi helyen.

- A „Szűrés” mezőbe gépelés közben a táblázat sorai közül eltűnnek azok, amelyeknek egyik cellájában sincs a beírt szöveg.



(a) Hozzáadás súgóval *(b)* Módosítás legördülő választómezővel

8. ábra. Két példa űrlapokra

  - „nincs” kulcsszó beírására csak azok a sorok jelennek meg, amelyeknek minimum egy cellájuk üres.
  - Nem érzékeny a kis- és nagybetűkre.
  - Ha számot írunk a keresőbe, és az oszlop csak számokat tartalmaz, akkor csak a pontos egyezés számít, így pl. ha „2”-t írtunk be, akkor a „21”-et tartalmazó cella sora nem fog megjelenni.
  - Tipp: Ha például az ’A’ betűvel kezdődő termeket szeretnénk csak megnézni, és beírunk egy nagy ’A’ betűt a keresőbe, akkor azt tapasztalhatjuk, hogy több sor látszik, mint amire elsőre számítunk. Ez azért van, mert nem szó elején, hanem a szón belül bárhol érvényes az egyezés, és a kis ’a’ betű is számít, ami lehet hogy a többi oszlopban, pl. a fantázianevében vagy a hozzá fűzött megjegyzésekben is megtalálható. Ilyenkor javasolt az oszlopszűrő használata.

- Fejléc. Egy oszlopcímre kattintva a sorok növekvő sorrendbe rendezhetők, második kattintásra pedig csökkenőbe.

  Tipp: Ugyan csak egy oszlopot lehet megadni, ami alapján rendezni szeretnénk, de ha fordított sorrendben kattintunk az oszlopokra (és közben nem töltjük újra az oldalt), akkor megmarad a rendezés, így pl. ha a kurzusoknál az egy tárgyhoz tartozó csopor-

tokat oktató alapján szeretnénk rendezni, akkor először rendezzük oktatók szerint, majd tárgy szerint, így az egyes tárgyakhoz tartozó oktatók sorrendben maradnak az előző rendezés miatt.

- Fejléc alatti sor, a gombokon kívül minden oszlop: oszlopszűrő. Működése megegyezik az előbb részletezett szűrőjével, de csak egy oszlopra szűr.

  Ha több oszlopszűrőbe is írunk keresési feltételeket, akkor mindegyiknek egyszerre kell teljesülnie a megfelelő oszlopokban ahhoz, hogy egy sor megjelenjen/ne tűnjön el.

  A „van” szó beírásával nemüres mezőre is lehet keresni. (A teljes szélességűnél nem, mert minden sorban van valami és felesleges lenne.)

- A sorok jobb oldalán törlés és módosítás gomb található.

  – Törlés: egy felugró ablak megkérdezi, hogy biztosan törölni szeretnénk-e.

  – Módosítás: a hozzáadáshoz hasonló űrlap jelenik meg. Az adatok ki vannak töltve az eddig megadottakkal.

A táblázatok oszlopait lásd az *Az adatbázis felépítése* részben.

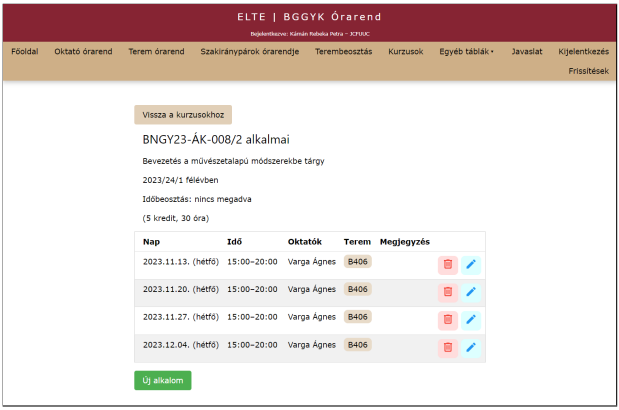
Összetettebb táblázatok

Kurzus Minden sorban a „Törlés” és „Módosítás” gomb mellett van egy „Alkalmak” gomb is, amire kattintva az adott kurzus eddig beosztott alkalmaira ugorhatunk (lásd következő, *Kurzus alkalmai* részben). A kurzusok táblázata látható az 1. ábrán.

Oszlopok:

- Az egyértelmű oszlopokon kívül színt is meg kell adni. A terembeosztásnál ilyen színnel jelenik meg az alkalom. Ez azért fontos, mert így könnyen lehet látni, hogy mikor vannak az egyes szakirányoknak azok az órái, amik nem ütközhetnek.

Kurzus alkalmai Egy adott kurzus adatai alatt az eddig beosztott alkalmai láthatóak. Az alkalmakat ugyanúgy lehet szerkeszteni, törölni és hozzáadni, mint az egyszerű táblázatoknál.



9. ábra. Egy kurzus alkalmai

Terembeosztás táblázat Ez a táblázat eltér az eddigiektől. Nem szerkesztésre szolgál, hanem az eddig beosztott alkalmak megjelenítésére (10. ábra).

Ki kell választani egy dátumot, majd megjelenik az ahhoz a naphoz tartozó táblázat.

Minden sor egy teremhez tartozik, az oszlopok pedig negyedóránként az időt mutatják.

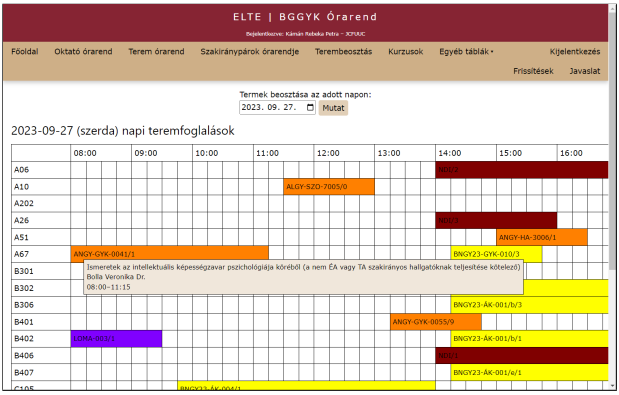
A színes cellákban lévő kurzuskód mutatja, ha a terem foglalt. Ha az ilyen cellákra visszük az egeret, láthatjuk az alkalom részleteit: kurzusok, tárgynév, időpont, oktatók, megjegyzés; rákattintva pedig szerkeszteni lehet. A bal oldali cellákban lévő teremneveknél ugyanilyen módon megjelenik a terem fantázianeve, létszáma és a hozzá fűzött megjegyzés (ami általában felszereltség).

2.5. Mobil nézet

Az oldal reszponzív, így mobilon is használható.

A reszponzivitás a publikus órarendeknél nagyon fontos, hogy gyorsan és egyszerűen hozzáférhetőek legyenek a kívánt adatok (11. ábra).

A szerkesztő táblázatokat valószínűleg nem fogják mobilon használni, ennek ellenére az is megfelelően jelenik meg. Néhány keskenyebb táblázat (pl. szakirány–tárgy, 12a. ábra) teljes szélességben is látszik,



10. ábra. Terembeosztás

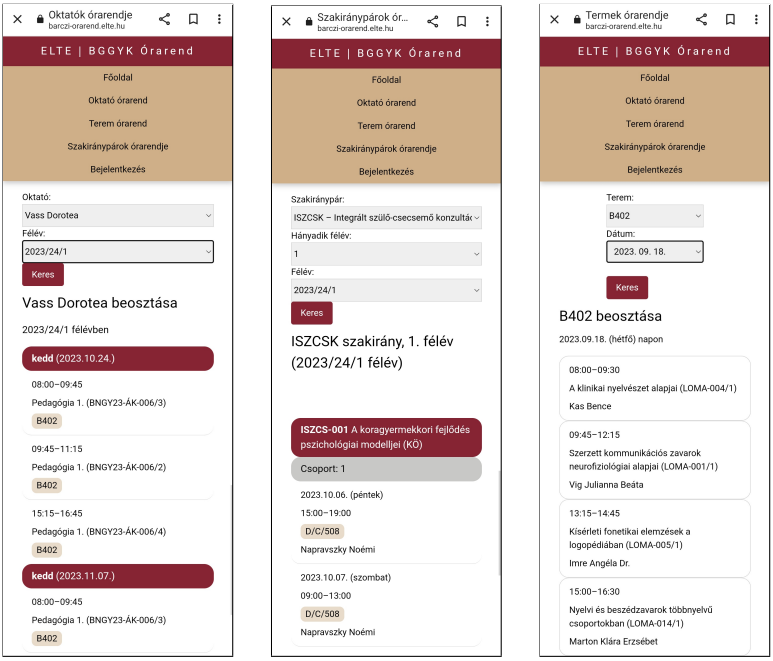
a széles táblázatok (pl. tárgy, 12b. ábra) pedig oldalra görgethetők. A terembeosztásnál a súgó hosszú rányomás hatására jelenik meg (12c. ábra).

2.6. Példa összevont alkalmak hozzáadásához

Cél: egy meglévő tárgyhoz szeretnénk (1) hozzáadni egy kurzust és (2) annak egy alkalmát, majd egy ugyanehhez a tárgyhoz tartozó másik, már meglévő kurzusához (3) rendelünk hozzá egy alkalmat úgy, hogy az közös legyen az elsőként említett kurzus új alkalmával.

1. Rámegyünk a Kurzusok menüpontra. Az „Új felvitel” gombra kattintás után kiválasztjuk a megfelelő tárgyat, és kitöltjük a többi adatot. Amennyiben nincs a választási lehetőségek között a kívánt félév, először elmegyünk a Félévek menüpontra és ott adjuk hozzá. Ha jobb klikkel új oldalon nyitottuk meg, akkor azt bezárva egyből újra a kurzusoknál találjuk magunkat, és az oldal frissítése után folytathatjuk a munkát. Érdemes közben az oktatók órarendjét és a terembeosztást nézni a megfelelő alkalom kiválasztásához. Ha végeztünk, a „Hozzáadás” gombbal elmentjük.

2. Mentés után újra a kurzusoknál vagyunk. Keressük meg az imént hozzáadott kurzust és kattintsunk a sorában a jobb oldali „Al-



(a) Oktatók *(b)* Szakirányok *(c)* Termek

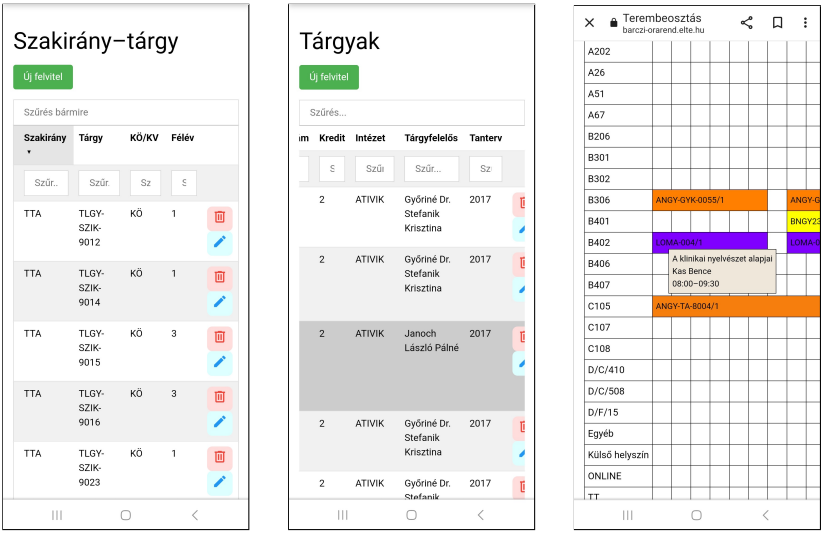
11. ábra. Publikus órarendek mobilon

kalmak” gombra (naptárszerű ikon). Látjuk, hogy még nincsenek alkalmak beosztva, hiszen még csak most hoztuk létre a kurzust. Adjunk hozzá egyet az „Új alkalom” gombbal.

3. Kurzus új alkalmának létrehozásakor nem lehet egyből összevonni más alkalommal. Két lehetőségünk van erre:

   (a) Egy kurzus már létező alkalmát módosítjuk. Ilyenkor, az új létrehozásával ellentétben, több kurzust is választhatunk.

   (b) A Minden alkalom menüpontban választjuk ki a létező alkalmat. Itt új hozzáadása esetén lenne lehetőségünk két kurzushoz is egy-egy új alkalmat hozzáadni, amik össze vannak vonva, amennyiben két kurzust választunk.



(a) Szakirány–tárgy *(b)* Tárgyak *(c)* Terembeosztás

12. ábra. Szerkesztő táblázatok mobilon

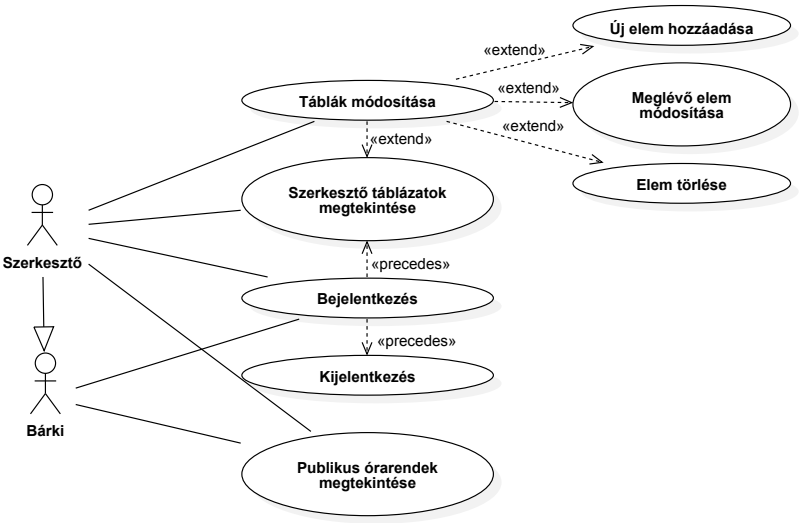
Egyik módnál sincs lehetőség új alkalmat meglévővel összevonni, csak meglévőhöz hozzáadni újat. (Ez később változhat.)

3. Fejlesztői dokumentáció

3.1. Alkalmazott technológiák

A weboldalhoz a következő nyelveket használtam:

- HTML
- CSS, W3CSS
- Javascript
- PHP 7.0
- MySQL 5.5



13. ábra. Felhasználói eset diagram

Keretrendszert nem használtam.

Az adatbázis és a forrásállományok mind az ELTE Caesar szerveren találhatók, ezek eléréséhez PuTTY és WinSCP használható. A `barczi-orarend` egy virtuális webszerverként üzemel, melyhez néhány megadott felhasználó fér hozzá fejlesztés céljából, akik saját felhasználónevükkel tudnak bejelentkezni rá.

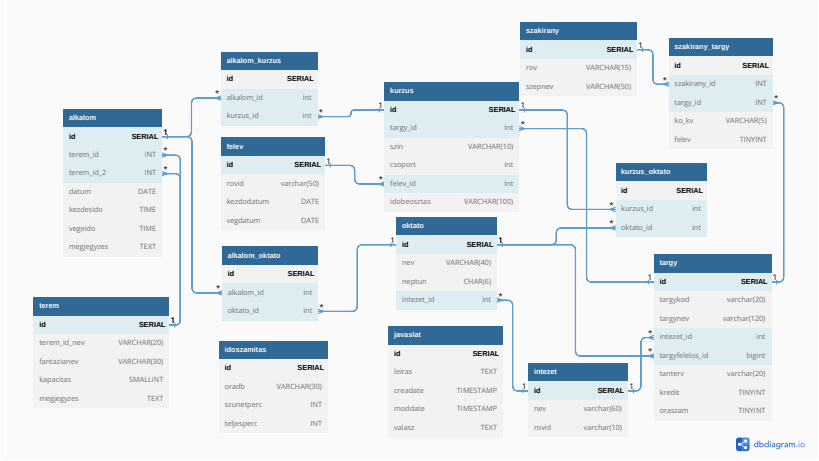
A webes bejelentkezés funkció a Caesar szerver központi bejelentkeztetés használatával készült, így biztonságos, illetve általa hozzáférhetünk a bejelentkezett felhasználó néhány ELTE-hez kapcsolódó adatához (pl. név, felvett kurzusok), és tudunk személyre szabott tartalmat megjeleníteni.

Modell–nézet architektúrát használok. Bár osztályok nincsenek, mégis elkülöníthetőek a kódnak az adatok adatbázisból kinyerésére, illetve a megjelenítésre használt részei. Előbbi jellemzően a PHP fájlok feladata, utóbbi a Javascripté.

3.2. Az adatbázis felépítése

A `dbdiagram.io`-t használtam az adatbázis-kapcsolatok vizuális megjelenítéséhez, ami kódból generál táblázatokat kapcsolatokkal. A DBML leírásokat SQL-ből is lehet generálni a CLI segítségével. Ez npm-mel telepíthető: `npm install -g @dbml/cli`.

Az alábbi ábrán láthatóak az adattáblák és a közöttük lévő kapcsolatok, amik nagyban segítik a rendszer átlátását.



14. ábra. Relációs adatmodell

3.3. Tippek a fejlesztéshez: hogyan működik egy menüpont

A Kurzusok menüpont kódjának részletes bemutatása. Ez alapján új hasonló menüpont is létrehozható.

Minden menüponthoz van egy php fájl a főmappában címmel, stíluslapokkal és scriptekkel. Ennek az oldalnak a betöltésekor lefutnak a scriptek is, amik egyből meg is jelenítenek valamit. A kurzusoknál ez a `kurzus.js`, aminek meghívódik a `kurzus_table_show` metódusa. Ez a `kurzus_table.php`-tól aszinkron módon kéri a megjelenítendő adatokat a `jsutils.js` `json_ajaxpost` függvényével, ami a kapott JSON objektumot átadja a szintén „közös” `table_htmlstr` függvénynek. Ez

egy html-ként értelmezhető stringet ad vissza, amit bele lehet írni a dokumentum megfelelő részébe (`torzs` azonosítójú `div` elem).

Megjelent a táblázat. A sorokban a gombokba generált `onclick` attribútum felhasználja a rekord azonosítóját. Ezekre kattintva a `kurzus.js` egy függvénye (Törlés: `kurzus_torol`, Módosítás: `kurzus_form_show`, Alkalmak: `kurzus_alkalmak`) hívódik meg.

Vegyük a módosítás gombot. A `kurzus_form_show` az előbb említett `json_ajaxpost` segítségével a táblázat helyén megjelenít egy űrlapot, amit kitölt az eddig megadott adatokkal, amiket az adatbázisból nyer ki (amennyiben kapott `id`-t, ellenkező esetben hozzáadásként kezeli az esetet, és üres formot jelenít meg). Mentéskor a `kurzus_insert_or_update` metódus összegyűjti a formba bevitt adatokat, amiket „előellenőrzés" (`kurzus_formell`: minden kötelező adat meg van-e adva) után POST-ol a `kurzus_insert_or_update.php` fájlnak, ami az adatbázisba mentést kezeli, majd visszajelzést küld annak sikerességéről, amit az újonnan megjelenített táblázat felett kiírunk.

A `kurzus_insert_or_update.php` fájl fogadja a POST változókat. Ezeket prepared statementek használatával beilleszti az adatbázisba (hozzáadás esetén `INSERT`, módosítás esetén `UPDATE`). A `kurzus_oktato` kapcsolótáblát is frissíti: kitörli az esetlegesen meglévő rekordokat, amik az adott kurzushoz tartoznak, aztán újakat szúr be (hozzáadásnál és módosításnál egyaránt). Ezt a két lépést egy tranzakción belül hajtja végre a hibák okozta esetleges inkonzisztenciák elkerülése érdekében.

Az Alkalmak gombra kattintva a kurzus alkalmainak oldalára irányít át. Az `id`-t ezesetben nem POST, hanem GET kéréssel adjuk át az URL-ben paraméterként, hogy az oldal újratöltése esetén, illetve az alkalmak hozzáadása vagy módosítása után ugyanide tudjunk visszatérni.

Az alkalmak beosztásánál az ellenőrzés szigorúbb, nem csak azt nézi, hogy minden adat meg van-e adva, hanem hogy nincs-e ütközés. Ezt már nem lehet egy egyszerű Javascript függvénnyel megoldani, mivel adatbázis-hozzáférés is szükséges hozzá. Ennek megoldása az `alkalom` mappában található. (A kurzusok és alkalmak szorosan összefüggnek, így, bár két külön mappára vannak osztva, használhatják egymás fájljait.) Az `alkalom.js` `alkalom_insert_or_update` függvénye nem egy sima Javascript függvényt használ az űrlap ellenőrzésére, hanem fetch API használatával az `alkalom_ellinfo.php`-tól megkér-

dezi, hogy ütközne-e az újonnam létrehozandó/módosítandó alkalom már meglévővel (módosítás esetén önmagán kívül). Ha igen, akkor nem enged menteni, ha nem, akkor elmenti az alkalmat.

Utószó és köszönetnyilvánítás

Mint sokan mások, akik a témabejelentő leadása előtt álltak, nekem is fejtörést okozott a leendő témám kitalálása. Volt egy pár ötletem, de egyik sem tűnt se eléggé jónak, se eléggé az „enyémnek", se olyannak, aminek a megvalósításához eléggé értenék ahhoz, hogy időben be tudjam fejezni majd. Ezért kezdtem már azzal próbálkozni, hogy először témavezetőt keresek, aki talán majd nekem dob egy témát, hogy ez legyen, de így sem jártam nagy sikerrel. Nagyon meglepődtem és megörültem a Bárczi megkeresésének, mert egyből tudtam, hogy érdekel a téma. Igazából ilyesmin is szoktam néha gondolkodni, de úgy voltam vele, hogy biztos már mindenki talált valami szoftvert, amit ilyenre használ, és biztos nem tudok jobbat írni. Tulajdonképpen lehet, hogy a bárczis órarendszerkesztők is találhattak volna ilyet és elkezdhették volna használni (ami talán hosszútávon fenntarthatóbb), de így talán zökkenőmentesebben ment az átmenet, hogy szépen egymás után látták megszületni a funkciókat, nem mindent egyszerre kellett egyből átlátniuk, és hogy a heti konzultációkon alkalmunk volt átbeszélni minden felmerülő kérdést, és elmondhatták a javaslataikat.

Külön öröm, hogy azóta, mióta elkészült, ismertem meg gyógypedagógiát tanuló hallgatót, aki pozitívan nyilatkozott a publikusan elérhető órarendekről, aki szintén azt mondta, hogy az Excelről való váltás sokkal egyszerűbbé és átláthatóbbá teszi a leendő órái megtervezését és a jelenlegiek számon tartását.

Tulajdonképpen mondhatjuk, hogy egy program sosem készül el, csak abbahagyni lehet (mint egy rajzot vagy házimunkát), ezért a leadás utáni nyáron és a mesterképzésemnek legalább az első félévében (az ezt követő folytatást még érthető okokból nem tudom elmesélni, mert még nem történt meg) is folytattam (bár jóval kevesebb időm volt rá), hogy minél több előbb-utóbb nélkülözhetetlen és kényelmi funkció legyen az órarendtervezőben.

Nem mindenki mondhatja el magáról, hogy használják is a szakdolgozatát, nem csak azért csinálta, hogy legyen valami, én meg még

élveztem is közben, még a(z előző, külföldi félévem miatti) túlzsúfolt utolsó félévem ellenére is. Ez azóta is nagy boldogsággal és motivációval tölt el, azonban mindez csak kis részben az én érdemem. Szeretném megköszönni a külső témavezetőmnek, Jakab Zoltán egyetemi docensnek a projekt támogatását, Czinkóczi Krisztina órarendszerkesztőnek a folyamatos tesztelést és visszajelzéseket, továbbá a témavezetőmnek, Lócsi Leventének azt, hogy ilyen lelkesen vállalta a témavezetést, és a projektre fordított rengeteg idejét és türelmét, a sok ötletet és terelgetést.

Formális nyelv = másodrendű algebrai elmélet

Kaposi Ambrus

ELTE Informatikai Kar
Programozási Nyelvek és Fordítóprogramok Tanszék

akaposi@inf.elte.hu

A formális nyelvek közé tartoznak a logikai és a programozási nyelvek. Ebben az írásban példákon keresztül megmutatjuk, hogyan lehet redundáns információ nélkül, magas szinten ilyen nyelveket megadni, és bennük programozni. A következő bontásban dolgozunk: elsőrendű algebrai elméletek (1. fejezet), elsőrendű általánosított algebrai elméletek (2. fejezet), másodrendű algebrai elméletek (3. fejezet), másodrendű általánosított algebrai elméletek (4. fejezet), végül az algebrai elméletek szignatúráit leíró nyelvek (nyelveket leíró nyelvek, 5. fejezet). A szerző honlapján elérhető ennek az írásnak legfrissebb változata és a hozzá tartozó számítógépes formalizálás.

1. Elsőrendű algebrai elméletek

Ebben a fejezetben a félcsoport és a kombinátor-kalkulus példáin keresztül bemutatjuk az algebrai elméleteket. Az algebrai elméletek (algebraic theory, AT) nagyon egyszerű programozási nyelveknek felelnek meg: olyan nyelveknek, melyekben nincsenek változók és típusok.

A legegyszerűbb Turing-teljes programozási nyelv Moses Schönfinkel kombinátor-kalkulusa. Az S és K kombinátorokat tudjuk egymás után írni zárójelezetten (például $(\mathsf{SK})\mathsf{K}$ illetve $\mathsf{S}(\mathsf{KK})$ két különböző kombinátor term), és két átírási szabályunk van: $(\mathsf{K}u)f \rightsquigarrow u$ illetve $((\mathsf{S}f)g)u \rightsquigarrow (fu)(gu)$, ahol u, f és g tetszőleges kombinátor termek.

Hogyan írjuk le ezt a nyelvet teljesen precízen? Először is, mik azok a termek? Tetszőleges sztringek (karaktersorozatok)? Sztringek, melyekben csak S, K, nyitó és záró zárójelek szerepelhetnek? Ezek közül csak azon sztringek, melyek jól zárójelezettek? Bináris fák, melyeknek kétféle levelük lehet? Kétféle levelű bináris fák ekvivalenciaosztályai, melyek a fenti két átírási szabályból képzett ekvivalencia-kongruencia reláció alapján vannak képezve?

Ebben a cikkben a legutolsó változatot fogjuk használni, de kicsit egyszerűbben fogalmazzuk meg ugyanezt: a kombinátor-kalkulus egy algebrai elmélet, melyben egy bináris művelet van (applikáció, egymás mellé írás), két nulláris művelet van (S és K), és két egyenlőség $(\mathsf{K}u)f = u$ és $((\mathsf{S}f)g)u = (fu)(gu)$. Konvenció, hogy az applikáció balra zárójeleződik, tehát a két egyenlőség rövidebben így is írható: $\mathsf{K}uf = u$ és $\mathsf{S}fgu = fu(gu)$. Az ilyen, három művelettel ellátott halmazokat, melyekre teljesül a fenti két egyenlőség, kombinátor-algebráknak nevezzük.

Mielőtt a kombinátor-algebrákat tárgyalnánk, megnézünk egy jól ismert algebrai elméletet.

1. Definíció (félcsoport)**.** *Egy félcsoport a következő komponensekből áll:*

$$\begin{aligned} \mathsf{C} \quad &: \mathsf{Set} \\ - \cdot - &: \mathsf{C} \to \mathsf{C} \to \mathsf{C} \\ \mathsf{ass} \quad &: (x \cdot y) \cdot z = x \cdot (y \cdot z) \end{aligned}$$

Egy félcsoport tehát egy halmaz, rajta egy bináris művelet (mely „curry-zett” módon, $\mathsf{C} \to (\mathsf{C} \to \mathsf{C})$ függvénnyel van megadva $\mathsf{C} \times \mathsf{C} \to \mathsf{C}$ helyett), mely asszociatív. Például félcsoportot alkotnak az egész számok az összeadással ($\mathsf{C} := \mathbb{Z}, x \cdot y := x + y$), hiszen az összeadás asszociatív. További példák: egész számok a szorzással; logikai értékek (kételemű halmaz) az és/vagy/kizáró vagy művelettel; sztringek az összefűzéssel (konkatenáció); $n \times n$-es valós mátrixok a mátrix-szorzással; $A \to A$ függvények a függvénykompozícióval tetszőleges A halmazra; A részhalmazai a metszettel/unióval; az egyelemű halmaz (mi a művelet?); az üres halmaz (mi a művelet?). Nem példa: egész számok a kivonással vagy hatványozással.

A kombinátor-algebrákat is megadjuk ugyanezzel a jelöléssel.

2. Definíció (kombinátor-algebra)**.**

$$
\begin{array}{ll}
\mathsf{Tm} & : \mathsf{Set} \\
- \cdot - & : \mathsf{Tm} \to \mathsf{Tm} \to \mathsf{Tm} \qquad \textit{(balra zárójeleződik)} \\
\mathsf{K} & : \mathsf{Tm} \\
\mathsf{S} & : \mathsf{Tm} \\
\mathsf{K}\beta & : \mathsf{K} \cdot u \cdot f = u \\
\mathsf{S}\beta & : \mathsf{S} \cdot f \cdot g \cdot u = f \cdot u \cdot (g \cdot u)
\end{array}
$$

A kombinátor-algebrákat szokás a kombinátor-kalkulus modelljeinek, a félcsoportokat a félcsoport-elmélet modelljeinek nevezni.

A triviális kombinátor-algebrában Tm az egyelemű halmaz.

3. Állítás. *Nincs kételemű kombinátor-algebra.*

Bizonyítás. K és S segítségével megadhatjuk a proj_1, proj_2, proj_3 termeket, melyekre $\mathsf{proj_i} \cdot u_1 \cdot u_2 \cdot u_3 = u_i$ teljesül. Tegyük fel, hogy a $\{0,1\}$ halmaz kombinátor-algebra. Ebben a modellben skatulya-elve alapján kettő proj_i megegyezik, feltehetjük, hogy mondjuk $\mathsf{proj}_1 = \mathsf{proj}_2$. Ekkor $0 = \mathsf{proj}_1 \cdot 0 \cdot 1 \cdot 1 = \mathsf{proj}_2 \cdot 0 \cdot 1 \cdot 1 = 1$. □

4. Feladat. *Nincs nemtriviális véges kombinátor-algebra (melyben* Tm *véges halmaz).*

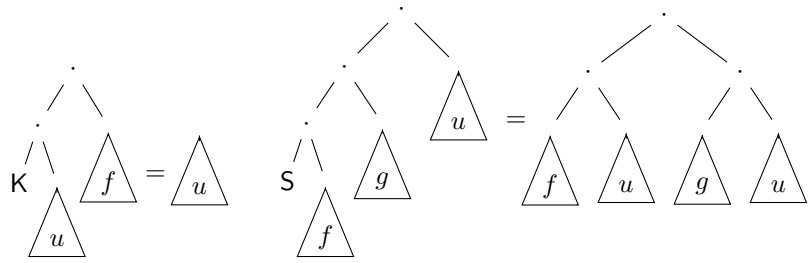
Nemtriviális kombinátor-algebrákat nem olyan könnyű találni, mint félcsoportokat. Ez nem meglepő, hiszen ez egy Turing-teljes nyelv – egy nemtriviális kombinátor-algebrában elkódolható az összes Turing-gép.

5. Gondolkodnivaló. *Van-e olyan kombinátor-algebra, melyben* Tm *a Turing-gépek kódjainak halmaza? Esetleg ennek valamely kvóciens halmaza?*

Van azonban egy módszer, amellyel tetszőleges algebrai elméletnek megadható egy nemtriviális modellje, így a kombinátor-kalkulusnak is: ezt szokás szabad algebrának vagy szintaxisnak nevezni. A kombinátor-kalkulus szintaxisa a következő.

6. Definíció (Kombinátor-kalkulus szintaxisa)**.** *Vesszük a bináris fákat, melyek leveleinél a kételemű halmaz valamely eleme szerepel (ezt a két elemet* K*-val illetve* S*-el fogjuk jelölni), és ezek közül a következő*

fákat megegyezőnek tekintjük (itt a háromszögekkel tetszőleges fákat jelölünk, melyeket elnevezünk u-nak, f-nek illetve g-nek):



A szintaxis „szabadsága” azt fejezi ki, hogy ez egy olyan algebra, mely csak annyit tud, amennyit az algebrai elmélet követel, sem többet (például nincs benne több egyenlőség), sem kevesebbet (hiszen az adott elmélet algebrája). A szintaxis elemeit szabadon generálja a három művelet, és az egyenlőségek szerint veszünk egy kvócienst. Ezt úgy is ki lehet fejezni, hogy a szintaxis elemei valójában ekvivalenciaosztályok: két term egy ekvivalenciaosztályban van, ha a két egyenlőség által generált ekvivalencia-kongruencia reláció szerint relációban állnak egymással. Ha két term a szintaxisban egyenlő, arra úgy gondolhatunk, hogy a két term mint program, ugyanazt a végeredményt adja (vagy ugyanúgy nem adnak végeredményt, ha végtelen sokáig futó programok). Ha valamit meg tudunk csinálni a szintaxisban, azt meg tudjuk csinálni tetszőleges kombinátor-algebrában is: ha egy termet fel tudunk építeni a szintaxisban, azt fel tudjuk építeni tetszőleges algebrában. Ha két term egyenlő a szintaxisban, a megfelelő termek egyenlőek tetszőleges kombinátor-algebrában is. A szintaxisban való munkát nevezzük programozásnak, ezt illusztráljuk most.

7. Feladat (Identitás)**.** *Tetszőleges kombinátor-algebrában megadható az algebra egy* I *eleme, melyre* $\mathsf{I} \cdot u = u$ *minden u-ra. Úgy is fogalmazhattunk volna, hogy a szintaxisban megadható egy* I *term, melyre* $\mathsf{I} \cdot u = u$ *minden u-ra.*

8. Feladat (kompozíció)**.** *A szintaxisban megadható egy* B *term, melyre* $\mathsf{B} \cdot f \cdot g \cdot u = f \cdot (g \cdot u)$ *minden f, g, u-ra.*

9. Feladat (C kombinátor)**.** *A szintaxisban megadható egy* C *term, melyre* $\mathsf{C} \cdot f \cdot u \cdot g = f \cdot g \cdot u$ *minden f, g, u-ra.*

Kombinátor termekkel reprezentálhatók a természetes számok: az

alapötlet, hogy egy n szám egy olyan függvény (term), melyre $n \cdot u \cdot f = f \cdot (f \cdot (\ldots (f \cdot u) \ldots))$, ahol f-et n-szer alkalmazzuk u-ra. A nulla nem más, mint K, a rákövetkező, összeadás, szorzás stb. definiálható.

10. Feladat (Rákövetkező)**.** *A szintaxisban megadható egy* suc *term, melyre* $\mathsf{suc} \cdot n \cdot z \cdot s = s \cdot (n \cdot z \cdot s)$ *minden* n, z, s*-re.*

11. Feladat (Összeadás)**.** *A szintaxisban megadható egy* P *term, melyre* $\mathsf{P} \cdot a \cdot b \cdot z \cdot s = a \cdot (b \cdot z \cdot s) \cdot s$.

12. Feladat (Fixpont kombinátor)**.** *A szintaxisban megadható egy* Y *term, melyre* $\mathsf{Y} \cdot f = f \cdot (\mathsf{Y} \cdot f)$.

Ennek segítségével tetszőleges rekurzív függvény implementálható.

13. Feladat (Fixpont kombinátor használata)**.** *Az összeadás megadható a fixpont kombinátort használva.*

14. Gondolkodnivaló (Extenzionalitás)**.** *Adjuk meg az* 1 *természetes szám két különböző implementációját! Tehát* t_0, t_1 *termeket, melyekre* $t_i \cdot u \cdot f = f \cdot u$, *de* $t_0 \neq t_1$. *Hogyan bizonyítjuk, hogy két term nem egyenlő?*

15. Gondolkodnivaló (Egyenlőségének eldönthetetlensége)**.** *Feltételezve, hogy a metaelméletünk konstruktív (pl. Martin-Löf típuselmélet), a kombinátor-kalkulus szintaxisában az egyenlőség eldönthetetlen. Tehát nem igaz, hogy bármely* t_0, t_1 *termekre* $t_0 = t_1$ *vagy* $t_0 \neq t_1$. *Ezt úgy tudjuk belátni, hogy megadjuk a metaelméletünk egy modelljét, amiben eldönthető (ez könnyű, tetszőleges klasszikus modell megfelel), és megadjuk egy modelljét, melyben ez az állítás hamis (ez már érdekesebb).*

16. Feladat. *Tetszőleges két kombinátor-algebrának képezhető a szorzat algebrája, ahol a termek halmaza a két halmaz Descartes-szorzata (meg kell mutatnunk, hogy a termek Descartes-szorzatán megadható egy kombinátor-algebra).*

17. Feladat. *Mutassuk meg, hogy van olyan kombinátor szintaktikus termek közötti egyenlőség, amely teljesül bizonyos kombinátor-algebrában, de nem a szintaxisban.*

18. Feladat. *Mutassuk meg, hogy ha egy egyenlőség teljesül az összes kombinátor-algebrában, akkor a szintaxisban is!*

19. Jelölés (Függvény-applikáció)**.** *Ha van egy* $f : A \to B$ *függvényünk és hozzá egy* $a : A$ *bemenetünk, a függvény alkalmazását a bemenetre a matematikában legtöbbször* $f(a)$ *módon jelöljük. Ebben a*

jegyzetben a funkcionális programozásban szokásos módon ehelyett f a-t írunk.

(Eddig csak infix függvényeink voltak, például $-\cdot-$, ezért nem kellett az applikációról külön beszélni. Ne tévessze meg az olvasót, hogy most már két különböző applikációról is beszélünk: a metanyelvi (metaelméleti) applikációt egyszerűen egymás mellé írással jelöljük, a tárgynyelvi (kombinátor-kalkulusbeli) applikációt $-\cdot-$-tal.)

Néhány további, jól ismert algebrai elmélet: egységelemes félcsoport (angolul monoid), csoport, gyűrű, tetszőleges test feletti vektortér. Nagyon sok algebrai elmélet van: annyi, ahányféle művelet és egyenlőség adható meg. Egy algebrai elméletet a *szignatúrája*[1] határoz meg: szignatúra az operátorok aritását és az egyenlőségek oldalait tartalmazza. Egy egyszerű definíció ezekre a következő.

20. Definíció (Algebra szignatúra)**.** *A szignatúra egyrészt egy $\nu : \mathbb{N} \to \mathbb{N}$ függvényből áll, ahol $\nu\, n$ megadja, hogy hány darab n-paraméteres műveletünk van. Másrészt tekintsük az alábbi szintaxist, amelynek segítségével az egyenlőségek két oldalát fogjuk elkódolni.*

$$\begin{aligned}
&\mathsf{tm} : \mathsf{Set}\\
&\mathsf{var} : \mathbb{N} \to \mathsf{tm}\\
&\mathsf{op} \ : (n : \mathbb{N}) \to \overline{\nu\, n} \to \underbrace{\mathsf{tm} \to \mathsf{tm} \to \ldots \to}_{n \ \mathit{darab}} \mathsf{tm}
\end{aligned}$$

Itt az $\overline{i}$ az i-elemű halmazt jelöli. Ez egy olyan algebrai elmélet, melyben nincsenek egyenlőségek. Végtelen sok nulláris operátor van (minden természetes számhoz egy), ezek felelnek meg a változóknak, melyek az egyenlőségekben szerepelnek. $\nu\, n$ darab n-áris műveletünk van. A ν függvényen túl a szignatúra tm*-párok egy halmazát tartalmazza.*

Például félcsoport szignatúrájában $\nu\, 2 = 1$ és minden $n \neq 2$-re $\nu\, n = 0$; a kombinátor-kalkulus szignatúrájában $\nu\, 0 = 2$, $\nu\, 2 = 1$, és minden egyéb n-re $\nu\, n = 0$. A félcsoport szignatúrájában egy egyenlőség van, melyet a következő pár határoz meg (0_1-el jelöljük az egyelemű

[1] A szignatúrába beleértjük az egyenlőségeket is, mert az általánosított algebrai elméleteknél (2. fejezet) a műveletek és az egyenlőségek nem választhatók el: lesznek olyan műveleteink, melyek aritása csak bizonyos egyenlőségek megléte miatt értelmes.

halmaz egyetlen elemét):

$$\Big(\mathsf{op}\,2\,0_1\,\big(\mathsf{op}\,2\,0_1\,(\mathsf{var}\,0)\,(\mathsf{var}\,1)\big)\,(\mathsf{var}\,2),$$
$$\mathsf{op}\,2\,0_1\,(\mathsf{var}\,0)\,\big(\mathsf{op}\,2\,0_1\,(\mathsf{var}\,1)\,(\mathsf{var}\,2)\big)\Big)$$

A kombinátor-kalkulus szignatúrájában két egyenlőség van, a $\mathsf{K}\beta$ egyenlőséget az alábbi pár kódolja (a kételemű halmaz elemeit 0_2-vel és 1_2-vel jelöljük):

$$\Big(\mathsf{op}\,2\,0_1\,\big(\mathsf{op}\,2\,0_1\,(\mathsf{op}\,0\,0_2)\,(\mathsf{var}\,0)\big)\,(\mathsf{var}\,1), \mathsf{var}\,0\Big)$$

Az $\mathsf{S}\beta$ egyenlőséget az alábbi pár kódolja:

$$\Big(\mathsf{op}\,2\,0_1\,\big(\mathsf{op}\,2\,0_1\,(\mathsf{op}\,2\,0_1\,(\mathsf{op}\,0\,1_2)\,(\mathsf{var}\,0))\,(\mathsf{var}\,1)\big)\,(\mathsf{var}\,2),$$
$$\mathsf{op}\,2\,0_1\,\big(\mathsf{op}\,2\,0_1\,(\mathsf{var}\,0)\,(\mathsf{var}\,2)\big)\,\big(\mathsf{op}\,2\,0_1\,(\mathsf{var}\,1)\,(\mathsf{var}\,2)\big)\Big)$$

21. Feladat. *Kódoljuk el a csoport szignatúráját a fenti módon!*

22. Feladat. *Adjunk meg olyan algebrai elméletet, amelynek csak triviális algebrája van!*

23. Feladat. *Adjuk meg az algebra-fogalmat tetszőleges szignatúrára!*

24. Feladat. *Mutassuk meg, hogy minden algebrai elméletnek van triviális algebrája!*

25. Feladat. *Mutassuk meg, hogy minden algebrai elméletben tetszőleges két algebrának van szorzat-algebrája!*

26. Megjegyzés. *Az 5. fejezetben megadjuk egy szebb leírását az algebrai elméleteknek.*

Az algebra homomorfizmus egy függvény a két algebra alaphalmaza között, mely megtartja a műveleteket. Például A és B félcsoportok között ez egy $f : \mathsf{C}_A \to \mathsf{C}_B$ függvény, melyre teljesül, hogy $f\,(x \cdot_A y) = f\,x \cdot_B f\,y$ minden x, y-ra. Hasonlóképp, A és B kombinátor-algebrákra ez egy $f : \mathsf{Tm}_A \to \mathsf{Tm}_B$ függvény, melyre $f\,(x \cdot_A y) = f\,x \cdot_B f\,y$, és $f\,\mathsf{K}_A = \mathsf{K}_B$ és $f\,\mathsf{S}_A = \mathsf{S}_B$.

27. Gondolkodnivaló. *Minden algebrai elmélethez megadható a homomorfizmus fogalom. A homomorfizmusok komponálhatók, és az identitás függvény mindig homomorfizmus.*

28. Definíció (Szintaxis)**.** *A szintaxis az iniciális algebra: ez azt jelenti, hogy minden más algebrába pontosan egy homomorfizmus megy a szintaxisból.*

29. Feladat. *Mutassuk meg, hogy a szintaxis egyértelmű: bármely két iniciális algebra között van egy izomorfizmus (oda-vissza homomorfizmusok, melyek kompozíciói identitások).*

30. Feladat. *Mutassuk meg, hogy a félcsoportok szintaxisa az üres halmaz!*

31. Feladat. *Mutassuk meg, hogy a 6. definícióban megadott algebra valóban a kombinátor-kalkulus szintaxisa!*

A szintaxisból egy adott algebrába menő homomorfizmust kiértékelésnek (interpretációnak) nevezzük.

32. Gondolkodnivaló (Üzemanyag szemantika)**.**
A kombinátor-normálformák ilyen alakú termek: K, $\mathsf{K}{\cdot}n$, S, $\mathsf{S}{\cdot}n$, $\mathsf{S}{\cdot}n{\cdot}n'$, *ahol n és n' normálformák. Meg szeretnénk adni egy függvényt, ami szintaktikus kombinátor-termekről velük egyenlő normálformákra képez. Ez a függvény parciális lesz, hiszen például a fixpont-kombinátor nem egyenlő semmilyen normálformával. (Ezt hogyan bizonyítjuk?) Másrészt az nem eldönthető, hogy egy termnek van-e normálformája (Rice tétele). Emiatt egy olyan szemantikát tudunk megadni, ahol üzemanyag (egy természetes szám) mennyiségű lépést hajthatunk létre, és azt adjuk meg, hogy ennyi lépésben megkapjuk-e a normálformát. Minden, a szintaxison megadott függvénynek meg kell tartania az egyenlőségeket, ezért a végeredményt faktorizálni kell a teljes relációval.*

33. Gondolkodnivaló. *Mutassuk meg, hogy a fixpont-kombinátornak nincs normálformája!*

34. Gondolkodnivaló. *Minden algebrai elméletnek van szintaxisa.*

Az induktívan megadott halmazok egyenlőségek nélküli algebrai elméletek szintaxisai: például a Peano természetes számokat az alábbi algebrai elmélet adja meg: $\mathsf{N} : \mathsf{Set}, \mathsf{zero} : \mathsf{N}, \mathsf{suc} : \mathsf{N} \to \mathsf{N}$.

Egy adott algebrai elmélet többféleképpen is prezentálható. Például a kombinátor-kalkulushoz hozzávehetjük az I műveletet (lásd a 7. feladatot), és ezzel egy vele izomorf algebrai elméletet kapunk, melynek modelljei egy az egyben megfeleltethetők a kombinátor-algebráknak. Vagy a félcsoportokhoz hozzávehetünk egy ternáris műveletet és egy egyenlőséget, mely kimondja, hogy a ternáris művelet egyenlő a bináris

művelet kétszeri alkalmazásával. Általánosságban nehéz kérdés, hogy van-e egyszerűbb prezentációja egy adott algebrai elméletnek, és néha nem világos, hogy melyik a „jobb” prezentáció.

A testek nem adhatók meg közvetlenül a szokásos módon algebrai elméletként, mert az osztás csak a nem-nulla elemekre működik, de a fenti definíció szerint minden műveletnek működnie kell a teljes alaphalmazra. Ez azonban még nem jelenti azt, hogy nincs valamilyen okosabb, izomorf megadása a testeknek. Ilyen azonban nincs.

35. Gondolkodnivaló. *A testek nem adhatók meg algebrai elméletként, mert tetszőleges algebrai elméletre vannak szorzat-algebrák, és van 2-elemű test és van 3-elemű test, de nincsen 6-elemű test.*

2. Elsőrendű általánosított algebrai elméletek

Ebben a fejezetben megadjuk a típusos kombinátor-kalkulust. Ennek a természetes prezentációja már nem egyszerű algebrai elmélet, hanem általánosított algebrai elmélet (generalised algebraic theory, GAT).

Egy általánosított algebrai elméletnek az algebrájában nem csak egy alaphalmaz, hanem az alaphalmaz feletti indexelt család is van. Egy típusos kombinátor-algebrában van a típusoknak egy alaphalmaza, és minden típushoz tartozik egy külön halmaz, mely az olyan típusú termek halmaza. A típusokat dokumentációnak is gondolhatjuk: a kombinátor-kalkulus ezen leírása intuitívabb, hiszen a típusok elmondják, hogy valójában mit csinál a K és az S kombinátor.

36. Definíció (Típusos kombinátor-algebra)**.**

$$
\begin{array}{ll}
\mathsf{Ty} & : \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} \\
\iota & : \mathsf{Ty} \\
- \Rightarrow - & : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} \\
- \cdot - & : \mathsf{Tm}\,(A \Rightarrow B) \to \mathsf{Tm}\,A \to \mathsf{Tm}\,B \qquad \textit{(balra zárójeleződik)} \\
\mathsf{K} & : \mathsf{Tm}\,(A \Rightarrow B \Rightarrow A) \\
\mathsf{S} & : \mathsf{Tm}\,((A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow A \Rightarrow C) \\
\mathsf{K}\beta & : \mathsf{K} \cdot u \cdot f = u
\end{array}
$$

$$\mathsf{S}\beta \qquad : \mathsf{S} \cdot f \cdot g \cdot u = f \cdot u \cdot (g \cdot u)$$

A $-\cdot-$ applikáció műveletnek van két implicit paramétere, ezek az A és a B. Az applikáció explicit leírása a következő lenne:

$$(A : \mathsf{Ty}) \to (B : \mathsf{Ty}) \to \mathsf{Tm}\,(A \Rightarrow B) \to \mathsf{Tm}\,A \to \mathsf{Tm}\,B$$

Nem akarjuk mindig kiírni az A-t és a B-t, ezért ezeket implicit módon mindig odaértjük. Az értékük általában kikövetkeztethető a másik két paraméterből. Hasonlóan, a K *műveletnek két, az* S *műveletnek három implicit* Ty*-paramétere van. Az egyenlőségeknél is odaértjük az implicit* Ty*/*Tm*-paramétereket.*

Az ilyen, általánosított algebrai elméletekre is igaz minden, amit a közönséges algebrai elméletekről elmondtunk:

- Megadható a szignatúrájuk (lásd a 4. fejezetben).
- Minden szignatúrához megadhatók a következő fogalmak: algebra/modell, homomorfizmus.
- Minden általánosított algebrai elméletnek van szintaxisa.
- Algebráknak van Descartes-szorzata, van triviális algebra.

37. Feladat. *Mutassuk meg, hogy minden típus nélküli kombinátor-algebra megad egy típusos kombinátor-algebrát! A másik irány miért nem teljesül?*

38. Gondolkodnivaló. *Mutassuk meg, hogy minden típusos kombinátor termnek van normálformája! (A normálformák ugyanazok, mint a típus nélküli esetben; Tait logikai predikátumok módszerét kell használni a bizonyításra.)*

Megadunk néhány további általánosított algebrai elméletet.

39. Definíció (Gráf)**.**

$$\mathsf{V} : \mathsf{Set}$$
$$\mathsf{E} : \mathsf{V} \to \mathsf{V} \to \mathsf{Set}$$

A gráfban csúcsok (vertex) és élek (edge) vannak.

40. Definíció (Előrendezés (preorder))**.**

$$\begin{aligned}
&\mathsf{Ob} && : \mathsf{Set}\\
&\mathsf{Mor} && : \mathsf{Ob} \to \mathsf{Ob} \to \mathsf{Set}\\
&- \circ - && : \mathsf{Mor}\ J\ I \to \mathsf{Mor}\ K\ J \to \mathsf{Mor}\ K\ I\\
&\mathsf{id} && : \mathsf{Mor}\ I\ I\\
&\mathsf{irr} && : (f\ g : \mathsf{Mor}\ J\ I) \to f = g
\end{aligned}$$

A preorder egy tranzitív és reflexív egyszerű gráf.

41. Definíció (Kategória)**.**

$$\begin{aligned}
&\mathsf{Ob} && : \mathsf{Set}\\
&\mathsf{Mor} && : \mathsf{Ob} \to \mathsf{Ob} \to \mathsf{Set}\\
&- \circ - && : \mathsf{Mor}\ J\ I \to \mathsf{Mor}\ K\ J \to \mathsf{Mor}\ K\ I\\
&\mathsf{id} && : \mathsf{Mor}\ I\ I\\
&\mathsf{ass} && : (f \circ g) \circ h = f \circ (g \circ h)\\
&\mathsf{idl} && : \mathsf{id} \circ f = f\\
&\mathsf{idr} && : f \circ \mathsf{id} = f
\end{aligned}$$

A kategória egy tranzitív és reflexív gráf, melyben az élek kompozíciója asszociatív, és a reflexív élekkel való kompozíció az identitás.

42. Feladat. *Minden egységelemes félcsoport megad egy kategóriát (ahol egy darab objektum van).*

43. Feladat. *Adjuk meg a gráf-, preorder- és kategória-homomorfizmus fogalmát! Utóbbit funktornak szokás nevezni.*

44. Feladat. *Adjuk meg a halmazok kategóriáját, melyben az objektumok halmazok, a morfizmusok függvények az adott halmazok között.*

45. Megjegyzés. *Az egyenlőségek nélküli általánosított algebrai elméletek induktív-induktív típusoknak felelnek meg. Ha egyenlőségeket is tartalmaznak, ezeket kvóciens induktív-induktív típusoknak nevezzük [10]. Az általánosított algebrai elméleteket először Cartmell írta le [5].*

3. Másodrendű algebrai elméletek

Ebben a fejezetben másodrendű algebrai elméleteket (second-order algebraic theory, SOAT) vizsgálunk.

A legtöbb nyelvben vannak változók, vagy más néven kötéssel rendelkező operátorok. Például az $\int_a^b \frac{1}{x^2}\,\mathrm{d}x$ kifejezésben az integrálás $\int - \mathrm{d}-$ operátor köti az x változót az $\frac{1}{x^2}$ kifejezésben, ez utóbbit nevezzük a kötés hatáskörének. Ez a kifejezés megegyezik az $\int_a^b \frac{1}{y^2}\,\mathrm{d}y$ kifejezéssel, hiszen a kötött változó neve nem számít, az csak egy mutató a kötés helyére. Hasonlóan, a $\sum_{x=1}^{10} x^2$ kifejezésben a $\sum$ a kötő operátor, a kötött változó az x, a kötés hatásköre az x^2 kifejezés. Az `int f(int i) { return(i+1); }` kifejezésben a függvénydefiníció a kötő operátor, a kötött név az `i`, a kötés hatásköre a kapcsos zárójelek (`{` és `}`) közötti rész.

46. Jelölés (Névnélküli függvény)**.** *A „plusz egy” $\mathbb{N} \to \mathbb{N}$ függvényt megadhatjuk az alábbi két, ekvivalens módon:* $\mathsf{pluszegy}\, n := n+1$ *illetve* $\lambda n.n+1$*. Utóbbinak előnye, hogy nem kell nevet adnunk a függvénynek.*

Az integrálás operátor például a következő halmazban van (ahol $a, b : \mathbb{R}$): $\int_a^b : (\mathbb{R} \to \mathbb{R}) \to \mathbb{R}$. A fenti példa kifejezést az alábbi módon írhatjuk: $\int_a^b (\lambda x.\frac{1}{x^2})$, erre a szokásos jelölés a fenti, ahol $\lambda x.$ helyett a kifejezés végére írunk $\mathrm{d}x$-et. Hasonlóan, az összegzés operátor precízen például mint $\sum : \mathbb{N} \to \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ adható meg, és a fenti kifejezés $\sum\, 1\, 10\, (\lambda x.x^2)$-et jelent.

A legegyszerűbb, kötésekkel rendelkező nyelv a lambda-kalkulus, amely a típus nélküli kombinátor-kalkulushoz hasonlóan Turing-teljes. Egy kötő operátor van, a $\mathsf{lam} : (\mathsf{Tm} \to \mathsf{Tm}) \to \mathsf{Tm}$, például az identitás függvényt a következőképp adjuk meg: $\mathsf{lam}\,(\lambda x.x)$. Ne tévessze meg az olvasót a két különböző „lambda”: a λ a metanyelvi (metaelméleti) lambda, hiszen a metanyelvünkben is tudunk függvényeket megadni, míg a lam a tárgynyelvi lambda.

47. Definíció (Lambda-kalkulus)**.** *Egy másodrendű lambda-kalkulus modell (algebra) a következő komponensekből áll.*

$$\begin{aligned} \mathsf{Tm} &: \mathsf{Set} \\ - \cdot - &: \mathsf{Tm} \to \mathsf{Tm} \to \mathsf{Tm} \\ \mathsf{lam} &: (\mathsf{Tm} \to \mathsf{Tm}) \to \mathsf{Tm} \end{aligned}$$

$$\beta \qquad : \mathsf{lam}\, f \cdot u = f\, u$$

A lambda-kalkulus hasonló a kombinátor-kalkulushoz, de a K és S helyett csak egy operátorunk van, a lam, és ennek megfelelően csak egy számítási szabályuk van. A lam azonban egy másodrendű operátor, hiszen a bemenete egy függvény. Ilyen másodrendű operátorok nem szerepelhetnek algebrai elméletekben, a 20. definícióval nem fejezhetők ki. A számítási szabály azt fejezi ki, hogy egy lam-mal megadott függvény tárgynyelvi applikációja ugyanaz, mint a metanyelvi applikáció.

Egy másodrendű lambda-kalkulus modellből készíthető egy kombinátor-algebra a következőképp.

$$\begin{aligned}
\mathsf{Tm} &:= \mathsf{Tm} \\
f \cdot u &:= f \cdot u \\
\mathsf{K} &:= \mathsf{lam}\left(\lambda u.\mathsf{lam}\left(\lambda f.u\right)\right) \\
\mathsf{S} &:= \mathsf{lam}\left(\lambda f.\mathsf{lam}\left(\lambda g.\mathsf{lam}\left(\lambda u.f \cdot u \cdot (g \cdot u)\right)\right)\right)
\end{aligned}$$

Az egyenlőségek is teljesülnek:

$$\begin{aligned}
&\mathsf{K}\beta : \mathsf{K} \cdot u \cdot f && = (\mathsf{K}\text{ definíciója}) \\
&\quad \left(\left(\mathsf{lam}\left(\lambda u.\mathsf{lam}\left(\lambda f.u\right)\right)\right) \cdot u\right) \cdot f && = (\beta) \\
&\quad \left(\left(\lambda u.\mathsf{lam}\left(\lambda f.u\right)\right) u\right) \cdot f && = (\text{metanyelvi függvényalkalmazás}) \\
&\quad \left(\mathsf{lam}\left(\lambda f.u\right)\right) \cdot f && = (\beta) \\
&\quad (\lambda f.u)\, f && = (\text{metanyelvi függvényalkalmazás}) \\
&\quad u
\end{aligned}$$

48. Feladat. *Mutassuk meg, hogy a fent megadott modellben az* $\mathsf{S}\beta$ *szabály is teljesül!*

49. Jelölés (Levezetési szabályok)**.** *A lambda-kalkulus operátorainak a megadására egy alternatív jelölés levezetési szabályokkal történik. Az elsőrendű függvénytér konklúziója helyett vízszintes vonalat írunk, a bemeneteket összegyűjtük (uncurry-zzük), a másodrendű függvénytér helyett* $\vdash$ *(turnstyle) szimbólumot.*

$$\frac{f : \mathsf{Tm} \qquad u : \mathsf{Tm}}{f \cdot u : \mathsf{Tm}} \qquad \frac{x : \mathsf{Tm} \vdash f : \mathsf{Tm}}{\mathsf{lam}\, x.f : \mathsf{Tm}} \qquad \frac{x : \mathsf{Tm} \vdash f : \mathsf{Tm} \qquad u : \mathsf{Tm}}{\beta : (\mathsf{lam}\, x.\, f) \cdot u = f[x \mapsto u]}$$

A vízszintes vonal feletti komponensek a bemenetek, az alatta levő rész a kimenet. Az első szabályt például a következőképp olvassuk: ha f és u termek, akkor $f \cdot u$ is term. A levezetési fás jelölésben a másodrendű függvénytér ($\vdash$) használata esetén nevet adunk a másodrendű paramétereknek, fent erre x-et használtunk. A választott név megjelenik a konklúzióban is. A metanyelvi applikációt $f\,u$ helyett $f[x \mapsto u]$-val jelöljük, amit úgy olvasunk, hogy f-ben az x értékét u-val helyettesítjük. A nevek nem számítanak, hiszen az $x : \mathsf{Tm} \vdash f : \mathsf{Tm}$ ugyanazt jelenti, mint hogy $f : \mathsf{Tm} \to \mathsf{Tm}$. A β szabály konklúzióját írhattuk volna például úgy is, hogy $\beta : \mathsf{lam}\, y.\, f \cdot u = f[y \mapsto u]$, a második szabály helyett írhattunk volna az alábbit is:

$$\frac{y : \mathsf{Tm} \vdash f : \mathsf{Tm}}{\mathsf{lam}\, y.f : \mathsf{Tm}}$$

Ha csak egy szortunk van, akkor a sok $: \mathsf{Tm}$-et nem szoktuk kiírni, hanem az alábbi, tömörebb módon is megadhatjuk a szabályokat.

$$\frac{f \qquad u}{f \cdot u} \qquad \frac{x \vdash f}{\mathsf{lam}\, x.f} \qquad \frac{x \vdash f \qquad u}{\beta : (\mathsf{lam}\, x.f) \cdot u = f[x \mapsto u]}$$

A levezetési szabályos jelölés egyik előnye, hogy nem kell λ-kat írni a kötő operátorok számítási szabályainál, hátránya, hogy minden kötésnél meg kell adni egy nevet. Egy másik előny, hogy kikényszeríti a másodrendűséget, vagyis harmadrendű műveletek (például $((\mathsf{Tm} \to \mathsf{Tm}) \to \mathsf{Tm}) \to \mathsf{Tm}$) nem adhatók meg, mert a $\vdash$ bal oldalán nem szerepelhet függvény, vízszintes vonal vagy $\vdash$.

A levezetési szabályok között nincsenek változókra vonatkozóak, mint például az alábbi.

$$\frac{}{x : \mathsf{Tm} \vdash x : \mathsf{Tm}}$$

Ez a szabály a másodrendű algebrai leírásban egyszerűen a $\mathsf{Tm} \to \mathsf{Tm}$ identitás függvény lenne. Az ilyen és ehhez hasonló szabályokat implicit módon beleértjük a levezetési szabályos leírásba, hiszen az ilyen szabályok formális változata a másodrendű függvényekkel megadott. A levezetési szabályos változatot azért használjuk, mert a programozási nyelvek és bizonyításelméleti nyelvek ilyen jelölést szoktak használni.

Másodrendű algebrai elméleteknél nincs értelmes homomorfizmusfogalom, például hogyan mondanánk ki azt, hogy egy $f : \mathsf{Tm}_A \to \mathsf{Tm}_B$ függvény megőrzi a lam operátort? Azt szeretnénk, hogy $f\,(\mathsf{lam}_A\, g) =$

$\mathsf{lam}_B\,(f \circ g \circ ?)$, de nem tudunk a ? helyére mit írni, hiszen $\mathsf{Tm}_B \to \mathsf{Tm}_A$-ban nincs függvényünk ($\circ$ a szokásos függvénykompozíció). Ha f izomorfizmus, tehát egy olyan homomofizmus, melynek van inverze, akkor ez működik, de az egy nagyon megszorított helyzet, ha csak izomorfizmusaink vannak. Ugyanez a probléma a szintaxis megadásánál. Emiatt a következő módszer [4] működik a homomorfizmus-fogalomra, illetve a szintaxisra: a másodrendű elméletet lefordítjuk elsőrendűre, és ott tekintjük ezeket a fogalmakat. A fordítás alapötlete, hogy bevezetjük a környezetek és a helyettesítések szortját, és minden operátort és egyenlőséget ezekkel is indexelünk. A környezetek gyűjtik össze a másodrendű paramétereket. Ha van egy másodrendű modellünk, abból mindig képezhető egy modellje a megfelelő elsőrendű elméletnek. Ennél részletesebben ezt itt nem fejtjük ki, az érdeklődők utánanézhetnek [4, 14].

50. Megjegyzés. *A legtöbb programozási nyelvekről szóló tankönyv nem másodrendű nyelvként, hanem elsőrendű nyelvként adja meg a nyelveket, tehát lényegileg közvetlenül a másodrendű → elsőrendű fordítás eredményét adják meg. Sőt, legtöbbször még ennél is alacsonyabb szinten, típusozatlan pretermekként adják meg a szintaxist – nem világos azonban, hogy az általunk használt magas szinten tárgyalható-e minden fontos tulajdonsága a programozási nyelveknek. A következő tankönyveket javasoljuk az érdeklődőknek [8, 11–13, 15]. Bizonyos nyelvekben a változók használatára korlátozások vannak: például egy változó maximum egyszer vagy pontosan egyszer használható fel: ezeket szubstrukturális nyelveknek nevezzük. Ilyenek például a lineáris logika [6], lineáris típusrendszer [2], modális típuselméletek [7]. Ilyen nyelveket nem tárgyalunk ebben a jegyzetben, a másodrendű algebrai leíráshoz további kiegészítésekre lenne szükség.*

A nyelv fogalma illetve a szintaxis használata (a szintaxisban való programozás) azonban teljesen jól működik másodrendben is, ezt fogjuk itt illusztrálni. A programozás úgy működik, hogy feltételezünk egy másorendű modellt, és ennek a komponenseit használjuk.

Például (feltételezve egy tetszőleges másodrendű lambda-kalkulus modellt, és annak komponenseit „levezetési szabályok” jelöléssel használva) az Y-kombinátor a következőképp adható meg.

$$\mathsf{Y} := \mathsf{lam}\, f.\big(\mathsf{lam}\, x.f \cdot (x \cdot x)\big) \cdot \big(\mathsf{lam}\, x.f \cdot (x \cdot x)\big)$$

Megmutatjuk, hogy ez egy fixpont-kombinátor:

$$\begin{aligned}
&\mathsf{Y}\cdot f && = (\mathsf{Y}\text{ definíciója})\\
&\Big(\mathsf{lam}\,f.\big(\mathsf{lam}\,x.f\cdot(x\cdot x)\big)\cdot\big(\mathsf{lam}\,x.f\cdot(x\cdot x)\big)\Big)\cdot f && = (\beta)\\
&\big(\mathsf{lam}\,x.f\cdot(x\cdot x)\big)\cdot\big(\mathsf{lam}\,x.f\cdot(x\cdot x)\big) && = (\beta)\\
&f\cdot\Big(\big(\mathsf{lam}\,x.f\cdot(x\cdot x)\big)\cdot\big(\mathsf{lam}\,x.f\cdot(x\cdot x)\big)\Big) && = (\mathsf{Y}\text{ definíciója})\\
&f\cdot(\mathsf{Y}\cdot f)
\end{aligned}$$

Feltételezve, hogy vannak $\mathsf{isZero} : \mathsf{Tm}\to\mathsf{Tm}$, kivonás $--- : \mathsf{Tm}\to\mathsf{Tm}\to\mathsf{Tm}$, logikai elágazás $\mathsf{if}{-}\mathsf{then}{-}\mathsf{else}{-}$ műveleteink és számliteráljaink (ezek mind definiálhatók, csakúgy mint a típus nélküli kombinátorkalkulus esetén), szeretnénk megadni egy fac termet, mely a faktoriális függvényt implementálja, vagyis az alábbi egyenlőség igaz rá:

$$\mathsf{fac}\cdot n = \mathsf{if}\,\mathsf{isZero}\,n\,\mathsf{then}\,1\,\mathsf{else}\,\mathsf{fac}\cdot(n-1)$$

Először is megadunk egy f termet, amely a rekurzív hívás struktúráját adja meg.

$$\mathsf{f} := \mathsf{lam}\,r.\mathsf{lam}\,n.\mathsf{if}\,\mathsf{isZero}\,n\,\mathsf{then}\,1\,\mathsf{else}\,r\cdot(n-1)$$

Ennek segítségével a faktoriális függvény a következő.

$$\mathsf{fac} := \mathsf{Y}\cdot\mathsf{f}$$

És ez a definíció teljesíti a fenti egyenlőséget:

$$\begin{aligned}
&\mathsf{fac}\cdot n && = (\mathsf{fac}\text{ definíciója})\\
&\mathsf{Y}\cdot\mathsf{f}\cdot n && = (\mathsf{Y}\text{ fixpont-kombinátor})\\
&\mathsf{f}\cdot(\mathsf{Y}\cdot\mathsf{f})\cdot n && = (\mathsf{f}\text{ definíciója})\\
&\big(\mathsf{lam}\,r.\mathsf{lam}\,n.\mathsf{if}\,\mathsf{isZero}\,n\,\mathsf{then}\,1\,\mathsf{else}\,r\cdot(n-1)\big)\cdot(\mathsf{Y}\cdot\mathsf{f})\cdot n\\
&\qquad && = (\beta\text{ kétszer})\\
&\mathsf{if}\,\mathsf{isZero}\,n\,\mathsf{then}\,1\,\mathsf{else}\,\mathsf{Y}\cdot\mathsf{f}\cdot(n-1) && = (\mathsf{fac}\text{ definíciója})\\
&\mathsf{if}\,\mathsf{isZero}\,n\,\mathsf{then}\,1\,\mathsf{else}\,\mathsf{fac}\cdot(n-1)
\end{aligned}$$

51. Megjegyzés. *A lambda-kalkulus (elsőrendű) szintaxisa nem teljesen izomorf a kombinátor-kalkulus szintaxisával, de ha mindkettőhöz hozzáveszünk néhány újabb egyenlőséget, akkor már izomorfak lesznek [1].*

4. Másodrendű általánosított algebrai elméletek

A legtöbb nyelv tartalmaz változókat, ezért a megadásához másodrendű algebrai elméletre van szükségünk. A legtöbb nyelvben van valamilyen típus-fogalom, ezért általánosított algebrai elméletre van szükségünk. Ebben a fejezetben ezt a két általánosítást kombinálva leírunk néhány alapvető logikai és programozási nyelvet másodrendű általánosított algebrai elméletként (second-order generalised algebraic theory, SOGAT).

Az alábbiakban megadjuk a minimális elsőrendű intuicionista logika nyelvét természetes levezetéses bizonyításelmélettel. A „minimális" azt jelenti, hogy csak implikációnk, univerzális kvantorunk van, és egy relációnk, az egyenlőség. Az egyszerűség kedvéért a bizonyítások szerkezetével nem foglalkozunk, ezt az az egyenlőség fejezi ki, amely azt mondja, hogy tetszőleges két bizonyítás egyenlő. Emiatt a bizonyításokra vonatkozó hipotéziseknek nem adunk nevet, tehát nem írunk kettőspontot és elé nem írunk nevet (sem a $\vdash$ előtt, sem a feltételekben és a konklúziókban).

52. Definíció (Minimális intuicionista logika)**.**

$$\frac{}{\mathsf{For} : \mathsf{Set}} \qquad \frac{}{\mathsf{Tm} : \mathsf{Set}} \qquad \frac{A : \mathsf{For} \qquad B : \mathsf{For}}{A \supset B : \mathsf{For}} \qquad \frac{x : \mathsf{Tm} \vdash A : \mathsf{For}}{\forall x.A : \mathsf{For}}$$

$$\frac{t : \mathsf{Tm} \qquad t' : \mathsf{Tm}}{\mathsf{Eq}\, t\, t' : \mathsf{For}} \qquad \frac{A : \mathsf{For}}{\mathsf{Pf}\, A : \mathsf{Set}} \qquad \frac{p : \mathsf{Pf}\, A \qquad q : \mathsf{Pf}\, A}{p = q}$$

$$\frac{\mathsf{Pf}\, A \vdash \mathsf{Pf}\, B}{\mathsf{Pf}\, (A \supset B)} \qquad \frac{\mathsf{Pf}\, (A \supset B) \qquad \mathsf{Pf}\, A}{\mathsf{Pf}\, B}$$

$$\frac{x : \mathsf{Tm} \vdash \mathsf{Pf}\, A}{\mathsf{Pf}\, (\forall x.A)} \qquad \frac{\mathsf{Pf}\, (\forall x.A) \qquad t : \mathsf{Tm}}{\mathsf{Pf}\, (A[x \mapsto t])} \qquad \frac{t : \mathsf{Tm}}{\mathsf{Pf}\, (\mathsf{Eq}\, t\, t)} \qquad \frac{\mathsf{Pf}\, (\mathsf{Eq}\, t\, t')}{t = t'}$$

Példaként bebizonyítjuk, hogy $(A \supset B \supset C) \supset (B \supset A \supset C)$

minden A, B, C állításra. A leveleknél változók vannak.

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{\Gamma \vdash \mathsf{Pf}\,(A \supset B \supset C)} \qquad \overline{\Gamma \vdash \mathsf{Pf}\,A}}{\Gamma \vdash \mathsf{Pf}\,(B \supset C)} \qquad \overline{\Gamma \vdash \mathsf{Pf}\,B}}{\underbrace{\mathsf{Pf}\,(A \supset B \supset C), \mathsf{Pf}\,B, \mathsf{Pf}\,A}_{=:\Gamma} \vdash \mathsf{Pf}\,C}}{\mathsf{Pf}\,(A \supset B \supset C), \mathsf{Pf}\,B \vdash \mathsf{Pf}\,(A \supset C)}}{\mathsf{Pf}\,(A \supset B \supset C) \vdash \mathsf{Pf}\,(B \supset A \supset C)}}{\mathsf{Pf}\,\big((A \supset B \supset C) \supset (B \supset A \supset C)\big)}$$

53. Feladat. *Adjuk meg az intuicionista logika másodrendű standard modelljét (Tarski-szemantikáját)! A formulák (metanyelvi) állításokkal, a termek valamilyen posztulált halmazzal, a bizonyítások metanyelvi bizonyításokkal vannak modellezve.*

54. Feladat. *Egészítsük ki a nyelvet általános reláció- és termszimbólumokkal, melyeket egy elsőrendű szignatúrából veszünk!*

55. Feladat. *Egészítsük ki a nyelvet a logikai konjunkció, diszjunkció, tagadás, igaz, hamis és egzisztenciális kvantor szabályaival!*

56. Feladat. *Mutassuk meg, hogy ha az így kiegészített nyelvhez hozzávesszük az alábbi szabályt, akkor a formulák Heyting-algebrát alkotnak!*

$$\dfrac{\mathsf{Pf}\,(A \supset B) \qquad \mathsf{Pf}\,(B \supset A)}{A = B}$$

57. Feladat. *Vezessük le tetszőleges A-ra, hogy* $\mathsf{Pf}\,(\neg\neg(A \vee \neg A))$*!*

58. Feladat. *Mutassuk meg, hogy akkor és csak akkor vezethető le minden A-ra* $\mathsf{Pf}\,(A \vee \neg A)$*, ha minden A-ra* $\mathsf{Pf}\,(\neg\neg A \supset A)$ *levezethető!*

59. Feladat. *Tegyük az előző feladat nyelvét klasszikussá: adjuk hozzá a kizárt harmadik elvét, tehát az*

$$\overline{\mathsf{Pf}\,(A \vee \neg A)}$$

szabályt! Mutassuk meg, hogy ha a klasszikus nyelvhez hozzávesszük az alábbi szabályt, akkor a formulák Boole-algebrát alkotnak!

$$\dfrac{\mathsf{Pf}\,(A \supset B) \qquad \mathsf{Pf}\,(B \supset A)}{A = B}$$

A következő nyelvet egyszerű típuselméletnek is szokás nevezni.

60. Definíció (Egyszerű típusos lambda-kalkulus)**.**

$$\frac{}{\mathsf{Ty} : \mathsf{Set}} \qquad \frac{A : \mathsf{Ty}}{\mathsf{Tm}\, A : \mathsf{Set}} \qquad \frac{}{\iota : \mathsf{Ty}} \qquad \frac{A : \mathsf{Ty} \qquad B : \mathsf{Ty}}{A \Rightarrow B : \mathsf{Ty}}$$

$$\frac{x : \mathsf{Tm}\, A \vdash b : \mathsf{Tm}\, B}{\mathsf{lam}\, x.b : \mathsf{Tm}\, (A \Rightarrow B)} \qquad \frac{f : \mathsf{Tm}\, (A \Rightarrow B) \qquad a : \mathsf{Tm}\, A}{f \cdot a : \mathsf{Tm}\, B}$$

$$\frac{}{\Rightarrow\beta : (\mathsf{lam}\, x.b) \cdot a = t[x \mapsto a]} \qquad \frac{f : \mathsf{Tm}\, (A \Rightarrow B)}{\Rightarrow\eta : f = \mathsf{lam}\, x.f \cdot x}$$

Az olyan nyelveknél, melyeknél ugyanez a két szort van, megadhatunk egy rövidebb jelölést is. Egyszerűen elhagyjuk a Ty-t és a Tm-et. Egyértelmű, hogy adott esetben melyikről van szó, mert a termeknek mindig oda van írva a típusa. Ezenkívül az egyenlőségeknek nem adunk nevet. Ezzel a spórolósabb jelöléssel ugyanez a kalkulus a következő.

$$\bar{\iota} \qquad \frac{A \qquad B}{A \Rightarrow B} \qquad \frac{x : A \vdash b : B}{\mathsf{lam}\, x.b : A \Rightarrow B} \qquad \frac{f : A \Rightarrow B \qquad a : A}{f \cdot a : B}$$

$$\frac{}{(\mathsf{lam}\, x.b) \cdot a = t[x \mapsto a]} \qquad \frac{f : A \Rightarrow B}{f = \mathsf{lam}\, x.f \cdot x}$$

Megadjuk az egyszerű típuselméletet algebrai jelöléssel is.

$$\begin{array}{ll}
\mathsf{Ty} & : \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} \\
\iota & : \mathsf{Ty} \\
- \Rightarrow - & : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} \\
\mathsf{lam} & : (\mathsf{Tm}\, A \to \mathsf{Tm}\, B) \to \mathsf{Tm}\, (A \Rightarrow B) \\
- \cdot - & : \mathsf{Tm}\, (A \Rightarrow B) \to \mathsf{Tm}\, A \to \mathsf{Tm}\, B \\
\Rightarrow\beta & : \mathsf{lam}\, b \cdot a = b\, a \\
\Rightarrow\eta & : f = \mathsf{lam}\, \lambda x.f \cdot x
\end{array}$$

61. Jelölés. *Az utolsó négy sort még rövidebben is meg tudjuk adni, a fentivel megegyezőt jelent az alábbi jelölés.*

$$\mathsf{Ty} \qquad : \mathsf{Set}$$

$$\begin{aligned}
&\mathsf{Tm} && : \mathsf{Ty} \to \mathsf{Set}\\
&\iota && : \mathsf{Ty}\\
&- \Rightarrow - && : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty}\\
&\mathsf{lam} && : (\mathsf{Tm}\, A \to \mathsf{Tm}\, B) \simeq \mathsf{Tm}\,(A \Rightarrow B) : - \cdot -
\end{aligned}$$

Azt mondjuk, hogy a $(\mathsf{Tm}\, A \to \mathsf{Tm}\, B)$ *és a* $\mathsf{Tm}\,(A \Rightarrow B)$ *halmazok ekvivalensek. Az ekvivalencia két oldalára írjuk az operátorokat, amelyek a két oldal között képeznek. Általánosságban egy ekvivalencia* $(f : X \simeq Y : g)$ *a következőt rövidíti:*

$$\begin{aligned}
&(f : X \to Y) \times (g : Y \to X) \times\\
&\big((x : X) \to g\,(f\,x) = x\big) \times \big((y : Y) \to f\,(g\,y) = y\big)
\end{aligned}$$

62. Feladat. *Adjuk meg az egyszerű típusos lambda-kalkulus standard modelljét! A típusok metanyelvi halmazokkal (metanyelvi típusokkal), a termek ezek elemeivel vannak modellezve.*

A következő nyelvnél is a spórolós jelölést használjuk.

63. Definíció (Gödel-féle System T)**.**

$$\begin{aligned}
&\mathsf{Ty} && : \mathsf{Set}\\
&\mathsf{Tm} && : \mathsf{Ty} \to \mathsf{Set}\\
&- \Rightarrow - && : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty}\\
&\mathsf{lam} && : (\mathsf{Tm}\, A \to \mathsf{Tm}\, B) \simeq \mathsf{Tm}\,(A \Rightarrow B) : - \cdot -\\
&\mathsf{Nat} && : \mathsf{Ty}\\
&\mathsf{zero} && : \mathsf{Tm}\,\mathsf{Nat}\\
&\mathsf{suc} && : \mathsf{Tm}\,\mathsf{Nat} \to \mathsf{Tm}\,\mathsf{Nat}\\
&\mathsf{rec} && : \mathsf{Tm}\, A \to (\mathsf{Tm}\,\mathsf{Nat} \to \mathsf{Tm}\, A \to \mathsf{Tm}\, A) \to \mathsf{Tm}\,\mathsf{Nat} \to \mathsf{Tm}\, A\\
&\mathsf{Nat}\beta_1 && : \mathsf{rec}\, z\, s\, \mathsf{zero} = z\\
&\mathsf{Nat}\beta_1 && : \mathsf{rec}\, z\, s\, (\mathsf{suc}\, n) = s\, n\, (\mathsf{rec}\, z\, s\, n)
\end{aligned}$$

A rec operátort leírjuk a levezetési szabályos jelöléssel is: ez azért érdekes, mert a rec a második paraméterében két változót is köt, ezeket ilyenkor vesszővel elválasztva soroljuk fel, a kötött változók nevét pedig az operátor jelölésekor ponttal választjuk el:

$$\frac{z : A \qquad i : \mathsf{Nat}, a : A \vdash s : A \qquad n : \mathsf{Nat}}{\mathsf{rec}\, z\, (i.a.s)\, n : A}$$

A számítási szabályok a levezetési szabályos jelöléssel a következők.

$$\frac{}{\mathsf{rec}\,z\,(i.a.s)\,\mathsf{zero} = z} \qquad \frac{}{\mathsf{rec}\,z\,(i.a.s)\,(\mathsf{suc}\,n) = s[i \mapsto n, a \mapsto \mathsf{rec}\,z\,(i.a.s)\,n]}$$

A suc-ra vonatkozó számítási szabályában ($\mathsf{Nat}\beta_2$) párhuzamosan helyettesítjük az i és az a változók értékét. Algebrai jelöléssel explicit változónevekkel ez a következőnek felel meg.

$$\mathsf{rec}\,z\,(\lambda i\,a.s\,i\,a)\,(\mathsf{suc}\,n) = s\,n\,(\mathsf{rec}\,z\,s\,n)$$

A természetes számok beágyazhatók System T-be: megadható egy $\ulcorner\_\urcorner : \mathbb{N} \to \mathsf{Tm}\,\mathsf{Nat}$ függvény, mely n-hez $\mathsf{suc}^n\,\mathsf{zero}$-t rendeli, például $\ulcorner 3 \urcorner = \mathsf{suc}\,(\mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero}))$. Egy $f : \mathbb{N} \to \mathbb{N}$ függvény definiálható System T-ben, ha létezik olyan $t : \mathsf{Tm}\,(\mathsf{Nat} \to \mathsf{Nat})$, melyre minden n-re $\ulcorner f\,n \urcorner = t \cdot \ulcorner n \urcorner$.

64. Feladat. *Az összeadás, szorzás, hatványozás és az Ackermann-függvény definiálható System T-ben.*

65. Feladat. *Van olyan függvény, mely nem definiálható System T-ben.*

66. Feladat. *Adjuk meg a System T standard modelljét!*

67. Feladat. *Adjuk meg a* $\mathsf{pred} : \mathsf{Nat} \Rightarrow \mathsf{Nat}$ *függvényt, melyre teljesül, hogy* $\mathsf{pred}\,(\mathsf{suc}\,n) = n$ *minden* $n : \mathsf{Nat}$*-ra!*

68. Gondolkodnivaló. *Ha rekurzió helyett csak iteráció van, vagyis az s paraméter nem kapja meg az $i : \mathsf{Nat}$-ot, akkor is megadható a* pred *függvény, de az egyenlőség nem teljesül minden $n : \mathsf{Nat}$-ra, csak az $n = \mathsf{suc}^n\,\mathsf{zero}$ alakúakra.*

A következő nyelv ismét Turing-teljes, csakúgy, mint a típus nélküli kombinátor/lambda-kalkulus, de vannak benne típusok. Lényegileg a System T kiegészítése egy fixpont-kombinátorral, de nincs η-szabály a függvényekre. És mivel van fixpont-kombinátor, ezért nincs szükség rekurzorra a természetes számokhoz, elég egy ifZero C-stílusú if-then-else elágazás.

69. Definíció (PCF)**.** *Két szortunk van, a típusok és a típusokkal indexelt termek.*

$$\frac{A \qquad B}{A \Rightarrow B} \qquad \frac{x : A \vdash b : B}{\mathsf{lam}\,x.b : A \Rightarrow B} \qquad \frac{f : A \Rightarrow B \qquad a : A}{f \cdot a : B}$$

$$\frac{}{(\mathsf{lam}\, x.b) \cdot a = t[x \mapsto a]} \qquad \frac{x : A \vdash t : A}{\mathsf{fix}\, x.t : A} \qquad \frac{}{\mathsf{fix}\, x.t = t[x \mapsto \mathsf{fix}\, x.t]}$$

$$\frac{}{\mathsf{Nat}} \qquad \frac{}{\mathsf{zero} : \mathsf{Nat}} \qquad \frac{n : \mathsf{Nat}}{\mathsf{suc}\, n : \mathsf{Nat}} \qquad \frac{b : \mathsf{Nat} \qquad t : A \qquad f : A}{\mathsf{ifZero}\, b\, t\, f : A}$$

$$\frac{n : \mathsf{Nat}}{\mathsf{pred}\, n : \mathsf{Nat}} \qquad \frac{}{\mathsf{ifZero}\, \mathsf{zero}\, t\, f = t} \qquad \frac{}{\mathsf{ifZero}\, (\mathsf{suc}\, n)\, t\, f = f}$$

$$\frac{}{\mathsf{pred}\, (\mathsf{suc}\, n) = n} \qquad \frac{}{\mathsf{pred}\, \mathsf{zero} = \mathsf{zero}}$$

Vegyük észre, hogy a függvényekre nincs η-szabályunk.

70. Feladat. *Az összeadás, szorzás, hatványozás és az Ackermann függvény definiálható PCF-ben.*

71. Definíció (Összeg és szorzat típusok)**.** *Az egyszerű típusos lambda-kalkulust az alábbi szabályokkal egészítjük ki.*

$$\frac{}{\bot} \qquad \frac{b : \bot}{\mathsf{exfalso}\, b : A} \qquad \frac{x : \bot \vdash a : A}{a = \mathsf{exfalso}\, x}$$

$$\frac{A \qquad B}{A + B} \qquad \frac{a : A}{\mathsf{inl}\, a : A + B} \qquad \frac{b : B}{\mathsf{inr}\, b : A + B}$$

$$\frac{t : (A + B) \qquad x : A \vdash c_0 : C \qquad y : B \vdash c_1 : C}{\mathsf{case}\, t\, (x.c_0)\, (y.c_1) : C}$$

$$\frac{}{\mathsf{case}\, (\mathsf{inl}\, a)\, (x.c_0)\, (y.c_1) = c_0[x \mapsto a]} \qquad \frac{}{\mathsf{case}\, (\mathsf{inr}\, b)\, (x.c_0)\, (y.c_1) = c_1[y \mapsto b]}$$

$$\frac{z : A + B \vdash t : C}{t = \mathsf{case}\, z\, (x.t[z \mapsto \mathsf{inl}\, x])\, (y.t[z \mapsto \mathsf{inr}\, y])}$$

$$\frac{}{\top} \qquad \frac{}{\mathsf{tt} : \top} \qquad \frac{t : \top}{t = \mathsf{tt}}$$

$$\frac{A \qquad B}{A \times B} \qquad \frac{a : A \qquad b : B}{(a, b) : A \times B} \qquad \frac{t : A \times B}{\mathsf{fst}\, t : A} \qquad \frac{t : A \times B}{\mathsf{snd}\, t : B}$$

$$\frac{}{\mathsf{fst}\, (a, b) = a} \qquad \frac{}{\mathsf{snd}\, (a, b) = b} \qquad \frac{t : A \times B}{t = (\mathsf{fst}\, t, \mathsf{snd}\, t)}$$

72. Feladat. *Adjuk meg a logikai értékek típusát a következőképp:* $\mathsf{Bool} := \top + \top$*! Mutassuk meg, hogy az alábbi operátorok és egyenlőségek definiálhatóak!*

$$\frac{}{\mathsf{true} : \mathsf{Bool}} \qquad \frac{}{\mathsf{false} : \mathsf{Bool}} \qquad \frac{b : \mathsf{Bool} \qquad a_0 : A \qquad a_1 : A}{\mathsf{ite}\, b\, a_0\, a_1 : A}$$

$$\frac{}{\mathsf{ite}\,\mathsf{true}\,a_0\,a_1 = a_0} \qquad \frac{}{\mathsf{ite}\,\mathsf{false}\,a_0\,a_1 = a_1}$$

$$\frac{x : \mathsf{Bool} \vdash t : A}{t = \mathsf{ite}\,x\,(t[x \mapsto \mathsf{true}])\,(t[x \mapsto \mathsf{false}])}$$

73. Definíció (Izomorfizmus)**.** *A és B típusok izomorfak, ha létezik $x : A \vdash t : B$ és $y : B \vdash t' : A$ term, melyekre $t[x \mapsto t'] = y$ és $t'[y \mapsto t] = x$. Ebben az esetben $A \cong B$-t írunk.*

74. Feladat. *Mutassuk, meg hogy a $\cong$ reláció reflexív, szimmetrikus és tranzitív, valamint kongruencia a $\times$-ra és a $+$-ra vontakozóan! (Tehát például $A \cong A'$-ből következik $A \times B \cong A' \times B$.)*

75. Feladat. *Mutassuk meg, hogy az összeg és szorzat típusokkal rendelkező nyelvben a $\cong$ relációval a típusok exponenciális semiring-et (félgyűrű, rig) alkotnak!*

76. Feladat. *Adjuk meg a fenti nyelv standard modelljét!*

Az alábbi nyelvben van egy generikus map függvényünk, ami nagyon sok $F : \mathsf{Ty} \to \mathsf{Ty}$ típusfüggvényre (a pozitív típusfüggvényekre) képes egy $(\mathsf{Tm}\,A \to \mathsf{Tm}\,B) \to \mathsf{Tm}\,(F\,A) \to \mathsf{Tm}\,(F\,B)$ függvényt megadni. A típusfüggvényekre megadjuk, hogy melyik pozitív és melyik negatív. Például pozitív az $F\,X = ((X + X) \Rightarrow \mathsf{Nat}) \Rightarrow X$, negatív az $F\,X = X \Rightarrow \mathsf{Nat}$, és az $F\,X = X \Rightarrow X$ se nem pozitív, se nem negatív. Pozitív és negatív helyett szokás kovariánsat és kontravariánsat mondani. Ebben és az induktív típusok nyelvében Robert Harpert követjük [8].

77. Definíció (Generikus programozás)**.** *Az összeg és szorzat típusok nyelvét (71. definíció) kiegészítjük az alábbiakkal.*

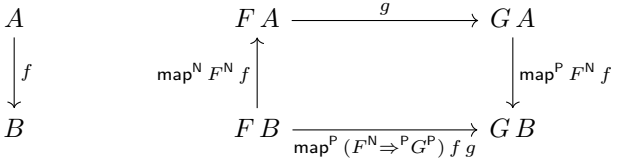
$$\begin{aligned}
&\mathsf{Pos} && : (\mathsf{Ty} \to \mathsf{Ty}) \to \mathsf{Set}\\
&\mathsf{Neg} && : (\mathsf{Ty} \to \mathsf{Ty}) \to \mathsf{Set}\\
&\mathsf{id}^{\mathsf{P}} && : \mathsf{Pos}\,(\lambda X.X)\\
&\mathsf{const}^{\mathsf{P}} && : (A : \mathsf{Ty}) \to \mathsf{Pos}\,(\lambda_.A)\\
&\mathsf{const}^{\mathsf{N}} && : (A : \mathsf{Ty}) \to \mathsf{Neg}\,(\lambda_.A)\\
&- +^{\mathsf{P}} - && : \mathsf{Pos}\,F \to \mathsf{Pos}\,G \to \mathsf{Pos}\,(\lambda X.F\,X + G\,X)\\
&- +^{\mathsf{N}} - && : \mathsf{Neg}\,F \to \mathsf{Neg}\,G \to \mathsf{Neg}\,(\lambda X.F\,X + G\,X)\\
&- \times^{\mathsf{P}} - && : \mathsf{Pos}\,F \to \mathsf{Pos}\,G \to \mathsf{Pos}\,(\lambda X.F\,X \times G\,X)\\
&- \times^{\mathsf{N}} - && : \mathsf{Neg}\,F \to \mathsf{Neg}\,G \to \mathsf{Neg}\,(\lambda X.F\,X \times G\,X)
\end{aligned}$$

$$- \Rightarrow^{\mathsf{P}} - : \mathsf{Neg}\, F \to \mathsf{Pos}\, G \to \mathsf{Pos}\, (\lambda X.F\, X \Rightarrow G\, X)$$
$$- \Rightarrow^{\mathsf{N}} - : \mathsf{Pos}\, F \to \mathsf{Neg}\, G \to \mathsf{Neg}\, (\lambda X.F\, X \Rightarrow G\, X)$$
$$\mathsf{map}^{\mathsf{P}} : \mathsf{Pos}\, F \to (f : \mathsf{Tm}\, A \to \mathsf{Tm}\, B) \to \mathsf{Tm}\, (F\, A) \to \mathsf{Tm}\, (F\, B)$$
$$\mathsf{map}^{\mathsf{N}} : \mathsf{Neg}\, F \to (f : \mathsf{Tm}\, A \to \mathsf{Tm}\, B) \to \mathsf{Tm}\, (F\, B) \to \mathsf{Tm}\, (F\, A)$$

A számítási szabályok a következők, nem adunk nekik nevet.

$$\begin{aligned}
&\mathsf{map}^{\mathsf{P}}\ \mathsf{id}^{\mathsf{P}} && f\, a && = f \cdot a\\
&\mathsf{map}^{\mathsf{P}}\ (\mathsf{const}^{\mathsf{P}}\ A) && f\, u && = u\\
&\mathsf{map}^{\mathsf{N}}\ (\mathsf{const}^{\mathsf{N}}\ A) && f\, u && = u\\
&\mathsf{map}^{\mathsf{P}}\ (F^{\mathsf{P}} +^{\mathsf{P}} G^{\mathsf{P}}) && f\, (\mathsf{inl}\, u) && = \mathsf{inl}\, (\mathsf{map}^{\mathsf{P}}\ F^{\mathsf{P}}\ f\, u)\\
&\mathsf{map}^{\mathsf{P}}\ (F^{\mathsf{P}} +^{\mathsf{P}} G^{\mathsf{P}}) && f\, (\mathsf{inr}\, v) && = \mathsf{inr}\, (\mathsf{map}^{\mathsf{P}}\ G^{\mathsf{P}}\ f\, v)\\
&\mathsf{map}^{\mathsf{N}}\ (F^{\mathsf{N}} +^{\mathsf{N}} G^{\mathsf{N}}) && f\, (\mathsf{inl}\, u) && = \mathsf{inl}\, (\mathsf{map}^{\mathsf{N}}\ F^{\mathsf{N}}\ f\, u)\\
&\mathsf{map}^{\mathsf{N}}\ (F^{\mathsf{N}} +^{\mathsf{N}} G^{\mathsf{N}}) && f\, (\mathsf{inr}\, v) && = \mathsf{inr}\, (\mathsf{map}^{\mathsf{N}}\ G^{\mathsf{N}}\ f\, v)\\
&\mathsf{map}^{\mathsf{P}}\ (F^{\mathsf{P}} \times^{\mathsf{P}} G^{\mathsf{P}}) && f\, (u, v) && = (\mathsf{map}^{\mathsf{P}}\ F^{\mathsf{P}}\ f\, u, \mathsf{map}^{\mathsf{P}}\ G^{\mathsf{P}}\ f\, v)\\
&\mathsf{map}^{\mathsf{N}}\ (F^{\mathsf{N}} \times^{\mathsf{N}} G^{\mathsf{N}}) && f\, (u, v) && = (\mathsf{map}^{\mathsf{N}}\ F^{\mathsf{N}}\ f\, u, \mathsf{map}^{\mathsf{N}}\ G^{\mathsf{N}}\ f\, v)\\
&\mathsf{map}^{\mathsf{P}}\ (F^{\mathsf{N}} \Rightarrow^{\mathsf{P}} G^{\mathsf{P}})\ f\, g && && = \mathsf{lam}\, \lambda x.\mathsf{map}^{\mathsf{P}}\ G^{\mathsf{P}}\ f\, \big(g \cdot (\mathsf{map}^{\mathsf{N}}\ F^{\mathsf{N}}\ f\, x)\big)\\
&\mathsf{map}^{\mathsf{N}}\ (F^{\mathsf{P}} \Rightarrow^{\mathsf{N}} G^{\mathsf{N}})\ f\, g && && = \mathsf{lam}\, \lambda x.\mathsf{map}^{\mathsf{N}}\ G^{\mathsf{N}}\ f\, \big(g \cdot (\mathsf{map}^{\mathsf{P}}\ F^{\mathsf{P}}\ f\, x)\big)
\end{aligned}$$

A map függvény $\Rightarrow^{\mathsf{P}}$ esetét a következő diagram illusztrálja.



Egy induktív típust egy szigorúan pozitív típusfüggvény határoz meg (lásd az induktív típusok iniciális algebra szemantikáját). Először is map-pelnünk kell a szigorúan pozitív típusoperátorokon, majd megadhatjuk az induktív típusokat, mint szigorúan pozitív típusfüggvények legkisebb fixpontját.

78. Definíció (Induktív típusok)**.** *Az összeg és szorzat típusok nyelvét*

(71. definíció) kiegészítjük az alábbiakkal.

$$
\begin{array}{ll}
\mathsf{SPos} & : (\mathsf{Ty} \to \mathsf{Ty}) \to \mathsf{Set} \\
\mathsf{id} & : \mathsf{SPos}\,(\lambda X.X) \\
\mathsf{const} & : (A : \mathsf{Ty}) \to \mathsf{SPos}\,(\lambda\_.A) \\
- \mathbin{\bar{+}} - & : \mathsf{SPos}\,F \to \mathsf{SPos}\,G \to \mathsf{SPos}\,(\lambda X.F\,X + G\,X) \\
- \mathbin{\bar{\times}} - & : \mathsf{SPos}\,F \to \mathsf{SPos}\,G \to \mathsf{SPos}\,(\lambda X.F\,X \times G\,X) \\
- \mathbin{\bar{\Rightarrow}} - & : (A : \mathsf{Ty}) \to \mathsf{SPos}\,F \to \mathsf{SPos}\,(\lambda X.A \Rightarrow F\,X) \\
\mathsf{map} & : \mathsf{SPos}\,F \to (f : \mathsf{Tm}\,A \to \mathsf{Tm}\,B) \to \mathsf{Tm}\,(F\,A) \to \mathsf{Tm}\,(F\,B) \\
 & : \mathsf{map}\ \ \mathsf{id}\ \ \ \ \ \ f\,a \ \ \ \ \ \ = f \cdot a \\
 & : \mathsf{map}\ \ (\mathsf{const}\,A)\,f\,u \ \ \ \ \ \ = u \\
 & : \mathsf{map}\ \ (\bar{F} \mathbin{\bar{+}} \bar{G})\ \ f\,(\mathsf{inl}\,u) = \mathsf{inl}\,(\mathsf{map}\,\bar{F}\,f\,u) \\
 & : \mathsf{map}\ \ (\bar{F} \mathbin{\bar{+}} \bar{G})\ \ f\,(\mathsf{inr}\,v) = \mathsf{inr}\,(\mathsf{map}\,\bar{G}\,f\,v) \\
 & : \mathsf{map}\ \ (\bar{F} \mathbin{\bar{\times}} \bar{G}), f\,(u,v)\ \ = (\mathsf{map}\,\bar{F}\,f\,u, \mathsf{map}\,\bar{G}\,f\,v) \\
 & : \mathsf{map}\ \ (A \mathbin{\bar{\Rightarrow}} \bar{F})\ f\,g \ \ \ \ \ \ = \mathsf{lam}\,\lambda a.\mathsf{map}\,\bar{F}\,f\,(g \cdot a) \\
\mathsf{Ind} & : \mathsf{SPos}\,\mathsf{F} \to \mathsf{Ty} \\
\mathsf{con} & : (\bar{F} : \mathsf{SPos}\,\mathsf{F}) \to \mathsf{Tm}\left(F\,(\mathsf{Ind}\,\bar{F})\right) \to \mathsf{Tm}\,(\mathsf{Ind}\,\bar{F}) \\
\mathsf{ite} & : (\bar{F} : \mathsf{SPos}\,\mathsf{F})(A : \mathsf{Ty}) \to \left(\mathsf{Tm}\,(F\,A) \to \mathsf{Tm}\,A\right) \to \mathsf{Tm}\,(\mathsf{Ind}\,\bar{F}) \to \\
 & \ \ \mathsf{Tm}\,A \\
\mathsf{Ind}\beta & : \mathsf{ite}\,\bar{F}\,A\,f\,(\mathsf{con}\,\bar{F}\,x) = f\left(\mathsf{map}\,\bar{F}\,(\mathsf{ite}\,\bar{F}\,A\,f)\,x\right)
\end{array}
$$

79. Feladat. *A természetes számokat a $\lambda X.\top + X$ típusfüggvény segítségével a* $\mathsf{Nat} := \mathsf{Ind}\,(\mathsf{const}\,\top \mathbin{\bar{+}} \mathsf{id})$ *típussal adjuk meg. Adjuk meg ehhez a* zero *és* suc *konstruktorokat, valamint az iterátort és mutassuk meg, hogy a számítási szabályok teljesülnek!*

80. Feladat. *Kódoljuk el a következő induktív típusokat: természetes számok listái, bináris fák, háromfelé ágazó fák, végtelenfelé ágazó fák.*

A koinduktív típusokat terminális koalgebrák adják meg. Minden szigorúan pozitív típusfüggvény meghatároz egy koinduktív típust. A következőkben kiegészítjük az induktív típusok nyelvét koinduktív típusokkal is (igazából az induktív típusokra nincs szükségünk, csak SPos-ra és map-re).

81. Definíció ((Ko)induktív típusok)**.** *Az induktív típusok nyelvét (78.*

definíció) kiegészítjük az alábbiakkal.

$$
\begin{aligned}
&\mathsf{Coind} && : \mathsf{SPos}\,\mathsf{F} \to \mathsf{Ty}\\
&\mathsf{des} && : (\bar{F} : \mathsf{SPos}\,\mathsf{F}) \to \mathsf{Tm}\,(\mathsf{Coind}\,\bar{F}) \to \mathsf{Tm}\,\big(F\,(\mathsf{Coind}\,\bar{F})\big)\\
&\mathsf{gen} && : (\bar{F} : \mathsf{SPos}\,\mathsf{F})(A : \mathsf{Ty}) \to \big(\mathsf{Tm}\,A \to \mathsf{Tm}\,(F\,A)\big) \to \mathsf{Tm}\,A \to\\
& && \quad \mathsf{Tm}\,(\mathsf{Coind}\,\bar{F})\\
&\mathsf{Coind}\beta && : \mathsf{des}\,\bar{F}\,(\mathsf{gen}\,\bar{F}\,A\,f\,a) = \mathsf{map}\,\bar{F}\,(\mathsf{gen}\,\bar{F}\,A\,f)\,(f\,a)
\end{aligned}
$$

A koinduktív típusoknak destruktorai vannak, és generátoruk. A legalapvetőbb példa az A típusú elemeket tartalmazó végtelen lista (folyam, stream), melyet a $\lambda X.A \times X$ típusfüggvény határoz meg.

82. Feladat. *Adjuk meg a folyamokat koinduktív típusként, és adjuk meg az* $[1, 2, 3, \dots]$ *folyamot!*

83. Feladat. *Adjuk meg az olyan gépek koinduktív típusát, melyekbe be lehet írni egy (természetes) számot, ki lehet íratni velük egy számot, és meg lehet nyomni rajtuk egy gombot! Ennek a típusnak az elemeként készítsünk egy összeadó-gépet, mely a beírt számokat összeadja, az összeget kiírja, és a gomb megnyomásával nullázza az összeget.*

A következő nyelvben vannak polimorf típusok. Ezt úgy érjük el, hogy vannak típusváltozók, tehát a $\vdash$ bal oldalán típusok is szerepelhetnek. Például az identitás függvényt egyszerű típuselméletben külön-külön meg kell adni az összes típusra, például $\mathsf{Nat} \Rightarrow \mathsf{Nat}$, $\mathsf{Bool} \Rightarrow \mathsf{Bool}$ stb., míg polimorf típusként meg tudjuk adni az összes típusra: $\forall A.A \Rightarrow A$. A következő típusrendszerben megkülönböztetjük a monomorf (MTy) és a polimorf típusokat (Ty), és az univerzális kvantor csak egy polimorf típus elején szerepelhet. Az alaptípusok (például Nat) monomorfak.

84. Definíció (Hindley–Milner-típusrendszer)**.**

$$
\begin{aligned}
&\mathsf{MTy} && : \mathsf{Set}\\
&\mathsf{Ty} && : \mathsf{Set}\\
&\mathsf{Tm} && : \mathsf{Ty} \to \mathsf{Set}\\
&- \Rightarrow - && : \mathsf{MTy} \to \mathsf{MTy} \to \mathsf{MTy}\\
&\forall && : (\mathsf{MTy} \to \mathsf{Ty}) \to \mathsf{Ty}\\
&\mathsf{i} && : \mathsf{MTy} \to \mathsf{Ty}
\end{aligned}
$$

$$\begin{aligned}\mathsf{lam} &: \big(\mathsf{Tm}\,(\mathsf{i}\,A) \to \mathsf{Tm}\,(\mathsf{i}\,B)\big) \simeq \mathsf{Tm}\,(\mathsf{i}\,(A \Rightarrow B))\\ \mathsf{Lam} &: \big((A : \mathsf{MTy}) \to \mathsf{Tm}\,(B\,A)\big) \simeq \mathsf{Tm}\,(\forall\, B)\end{aligned}$$

A Hindley–Milner-típusrendszerben nincsenek induktív és koinduktív típusok, kézzel kell őket hozzáadni, csakúgy mint egyszerű típuselméletben. A szokásos módszer az, hogy minden típusra van egy if-then-else/case jellegű operátorunk, és van egy központi fixpontoperátorunk, mint PCF-ben.

85. Feladat. *Egészítsük ki a Hindley–Milner-típusrendszert fixpontoperátorral, egész számokkal (egész számokat egy $\mathbb{Z} \to \mathsf{Tm}\,\mathsf{Int}$ operátorral lehet bevezetni), melyekre van néhány primitív operátor: összeadás, szorzás, hatványozás, és $\leq$ összehasonlítás, mely C-stílusú logikai értéket ad vissza.*

86. Feladat. *Mutassuk meg, hogy az így megadott nyelven definiálható a faktoriális, a Fibonacci- és az Ackermann-függvény!*

87. Gondolkodnivaló. *A Hindley–Milner-nyelvre megadható egy típuskikövetkeztető, mely típusinformáció nélküli pretermekből szintaktikus termeket készít.*

A következő nyelvben a $\forall$ már bárhol szerepelhet, nem csak a típusok legelején.

88. Definíció (System F)**.** *Két szortunk van, a típusok és a típusokkal indexelt termek.*

$$\frac{A \qquad B}{A \Rightarrow B} \qquad \frac{x : A \vdash b : B}{\mathsf{lam}\,x.b : A \Rightarrow B} \qquad \frac{f : A \Rightarrow B \qquad a : A}{f \cdot a : B}$$

$$\frac{}{(\mathsf{lam}\,x.b) \cdot a = t[x \mapsto a]} \qquad \frac{f : A \Rightarrow B}{f = \mathsf{lam}\,x.f \cdot x}$$

$$\frac{X \vdash A}{\forall X.A} \qquad \frac{X \vdash a : A}{\mathsf{Lam}\,X.a : \forall X.A} \qquad \frac{f : \forall X.A \qquad C}{f \bullet C : A[X \mapsto C]}$$

$$\frac{}{(\mathsf{Lam}\,X.a) \bullet C = a[X \mapsto C]} \qquad \frac{f : \forall X.A}{f = \mathsf{Lam}\,X.f \bullet X}$$

A System F rövidebb megadása:

$$\begin{aligned}\mathsf{Ty} &: \mathsf{Set}\\ \mathsf{Tm} &: \mathsf{Ty} \to \mathsf{Set}\end{aligned}$$

$$
\begin{array}{ll}
- \Rightarrow - & : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} \\
\mathsf{lam} & : (\mathsf{Tm}\, A \vdash \mathsf{Tm}\, B) \simeq \mathsf{Tm}\, (A \Rightarrow B) : - \cdot - \\
\forall & : (\mathsf{Ty} \vdash \mathsf{Ty}) \to \mathsf{Ty} \\
\mathsf{Lam} & : (X : \mathsf{Ty} \vdash \mathsf{Tm}\, (A\, X)) \simeq \mathsf{Tm}\, (\forall\, A) : - \bullet -
\end{array}
$$

89. Gondolkodnivaló. *System F-ben az összes induktív és koinduktív típus elkódolható (Church-kódolás). Például* $Bool := \forall X.X \to X \to X$.

System F-ben vannak polimorf függvények, de nincsenek típus szintű függvények. Például a listák nem adhatók meg, ami egy típusról típusra képező függvény. A következő rendszer ezt kijavítja. A típusokon és a termeken túl van egy újabb szortunk, a fajták (Kind-ok) szortja, a típusok pedig indexelve vannak az ő „típusukkal" (fajtájukkal). A $*$ fajtájú típusok a tulajdonképpeni típusok, nekik lehetnek termei. A $* \Rrightarrow *$ fajtájú típusok a típus-függvények, melyek egy tulajdonképpeni típusból képeznek egy tulajdonképpeni típusba.

90. Definíció (System F_ω)**.**

$$
\begin{array}{ll}
\mathsf{Kind} & : \mathsf{Set} \\
\mathsf{Ty} & : \mathsf{Kind} \to \mathsf{Set} \\
- \Rrightarrow - & : \mathsf{Kind} \to \mathsf{Kind} \to \mathsf{Kind} \\
\mathsf{LAM} & : (\mathsf{Ty}\, K \to \mathsf{Ty}\, L) \simeq \mathsf{Ty}\, (K \Rrightarrow L) : - \bullet\!\!\!\bullet - \\
* & : \mathsf{Kind} \\
\mathsf{Tm} & : \mathsf{Ty} * \to \mathsf{Set} \\
\forall & : (\mathsf{Ty}\, K \to \mathsf{Ty} *) \to \mathsf{Ty} * \\
\mathsf{Lam} & : ((X : \mathsf{Ty}\, K) \to \mathsf{Tm}\, (A\, X)) \simeq \mathsf{Tm}\, (\forall\, A) : - \bullet - \\
- \Rightarrow - & : \mathsf{Ty} * \to \mathsf{Ty} * \to \mathsf{Ty} * \\
\mathsf{lam} & : (\mathsf{Tm}\, A \to \mathsf{Tm}\, B) \simeq \mathsf{Tm}\, (A \Rightarrow B) : - \cdot -
\end{array}
$$

A fajták és típusok a $\Rrightarrow$-val úgy viselkednek, mint egyszerű típuselméletben a típusok és a termek. A System F_ω érdekessége, hogy az egyik szort felhasznál egy operátort ($*$-ot), tehát a szortok nem adhatók meg az operátoroktól elkülönülten.

A lista típus a következőképp Church-kódolható:

$$
\begin{array}{l}
\mathsf{List} : \mathsf{Ty}\, (* \Rrightarrow *) \\
\mathsf{List} := \mathsf{LAM}\, \lambda A. \forall\, \lambda B. B \Rightarrow (A \Rightarrow B \Rightarrow B) \Rightarrow B
\end{array}
$$

A Haskell bind operátorának a következő a típusa:

$$\forall\, \lambda M. \forall\, \lambda A. \forall\, \lambda B. (A \Rightarrow M \bullet B) \Rightarrow M \bullet A \Rightarrow M \bullet B$$

A λ-kat kihagyva és a $\forall$ által kötött típusok fajtáit odaírva a következőt kapjuk:

$$\forall (M : * \Rrightarrow *). \forall (A : *). \forall (B : *). (A \Rightarrow M \bullet B) \Rightarrow M \bullet A \Rightarrow M \bullet B$$

91. Feladat. *Definiáljuk a Church-kódolt lista típus konstruktorait és iterátorát (fold operátorát)!*

92. Feladat. *Adjuk meg a Church-kódolt lista típus return és bind operátorait és vizsgáljuk meg, hogy teljesítik-e a monád törvényeket!*

93. Feladat. *Adjuk meg a lambda-kocka (lambda cube [3]) összes nyelvét SOGAT-ként! Adjuk meg a legáltalánosabb nyelvet (CC) és fejezzük ki, hogy a speciálisabb nyelvek milyen extra kritériumokkal adódnak ebből.*

A függő típuselméletben a típusok termektől is függhetnek (System F-ben a típusok csak típusoktól függhettek).

94. Definíció (Martin-Löf típuselmélete)**.** *Két szortunk van, a típusok és a típusokkal indexelt termek. A típusok továbbá a szintjükkel vannak indexelve, ami egy természetes szám. A 61. jelölést használjuk.*

$$
\begin{array}{ll}
\mathsf{Ty} & : \mathbb{N} \to \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Ty}\, i \to \mathsf{Set} \\
\Pi & : (A : \mathsf{Ty}\, i) \to (\mathsf{Tm}\, A \to \mathsf{Ty}\, i) \to \mathsf{Ty}\, i \\
\mathsf{lam} & : ((a : \mathsf{Tm}\, A) \to \mathsf{Tm}\, (B\, a)) \simeq \mathsf{Tm}\, (\Pi\, A\, B) : - \cdot - \\
\Sigma & : (A : \mathsf{Ty}\, i) \to (\mathsf{Tm}\, A \to \mathsf{Ty}\, i) \to \mathsf{Ty}\, i \\
(-, -) & : (a : \mathsf{Tm}\, A) \times \mathsf{Tm}\, (B\, a) \simeq \mathsf{Tm}\, (\Sigma\, A\, B) : \mathsf{fst}, \mathsf{snd} \\
\mathsf{U} & : (i : \mathbb{N}) \to \mathsf{Ty}\, (i + 1) \\
\mathsf{c} & : \mathsf{Ty}\, i \simeq \mathsf{Tm}\, (\mathsf{U}\, i) : \mathsf{El} \\
\mathsf{Lift} & : \mathsf{Ty}\, i \to \mathsf{Ty}\, (i + j) \\
\mathsf{mk} & : \mathsf{Tm}\, A \simeq \mathsf{Tm}\, (\mathsf{Lift}\, A) : \mathsf{un} \\
\bot & : \mathsf{Ty}\, 0 \\
\mathsf{exfalso} & : \mathsf{Tm}\, \bot \to \mathsf{Tm}\, A \\
\top & : \mathsf{Ty}\, 0
\end{array}
$$

$$
\begin{aligned}
&\mathsf{tt} && : \top \simeq \mathsf{Tm}\,\top\\
&\mathsf{Bool} && : \mathsf{Ty}\,0\\
&\mathsf{true} && : \mathsf{Tm}\,\mathsf{Bool}\\
&\mathsf{false} && : \mathsf{Tm}\,\mathsf{Bool}\\
&\mathsf{indBool} && : (C : \mathsf{Tm}\,\mathsf{Bool} \to \mathsf{Ty}\,i) \to \mathsf{Tm}\,(C\,\mathsf{true}) \to \mathsf{Tm}\,(C\,\mathsf{false}) \to\\
& && \quad (b : \mathsf{Tm}\,\mathsf{Bool}) \to \mathsf{Tm}\,(C\,b)\\
&\mathsf{Bool}\beta_1 && : \mathsf{indBool}\,t\,f\,\mathsf{true} = t\\
&\mathsf{Bool}\beta_2 && : \mathsf{indBool}\,t\,f\,\mathsf{false} = f\\
&\mathsf{Id} && : (A : \mathsf{Ty}\,i) \to \mathsf{Tm}\,A \to \mathsf{Tm}\,A \to \mathsf{Ty}\,i\\
&\mathsf{refl} && : (a : \mathsf{Tm}\,A) \to \mathsf{Tm}\,(\mathsf{Id}\,a\,a)\\
&\mathsf{J} && : \big(C : (x : \mathsf{Tm}\,A) \to \mathsf{Tm}\,(\mathsf{Id}\,A\,a\,x) \to \mathsf{Ty}\,i\big) \to\\
& && \quad \mathsf{Tm}\,\big(C\,a\,(\mathsf{refl}\,a)\big) \to (x : \mathsf{Tm}\,A)(e : \mathsf{Tm}\,(\mathsf{Id}\,A\,a\,x)) \to \mathsf{Tm}\,(C\,x\,e)\\
&\mathsf{Id}\beta && : \mathsf{J}\,C\,w\,a\,(\mathsf{refl}\,a) = w\\
&\mathsf{W} && : (S : \mathsf{Ty}\,i) \to (\mathsf{Tm}\,S \to \mathsf{Ty}\,i) \to \mathsf{Ty}\,i\\
&\mathsf{sup} && : (s : \mathsf{Tm}\,S) \to (\mathsf{Tm}\,(P\,s) \to \mathsf{Tm}\,(\mathsf{W}\,S\,P)) \to \mathsf{Tm}\,(\mathsf{W}\,S\,P)\\
&\mathsf{indW} && : (C : \mathsf{Tm}\,(\mathsf{W}\,S\,P) \to \mathsf{Ty}\,i) \to\\
& && \quad \Big(\big((p : \mathsf{Tm}\,(P\,s)) \to \mathsf{Tm}\,(C\,(f\,p))\big) \to \mathsf{Tm}\,\big(C\,(\mathsf{sup}\,s\,f)\big)\Big) \to\\
& && \quad (w : \mathsf{Tm}\,(\mathsf{W}\,S\,P)) \to \mathsf{Tm}\,\big(C\,w\big)\\
&\mathsf{W}\beta && : \mathsf{indW}\,C\,h\,(\mathsf{sup}\,s\,f) = h\,(\lambda p.\mathsf{indW}\,C\,h\,(f\,p))
\end{aligned}
$$

95. Feladat. *Adjuk meg a Martin-Löf-típuselmélet másodrendű standard modelljét!*

96. Jelölés. *Rövidítések:*

$$
\begin{aligned}
A \Rightarrow B &:= \Pi\,A\,(\lambda\_.B)\\
A \times\ B &:= \Sigma\,A\,(\lambda\_.B)
\end{aligned}
$$

97. Feladat. *Adjunk meg* $\mathsf{Tm}\,(\mathsf{Id}\,\mathsf{Bool}\,\mathsf{true}\,\mathsf{false} \Rightarrow \bot)$ *egy elemét!*

98. Feladat. *Fogalmazzuk meg, majd bizonyítsuk be, hogy az* $\mathsf{Id}\,A$ *tetszőleges* A*-ra egy ekvivalencia-reláció!*

99. Feladat. *Adjuk meg a természetes számokat a* W *típus segítségével! Bizonyítsuk be a teljes indukció elvét! (Szükség van Hugunin trükkjére [9].)*

100. Feladat. *Adjuk meg a koinduktív típusok szabályait* M *típusok segítségével!*

101. Feladat. *Mutassuk meg, hogy ha az univerzum szabálya* $\mathsf{U}\,i : \mathsf{Ty}\,i$ *lenne, akkor van* $\mathsf{Tm}\,\bot$*-nak eleme.*

102. Feladat. *Fogalmazzuk meg, majd mutassuk meg, hogy az elsőrendű logika beágyazható Martin-Löf-típuselméletbe!*

103. Definíció (Russell)**.** *A Martin-Löf-típuselmélet Russell típusú, ha a következő szort-egyenlőség teljesül:*

$$\mathsf{Ty}\,i = \mathsf{Tm}\,(\mathsf{U}\,i)$$

104. Feladat. *Mutassunk egy másodrendű modellt, amelyben ez teljesül! Mutassuk meg, hogy tetszőleges másodrendű modellből képezhető olyan nemtriviális másodrendű modell, mely Russell.*

105. Gondolkodnivaló. *A következő nyelvek (elsőrendű) szintaxisában minden termnek van normálformája: egyszerű típuselmélet, System F, Hindley–Milner, Martin-Löf-típuselmélet.*

5. Másodrendű általánosított algebra szignatúrák megadása másodrendű általánosított algebrai elmélettel

Ebben a fejezetben univerzális algebrával foglalkozunk. Ahogy a 20. definícióban megadtuk az algebrai elméletek szignatúráit, most megint megadjuk az AT szignatúrákat, de a szignatúrák nyelve egy konkrét SOGAT szintaxisa lesz. Látni fogjuk, hogy ez kényelmes definícióhoz vezet, ahol kevesebb kódolási költség van. A SOGAT-tal megadott szignatúra módszer továbbá nem csak AT szignatúrákra, hanem GAT és SOGAT szignatúrákra is működik.

A következő SOGAT szintaxisa írja le az algebrai elméletek szignatúráit, ez egy alternatívája a 20. definíciónak. Egy típus felel meg egy szignatúrának. Az algebrai szignatúrák nyelve Martin-Löf típuselméletének egy megszorítása: a Π típusnak például fix az értelmezési tartománya. Ezzel biztosítjuk, hogy ne lehessenek magasabbrendű függvények, és a függvények bemenete mindig az adott algebrai elmélet egyetlen szortja legyen. Egy alaptípusuk van, az algebrai elmélet szortja (Srt).

106. Definíció (AT)**.**

$$\begin{aligned}
\mathsf{Ty} &: \mathsf{Set}\\
\mathsf{Tm} &: \mathsf{Ty} \to \mathsf{Set}\\
\Sigma &: (A : \mathsf{Ty}) \to (\mathsf{Tm}\, A \to \mathsf{Ty}) \to \mathsf{Ty}\\
\mathsf{Srt} &: \mathsf{Ty}\\
\Pi\mathsf{Srt} &: (\mathsf{Tm}\,\mathsf{Srt} \to \mathsf{Ty}) \to \mathsf{Ty}\\
\cdot &: \mathsf{Tm}\,(\Pi\mathsf{Srt}\, B) \to (x : \mathsf{Tm}\,\mathsf{Srt}) \to \mathsf{Tm}\,(B\,x)\\
\mathsf{Id} &: \mathsf{Tm}\,\mathsf{Srt} \to \mathsf{Tm}\,\mathsf{Srt} \to \mathsf{Ty}
\end{aligned}$$

Minden $B : \mathsf{Ty}$-ra bevezetjük a $\mathsf{Srt} \Rightarrow B := \Pi\mathsf{Srt}\,(\lambda\_.B)$ rövidítést.

Például a félcsoportok (1. definíció) a következőképp adhatók meg, a szignatúra Ty egy eleme.

$$\Sigma\,(\mathsf{Srt} \Rightarrow \mathsf{Srt} \Rightarrow \mathsf{Srt})\lambda op.$$
$$\Pi\mathsf{Srt}\,\lambda x.\Pi\mathsf{Srt}\,\lambda y.\Pi\mathsf{Srt}\,\lambda z.\mathsf{Id}\,\big(op \cdot (op \cdot x \cdot y) \cdot z\big)\,\big(op \cdot x \cdot (op \cdot y \cdot z)\big)$$

Összehasonlítva az 1. definícióval, a szortot nem kell megadnunk, hiszen algebrai elméletekben (AT-ben) csak egy szort van, a Σ típus első komponense a bináris operátor lesz, melynek curry-zett típusát megadjuk és *op*-nak nevezzük el. A Σ típus második komponense kvantifikál három Srt-beli elem fölött, majd visszaadja az egyenlőség (identitás) típus megfelelő elemét. az 1. definícióban megadott verzió tekinthető ennek a formális definíciónak a kényelmesebb megadásának.

107. Feladat. *Adjuk meg az 1. fejezetben megadott összes AT szignatúráját a fenti módon!*

108. Megjegyzés. *A definíciónk szerint az üres algebrai elmélet (egy szort, nulla operátor) nem AT. Ez is bevehető, ha az AT elmélethez hozzávesszük az egyelemű $\top$ típust.*

A következő SOGAT szintaxisa írja le a zárt GAT-ok szignatúráit. Egy típus felel meg egy szignatúrának, a szignatúra szortjai U-ban vannak, az operátorait (és az indexelt szortokat) a Π függvénytípussal írjuk le.

109. Definíció (Zárt GAT)**.**

$$\begin{aligned}
\mathsf{Ty} &: \mathsf{Set}\\
\mathsf{Tm} &: \mathsf{Ty} \to \mathsf{Set}
\end{aligned}$$

$$
\begin{aligned}
\Sigma \quad &: (A : \mathsf{Ty}) \to (\mathsf{Tm}\, A \to \mathsf{Ty}) \to \mathsf{Ty} \\
(-,-) &: (a : \mathsf{Tm}\, A) \times \mathsf{Tm}\,(B\, a) \simeq \mathsf{Tm}\,(\Sigma\, A\, B) : \mathsf{fst}, \mathsf{snd} \\
\mathsf{U} \quad &: \mathsf{Ty} \\
\mathsf{El} \quad &: \mathsf{Tm}\, \mathsf{U} \to \mathsf{Ty} \\
\Pi \quad &: (a : \mathsf{Tm}\, \mathsf{U}) \to (\mathsf{Tm}\,(\mathsf{El}\, a) \to \mathsf{Ty}) \to \mathsf{Ty} \\
- \cdot - \quad &: \mathsf{Tm}\,(\Pi\, a\, B) \to (x : \mathsf{Tm}\,(\mathsf{El}\, a)) \to \mathsf{Tm}\,(B\, x) \\
\mathsf{Id} \quad &: (a : \mathsf{Tm}\, \mathsf{U}) \to \mathsf{Tm}\,(\mathsf{El}\, a) \to \mathsf{Tm}\,(\mathsf{El}\, a) \to \mathsf{Ty} \\
\mathsf{reflect} &: \mathsf{Tm}\,(\mathsf{Id}\, a\, u\, v) \to u = v
\end{aligned}
$$

Itt is használjuk a Martin-Löf-típuselméletnél bevezetett $A \Rightarrow B := \Pi\, A\, (\lambda\_.B)$ és $A \times B := \Sigma\, A\, (\lambda\_.B)$ rövidítéseket.

Például az előrendezés (40. definíció) szignatúrája a fenti SOGAT szintaxisában a következő típus (Ty egy eleme):

$$
\begin{aligned}
&\Sigma\, \mathsf{U}\, \lambda Ob.\Sigma \\
&\quad (Ob \Rightarrow Ob \Rightarrow \mathsf{U})\, \lambda Mor. \\
&\quad \big(\Pi\, Ob\, \lambda I.\Pi\, Ob\, \lambda J.\Pi\, Ob\, \lambda K.Mor \cdot J \cdot I \Rightarrow Mor \cdot K \cdot J \Rightarrow \\
&\quad\ \mathsf{El}\,(Mor \cdot K \cdot I)\big) \times \\
&\quad \big(\Pi\, Ob\, \lambda I.\mathsf{El}\,(Mor \cdot I \cdot I)\big) \times \\
&\quad \big(\Pi\, Ob\, \lambda I.\Pi\, Ob\, \lambda J.\Pi\,(Mor \cdot J \cdot I)\, \lambda f.\Pi\,(Mor \cdot J \cdot I)\, \lambda g. \\
&\quad\ \mathsf{Id}\,(Mor \cdot J \cdot I)\, f\, g\big)
\end{aligned}
$$

Látjuk, hogy a felsorolás helyett Σ típusokat használunk, kötéseknél metanyelvi λ-t, nincsenek implicit paramétereink, és ki kell írnunk az El-t, amikor egy U típusú termből típust csinálunk. Mindez csak egy nagyon explicit leírás a 40. definícióra.

110. Feladat. *Adjuk meg a 2. fejezetben megadott összes GAT szignatúráját a fenti módon!*

111. Definíció (Nyílt GAT)**.** *A zárt GAT szignatúrák elméletét az alábbiakkal egészítjük ki:*

$$
\begin{aligned}
\hat{\Pi} \quad &: (T : \mathsf{Set}) \to (T \to \mathsf{Ty}) \to \mathsf{Ty} \\
- \hat{} - &: \mathsf{Tm}\,(\hat{\Pi}\, T\, B) \to (\alpha : T) \to \mathsf{Tm}\,(B\, \alpha)
\end{aligned}
$$

A nyílt általánosított algebrai elméletek már nem adhatók meg nyílt másodrendű általánosított algebrai elméletként, csak végtelenül elága-

zódó (infinitary) másodrendű általánosított algebrai elméletként, de ezzel semmi gond nincs, lásd [10].

A másodrendű általánosított algebrai elméleteket a következő szignatúrával adjuk meg. Van egy új szortunk, U^+, az ebben levő szortokat lehet másodrendű függvénytérben használni, más néven az U^+-ban levő szortjainkhoz tartoznak változók. Persze tehetjük az összes szortot U^+-ba, ezzel a szemlélettel írtuk le a korábbi SOGAT-jainkat.

112. Definíció (SOGAT)**.** *A zárt GAT szignatúrák elméletét az alábbiakkal egészítjük ki:*

$$
\begin{aligned}
&\mathsf{U}^+ && : \mathsf{Ty}\\
&\mathsf{el}^+ && : \mathsf{Tm}\,\mathsf{U}^+ \to \mathsf{Tm}\,\mathsf{U}\\
&\pi^+ && : (a^+ : \mathsf{Tm}\,\mathsf{U}^+) \to \Big(\mathsf{Tm}\,\big(\mathsf{El}\,(\mathsf{el}^+\,a^+)\big) \to \mathsf{Tm}\,\mathsf{U}\Big) \to \mathsf{Tm}\,\mathsf{U}\\
&\mathsf{lam}^+ && : \Big(\big(x : \mathsf{El}\,(\mathsf{el}^+\,a^+)\big) \to \mathsf{Tm}\,(\mathsf{El}\,(b\,x))\Big) \simeq \mathsf{Tm}\,\big(\mathsf{El}\,(\pi^+\,a^+\,b)\big) : \_ \cdot^+ \_
\end{aligned}
$$

Például az egyszerű típuselmélet (60. definíció) szignatúrája a következő (mivel típusok nem szerepelnek nyíl bal oldalán, más szóval nincsenek típusváltozóink, ezért a típusok szortja U-ban van).

$$
\begin{aligned}
&\Sigma\,\mathsf{U}\lambda Ty.\Sigma\\
&(Ty \Rightarrow \mathsf{U}^+)\,\lambda Tm.\\
&\mathsf{El}\,Ty \times \Sigma\\
&(Ty \Rightarrow Ty \Rightarrow \mathsf{El}\,Ty).\lambda arr.\Sigma\\
&\Big(\Pi\,Ty\,\lambda A.\Pi\,Ty\,\lambda B.\\
&\quad \big(Tm\cdot A \Rightarrow^+ \mathsf{el}^+\,(Tm\cdot B)\big) \Rightarrow \mathsf{El}\,\big(\mathsf{el}^+\,(Tm\cdot(arr\cdot A\cdot B))\big)\Big)\,\lambda lam.\Sigma\\
&\Big(\Pi\,Ty\,\lambda A.\Pi\,Ty\,\lambda B.\\
&\quad \mathsf{el}^+\,(Tm\cdot(arr\cdot A\cdot B)) \Rightarrow \mathsf{el}^+\,(Tm\cdot A) \Rightarrow \mathsf{El}\,\big(\mathsf{el}^+\,(Tm\cdot B)\big)\Big)\,\lambda app.\\
&\Big(\Pi\,Ty\,\lambda A.\Pi\,Ty\,\lambda B.\Pi\,\big(Tm\cdot A \Rightarrow^+ \mathsf{el}^+\,(Tm\cdot B)\big)\,\lambda b.\\
&\quad \Pi\,\big(\mathsf{el}^+\,(Tm\cdot A)\big)\,\lambda a.\\
&\quad \mathsf{Id}\,\big(\mathsf{el}^+\,(Tm\cdot B)\big)\,\big(app\cdot A\cdot B\cdot(lam\cdot A\cdot B\cdot b)\cdot a\big)\,(b\cdot^+ a)\Big)\times
\end{aligned}
$$

$$\Big(\Pi\, Ty\, \lambda A.\Pi\, Ty\, \lambda B.\Pi\, \big(Tm \cdot (arr \cdot A \cdot B)\big)\, \lambda f.$$
$$\mathsf{Id}\, \Big(\mathsf{el}^{+}\, \big(Tm \cdot (arr \cdot A \cdot B)\big)\Big)$$
$$f\, \big(lam \cdot A \cdot B \cdot (\mathsf{lam}^{+}\, \lambda x.app \cdot A \cdot B \cdot f \cdot x)\big)\Big)$$

113. Feladat. *Mutassuk meg, hogy a 4. fejezetben megadott összes nyelv leírható SOGAT szignatúrával. Csak azokat a szortokat tegyük* U^{+}*-ba, melyeket feltétlenül szükséges (például egyszerű típuselméletnél vagy Martin-Löf-típuselméletnél csak* Tm *van* U^{+}*-ban, míg System F-nél* Ty *is).*

114. Feladat. *Mutassuk meg, hogy a SOGAT-okat leíró SOGAT megadható SOGAT-ként (saját farkába harap a kígyó).*

115. Gondolkodnivaló. *Adjuk meg a SOAT szignatúrákat SOGAT-leírással!*

116. Gondolkodnivaló. *Tetszőleges másodrendű általánosított algebrai elmélet másodrendű modelljéből képezhető nemtriviális elsőrendű modell. Lásd [4].*

117. Gondolkodnivaló. *A Martin-Löf-típuselméletet egyszer szeretnénk SOGAT-okkal kiegészíteni: például olyan szabályokkal, melyek tetszőleges SOGAT szignatúrához megadják annak elsőrendű szintaxisát.*

Hivatkozások

[1] Thorsten Altenkirch, Ambrus Kaposi, Artjoms Sinkarovs, Tamás Végh, Combinatory logic and lambda calculus are equal, algebraically, Eds. Marco Gaboardi, Femke van Raamsdonk, *FSCD 2023, July 3–6, 2023, Rome, Italy,* 260 *LIPIcs* (2023), pp. 24:1–24:19.

[2] Robert Atkey, Syntax and semantics of quantitative type theory, Eds. Anuj Dawar, Erich Grädel, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09–12, 2018,* pp. 56–65.

[3] Henk Barendregt, Introduction to generalized type systems, *J. Funct. Program.,* 1(2) (1991), pp. 125–154.

[4] Rafaël Bocquet, Ambrus Kaposi, Christian Sattler, For the metatheory of type theory, internal sconing is enough, Eds. Marco Gaboardi, Femke van Raamsdonk, *FSCD 2023, July 3–6, 2023, Rome, Italy*, 260 *LIPIcs* (2023), pp. 18:1–18:23.

[5] John Cartmell, Generalised algebraic theories and contextual categories, *Ann. Pure Appl. Log.,* 32 (1986), pp. 209–243.

[6] Jean-Yves Girard, Linear logic, *Theor. Comput. Sci.,* 50 (1987), pp. 1–102.

[7] Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, Lars Birkedal, Multimodal dependent type theory, *Log. Methods Comput. Sci.,* 17(3) (2021).

[8] Robert Harper, *Practical Foundations for Programming Languages,* Cambridge University Press, New York, NY, USA, 2nd edition, 2016.

[9] Jasper Hugunin, Why not W?, Eds. Ugo de'Liguoro, Stefano Berardi, Thorsten Altenkirch, *TYPES 2020, March 2–5, 2020, University of Turin, Italy,* 188 *LIPIcs,* (2020) pp. 8:1–8:9.

[10] András Kovács, Type-theoretic signatures for algebraic theories and inductive types, *CoRR,* abs/2302.08837, 2023.

[11] John Longley, Dag Normann, *Higher-Order Computability,* Theory and Applications of Computability, Springer, 2015.

[12] Benjamin C. Pierce, *Types and programming languages,* MIT Press, 2002.

[13] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, Brent Yorgey, *Logical Foundations,* 1 *Software Foundations*. Electronic textbook, 2023. Version 6.5. `http://softwarefoundations.cis.upenn.edu`

[14] Taichi Uemura, A general framework for the semantics of type theory, *CoRR,* abs/1904.04097, 2019.

[15] Philip Wadler, Wen Kokke, Jeremy G. Siek, *Programming Language Foundations in Agda,* August 2022.

RSA akkumulátorok decentralizációjának vizsgálata[‡]

Kocsis Ábel[*]

Eötvös József Collegium[**]

abelko@duck.com

Témavezetők: Myrto Arapinis és Nicolaos Labrou
University of Edinburgh

Az E-cclesia egy önjegyző e-szavazó protokoll, mely RSA akkumulátorokat használ a szavazók hitelesítésére. A RSA akkumulátor modulusának generálása azonban lehetőséget nyújt a központi Választási Szervnek, hogy érvénytelen szavazatokat adjon le.

Ezen cikk bemutatja, hogy miképpen lehet az RSA akkumulátorok modulusának generálását olyan módon végezni, hogy a szám teljes prímfelbontása senkinek se álljon rendelkezésére. A cikkben ezt RSA-UFO-k használatával, vagy a szavazásban résztvevők bevonásával járó új protokoll tervezésével érjük el. Ezen túl a cikk kitér a tervezett protokollok elemzésére mind biztonság, mind hatékonyság szempontjából.

1. Bevezetés

A demokrácia megszületése óta a szavazás fontos része a különböző társadalmaknak, ahol a szavazati joggal rendelkezők döntenek különböző fontos kérdésekről. Sokan azonban gyakran megkérdőjelezik a

[‡] A szerző Edinburghi Egyetemen készült diplomamunkája első felének magyar nyelvű összefoglalója.

[*] ELTE Informatikai Kar (2017–2020), University of Edinburgh (2020–2021)

[**] 2017–2020

szavazások végeredményének igazságosságát [25], valamint a postai szavazatokkal kapcsolatban is számos kérdés felmerül [29].

Bár több nemzeti szintű elektronikus szavazás (e-szavazás) lebonyolításra került már [14, 15], ezeket központi szerverek bonyolították le, és eredményük ugyancsak megkérdőjelezhető [8]. Emiatt tehát hiába tűnhet technikai előrelépésnek egy e-szavazás, amíg egy központi intézmény felel ennek lebonyolításáért, legalább annyi támadás érheti a rendszert, mint egy hagyományos offline választás esetén.

Számos elektronikus nem teljesen centralizált szavazóprotokoll létezik, melyek közül az E-CCLESIA több tulajdonságával kiemelkedik: bizonyítható, például bármely szavazó esetén matematikailag belátható, hogy a szavazat beszámításra került a végeredménybe, valamint anonim: a szavazókról nem lehet megállapítani, hogy mire szavaztak [2]. A protokoll úgynevezett RSA akkumulátort használ a szavazók névtelen hitelesítéséhez. A protokollban azonban ezen RSA akkumulátor felállításához szükség van egy központi Választási Szervre (**VSZ**). A **VSZ**-nek azonban ezen feladatát kihasználva lehetősége adódik arra, hogy megtámadja a protokollt, és annyiszor szavazzon, ahányszor csak akar.

Ezen cikk az Edinburghi Egyetemi diplomamunkám [11] első felének összefoglalója. A kutatásom célja az E-CCLESIA protokoll olyan módon való fejlesztése és vizsgálata volt, hogy a központi **VSZ** felé semmilyen bizalom ne legyen szükséges, azaz **VSZ** képtelen legyen a rendszer megtámadására. Egy ilyen protokoll dizájnján kívül igyekeztem megválaszolni azt a kérdést is, hogy egy ilyen algoritmus használható lenne-e a való világban, illetve milyen hátrányai lennének.

A cikk felépítése a következő. Először a 2. fejezetben bemutatom az elektronikus szavazórendszereket általánosságban, valamint az E-CCLESIA protokollt nagy vonalakban. Ezután a 3. fejezet felvázolja a kutatás eredményeként kapott néhány protokollt, valamint azoknak biztonsági és gyakorlati tulajdonságait. Végül pedig a 4. fejezet összefoglalja diplomamunkám ezen részét.

2. Háttér

A fejezetben áttekintjük az e-szavazórendszerek általános jellemzőit és kihívásait (2.1. alfejezet), ezután pedig rátérünk az E-CCLESIA szavazóprotokoll bemutatására a 2.2. alfejezetben.

2.1. E-szavazórendszerek

E-szavazórendszereknek az olyan szavazórendszereket vagy protokollokat hívjuk, ahol a szavazás menete többnyire elektronikus eszközökön alapul [8]. Ezen rendszerektől ideális esetben a következő tulajdonságokat várjuk el a források alapján [13]:

1. *Helyesség*: csak a jogosultsággal rendelkező résztvevők szavazhatnak, mindegyik résztvevő egyszer szavazhat, a győztes az, aki a legtöbb szavazatot kapta;

2. *Bizonyíthatóság*: univerzálisan bárki be tudja bizonyítani, hogy a szavazás végeredménye a helyes végeredmény. Ezentúl *privát bizonyíthatóságnak* nevezzük, amikor bármely szavazó be tudja bizonyítani, hogy az ő szavazata bele lett számolva a végeredménybe;

3. *Szavazóanonimitás*: a szavazó kiléte nem kapcsolható a leadott szavazatához;

4. *Bizonylatmentesség*: a szavazó nem tudja bebizonyítani, hogy egy adott módon szavazott. Ezen tulajdonság a szavazatvásárlás megelőzése miatt fontos.

Ezentúl egy *fairség* tulajdonságot is meg lehet követelni egy e-szavazó rendszertől, ami elvárja a protokolltól, hogy ne szivárogtasson az eredményről semmilyen részeredményt akár időben (részeredmény publikálása), akár térben (adott kerület melyik jelöltre szavazott) [12].

Mivel a centralizált szavazórendszerek sokszor számos támadási faktornak vannak kitéve [3, 22], teljesen decentralizált szavazóprotokoll azonban jelen tudásunk alapján nem ismert, kénytelenek vagyunk a két szélsőség között vizsgálódni. Az *önjegyző szavazóprotokollok*, amit elsőként Kiayias és Yung írt le [10], olyan szavazóprotokollok csoportja, ahol a szavazás elején szükség van egy központi Választási Szervre (**VSZ**), de a szavazás felállítása után a szavazók maguk publikálják valamilyen formában szavazatukat, számítják ki az eredményt, és bizonyítják a protokoll helyes működését. A számos önjegyző protokoll [9, 18, 21] egyike az E-CCLESIA [2].

2.2. E-cclesia, egy önjegyző szavazóprotokoll

Az E-CCLESIA tehát egy önjegyző protokoll, melyet elsőként Arapinis és társai mutattak be [2]. A protokoll bizonyítottan teljesíti a helyesség, szavazóanonimitás [16], valamint a fairség feltételeket.

Fogalmak

A protokollok és kriptográfiai eszközök magyarázata során a következő fogalmakra fogunk építeni. $\mathbf{V_i}$ egy szavazót jelöl, **PT** pedig egy publikus tábla, amire bárki tud írni, de amiről törölni senki sem tud (például blokklánc). $a \leftarrow \texttt{Alg}(params)$ jelentése, hogy az `Alg` algoritmus futtatva lett *params* paraméterekre, és a futás eredménye a. $\{m\}_P$ egy m üzenet P résztvevő által digitálisan aláírva, ami azt jelenti, hogy bárki által kriptográfiailag ellenőrizhető, hogy az üzenetet P hitelesítette.

A Zerocointól kölcsönzött ötletek

Az E-CCLESIA nagyban épít a Zerocoin [17] protokoll által bemutatott ötletekre, ami RSA akkumulátorokat [5] és Pedersen elköteleződés sémáját [19] használja egy decentralizált pénzügyi rendszer létrehozására. Bár sebezhetőségek miatt a Zerocoin használata már kifejezetten nem javasolt, kisebb módosításokkal a használt kriptográfiai primitíveket fel lehet használni egy szavazórendszer tervezéséhez.

Pedersen elköteleződés sémája

Pedersen elköteleződés sémája [19] röviden egy „ígéret" érték c egy titkos érték S-hez. A Zerocoin protokollban egy titkos sorozatszám S_i-hez tartozó ígéret c_i publikálásra kerül minden résztvevőtől. Ezután minden c_i értékhez egy tanú w_i generálható, ami felhasználható egy nem interaktív nem feltáró bizonyítás [6,17] generálására. Így egy résztvevő tehát ha publikálta c_i-t, később be tudja bizonyítani, hogy van jogosultsága a protokollban részt venni anélkül, hogy felfedné kilétét, azaz hogy melyik c_i-hez tartozó tanú értéket használja. Az elköteleződés séma miatt azonban a publikált c_i-hez tartozó S_i-t változtatni már nem lehet.

Ezen elköteleződés séma RSA akkumulátort használ, ami azonban függ egy előre meghatározott RSA szám N-től, aminek faktorizációjának kellően nehéznek kell lenni ahhoz, hogy a protokoll biztonságos legyen. Ezt az RSA számot a rendszer felállításakor egy központi szerv generálja, aki ezután megsemmisíti ennek a prímtényezőit. Ha azonban nem így tesz, vagy a prímtényezők kiszivárognak, amire nincs semmilyen ellenbizonyítéka a résztvevőknek, a prímfelbontással rendelkező személy könnyedén megtámadhatja a protokollt, ahogy erre a szerzők is kitérnek [17]. Erre egy megoldás lehet RSA-UFO-k használata.

RSA akkumulátorok

Az RSA akkumulátorok [4–6] olyan kriptográfiai eszközök, melyek résztvevők hitelesítésére használhatóak konstans futási időben. Az RSA akkumulátorok építőblokkjai a következők: bázis érték u; résztvevők jogosultságát igazoló prímek c_i, és egy RSA szám modulus N. Ekkor az akkumulátor érték $a = u^{\Pi c_i} \mod N$.

Az akkumulátor érték a, modulus érték N, a bázis szám u, valamint a c_i értékek publikusan elérhetők minden résztvevő számára. Ahhoz, hogy egy résztvevő bebizonyítsa, hogy a hozzá tartozó c_i része az akkumulátornak, egy tanú értékre van szüksége, ami a $w_i = u^{c_1 \cdots c_{i-1} \cdot c_{i+1} \cdots c_v} \mod N$. Ezen tanú érték fontos tulajdonsága, hogy $w_i^{c_i} \mod N = a$, valamint bizonyított, hogy egy x, y pár megtalálása, ahol $x^y \mod N = a$ egy matematikailag nehéz probléma az N faktorizációja nélkül [4].

Az E-cclesia szakaszai

Ebben a fejezetben az E-cclesia protokoll szakaszai kerülnek nagy vonalakban bemutatásra. Az E-cclesia biztonságát részletesebben formálisan és informálisan más források taglalják [2, 11, 16].

A protokoll alapvetően négy fázisból áll. Ezek a fázisok időben elkülönülnek: csak az előző hivatalos lezárulta után kezdődik el a következő. Ezen szakaszok kezdő és befejező időpontjai előre bejelentésre kerülnek a **VSZ** által. Az első szakasz, tehát a protokoll megkezdése előtt minden szavazó regisztrációra kerül személyesen, ahol megkapja a személyre szóló azonosító számát.

Az első fázis a **Kezdeti beállítás** szakasz, ahol a **VSZ** generál különböző paramétereket a szavazáshoz, és ezeket közzéteszi. Ezen pa-

raméterek közül kiemelten fontos az RSA szám modulus $N = p \cdot q$ generálása, ahol p es q (egyéb feltételek mellett) nagy prímszámok. A **VSZ** az N számot publikálja, a p és q prímeket pedig törli. A **VSZ**-nek egyedül ebben a fázisban van szerepe, ezután minden ellenőrző vagy összesítő tevékenység minden szavazó vagy megfigyelő által elvégezhető.

A második fázis a **Bizonyítvány generálás** fázis, ahol minden szavazó generál egy jogosultságot igazoló értéket, c_i-t, mely függ a szavazás paramétereitől és az általa választott S_i szériaszám és r_i véletlen szám értékektől. Ezen c_i értéket minden szavazó aláírva, $\{c_i\}_{v_i}$ formában publikálja a **PT**-ra, míg az (S_i, r_i) párt titokban tarja. A protokoll során ez az egyetlen rész, amikor a szavazó aláírásával felfedi kilétét. Ezen részben biztosítható, hogy minden szavazó legfeljebb egy c_i értéket publikál, valamint itt ellenőrizhető, hogy csak erre jogosultak publikálnak ilyen értéket. Fontos, hogy a később felhasználásra kerülő S_i érték nem kapcsolható össze a hozzá tartozó c_i-vel.

A harmadik fázisban, ami a **Szavazás** fázis, minden szavazó leadja szavazatát a következő módon: miután választott egy o_i résztvevőt akire szavazni szeretne, generál egy szavazatot: $v_i \leftarrow \texttt{GenBallot}(params, o_i)$. Ezután a szavazatát hitelesíti: $\sigma_i \leftarrow \texttt{AuthBallot}(v_i, S_i, r_i, params)$, majd a szavazatot (v_i), hitelesítést (σ_i), valamint a hozzá tartozó szériaszámot (S_i) publikálja. A hitelesítés egy nem feltáró bizonyítás, ami garantálja, hogy ezen értékek csak akkor helyesek, ha létezik korábban publikált c_i érték ami egy elköteleződés volt (S_i, r_i)-re, de a bizonyítás nem fedi fel, hogy mely c_i-hez tartozik (S_i, r_i), tehát mely szavazó adta le ezt a szavazatot. Ezekkel együtt a lépés biztosítja, hogy minden szavazó legfeljebb egyszer tud szavazni, és senki más nem tud leadni szavazatot mint az erre jogosultak.

Az `AuthBallot` függvény futtatása közben a következő történik: a szavazó kiszámolja a c_i-hez tartozó tanú értéket (w_i), amit felhasznál a nem feltáró bizonyításhoz. A tanú kiszámítása a protokoll legnagyobb számítási igényű része, ugyanis a szavazónak a $w_i = u^{c_0 \cdot c_1 \cdots c_{i-1} \cdot c_{i+1} \cdots c_v} \mod N$ értéket kell kiszámítania, ami a szavazószám és N nagyságától függően nő. Vegyük azonban észre, hogy a w_i értéke csak publikus értékektől függ (N, u, c_i), ezért ezt bárki kiszámíthatná, de a bizonyításhoz már szükség van a titkos (S_i, r_i) értékekre.

Az utolsó, **Összeszámlálás** részben a szavazók kinyitják szavazataikat, és összeszámolják a végeredményt. A kinyitatlan szavazatokat a közösség fel tudja törni, amire a `GenBallot` által használt időzáras

kódolás miatt van lehetőség. Ebben a részben tehát bárki számára lehetőség nyílik a szavazás végkimenetelének ellenőrzésére.

Az E-CCLESIA protokoll bizonyítottan biztonságos [16] a „Universal Composable" keretrendszerben [7].

2.3. Probléma és a cikk fókusza

Az eredeti E-CCLESIA protokollban tehát egy centrális **VSZ** generálja a hitelesítéshez szükséges RSA modulus N-et. Ez azonban lehetőséget ad arra, hogy **VSZ** tetszőleges számú érvényes szavazatot generáljon a Szavazás fázisban, tehát csaljon [11, 3.1. fejezet]. Hogy megtegyük a következő lépést a decentralizáció felé, a következő kérdést kell feltennünk: lehetséges-e egy N RSA vagy RSA-szerű szám generálása úgy, hogy egyik résztvevő sem tudja a teljes faktorizálását a generált N-nek? Ha igen, használható-e a gyakorlatban egy ilyen algoritmus? Milyen következményei lennének egy ilyen stratégiának a protokoll biztonságára?

A cikk következő fejezete ezen kérdésekre igyekszik választ adni, bemutatva a diplomamunka eredményei közül néhányat. A munka egyes részei, melyek átmeneti eredmények voltak, kihagyásra kerülnek. Ugyancsak kihagyásra kerül az E-CCLESIA protokollhoz csak lazán kötődő egyidejű broadcast csatorna rész [11, 4. fejezet], melyben egy ilyen csatorna implementálását mutattam be a „Universally Composable" keretrendszerben [7], és mely munka eredményeképp később közös publikáció is született [1].

3. Megoldási lehetőségek

Ezen fejezet bemutatja az E-CCLESIA decentralizálásához szükséges következő lépés lehetőségeit, a modulus N szám decentralizált generálására talált megoldásokat.

3.1. RSA-UFO-k használata

RSA akkumulátorok decentralizálása nem egy teljesen új feladat. Sander tanulmányában RSA-UFO-knak nevezte azon RSA akkumulátorokat, melyek esetében egyik résztvevő sem ismeri a modulus prímfelbontását [20]. Mivel annak valószínűségére, hogy egy adott nagy szám

RSA szám létezik számelméleti alsó becslés ($P[a$ RSA szám$] > p$), Sander ötlete alapján ha megfelelő mennyiségű (n) nagy számot veszünk, annak valószínűségét is tudjuk alulról becsülni, hogy ezeknek legalább egyike RSA szám: $(1-(1-p)^n)$. Ezen számok szorzata pedig, bár nem RSA szám, mégis legalább ugyanolyan nehéz a teljes faktorizációja, mint a benne található legnehezebben faktorizálható számnak. Ha tehát ezt a szorzatot használjuk az akkumulátor modulusának, $(1-(1-p)^n)$ valószínűséggel a kapott szorzatot legalább olyan nehéz teljesen felbontani prímtényezőkre, mint egy RSA számot, tehát bizonyos valószínűséggel egyik résztvevő sem tudja a modulus prímfelbontását.

Ezt ez ötletet felhasználhatjuk az E-CCLESIA esetén is. Ha a központi **VSZ** generálása helyett ugyanis a modulus egy előre meghatározott publikus forrás véletlen számai közül n darab, például egy adott blokklánc hash értékei, akkor azzal a modulus egy RSA-UFO, melynek faktorizációjának megfelelősége az n tekintetében becsülhető.

Ezen stratégia elméleti hátránya így egyből az ölünkbe hullik: mivel véletlenszerű számok mennyiségének függvényében lehet a hibalehetőséget becsülni, a protokoll sosem érhet el tökéletes biztonságot. Valamekkora esély tehát mindig lesz arra, hogy a generált szám felbontása nem megfelelően nehéz, és a protokoll feltörhető.

Ezen protokoll gyakorlati vizsgálatához tudnunk kell, hogy egy véletlenszerű szám RSA valószínűsége alulról becsülhető a $p = 0.08$ értékkel [11, 20]. A számításokat elvégezve azt kapjuk, hogy ha legfeljebb $\frac{1}{2^l}$ hibalehetőséggel szeretnénk RSA-UFO-t generálni, legalább $n = \left\lfloor \frac{-l}{\log_2(1-p)} \right\rfloor + 1$ véletlen számra lesz szükségünk [11, 3.3.1. fejezet].

A vizsgálatból azonban az adódik, hogy már nagyon kis biztonsági paraméterek esetén ($RSA\ prímtényező\ bithosszúság = 512, l = 80, szavazószám = 5$), bár az RSA-UFO szám generálása rövid időbe telik ($< 2s$), a w_i tanúgenerálás c_i-hez N RSA-UFO esetén több, mint 10 percet vesz igénybe. Növelve a paramétereket természetesen a számítási igény is nő, ezért ezt a megoldást nem tudjuk felhasználni a gyakorlatban az E-CCLESIA esetén.

3.2. Modulus generálása a szavazók által

Vegyük észre, hogy Sanders ötlete azért eredményezett túl hosszú tanúszámításokat, mert a modulus N túl nagy volt a véletlenszerű szá-

mok szorzása miatt. Az ötletet azonban, miszerint ha legalább egy szám faktorizálása az N létrehozásakor RSA szám, akkor a végső N faktorizációja egy nehéz feladat, fel tudjuk használni másképpen is.

Ehhez munkám következő fejezetében bemutattam egy algoritmust, ahol a szavazók mindegyike generál egy RSA számot, legyen N_i, majd a végső modulust ezen számok szorzata adja ki $N = \Pi N_i$ [11, 3.4 fejezet]. Ekkor a modulus egyedül akkor nem biztonságos, ha egyik szavazó sem megbízható: mindenki vagy közzéteszi az általa generált szám prímfaktorizációját, vagy eleve nem RSA számot generál. Ha senki sem megbízható, akkor viszont az egész szavazás értelmetlen, tehát bátran állíthatjuk, hogy ebben az esetben elértük a tökéletes decentralizációt.

Ez a módszer azonban 100 fölötti résztvevő esetén már ismét olyan nagy modulust hozna létre, ami a tanúgenerálás idejét megnövelné több, mint 10 percre, ezért ugyancsak nem alkalmazható a gyakorlatban. Vegyük észre, hogy ebben az esetben legalább egy őszinte szavazónak részt kellett vennie a generálásban, hogy a modulus biztonságos legyen, a többiekről pedig feltételezhettük, hogy meg akarják támadni a protokollt.

A következő megoldási lehetőségben gyengítsük a feltételeinket, és tegyük fel, hogy egy nagyobb része őszinte a szavazóknak. Jelöljünk ki az összes v szavazó közül véletlenszerűen (például [11, 3.5.4 fejezet] szerint) k darabot, akik részt vesznek a modulus generálásában: ők mindannyian generálnak egy RSA számot és közzéteszik. Ekkor a végső modulus legyen $N = \Pi_{N_i \in N_s} N_i$, ahol N_s az RSA szám generálására kiválasztott szavazók által generált számok halmaza. A fentebb említett módszer az összes szavazó bevonásával egy szélsőséges esete ennek az ötletnek.

A módszer biztonságának vizsgálata

Mint azt korábban is láthattuk, a generált N modulus biztonságos, ha legalább egy faktora RSA szám. Ebből következik, hogy a protokoll biztonságos, ha legalább egy szavazó az RSA generálók közül megbízható: RSA számot generál, a faktorizációt titokban tartja, és törli. Fontos megemlíteni továbbá, hogy a protokoll megtámadásához nem elég, ha minden RSA szavazó rosszindulatú: nekik együtt is kell működni ahhoz, hogy megtalálják a modulus prímfaktorizációját.

Ugyancsak emeljük ki, hogy ha **VSZ** részese az RSA generálók halmazának, akkor a protokoll legalább annyira biztonságos, mint az eredeti E-CCLESIA protokoll, ahol az **VSZ** egyedül generálta a modulus értéket. Ezentúl lehet gondolkodni olyan megoldásokon, mint például különböző politikai pártok vagy semleges megfigyelők bevonása az RSA generálásába, ahol mindannyiuk rosszindulatú együttműködésének a valószínűsége kicsi.

Vizsgáljuk meg azonban elméleti esetben a protokoll biztonságát (S) különböző számú szavazók (v), rossz indulatú szavazók ($d \leq v$), valamint RSA generálók ($k \leq v$) esetén. Látható, hogy a skatulya elv alapján ha $d < k$, a protokoll mindig tökéletesen biztonságos. Vizsgáljuk tehát az esetet, amikor $d \geq k$:

$$\text{RSA generálók összes esete: } \binom{v}{k}$$

$$\text{Nem biztonságos RSA generálók (csak rosszindulatúak): } \binom{d}{k}$$

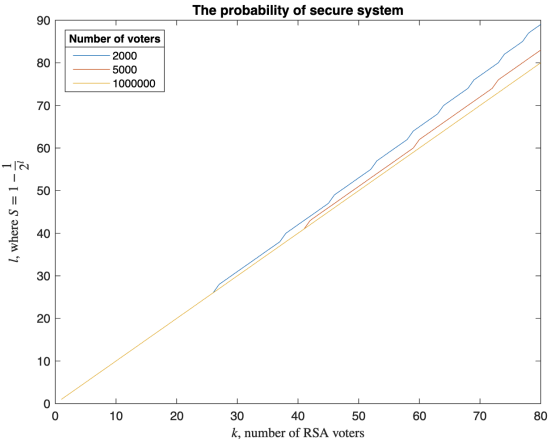
$$\overline{S} = \frac{\binom{d}{k}}{\binom{v}{k}} \quad \Longrightarrow \quad S = 1 - \frac{\binom{d}{k}}{\binom{v}{k}} =$$

$$= 1 - \frac{d!}{k!(d-k)!} \cdot \frac{k!(v-k)!}{v!} = 1 - \frac{d!\,(v-k)!}{(d-k)!\,v!} = 1 - \frac{\prod_{i=d-k+1}^{d} i}{\prod_{i=v-k+1}^{v} i}$$

Tegyük fel – mint ahogy azt gyakran megengedik –, hogy a szavazók fele rosszindulatú ($d = \frac{v}{2}$), és ők együttműködnek. Ebben az esetben:

$$S = 1 - \frac{\prod_{i=\frac{v}{2}-k+1}^{\frac{v}{2}} i}{\prod_{i=v-k+1}^{v} i} = 1 - \frac{(\frac{v}{2}-k+1)\cdots\frac{v}{2}}{(v-k+1)\cdots v} \geq 1 - \left(\frac{\frac{v}{2}-k+1}{v}\right)^k$$

Ezen eredményt felhasználva, egy nem biztonságos rendszer valószínűsége különböző RSA generálók függvényében az 1. ábrán látható. Minél nagyobb l, annál biztonságosabb a rendszer. Ahogy látható, ha a szavazók fele rosszindulatú és 1 millió szavazó vesz részt a szavazásban, 80 RSA generáló $1 - \frac{1}{2^{80}}$ valószínűségű biztonságot eredményez, ami a gyakorlatban elfogadható.



1. ábra. A rendszer biztonsága különböző szavazók és RSA generálók esetén. Minél nagyobb l, a protokoll annál biztonságosabb. [11, 24. oldal]

A módszer hatékonyságának vizsgálata

A módszer hatékonyságát, gyorsaságát egyrészt vizsgálhatjuk a modulus szám generálásának függvényében, másrészt pedig a tanú kiszámítása alapján. Mivel a modulus szám kiszámítása szinte mindig gyorsabb lesz, ezért a módszer hatékonyságát a tanú kiszámításának gyorsaságával mérjük (lásd 2.2. alfejezet).

Ezen számításokat egy néhány éves MacBook Pro modellen végeztük, feltételezésünk pedig, hogy egy átlagos szavazónak nem áll rendelkezésre erősebb számítógépe, és legfeljebb 10 percet hajlandó a szavazásra fordítani. Az implementációhoz C++ programozási nyelven a Bitcoin [24] és Zerocoin által is használt CBignum [26,27] osztályt használtuk, ami az OpenSSL könyvtárat használja [28].

A 1. táblázatban láthatjuk a tanúgenerálás idejét különböző számú szavazók és RSA generálók esetén. A teszt során a legkisebb elfogadható biztonsági paramétereket használtuk.

Ahogy a táblázatban is látszik, ha az RSA generálók száma nem túl nagy, például 5, a protokoll hatékonyan használható nagy számú szavazó esetén is, példánkban akár 25000. Így a tanúgenerálás egy szavazónak körülbelül 8 percbe telne.

Ugyancsak megemlítendő, hogy az RSA generálók számának változása gyorsan változtatja a tanúgenerálás gyorsaságát: több RSA generáló drasztikusan növeli egy tanú kiszámításának az idejét.

Szavazószám	RSA generálók			
	1	**10**	**20**	**30**
10	9ms	617ms	2s 265ms	5s 113ms
100	96ms	6s 214ms	23s 812ms	52s 893ms
500	463ms	30s 890ms	2m 2s	4m 30s
1 000	917ms	1m 0s	3m 55s	8m 51s
5 000	4s 471ms	5m 7s	19m 44	44m 27s
10 000	8s 828ms	10m 24s	39m 34s	
20 000	17s 325ms	21m 35s		
25 000	23s 609ms			
50 000	46s 25ms			
100 000	1m 27s			
700 000	10m 20s			
1 000 000	14m 44s			

1. táblázat. A tanú kiszámításának sebessége különböző számú szavazók és RSA generálók esetében [11, 25. oldal]

Párhuzamos tanúkalkuláció

A nagy modulusból származó alacsony hatékonyságot megpróbálhatjuk kiküszöbölni egy további, a korábbi protokolltól független stratégiával. Alkalmazhatjuk ugyanis az Anoncoin [23] által használt ötletet: egyetlen nagy N modulus helyett használjunk több kisebb N_j modulust, ahol ezek a számok az RSA generálók által generált számok, amiknek halmazát jelöljük N_s-vel, és amikre igaz, hogy $N = \Pi_{N_j \in N_s} N_j$. Ekkor egy szavazó a szokásos $w_i = u^{c_1 \cdots c_{i-1} \cdot c_{i+1} \cdots c_v} \mod N$ kiszámítása helyett ahhoz, hogy bebizonyítsa a szavazásra való jogosultságát, a következő értékeket kell kiszámítania: $w_{i,j} = u^{c_1 \cdots c_{i-1} \cdot c_{i+1} \cdots c_v} \mod N_j$ minden $N_j \in N_s$ esetre.

Vegyük észre azonban, hogy egy V_i által kiszámítandó $w_{i,1}$, $w_{i,2}$, $w_{i,3}, \ldots$ egymástól független értékek, tehát ezek kiszámítása párhuzamosítható. Ezt kihasználva a szavazók előnyt tudnak kovácsolni több

magos számítógépeikből vagy annak videókártyájából, és az összes tanúgenerálásra fordítandó idő is jelentősen csökkenthető.

A tanúgenerálás videókártyával ezen munkának nem része, de hogy érzékeltessük a módszer gyorsaságát, bemutatjuk egy CPU esetén egy ilyen tanú kiszámítását a 2. táblázatban párhuzamosítás nélkül. A teljes kiszámítás gyorsasága tehát majdnem annyival lenne gyorsabb, ahányszor több mag áll a rendelkezésünkre.

Szavazószám	RSA-2048		RSA-4096	
	p_1	p_2	p_1	p_2
10	9ms	21ms	34ms	63ms
100	96ms	185ms	349ms	701ms
25 000	23s 609ms	44s 191ms	1m 22s	2m 41s
50 000	46s 25ms	1m 27s	2m 45s	5m 19s
100 000	1m 27s	2m 53s	5m 27s	10m 50s
200 000	2m 54s	5m 43s	11m 0s	22m 26s
500 000	7m 16s	14m 38s	26m 58s	
700 000	10m 20s			
1 000 000	14m 44s			

2. táblázat. Egy tanú kiszámításához szükséges idő különböző p_1 és p_2 biztonsági paraméterek és RSA számok esetén [11, 26. oldal]

Ezen eredmények alapján végezzünk egy elméleti szintű becslést a tanúk kiszámítására. Tegyük fel a fenti táblázat alapján, hogy egy tanú kiszámítása $t_{|N|=4096,p=p_2}(w) \approx 63\,\text{ms}$. Egy ilyen videókártya használata 1000 maggal ($g = 1000$) és a CPU-hoz hasonló t-vel, $v = 50000$ szavazó esetén és $k = 100$ RSA generálónál az összes becsült idő az összes tanú kiszámítására $T \approx \left\lceil \frac{100}{1000} \right\rceil \cdot 50000 \cdot 63\,\text{ms} = 315000\ \text{s} = 5\ \text{p}\ 15\ \text{mp}$.

Egy hatékony párhuzamos tanú számító algoritmus esetén tehát láthatjuk, hogy az RSA generálók száma egy bizonyos fokig lényeges időnövekedés nélkül növelhető. A protokoll ilyen módon való gyakorlati fejlesztése azonban jövőbeni feladat marad.

3.3. Tanúgenerálás a központi VSZ által

Az eddigi fejezeteknél, ahogy az eredeti E-CCLESIA protokollban is, feltételeztük, hogy a tanúgenerálást a szavazók végzik lokálisan saját

elektronikus eszközeiken. Azonban ahogy ezt a 2.2. fejezetben is említettük, a tanú kiszámítását bárki meg tudja tenni, hiszen az ehhez szükséges N, u, és c_i értékek a publikus táblán szerepelnek.

Tehát egy másik opció egy biztonságos és hatékony protokoll tervezésére, ha a tanú számításból származó számítási nehézségeket áthelyezzük a központi **VSZ**-re. A **VSZ** tehát protokoll szerint kötelezhető lenne, hogy kiszámítsa a tanú értékeket minden c_i-hez a Szavazás fázis előtt, például egy szuperszámítógép segítségével. Ez a megoldás használható lenne például a 3.2. alfejezetben bemutatott ötlettel, ahol így választhatnánk magasabb számú RSA generáló halmazt.

A protokoll biztonságát vizsgálva azt tapasztaljuk, hogy ez a módszer nem gyengíti a biztonságot, ugyanis a nem feltáró bizonyítást továbbra sem képes kiszámítani az **VSZ** bármely c_i vagy w_i-hez tartozó (S_i, r_i) értékek hiányában. Ha a **VSZ** rosszindulatú, legrosszabb esetben érvénytelen tanút generál, amit a szavazók azonnal észre fognak venni, ugyanis nem fogják tu dni a szavazatukat leadni, és a **VSZ** csalását tudják bizonyítani a többi szavazó felé.

Ezen megoldás hatékonyságát nem áll módunkban vizsgálni, ugyanis a szerzőnek a cikk írásakor nem állnak rendelkezésére olyan gyors számítógépek, amivel az esetet tesztelni lehetne.

4. Összefoglalás

Ezen cikkben bemutattam tehát a diplomamunkám [11] egyes eredményeit, mely az E-CCLESIA [2] szavazó rendszer decentralizálásához vezető következő lépést vizsgálta. Ezen következő lépés az akkumulátor modulus decentralizált generálásával valósult meg, amire több példát is láthattunk.

Ahogy bemutattuk, ezen decentralizáció könnyen hátrányokkal is járhat. A 3.1. alfejezetben véletlen számokat generáltunk egy RSA-szerű szám létrehozásához, azonban ez olyan nagy modulus értéket eredményezett, hogy a protokoll gyakorlatban használhatatlanná vált, ezen felül tökéletes biztonságot sosem tudott elérni. A 3.2. alfejezetben bemutatott első ötlet alapján pedig, amikor az RSA modulust minden szavazó generálta, még mindig túl nagy számot kaptunk.

A 3.2. alfejezet hozott azonban áttörést is, amikor bevezettük az RSA szavazók fogalmát, ami a szavazók egy véletlenszerű részhalmaza, akik részt vesznek az RSA modulus generálásában. Ez a módszer ugyanis lényegesen hatékonyabbnak bizonyult, miközben életszerű feltételek mellett bár nem tökéletes, de elegendő biztonságot is nyújtott.

Ezt követően a 3.2. alfejezet bemutatta, hogy a gyakorlatban párhuzamos számításokkal hogyan lehet a tanúgenerálásra való időt jelentősen csökkenteni. Ehhez nagy teljesítményű, több magos CPU vagy GPU, illetve megfelelő szoftverre lenne szükség, de így a fent részletezett protokollok már nagy mennyiségű szavazó esetében is használhatóak lennének biztonságosan.

Végül a 3.3. alfejezet egy elméleti megoldással foglalkozott, ami a központi **VSZ** nagy energia- és időigényű számításokba való szerepvállalását taglalta anélkül, hogy a biztonságból fel kellene adni. Ezen lehetőséget gyakorlati szinten azonban nem volt lehetőségünk vizsgálni.

Összességében tehát az E-CCLESIA protokoll, általánosságban pedig az RSA akkumulátorok számos módon továbbfejleszthetők a teljes decentralizáció irányába. Amint azt megmutattuk, ezen fejlesztések egy része a gyakorlatban is használható, és utat mutat egy szebb, biztonságosabb, fairebb és decentralizáltabb világ felé.

Hivatkozások

[1] M. Arapinis, A. Kocsis, N. Lamprou, L. Medley, T. Zacharias, Universally composable simultaneous broadcast against a dishonest majority and applications, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, ACM PODC '23, pp. 200–210, New York, NY, USA, 2023.

[2] M. Arapinis, N. Lamprou, L. Marekova, T. Zacharias, E-cclesia: Universally composable self-tallying elections, *IACR Cryptol. ePrint Arch.*, 2020:513, 2020.

[3] J. Bannet, D. Price, A. Rudys, J. Singer, D. Wallach, Hack-a-vote: Security issues with electronic voting systems, *IEEE security & privacy*, 2(1) (2004), pp. 32–37.

[4] N. Baric, B. Pfitzmann, Collision-free accumulators and fail-stop signature schemes without trees, *International Conference on the Theory and Application of Cryptographic Techniques,* 1997.

[5] J. Benaloh, M. de Mare, One-way accumulators: A decentralized alternative to digital signatures, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),* 765 (1994), pp. 274–285.

[6] J. Camenisch, A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Berlin, Springer, 2442 (2002), pp. 61–76.

[7] R. Canetti, Universally composable security: a new paradigm for cryptographic protocols, *Proceedings 42nd IEEE Symposium on Foundations of Computer Science,* (2001), pp. 136–145.

[8] J. P. Gibson, R. Krimmer, V. Teague, J. Pomares, A review of e-voting: the past, present and future, *Annales des télécommunications,* 71(7) (2016), pp. 279–286.

[9] J. Groth, Efficient maximal privacy in boardroom voting and anonymous broadcast *Lecture Notes in Computer Science,* Berlin, Springer, 3110 (2004), pp. 90–104.

[10] A. Kiayias, M. Yung, Self-tallying elections and perfect ballot secrecy, *Public Key Cryptography, Lecture Notes in Computer Science,* Berlin, Heidelberg, Springer, 2274 (2002), pp. 141–158.

[11] A. Kocsis, RSA accumulators and simultaneous broadcast channel for decentralised e-voting, Postgraduate dissertation, University of Edinburgh, Edinburgh, UK, 2021.

[12] J. Levi, E-cclesia implementation using time-lock encryption, Undergraduate project dissertation, University of Edinburgh, Edinburgh, UK, 2019.

[13] B. Lynn, Cryptography – Electronic Voting, 2002. `https://crypto.stanford.edu/pbc/notes/crypto/voting.html` (2021.08.03.)

[14] E. Maaten, Towards remote e-voting: Estonian case, *Electronic voting in Europe – Technology, law, politics and society, workshop of the ESF TED programme together with GI and OCG,* Eds.: A. Prosser, R. Krimmer, Gesellschaft für Informatik e.V., Bonn, 2004, pp. 83–90.

[15] î Madise, T. Martens, E-voting in estonia 2005. the first practice of country-wide binding internet voting in the world, *Electronic Voting 2006 – 2nd International Workshop, Co-organized by Council of Europe, ESF TED, IFIP WG 8.6 and E-Voting.CC,* Ed.: R. Krimmer, Gesellschaft für Informatik e.V., Bonn, 2006, pp. 15–26.

[16] L. Marekova, Zerovote: Self-tallying e-voting protocol in the UC framework, Undergraduate honors thesis, University of Edinburgh, Edinburgh, UK, 2018.

[17] I. Miers, C. Garman, M. Green, A. D. Rubin., Zerocoin: Anonymous distributed e-cash from bitcoin, *2013 IEEE Symposium on Security and Privacy,* IEEE, 2013, pp. 397–411.

[18] T. Okamoto, Receipt-free electronic voting schemes for large scale elections, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),*, Springer-Verlag, New York NY, 1361 (1998), pp. 25–35.

[19] T. P. Pedersen, Non-interactive and information-theoretic secure verifiable secret sharing, *Advances in Cryptology – CRYPTO '91, Lecture Notes in Computer Science,* Springer, Berlin, Heidelberg, 2001, pp. 129–140.

[20] T. Sander, Efficient accumulators without trapdoor (extended abstract), *Information and Communication Security, Lecture Notes in Computer Science,* Springer, Berlin, Heidelberg, 1726 (1999), pp. 252–262..

[21] A. Szepieniec, B. Preneel, New techniques for electronic voting, *USENIX Journal of Election Technology and Systems (JETS),* (Aug 2015)

[22] S. Wolchok, E. Wustrow, D. Isabel, J. A. Halderman, Attacking the Washington, D.C. internet voting system, *Financial Cryptography and Data Security, Lecture Notes in Computer Science,* Springer, Berlin, Heidelberg, 7397 (2012), pp. 114–128.

[23] Ancoin. `https://anoncoin.github.io/RSA_UFO/` (2021.07.09.)

[24] Bitcoin Project. `https://bitcoin.org/` (2021.07.09.)

[25] Election Integrity: 62% Don't Think Voter ID Laws Discriminate, Rasmussen Reports, Apr. 2021. `https://www.rasmussenreports.com/public_content/politics/general_politics/april_2021/election_integrity_62_don_t_think_voter_id_laws_discriminate` (2021.04.28.)

[26] libzerocoin/bitcoin_bignum/bignum.h. `https://github.com/Zerocoin/libzerocoin/blob/master/bitcoin_bignum/bignum.h` (2021.08.05.)

[27] OpenSSL Software Foundation, bn. `https://www.openssl.org/docs/man1.0.2/man3/bn.html` (2021.07.09.)

[28] OpenSSL Software Foundation. `https://www.openssl.org/` (2021.07.09.)

[29] US election 2020: Fact-checking Trump team's main fraud claims, *BBC News*, Nov. 2020. `https://www.bbc.com/news/election-us-2020-55016029` (2021.08.03.)

Mozgásmásolás és inverz kinematika használata robotok vezérlésére

Kovács Lehel István*

Kiss Elemér Szakkollégium**

klehel@ms.sapientia.ro

1. Bevezetés

A Kiss Elemér Szakkollégium 2010 novemberében alakult a Sapientia Erdélyi Magyar Tudományegyetem Marosvásárhelyi Karán a Matematika-Informatika Tanszék és MITIS Egyesület keretében. Mindjárt a megalakulása után együttműködési szerződést kötött a budapesti Eötvös Collegiummal.

Névadónk, Kiss Elemér 1929. augusztus 25-én született Brassóban. Gyermekkorát Csíkmenaságon töltötte. A középiskolát a csíkszeredai gimnáziumban végezte. Egyetemi diplomát a kolozsvári Bolyai Tudományegyetemen szerzett 1951-ben. 1961-ig a Bolyai Farkas Líceumban tanított, majd a Marosvásárhelyi Tanárképző Főiskola matematikai tanszékének adjunktusa volt. Az intézet megszűnése után a Petru Maior Tudományegyetem elődintézetében folytatta munkáját, ahol 1976-tól 1985-ig tanszékvezetői feladatokat is ellátott. Doktori disszertációját, amelyben a modern algebra témaköréhez tartózó kérdésekkel foglalkozott, 1974-ben védte meg Gh. Pick professzor irányítása alatt. Részt vett a Sapientia Erdélyi Magyar Tudományegyetem alapításában, és

* Sapientia Erdélyi Magyar Tudományegyetem, Marosvásárhelyi Kar

** A Marosvásárhelyen működő Kiss Elemér Szakkollégium vezetője 2013 óta

haláláig professzora volt. Élete utolsó két évtizedében Bolyai Jánosnak a marosvásárhelyi Teleki–Bolyai könyvtárban található kéziratos hagyatékát tanulmányozta. Kutatói munkája eredményeként a Magyar Tudományos Akadémia 2001-ben külső tagjai közé választotta. 2006. augusztus 23-án halt meg Marosvásárhelyen.

Jelen dolgozat a Kiss Elemér Szakkollégium keretei között futó egyik kutatásról számol be. Az eredmények részleteit kétszer is bemutattam az Eötvös Collegiumban: 2020 februárjában, illetve 2022 szeptemberében.

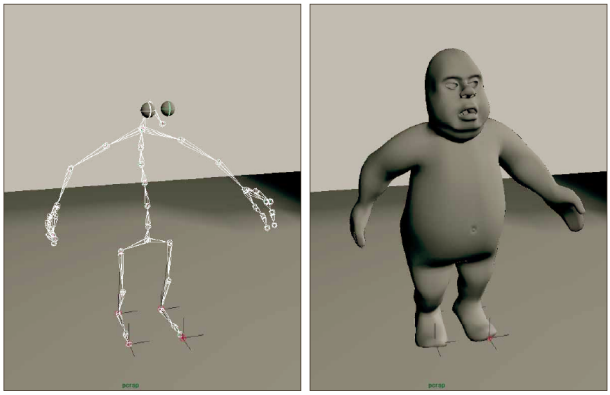
2. Animáció, inverz kinematika

Az animáció olyan filmkészítési technika, amely élettelen tárgyak (általában bábok vagy rajzok, ábrák, stb.) „kockázásával", azaz képkockánkénti rögzítésével, olyan illúziót kelt a nézőben, mintha a szereplők megelevenednének vagy élnének. A számítógépes animáció előnye a hagyományos, kézzel előállított animációval szemben az, hogy nem kell az összes mozgásfázist lerajzolni, a képkockákat és háttereket kézzel kifesteni, kevesebb animátorra, rajzolóra, színezőre van szükségünk, és az egyes műveletek felgyorsultak.

Ha számítógépes animációról beszélünk, a következő válfajokat különböztethetjük meg: *egyszerű animáció* (kulcspozíciók, programvezérelt), valamint *összetett animáció* (inverz kinematika, forward kinematika, motion capture, vagyis mozgásmásolás).

Az összetett animáció egy csont/ízület rendszert feltételez [1]. A megmozgatni (animálni) kívánt modellhez a topológiája alapján egy csont/ízület rendszert rendelünk (1. ábra). Csontokat és őket összekötő ízületeket hozunk létre, amelyek a valódi csontvázhoz hasonlóan mozgatják a felületeket. Az ízületek a csontokat kötik össze, az ízületek körül forognak a csontok egy bizonyos szögtartományban. Minden animáció vagy a robotok végtagjainak a mozgatása tehát forgatásra vezethető vissza: egy csont egy adott szöggel elfordul egy ízület körül. Fontos, hogy a csontok hossza a mozgás során nem változik.

A *direkt (forward) kinematika* a figurák „hagyományos" mozgatását jelenti, vagyis először a karok, lábak felső csuklóit állítjuk be, és egyesével haladunk a csontrendszer utolsó csuklói, az ujjak felé. Olyan



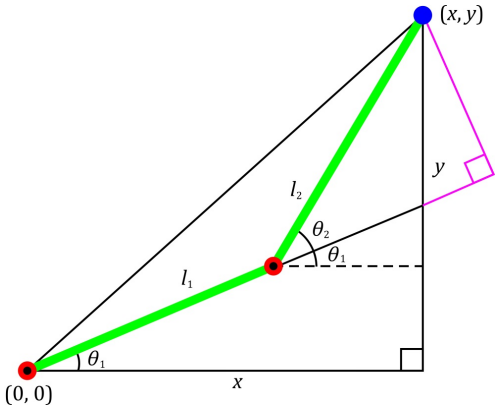
1. ábra. Csont/ízület rendszerek [`www.vassg.hu/pdf/grafika6.pdf`]

bonyolult mozgások, mint például lépcsőkön menés, robotok számára nem valósítható meg direkt kinematikával, csak inverz kinematikával. Az *inverz kinematika* sokkal kényelmesebb és hatékonyabb, mint a direkt. Az animátornak nincsen más feladata, mint a csontrendszer utolsó csontját mozgatni, a többiről a számítógép gondoskodik. A gyakorlatban ez az jelenti, hogy elég a figurának a kézfejét mozgatni, a könyök és a váll mozgását a program kikövetkezteti. Komplex mozgásokat szinte csak ezzel a célirányos technikával lehet elkészíteni. Matematikailag az inverz kinematika rendszere összetett. Pontos módszer csak két csontra létezik, többre már nagyon bonyolulttá válik, így iteratív megoldásra, nemlineáris optimalizálásra van szükség. Csont/ízület rendszerekkel nemcsak emberi csontvázak, hanem bármilyen alakzatok megvalósíthatók, mozgathatók. Ez a nagy előnye a motion capture technikához képest, amely főleg csak az ember mozgását tudja lemásolni.

A rendszer mozgatására a következő technikák születtek:

- Analitikus megoldás;
- Hierarchikus mozgás.

Az analitikus megoldás feladata egyszerű [2]: *Adott két csont. Az első egyik vége az origóban* $(0, 0)$*, a másik végéből pedig a második csont indul. Mekkora szögekkel kell elforgatnunk a két csontot ahhoz, hogy a második csont szabad vége egy adott* (x, y) *pozícióba kerüljön?*



2. ábra. Az analitikus megoldás

A 2. ábrán látható feladat megoldása még matematikailag eléggé egyszerű:

$$\cos\theta_2 = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1 l_2},$$

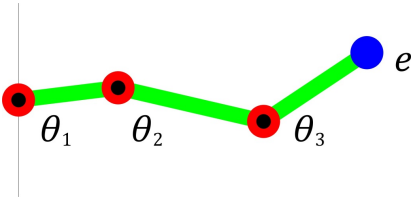
a másik szög meghatározása pedig:

$$\tan\theta_1 = \frac{y \cdot (l_1 + l_2 \cdot \cos\theta_2) - x \cdot l_2 \cdot \sin\theta_2}{x \cdot (l_1 + l_2 \cdot \cos\theta_2) + y \cdot l_2 \cdot \sin\theta_2}.$$

A hierarchikus mozgást akkor használjuk, amikor több mint két csontunk van, s így az analitikus megoldás már túl bonyolulttá válik. Ekkor iteratív megoldásokhoz folyamodunk: az effektortól iterálunk az alapig, és optimalizálunk minden egyes ízületet, hogy az utolsó olyan közel kerülhessen a célponthoz, amennyire csak lehet. Láthatjuk, hogy ekkor ugyanaz a megoldás születik több inverz kinematika feladatra, ám az iteráció miatt a megoldás költséges.

A 3. ábrán látható hierarchikus mozgás feladata: *Ismerjük az effektor koordinátáit* ($e = [e_1, e_2, \ldots, e_N]$), *határozzuk meg a szabadságfokot (Degrees of Freedom, DOF* : $\theta = [\theta_1, \theta_2, \ldots, \theta_M]$*)!*

Vagyis meg kell határozni egy $\theta = f^{-1}(e)$ függvényt. Mivel a függvény nemlineáris, az inverz függvény meghatározása nem triviális feladat. A függvény nem egy-egy értelmű, vagyis több állapothoz tartozhat ugyanaz az effektor-helyzet. Az inverzió nemlinearitásával és



3. ábra. Hierarchikus mozgás

többértelműségével egy iterációs eljárás segítségével birkózhatunk meg, amely a lehetséges megoldások közül egyet állít elő. Az iteráció alapötlete: ha egy t időpillanatban ismerjük az effektor helyzetét, akkor ebből következtethetünk a $t + \Delta t$ időpontban érvényes helyzetre. Ha Δt kicsiny, akkor a nemlineáris függvényt megközelíthetjük az érintőjével (lineáris approximáció). Az inverz kinematikai (IK) feladat megoldására három módszert próbáltunk ki:

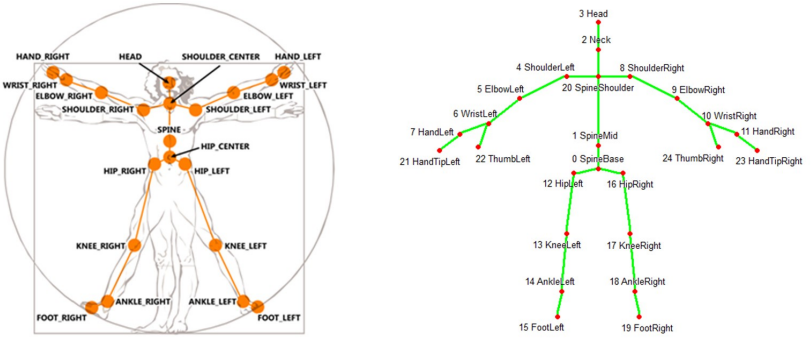
- Jacobi-mátrix;
- Ciklikus koordináta leereszkedés – Cyclic Coordinate Descent (CCD);
- Megkötések lazítása – Constraint Relaxation (CR).

A Jacobi-mátrix [3] a rendszer parciális deriváltjainak mátrixa.

$$J = \begin{bmatrix} \frac{\partial e_x}{\partial \theta_1} & \frac{\partial e_x}{\partial \theta_2} & \frac{\partial e_x}{\partial \theta_3} \\ \frac{\partial e_y}{\partial \theta_1} & \frac{\partial e_y}{\partial \theta_2} & \frac{\partial e_y}{\partial \theta_3} \end{bmatrix}.$$

Mivel általában ez nem egy négyzetes mátrix, csak pszeudoinverz számolható. A pszeudoinverzzel kapott állapotváltozások minimálisak, tehát a lehetséges mozgások közül azokat kapjuk meg, amelyekben az ízületekben a csontok relatív sebessége minimális. Iteratív megoldás: a t_0 kezdeti állapotban minden ismert. Az effektort az előírt pályán kis lépésekben mozgatjuk, és minden lépésben a Jacobi-mátrix pszeudoinverzének felhasználásával kiszámítjuk az állapotváltozást.

A CCD algoritmus [4, 5] vektorok különböző szorzatait (skaláris, vektoriális) használja fel a megoldás megkeresésére.



4. ábra. Kinect csontvázak (V1 és V2)

A CR algoritmus [6] képes bármilyen hosszú csontlánc csontjai pontos pozíciójának meghatározására, illetve könnyű implementálni nagyobb dimenziókban is. Az algoritmus lényege, hogy egyenként a csontokat megnyújtjuk a célpont felé, majd visszaállítjuk az eredeti méretet, ezzel egyre közelebb kerülve a megoldáshoz.

3. Tér, mozgás és személy érzékelése Kinect-tel, mozgásmásolás

A Kinect (kódnevén Project Natal) a Microsoft által fejlesztett speciális mozgásérzékelő eszköz az Xbox 360 és az Xbox One videojáték-konzolokhoz és Windows-os PC-khez. 2010-ben jelent meg, a Windows-os változat pedig 2012-ben.

A többféle kamerát és mikrofont tartalmazó eszköz segítségével kontroller nélkül játszhatóak egyes játékok. Az irányítás testmozgás, természetes gesztusok és szóbeli parancsok segítségével történik. A Kinect érzékeli a teret, ki tudja számítani az objektumok közötti távolságokat, valamint egy csont/ízület rendszert (csontvázat) is implementál.

Az 1-es és a 2-es verziójú Kinect csont/ízület rendszerét (csontvázát) a 4. ábrán láthatjuk.

A *motion capture*, magyarul *digitális mozgásrögzítés* vagy *mozgásmásolás* egy olyan eljárás, melynek során mozgást rögzítenek, majd azt egy digitális modellre ültetik át. A Kinect tökéletesen megfelel ennek a

célnak. Segítségével gesztusokat ismerhetünk fel, lemásolhatjuk ezeket, és felhasználhatjuk robotok vezérlésére.

A Kinect csontvázát a következő kóddal olvashatjuk a *NuiApi* segítségével:

```
#include <NuiApi.h>
#include <NuiImageCamera.h>
#include <NuiSensor.h>

Vector4 skeletonPosition[NUI_SKELETON_POSITION_COUNT];

void getSkeletalData()
{
  NUI_SKELETON_FRAME skeletonFrame = {0};
  if (sensor->NuiSkeletonGetNextFrame(0, &skeletonFrame)
      >= 0)
  {
     sensor->NuiTransformSmooth(&skeletonFrame, NULL);
     for (int z = 0; z < NUI_SKELETON_COUNT; ++z)
     {
        const NUI_SKELETON_DATA& skeleton = skeletonFrame
            .SkeletonData[z];
        if (skeleton.eTrackingState ==
            NUI_SKELETON_TRACKED)
        {
           for (int i = 0; i <
               NUI_SKELETON_POSITION_COUNT; ++i)
           {
              skeletonPosition[i] = skeleton.
                  SkeletonPositions[i];
              if (skeleton.eSkeletonPositionTrackingState
                  [i] ==
                  NUI_SKELETON_POSITION_NOT_TRACKED)
                  skeletonPosition[i].w = 0;
           }
           return;
        }
     }
  }
}
```

A *skeletonPosition* a 4. ábrán látható ízületek (x, y, z, w) homogén koordinátáit tartalmazza, így az ízületek a térben egy pontfelhőt alkotnak. Hivatkozni is így lehet rájuk:

```
Vector4& lf = skeletonPosition[
    NUI_SKELETON_POSITION_FOOT_LEFT];
```

```
2 Vector4& la = skeletonPosition[
      NUI_SKELETON_POSITION_ANKLE_LEFT];
3 Vector4& lk = skeletonPosition[
      NUI_SKELETON_POSITION_KNEE_LEFT];
4 Vector4& lp = skeletonPosition[
      NUI_SKELETON_POSITION_HIP_LEFT];
5 ...
```

4. Gesztusok felismerése

A *gesztus* a metakommunikáció egyik fajtája. Magában foglalja a fej, a kar és a láb mozgását. Általában olyan rövid, kifejező testmozgás vagy testmozgások sorozata, amely a közösség tagjai számára hírértékkel bír. Gesztus például egy bólintás, intés, karemelés stb.

A gesztusfelismerés célja az emberi gesztusok matematikai és informatikai algoritmusok segítségével történő értelmezése, akár mesterséges intelligencia technikák által. Mi csontváz alapú 3D gesztusfelismerési technikát alkalmaztunk.

Az általunk használt módszer feltételezi, hogy minden gesztust 33 képkocka ír le. Polárkoordinátákat használunk, mert ezeket a koordinátákat könnyebb normalizálni. Így nem kell tekintettel lennünk a felhasználó méreteire (például a karjának hosszúságára), csak az origótól való távolság változik [7].

A gesztusfelismerés javasolt algoritmusa 4 lépésből áll:

1. az emberi felhasználó felismerése,
2. a jellemzők kinyerése,
3. a vetemedés szakasza, amelyben a gesztusokat referencia gesztusokhoz hasonlítjuk,
4. a gesztus felismerése.

Az emberi felhasználó észlelését és felismerését a Kinect szenzor elősegíti, így az 1. lépés szinte triviális. A gesztusok legfontosabb jellemzői, itt, a végtagok mozgása. A 3. lépésben DTW-t használunk.

Az idősorelemzésben a *dinamikus idővetemítés* (Dynamic Time Warping, DTW) az egyik legjobb algoritmus két időbeli sorozat közötti hasonlóság mérésére, még akkor is, ha a két sorozat sebessége változó.

Például a DTW használatával hasonlóságokat lehet felfedezni két személy gyaloglásában, még akkor is, ha az egyik személy gyorsabban jár, mint a másik.

Az algoritmus a következő lépésekből áll:

- a személy felismerése,
- a jellemzők kinyerése,
- dinamikus idővetemítés (DTW),
- a gesztus felismerése,
- ha IGEN
  - a robot vezérlése,
  - a művelet megjegyzése.

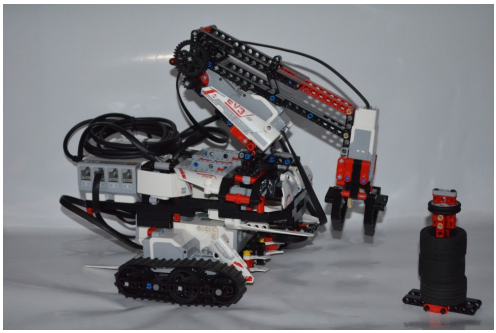
A gesztusfelismeréshez a következő lépések szükségesek:

- a 33 képkocka beazonosítása,
- az ízületek koordinátáinak meghatározása,
- polárkoordinátákká alakítás,
- a koordináták normalizálása,
- a jellemzők vektorának felépítése,
- dinamikus idővetemítés algoritmusa (DTW).

A DTW összehasonlítja az ismeretlen gesztusok sorozatát egy vagy több referenciamintával vagy sablonnal. Több sablon használatával nagyobb lesz a felismerési arány, de megnő a számítási idő.

Ha van két sorozatunk, amelyeket a következő idősorok képviselnek: $x = (x_1, x_2, ..., x_n)$, valamint $y = (y_1, y_2, ..., y_m)$, akkor felépíthetünk egy $n \times m$ elemű mátrixot, amelynek minden eleme az idősorok elemei közötti távolságot jelentik. Ezeket a *mátrix költségének* nevezünk.

Ahhoz, hogy megtaláljuk a legjobb egyezést a két sorozat között, olyan mátrix-útvonalat kell találni, amely minimalizálja az elemeik közötti kumulatív teljes távolságot.



5. ábra. LEGO robot

A *vetemítési útvonal* a két idősor elemei közötti leképezést határozza meg: $w = (w_1, w_2, ..., w_k, ...w_p)$, ahol $\mathrm{DTW}(x, y) = \min \sum_{k=1}^{p} d(w_k)$, és $d(w_k)$ az x_i és y_j az idősorok elemei közötti távolságot jelölik. Ez: $d(x_i, y_j) = |x_i - y_j|$.

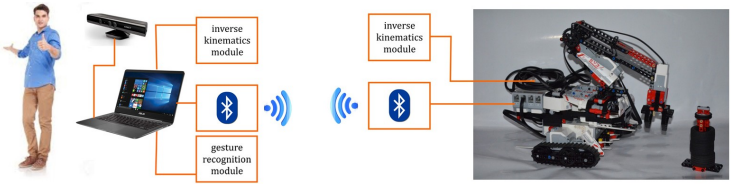
5. LEGO robotok vezérlése gesztusokkal

A LEGO Mindstorms EV3 harmadik generációs LEGO robot. 2013 szeptemberében e termékcsalád megjelentetésével ünnepelte a népszerű Mindstorms játék- és oktatóeszköz tizenötödik születésnapját a LEGO [8].

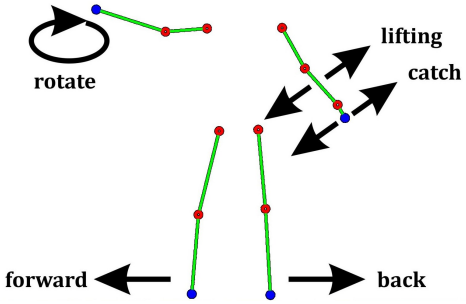
Kísérletünk számára elkészítettük az 5. ábrán látható LEGO robotot. Lánctalpakon előre és hátra tud menni, forgatható, emelhető robot markoló karral rendelkezik. A kívánt szabadságfok elérése érdekében több EV3 téglát kapcsoltunk össze, így egy programozható osztott rendszert kaptunk.

Rendszerünk a 6. ábrán látható. A személy mozgásait egy Kinect érzékeli, a számítógép elvégzi az átalakításokat a különböző koordinátarendszerek között (IK modul), kiszámítja az inverz kinematika mozgásegyenleteit, az adatokat pedig Bluetooth kapcsolaton átküldi az EV3 téglának, amely vezérelni fogja a robotot [9, 10].

A 7. ábrán az alkalmazás egyszerűsített csont/ízület rendszerét láthatjuk. A testet, fejet elhagytuk, a robotot ugyanis csak a lábakkal és a kezekkel vezéreljük a képen látható módon. Külön feladat a gesztusok



6. ábra. A LEGO rendszer két oldala: személy és robot



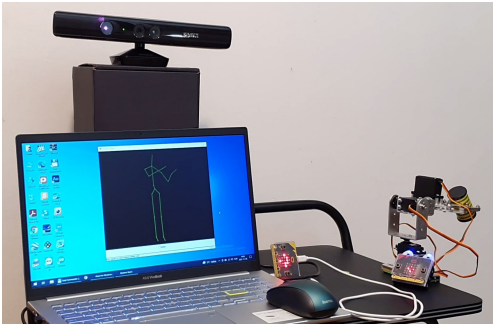
7. ábra. Parancsok, gesztusok a vezérlésre

felismerése. A csontváz mozgásából, az x, y, z koordináták változásából következtetni tudunk a parancsokra. Felismerjük a kéz forgatását, emelését, vagy a markolásra vonatkozó gesztusokat is. A lábak mozgásából következtetni tudunk az előre vagy hátra haladási irányokra.

6. Micro:bit robotok vezérlése gesztusokkal

Rendszerünket kipróbáltuk egy micro:bit vezérelte robotkarral is [11]. A rendszer nagyon hasonlított a LEGO robotos rendszerre, annyi különbséggel, hogy két micro:bitre volt szükség, amelyek rádiójellel kommunikáltak egymással, a micro:bit Bluetooth segítségével nem tud kommunikálni a számítógéppel (8. ábra). Mivel a micro:bit robotkarnak mások a méretei, a robot IK modulját erre kellett testre szabni.

A *BBC micro:bit* egy kifejezetten oktatási célra létrehozott, egylapkás mikrovezérlő, amely 4×5 cm-es méretével, 5×5-ös LED kijelzőjével, gyorsulásérzékelő, hőmérséklet érzékelő, fény-érzékelő, irány érzékelő

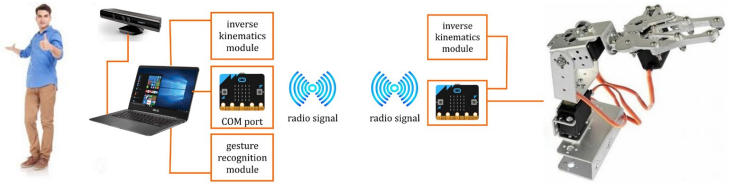


8. ábra. Micro:bit robot

szenzoraival, be- és kimeneti csatlakozóival, 2 gombjával, bluetooth/rádió kapcsolódási lehetőségével igen sokrétű alkalmazást tesz lehetővé, legyen az (akár többfelhasználós) játék fejlesztése, viselhető eszközök (pl. okosóra, lépésszámláló, okosruha) tervezése és megvalósítása, kísérletezés a szenzorok által mért adatok felhasználásával, vagy éppen külső eszközök vezérlése/irányítása. Mivel az eszköz egy mikrovezérlő, ezért a programozásához szükséges egy számítógép (asztali, notebook, vagy akár tablet és okostelefon), amelyhez vagy USB kábellel, vagy bluetooth kapcsolaton keresztül kapcsolódhatunk. PC-ről vagy mobilról is elérhető web-es felületen (`https://makecode.microbit.org/`) írhatunk programokat, amelyek USB-n vagy akár bluetooth-on keresztül tölthetők fel az eszközre. A programot egyszerűen fel kell másolni a micro:bit virtuális meghajtójára, és már működni is kezd [12].

Egy modulokból felépíthető, MG995-ös szervomotorokkal működtethető alumínium robotkart választottunk, amelynek több előnye is van:

- Az akril, műanyag robotkarok nem igazán voltak megfelelőek – bár sokat próbálkoztunk ezekkel is –, mert hamar törtek, eléggé kényesek voltak.
- A robotkarhoz tartozó MG995 szervomotorok nagyon pontosaknak és megbízhatóknak bizonyultak.
- Habár nem adtak a karhoz összeszerelési útmutatót (igaz, hogy más műanyag karokhoz sem adtak), elég könnyen össze lehetett szerelni.



9. ábra. A micro:bit rendszer két oldala: személy és robot

- A robotkar moduláris, könnyen kibővíthető.

Számtalan alkalmazás számára a 3 szabadságfok nem igazán elegendő, de a kibővített 6 DOF-os robotkar már jól üzemelt, és hat szervomotort GVS pinek segítségével jól is lehet vezérelni micro:bit segítségével (bőven van annyi szabad pin). Kísérleteink során már a 4 szabadságfok jól működött, elegendő volt, a 6 szabadságfok a non plus ultra.

7. Következtetések

Inverz kinematika segítségével sokkal könnyebb animálni, mint a direkt (forward) kinematika használata esetén. A robotkar és a teljes robot vezérlése megtörténhet inverz kinematikával is, ám ez nem épp annyira pontos, mint direkt kinematika esetén, viszont az optimalizáló algoritmusok jól meg tudják közelíteni a célt. Ha a robot működtetése nem annyira költséges, és nincs szükség arra, hogy minden ízület pontos helyét meghatározzuk a térben, vagy a mozgás nagyon összetett, az inverz kinematika célravezetőbb, mint a direkt kinematika.

A Kinect segítségével jól fel tudjuk ismerni a gesztusokat, és ezeket fel tudjuk használni robotok vezérlésére.

A LEGO, micro:bit vagy más robotokon is elvégzett kísérletek igazolják az elméletet.

Kísérleti eredményeink azt mutatják, hogy a Jacobi-mátrix megoldású lineáris megközelítés elég sok szingularitást tartalmaz, mindenképpen javítani kell, de gyors, hatékony módszer. A CCD jobban teljesít a mozgó célpont követésében, elkerülve a Jacobi módszerek által mutatott oszcillációkat és mozgási folytonossági zavarokat. A CCD alacsony számítási költséggel rendelkezik, és valós időben oldja meg az IK problémát. A CR algoritmus nehezebben érthető, de valós időben megfelelő

eredményeket produkál, ezért érdemes használni.

A bemutatott módszerekkel egyszerűen tudjuk irányítani a robotokat, természetesen a gesztusokat kellően be kell gyakorolnunk a kívánt pontosság eléréséhez.

A bemutatott módszerek elég általánosak ahhoz, hogy ne csak LEGO vagy micro:bit robotokat irányítsanak, hanem bármilyen robotot.

Az itt leírt megoldás kellően könnyű, kiválóan alkalmas oktatási célokra, de alkalmas laboratóriumi vagy szimulációs gyakorlatokra is.

Hivatkozások

[1] Szirmay-Kalos L., Antal Gy., Csonka F., *Háromdimenziós grafika, animáció és játékfejlesztés*, ComputerBooks, Budapest, 2006.

[2] Ryan Juckett, Analytic Two-Bone IK in 2D. `http://www.ryanjuckett.com/programming/analytic-two-bone-ik-in-2d/`

[3] Samuel R. Buss, *Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods*, University of California, San Diego, 2009.

[4] Stephen J. Wright, *Coordinate Descent Algorithms*, University of Wisconsin, 2010.

[5] Ryan Juckett, Cyclic Coordinate Descent in 2D. `http://www.ryanjuckett.com/programming/cyclic-coordinate-descent-in-2d/`

[6] Ryan Juckett, Constraint Relaxation IK in 2D. `http://www.ryanjuckett.com/programming/constraint-relaxation-ik-in-2d/`

[7] Boboc, Răzvan Gabriel, *Interacțiunea naturală om-robot pentru aplicații de robotică asistivă*, Universitatea Transilvania din Brașov, 2015.

[8] `http://education.lego.com/es-es/products`

[9] Kovács Lehel István, Gesztusokkal vezérelt robotkar, *SzámOkt 2017*, EMT, Kolozsvár, (2017), pp. 178–183.

[10] Lehel István Kovács, Gesture-Driven LEGO robots, *Acta Universitatis Sapientiae, Informatica*, 1(11), (2019), pp. 80–94.

[11] Lehel István Kovács, Roboţi LEGO şi micro:bit controlaţi cu gesturi, *Conferinţa ştiinţifică naţională cu participare internaţională „Integrare prin cercetare şi inovare”*, USM, 10-11 noiembrie 2021, Ştiinţe ale naturii şi exacte, Matematică şi Informatică, CEP USM, Chişinău, (2021), pp. 238–240.

[12] Jorge Pereira, Using micro:bit with the LEGO Mindstorms EV3. `https://github.com/JorgePe/microbit`

Érintők, normák, zérushelyek

Lócsi Levente*

Eötvös József Collegium**

locsi@inf.elte.hu

Bevezetés

Ezen kötet szerkesztői munkálatai mellett mindenképpen szerettem volna egy szakmai jellegű írással is hozzájárulni az Informatikai Műhely húszéves jubileumi kiadványához. Hasonló módon és stílusban tenném ezt ahhoz, ahogy tíz évvel ezelőtt, a műhely első évtizede előtt tisztelgő, Csörnyei Zoltán tanár úr, alapító műhelyvezetőnk által szerkesztett kötet esetében.

Egyúttal ez egy remek lehetőség arra is, hogy az ELTE Informatikai Kar Numerikus Analízis Tanszékének adjunktusaként az utóbbi években végzett kutatómunkámat röviden bemutassam, hangolódva valamelyest a habilitációs értekezésem megírására is, amelyet néhány éven belül tervezek benyújtani.

Kutató-, illetve alkotómunkám fókuszában mostanság olyan témák álltak, amelyek egy-egy informatikai vagy alkalmazásokhoz közeli probléma vagy eljárás matematikai hátterét képező fogalmak, jelenségek mélyebb megértését teszik lehetővé (tipikusan az analízis, numerikus módszerek, digitális jelfeldolgozás témaköreihez kapcsolódóan), eddig ismeretlen, esetleg feltérképezetlen – de legalábbis nem teljesen feltárt – részleteket hoznak napvilágra; még ha ezzel nem is a tudományok legnagyobb magaslatait ostromlom.

* ELTE Informatikai Kar

** 2003–2011, műhelyvezető: 2022–

Mindvégig törekedtem arra is, hogy az egyes területekhez igényes vizualizációk, minél szebb illusztrációk készüljenek, ezzel is hangsúlyozva a témákban rejlő szépséget, remélhetőleg közelebb hozva azokat az érdeklődők minél szélesebb rétegéhez. Igazán szeretem, ha egy-egy nem triviális, esetleg némileg mélyebb matematikai fogalmat, vadnak tűnő képletet néhány szép ábra segítségével sikerül érthetőbbé tennem, átadnom.

Ebben az írásban – annak összefoglaló jellegére való tekintettel – egy-egy kérdés részletes formális bemutatása helyett hagyom inkább a kapcsolódó ábrákat beszélni. Hasonló okokból a szakirodalmi hivatkozásokat is indirekt módon bocsájtom az érdeklődő Olvasó rendelkezésére: méghozzá saját publikációimra, témavezetésemmel készült dolgozatokra (ezek irodalomjegyzékére) hivatkozva.

A cím három szavának megfelelően a következő három fejezetben az alábbi témaköröket foglaljuk össze: a CT felvételek mögött megbúvó Radon-transzformáció valós idejű webes vizualizációja; az indukált – avagy természetes – mátrixnormák definíciója alapján adódó halmazok (görbék, felületek) vizsgálata, felfedezve mátrixok p-sajátvektorait; valamint a rövid idejű – másképp: „ablakolt” – Fourier-transzformáció zérushelyeinek elhelyezkedése bizonyos speciális esetekben.

1. Ellipszisek érintőiről

A CT – *Computer Tomography,* komputer tomográfia – elterjedten használt eszköz az orvosi diagnosztikában. Ezen eljárás során lényegében sok irányból röntgenfelvételt készítenek a páciensről. A testen áthaladó röntgensugarak különböző mértékben nyelődnek el az egyes szövetekben (csont, izom, szervek, zsír, bőr), és végül mindezen információk alapján nem csak egyetlen vetület tanulmányozhatunk, hanem a test keresztmetszeteiről is képet alkothatunk. A problémát matematikai tekintetben Johann Radon már több mint száz évvel ezelőtt, 1917-ben megoldotta *Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten*[1] című dolgozatában, bevezetve az azóta róla elnevezett transzformációt, illetve annak inverzét. A gyakorlatban való megvalósításig, a gyógyászatban való alkalmazhatóságig

[1] A cím magyar fordítása: „Függvények bizonyos sokaságokon vett integrálértékei által történő meghatározásáról”.

még el kellett telnie néhány évtizednek, s a technológia kidolgozásáért Allan M. Cormack és Godfrey N. Hounsfield elnyerték az 1979. évi fiziológiai és orvostudományi Nobel-díjat. Méltó tehát, hogy ilyen fontos gondolatokat igyekezzünk közelebb hozni az érdeklődőkhöz, akár a tudományos ismeretterjesztés zászlaja alatt is.

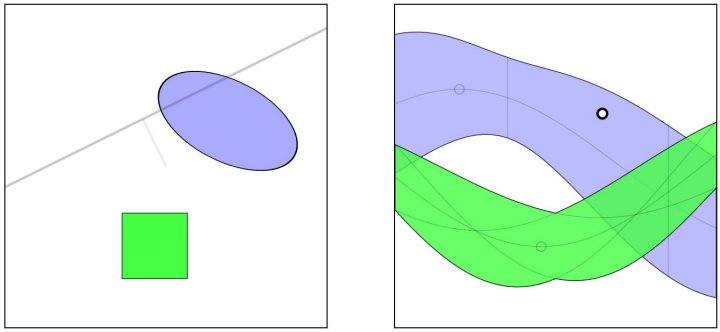
A Radon-transzformáció, a CT működésének tanulmányozása céljából igen gyakran alkalmaznak úgynevezett „fantomokat", egyszerű síkidomokból, tipikusan ellipszisekből összeállított modelleket, mint pl. a Shepp–Logan-fantom (az emberi koponya, az agy keresztmetszetéről), vagy a Bognár Gergő kollégám által nemrég kidolgozott tüdőfantom[2], amelyek transzformáltja is analitikusan számolható. Ezek segítségével többek között vizsgálhatók a diszkretizációs tulajdonságok, elemezhetők a potenciális hibák, vagy alapot adhatnak valós felvételek minőségének számszerűsítéséhez. A számítógépes szimulációkhoz, vagy akár már csak a szemléltetéshez is nem elhanyagolható programozási ismeretek, esetleg komolyabb programcsomagok szükségesek (pl. Matlab, Mathematica), de még sokszor így is mindez lassú, időigényes.

A mi munkánk célja mindezek figyelembevételével az volt, hogy egy olyan megoldást adjunk, amely segítségével lehetővé válik a Radon-transzformáció működésének szemléltetése úgy, hogy a modell interaktívan szerkeszthető, a transzformált valós időben megjeleníthető, valamint hogy mindez bárki számára könnyen és ingyenesen hozzáférhető. Egy böngészőben futó, az interneten elérhető alkalmazást terveztünk.

Gál Zsófia mesterszakos diplomamunkája keretében [2] egy szerveroldali, numerikus megközelítést valósított meg témavezetésemmel. A felhasználó a webes felületen egy tetszőleges szürkeárnyalatos képet összerakhatott téglalapokból és ellipszisekből, ezt a böngésző továbbította egy szervernek, ahol a Radon-transzformáltat (avagy szinogramot) egy külön program számolta, majd az elkészült képet visszaküldte a böngészőnek. Ezzel a célunk lényegében meg is valósult. Azonban a szerverszoftver által használt valamely programkönyvtár kísérleti mivolta miatt még nem volt alkalmas a nyilvánosságra hozásra, valamint a numerikus számítás és a kommunikáció miatt a transzformált megjelenítése nem volt elég gyors.

Ezen első megközelítés jó ötleteit továbbvíve, a tanulságokat levonva egy kliensoldali, analitikus megoldást készítettem. A felhasználó a

[2] Nevezhetjük Bognár-fantomnak.



1. ábra. Képernyőkép a webes alkalmazásról: négyzet és ellipszis, valamint Radon-transzformáltjuk

webes felületen színes ellipszisekből és téglalapokból összeállíthat egy képet, majd az egyszerűsített transzformált máris megjelenik mellette, és azonnal követi a képen alkalmazott módosításokat, továbbá láthatjuk a transzformált mutatott pontjainak megfelelő egyeneseket az eredeti képen. Az alapötlet az volt, hogy nem minden képpontra számítjuk ki a transzformált értékét, hanem azt számoljuk, hogy hol vannak bizonyos jellegzetes pontok, ahol pl. egy-egy röntgensugár először belemetsz egy objektumba. Úgymond a transzformált részegységeinek határát, illetve „tartóját" számoljuk. Mindez teljesen a böngészőben zajlik. A program elérhető a

`https://locsi.web.elte.hu/radon/`

oldalon, érdemes máris kipróbálni.[3] Az 1. ábrán látható felülettel találkozhatunk, itt már elhelyeztünk egy ellipszist és egy négyzetet.

A matematikai leíráshoz tekintsük a sík egy ℓ egyenesét az origótól vett r távolságával és normálvektorának φ irányszögével megadva. Ekkor valamely f integrálható függvény Radon-transzformáltja a függvény lehetséges egyenesek mentén vett integrálja, $\mathcal{R}f(\ell) = \mathcal{R}f(\varphi, r) = \int_\ell f$. Ennek egy hasznos megfogalmazási módja az ℓ egyenes paraméteres

[3] A jelenleg választható színeket a *Sunny Bunnies* (Napnyuszik) című rajzfilmsorozat szereplői ihlették. (Digital Light Studio, Minszk, Fehéroroszország, 2015.)

leírását véve a következő:

$$\mathcal{R}f(\varphi, r) = \int_{\mathbb{R}} f(r \cdot \cos\varphi - s \cdot \sin\varphi,\ r \cdot \sin\varphi + s \cdot \cos\varphi)\,\mathrm{d}s,$$

ahol tehát egyenesünk átmegy az $(r \cdot \cos\varphi, r \cdot \sin\varphi)$ ponton és irányvektora $(-\sin\varphi, \cos\varphi)$, valamint $r, \varphi \in \mathbb{R}$.

Most viszont nem akarjuk integrálok (akár csak közelítő) értékét számolni, hanem arra vagyunk kíváncsiak, hogy egy-egy alakzat esetében – maradjunk most csak ellipsziseknél – hol találjuk azokat az egyeneseket, amelyek pontosan érintik azt. (Az egyenest kicsit megmozgatva nulla, vagy nem nulla integrálokat tapasztalnánk: ezek adják tehát a transzformált „határát".) Jelölje tehát $r^+(\varphi)$ és $r^-(\varphi)$ az origótól vett távolságát a φ szöggel adott egyenesek közül azoknak, amelyek pontosan érintik a szóban forgó ellipszist.

Elég könnyen látható, hogy egy origóban elhelyezett $a > 0$ sugarú kör esetén $r^{\pm}(\varphi) = \pm a$. Kis trigonometriai meggondolás után kijelenthetjük, hogy a (φ_0, r_0) polárkoordinátákkal adott középpontú a sugarú körre pedig $r^{\pm}(\varphi) = r_0 \cdot \cos(\varphi - \varphi_0) \pm a$. Kicsit hosszabb számolgatás szükséges ahhoz, hogy belássuk, hogy az origóban elhelyezett a és b (vízszintes és függőleges) féltengelyű ellipszis esetén

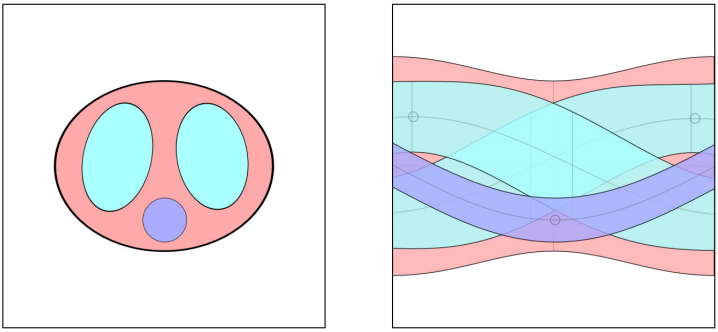
$$r^{\pm}(\varphi) = \pm\sqrt{b^2 \sin^2\varphi + a^2 \cos^2\varphi}$$

teljesül. Ellenőrizhetjük azonban, hogy mit kapunk a speciális $a = b$ esetben, illetve pl. 0 vagy $\pi/2$ szögekre. Innen viszont már csak egy kis lépés megfontolni, hogy egy általános helyzetű – a (φ_0, r_0) középpontba eltolt, valamint γ szöggel elforgatott, a és b féltengelyű – ellipszisre a végeredmény

$$r^{\pm}(\varphi) = r_0 \cdot \cos(\varphi - \varphi_0) \pm \sqrt{b^2 \sin^2(\varphi - \gamma) + a^2 \cos^2(\varphi - \gamma)}.$$

A Radon-transzformált szemléltetéséhez voltaképpen elegendő ezen függvényeket ábrázolnunk néhány pontjuk segítségével.

Így korábban megfogalmazott céljainkat elértük. Az elkészült szemléltető eszköz erejét jól mutatja, hogy egy tüdőfantom összeállítása és Radon-transzformáltjának vizsgálata immár bárki számára percek alatt kivitelezhető. Egy ilyet láthatunk a 2. ábrán; a világoskék ellipszisek a tüdőt, a lilás árnyalatú kör a gerincoszlopot kívánja modellezni.



2. ábra. Tüdőfantom és szinogramja a weben

Eredményeinket bemutattuk a 2020. évi MaCS konferencián *(13th Joint Conference on Mathematics and Computer Science),* mely online került megrendezésre, valamint összefoglaltuk a hányattatott sorsú [6] folyóiratcikkben. A témáról szívesen mesélek ezen túlmenően tehetséggondozó foglalkozásokon, szakkollégiumi előadásokon, vagy akár a Kutatók Éjszakáján mint tudománynépszerűsítő rendezvényen.

Számos továbbfejlesztési lehetőség elképzelhető a programhoz: pl. más síkidomok hozzáadása, zaj vizsgálata, GPU-s megvalósítás, szintvonalak vagy akár részletesebb szinogram hozzáadása, további funkciók (mentés, betöltés, színek, leírások stb.) Érdekes megismerni a Radon-transzformáció összefüggését a kétváltozós Fourier-transzformációval, valamint az inverz transzformáció működését is.

Érdeklődőknek ajánlom még Schipp Ferenc professzor úr jegyzetét, Samuli Siltanen professzor (Helsinki Egyetem, Finnország) előadásait, akinek látványos videóiban helyenként unikornisok is szerephez jutnak, valamint Todd Quinto professzor (Tufts Egyetem, Massachusetts, USA) munkásságát, akitől megtisztelő módon elismerő visszajelzést is kaptam a programmal kapcsolatban.

2. Mátrixok indukciós halmazairól

Eme témakör kutatása egy hallgató kérdésével indult, aki nagyon szerette volna megérteni – legalábbis a közelgő zárthelyi erejéig –, hogy mik is azok az indukált mátrixnormák; s a magyarázatok során olyan dolgokat is lerajzoltam, amilyeneket talán még mások sosem, és ezek meglepő felfedezésekhez vezettek, továbbá bőven kínálnak kutatási kérdéseket továbbra is.

Vektorok p-normáit, avagy hatványnormáit a szokásos módon adjuk meg $2 \leq n \in \mathbb{N}$ esetén:

$$\|.\|_p : \mathbb{R}^n \to \mathbb{R}, \qquad \|x\|_p = \left(\sum_{k=1}^{n} |x_k|^p \right)^{1/p} \qquad (p \in [1, \infty)),$$

valamint

$$\|x\|_\infty = \max_{k=1}^{n} |x_k| .$$

Ismeretes, hogy $\lim_{p \to +\infty} \|x\|_p = \|x\|_\infty$ $(x \in \mathbb{R}^n)$. Mátrixok indukált normáit a következőképpen (vektornormák által „indukálva") definiálhatjuk:

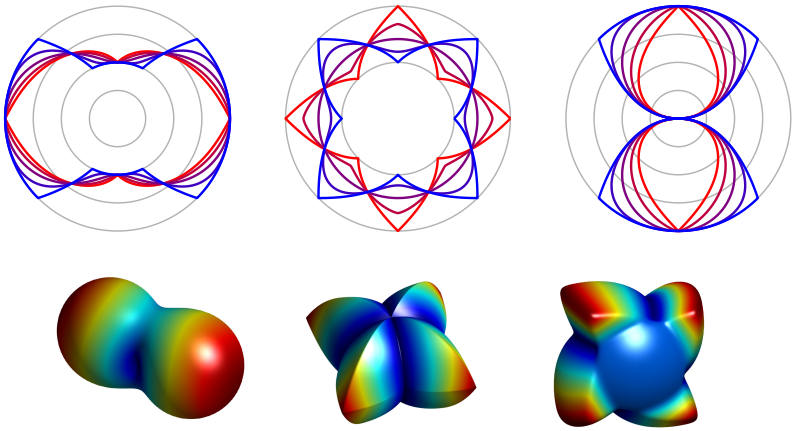
$$\|.\|_p : \mathbb{R}^{n \times n} \to \mathbb{R}, \qquad \|A\|_p = \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} \qquad (p \in [1, \infty]) .$$

Megtartjuk ugyanazt a jelölést mátrixok p-normáira, mint vektorok esetében. Tehát a mátrix „nagyságát" jellemzi, hogy a vele történő szorzás legfeljebb hányszorosára változtatja vektorok normáját („hosszát"). Ekvivalens módon tekinthetjük a szuprémumot csak az egységnyi hosszúságú vektorokra nézve, így megszabadulhatunk a törttől. (Sőt véges dimenziós terekben gondolhatunk egyszerűen maximumra.)

Definiáljuk most az indukciós halmazokat! Adott $A \in \mathbb{R}^{n \times n}$ mátrix, $2 \leq n \in \mathbb{N}$ és $p \in [1, \infty]$ esetén a

$$\mathcal{I}_p(A) := \left\{ \frac{\|Ax\|_p}{\|x\|_p} \cdot \frac{x}{\|x\|_2} \in \mathbb{R}^n \, : \, 0 \neq x \in \mathbb{R}^n \right\} \subset \mathbb{R}^n$$

halmazt az A mátrix p paraméterű *indukciós halmazának* nevezzük.



3. ábra. Indukciós görbék és felületek

Minden (kivéve a nulla) vektor helyett vizsgálhatunk irányonként egyetlen vektort (pl. az 1 normájút), hiszen az $x/\|x\|_2$ kifejezés szerepel a definícióban. Így voltaképpen ez a formula azt mondja, hogy vizsgáljuk meg minden irányban az $\|Ax\|_p / \|x\|_p$ tört értékét, azaz hogy adott irányban hányszorosára változik a vektorok hossza a mátrixszal való szorzás által. Végül minden irányban egy pontot kapunk egy origóra szimmetrikus objektumon. Az $n = 2$ esetben az előálló halmazt nevezhetjük *indukciós görbének,* $n = 3$ esetén *indukciós felületnek,* általánosan *indukciós sokaságnak.*

A 3. ábrán láthatunk néhány példát indukciós görbékre és felületekre. A felső sorban rendre egy diagonális, egy forgatási és egy szinguláris diagonális mátrix indukciós görbéit láthatjuk. Szürke körök az origótól vett távolságot jelölik, a színek (pirostól kékig) pedig a p értéket jelzik (1, $4/3$, 2, 4, $+\infty$). Az alsó sorban pedig néhány indukciós felületet találunk, melyek kódneve lehetne „földimogyoró”, „iránytű” és „propeller”. Mátrixok indukált normájához végül is a szóban forgó tört maximális értékét vesszük. Ez az ábrákon az origótól legmesszebb lévő pont(ok)nak felel meg.

Számos indukciós görbe és felület vizsgálata után arra a megfigyelésre juthatunk, hogy vannak olyan vektorok, irányok, amelyekhez tartozó

pontok adott mátrix bármely $p \in [1, +\infty]$ paraméterű indukciós halmazában benne vannak. A görbék esetében jól láthatók a közös metszéspontok. Ez azt jelenti, hogy ezen irányokban a $\|Ax\|_p / \|x\|_p$ tört értéke független p értékének megválasztásától. Mivel a mátrixok sajátvektorai ezzel a tulajdonsággal magától értetődően rendelkeznek, itt pedig jóval több ilyen vektort is megfigyelhetünk, ezért nevezhetjük ezeket *p-sajátvektoroknak*.

Mátrixok – hagyományos – sajátvektorai egyben p-sajátvektorok is tehát, viszont vannak olyan vektorok, amelyek p-sajátvektorok ugyan, de nem sajátvektorok; azaz nem triviális p-sajátvektorok. Szóval a sajátvektor fogalom egy általánosításához van szerencsénk. Analóg módon definiálhatók a p-sajátértékek is.

Néhány matematikai eredményt összefoglalva elmondhatjuk, hogy 2×2-es diagonális mátrixok esetén a (nem triviális) p-sajátértékek és p-sajátvektorok:

$$D = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}, \qquad \gamma = \sqrt{|ab|}, \qquad x_{1,2} = \begin{pmatrix} \pm\sqrt{|b|} \\ \sqrt{|a|} \end{pmatrix}.$$

Továbbá forgatási mátrixoknál

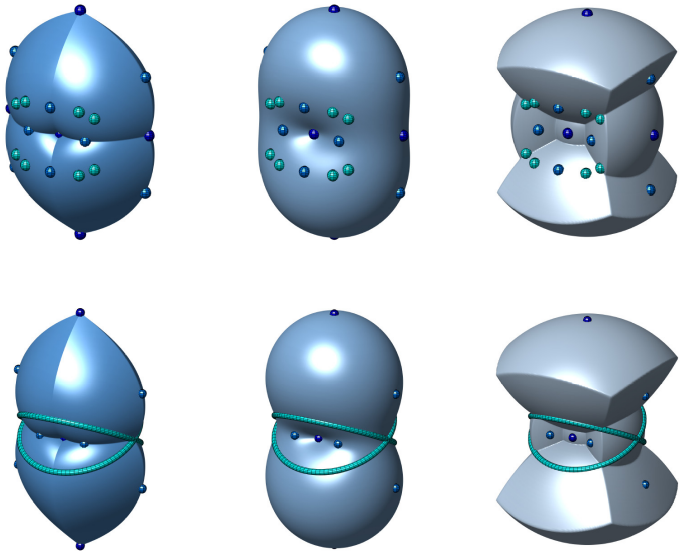
$$R(\varphi) = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix}, \qquad \gamma = 1, \qquad x_k = \begin{pmatrix} \cos\theta_k \\ \sin\theta_k \end{pmatrix},$$

ahol $\theta_k = -\frac{\varphi}{2} + k \cdot \frac{\pi}{4}$, valamint $\varphi \in \mathbb{R}$ és $k \in \{\,0, 1, 2, 3\,\} \subset \mathbb{Z}$. Eggyel magasabb dimenzióban, $\operatorname{diag}(a, b, c) \in \mathbb{R}^{3\times 3}$ diagonális mátrix esetén a fentiek megfelelőin túlmenően p-sajátértékek, -vektorok még:

$$\gamma = \sqrt[3]{abc}, \qquad x_1 = \begin{pmatrix} \sqrt[3]{bc^2} \\ \sqrt[3]{ca^2} \\ \sqrt[3]{ab^2} \end{pmatrix}, \qquad x_2 = \begin{pmatrix} \sqrt[3]{b^2c} \\ \sqrt[3]{c^2a} \\ \sqrt[3]{a^2b} \end{pmatrix}.$$

Általában pedig kiderül, hogy a p-sajátvektorok megadhatók az eredeti mátrix olyan variánsainak (klasszikus) sajátvektoraiként, amelyek úgy állnak elő, hogy a mátrix sorait permutáljuk, illetve ± 1-gyel szorozzuk.

A 4. ábrán a $\operatorname{diag}(1, 2, 3)$ és a $\operatorname{diag}(1, 2, 4)$ mátrixok esetében figyelhetjük meg az indukciós felületeket a $p = 1,\ 2,\ +\infty$ normákkal. Kis gömbök jelölik a p-sajátirányokat. Képzeljük el, ahogy soronként balról jobbra haladva p értékével bejárjuk a teljes $[1, +\infty]$ intervallumot, és a

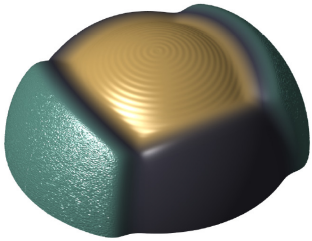


4. ábra. Két diagonális mátrix indukciós felületei három normával, p-sajátirányokkal

felület az ábrázolt képek között változik. A kis gömbök elhelyezkedése ugyanolyan: ezek egyúttal e végtelen sok felület közös pontjai, melyekből az első mátrix esetében összesen 34 van. A második esetben pedig egy olyan speciális esetet láthatunk, ahol bizonyos p-sajátirányok összeolvadnak, egy-egy kétdimenziós p-sajátalteret alkotva, mint egy öv. Ez a jelenség akkor lép fel, ha a három különböző diagonális elem közül az egyik a másik kettő mértani közepe.

Indukciós halmazokkal és p-sajátvektorokkal kapcsolatos első eredményeink a [3] írásban láttak napvilágot, majd a [4] cikkben vittük tovább a kutatást, és részleteztük p-sajátvektorok számítását, számosságát. Megfigyeléseimet előadtam 2019-ben Visegrádon a *Harmonikus Analízis és Rokon Területei* konferencián, majd 2022-ben az ausztriai Stroblban is bemutattam egy poszteren, a *Strobl22: Applied Harmonic Analysis and Friends* konferencián. Természetes kérdésként merült fel, hogy vajon a képét látva, a görbék néhány pontja alapján

meghatározhatók-e automatikusan az indukciós görbe paraméterei (a mátrix elemei és p értéke). Kapcsolódó vizsgálataimat a Nelder–Mead-féle szimplex algoritmus felhasználásával a 2022. évi MaCS konferencián ismertettem Kolozsváron, a Babeș–Bolyai Tudományegyetemen, majd az [5] dolgozatban jelent meg. Tekintettel arra, hogy $p = +\infty$ esetén az indukciós felületek koncentrikus gömbfelületekből, valamint ezeket folytonosan összekapcsoló részekből állnak, nagy p értékekre (pl. 8–64) gömbfelületek sima, differenciálható csatlakozását nyerhetjük. Egy – némi textúrával: hullámokkal, zajjal megspékelt – példát mutat erre az 5. ábra.[4] Az ilyen irányú alkalmazási lehetőségeket a számítógépes geometriai modellezésben a [7] cikkben jártam körül.



5. ábra. Koncentrikus, textúrázott gömbfelületek sima összekapcsolása indukciós felületek segítségével

A témakörben elkészült anyagok (cikkek, előadások) elérhetők a

`https://locsi.web.elte.hu/indsets/`

oldalon, köztük az itt bemutatottakhoz hasonló ábrák elkészítéséhez használható Matlab programokkal.

Itt is rengeteg további kutatási irány elképzelhető, számos nyitott kérdés van; többek között komplex p-sajátértékekkel, speciális mátrixok részletes jellemzésével, magasabb, akár végtelen dimenziós általánosításokkal (pl. Fourier-transzformáció, integráloperátorok), görbék és felületek identifikációjával, hatékony megjelenítéssel, valamint potenciális alkalmazási lehetőségekkel kapcsolatban.

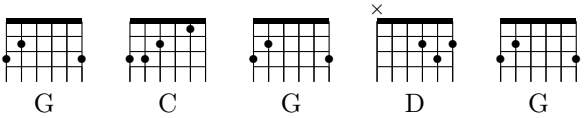
[4] A `https://coolors.co/` weboldal alapján az itt felhasznált színek a következők: Cambridge-kék, faszénszürke és Hunyadi-sárga.

3. Gábor-transzformáltak zérushelyeiről

A rövid idejű, avagy „ablakolt” Fourier-transzformáció *(STFT, Short-Time Fourier Transform)* a zenei kotta matematikai megfelelője. Egy jel – pl. hang mint időben változó rezgés – feldolgozása során általa nyerhetünk olyan információkat, hogy melyik időpillanatban milyen frekvenciakomponensek vannak jelen. Általában Gábor-transzformációról akkor beszélünk, ha a Gauss-féle haranggörbét használjuk „ablakfüggvénynek”. Neve Gábor Dénesnek állít emléket, akinek munkássága meghatározó volt a tudományterületen (és aki a fizikai Nobel-díjat 1971-ben a holográfia módszerének feltalálásáért és kidolgozásáért kapta). Ilyen transzformációk matematikai elemzése a harmonikus analízis témakörébe illeszkedik. Természetesen azóta sok más módszert is kifejlesztettek, melyek elmélete és alkalmazása ma is igen aktívan kutatott terület, és szinte mindenki élvezi mindennek gyümölcsét, mikor pl. `mp3` formátumú fájlok közvetítésével hallgat zenét. Sőt voltaképpen az emberi hallás is így működik.

6. ábra. Rövid-idejű Fourier-transzformáltak

A 6. ábrán két példát láthatunk egy-egy zenei motívum rövid-idejű Fourier-transzformáltjára. A bal oldalin felismerhetjük a *Megy a gőzös* kezdetű közismert dal első sorát (körtemuzsikán elfújva). A jobb oldalin pedig – bár nehezebben követhető – a



akkordok váltakozását figyelhetjük meg, a végén kis díszítéssel (gitáron pengetve).

Tipikusan a Gábor-transzformált domináns, nagy értékei a fontosak a jelfeldolgozás szempontjából, azonban az utóbbi évtizedekben egyre nagyobb figyelmet kaptak a transzformált zérushelyei is, és sok érdekes eredmény eredmény született. (Említendő pl. a fázis deriváltjának szinguláris viselkedése a zérushelyek közelében, zajos jelek transzformáltja zérushelyeinek elhelyezkedési statisztikái, zajszűrési lehetőségek a zérushelyek alapján.)

Ismert volt, hogy két azonos amplitúdójú egyszerű szinuszos jel összege Gábor-transzformáltjának zérushelyei periodikusan, az átlagfrekvencia mentén helyezkednek el. Még doktorandusz koromban találkoztam azzal a problémával, amely még nemrég is megoldatlannak bizonyult, hogy két *különböző* amplitúdójú szinuszos jel összege esetén hogyan módosulnak a zérushelyek. Ábráinkon az jól látható volt, hogy kicsit eltolódnak, de vajon kiszámolhatók-e pontosan, vagy legalább adhatunk-e jó közelítést elhelyezkedésükre?

Czirkos Angéla[5] vállalta, hogy témavezetésemmel készülő mesterszakos diplomamunkájában [1] utánajár ennek a kérdésnek. Számos numerikus vizsgálatot elvégzett a zérushelyek közelítő meghatározására, valamint analitikus számításokkal is sikerült eredményeket elérnie több ablakfüggvény, különböző típusú jelek esetében.

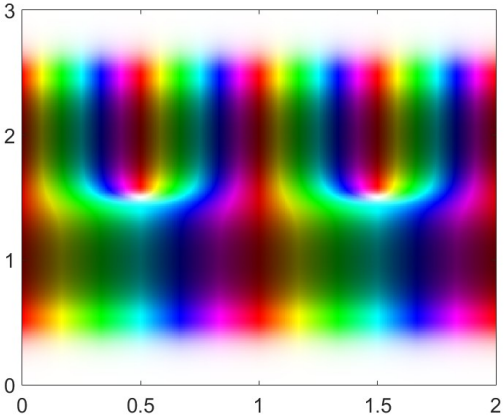
Legyen $f\colon \mathbb{R} \to \mathbb{C}$ a feldolgozni kívánt függvényünk, $g\colon \mathbb{R} \to \mathbb{R}$ pedig az ablakfüggvény. (A releváns függvényterek megnevezésén és részletezésén itt elegánsan átlépünk.) Az f függvény rövid-idejű Fourier-transzformáltját a g ablakfüggvénnyel az alábbi formulával adjuk meg:

$$V_g f\colon \mathbb{R}^2 \to \mathbb{C}, \qquad V_g f(x,\omega) = \int_{\mathbb{R}} f(t) \cdot \overline{g(t-x)} \cdot e^{-2\pi i t\omega}\,\mathrm{d}t.$$

Tehát a g eltoltjaival szorozzuk f-et, mintha annak csak egy rövid részletét látnánk (egy „ablakon” keresztül), majd ennek képezzük a Fourier-transzformáltját. Az itt szereplő exponenciális tényezőben ω jelöli a frekvenciát. Elterjedt a transzformált

$$W_g f\colon \mathbb{R}^2 \to \mathbb{C}, \qquad W_g f(x,\omega) = \int_{\mathbb{R}} f(t) \cdot \overline{g(t-x)} \cdot e^{-2\pi i (t-x)\omega}\,\mathrm{d}t$$

[5] EJC: 2018–2019.



7. ábra. Kétkomponensű jel Gábor-transzformáltja

változata is, ahol úgymond a fázis is csúszik az ablakkal együtt. A két változat végeredményben egy egységnyi abszolút értékű komplex szorzóban tér el egymástól, így a zérushelyeiket ugyanott találjuk. Ábrázolás szempontjából W előnyösebb, míg a számolások szempontjából V sokszor könnyebben kezelhető. Ablakfüggvénynek pedig vegyük az egyszerű e^{-x^2} Gauss-féle haranggörbe $\sigma/\sqrt{\pi}$ szórású változatát:

$$g\colon \mathbb{R} \to \mathbb{R}, \qquad g(t) = \exp \frac{-\pi t^2}{2\sigma^2}.$$

Tekintsünk először egy ω_0 frekvenciájú komplex szinuszos jelet:

$$f(t) := \exp\left(2\pi i t \omega_0\right).$$

Ennek valós része az Euler-formula alapján a koszinusz, képzetes része pedig a szinusz függvénnyel írható fel. Ekkor

$$W_g f(x,\omega) = \exp\left(2\pi i x \omega_0\right) \cdot \exp\left(-2\pi\sigma^2(\omega-\omega_0)^2\right)$$

adódik, vagyis az időtengely mentén periodikusan ismétlődő fázissal, a frekvenciatengely mentén egy eltolt haranggörbével találkozhatunk.

Legyen most a függvényünk két különböző (ω_1 és ω_2) frekvenciájú, azonos amplitúdójú komplex szinuszos jel összege.

$$f(t) := \exp(2\pi i t\omega_1) + \exp(2\pi i t\omega_2).$$

Ekkor a Gábor-transzformált – lévén lineáris – a fentivel egyező két tag összegeként írható fel; zérushelyei pedig ismertek, koordinátáikat az

$$(x, \omega) = \left(\frac{2k+1}{2(\omega_1 - \omega_2)}, \frac{\omega_1 + \omega_2}{2} \right) \qquad (k \in \mathbb{Z})$$

formula adja meg. Vagyis az átlagfrekvencia mentén helyezkednek el, időben pedig periódusuk a két frekvencia különbségétől függ. Erre láthatunk egy példát a 7. ábrán, ahol $\omega_1 = 1$, $\omega_2 = 2$. A vízszintes tengelyen az időt, a függőleges tengelyen a frekvenciát ábrázoltuk. A sötétebb és világosabb árnyalatok most is a nagyobb és kisebb abszolút értékeket jelzik (a valódi értékek logaritmusával arányosan) – látható a két sötét sáv a megjelenő frekvenciáknál –, viszont ezúttal színekkel jelöltük a komplex értékek fázisát.[6] Így gyönyörűen kirajzolódnak a zérushelyek, összhangban a fenti formulával.

Végül vegyük két különböző frekvenciájú és *különböző amplitúdójú* komplex szinuszos jel összegét. A frekvenciákat jelölje ismét ω_1 és ω_2, az amplitúdók pedig legyenek α és β pozitív számok. Vagyis

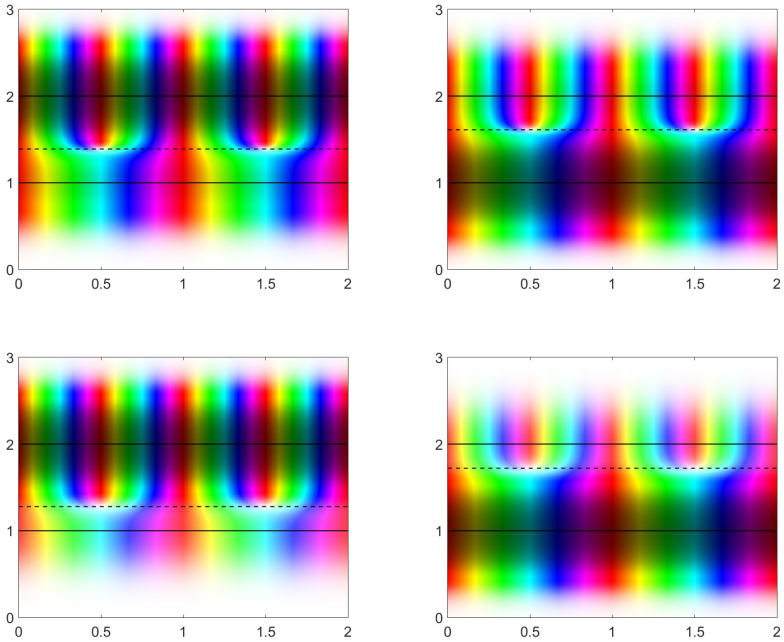
$$f(t) := \alpha \cdot \exp(2\pi i t\omega_1) + \beta \cdot \exp(2\pi i t\omega_2).$$

Kiderült, hogy a zérushelyek ebben az esetben is felírhatók, méghozzá a következőképpen:

$$(x, \omega) = \left(\frac{2k+1}{2(\omega_1 - \omega_2)}, \frac{\omega_1 + \omega_2}{2} + \frac{\ln(\beta/\alpha)}{4\pi\sigma^2(\omega_1 - \omega_2)} \right) \qquad (k \in \mathbb{Z}).$$

Ez tehát azt jelenti, hogy a zérushelyek az amplitúdók arányának logaritmusával tolódnak el az átlagfrekvenciához képest, méghozzá a gyengébb komponens irányába. Az eltolódás mértéke pedig függ a Gauss-ablak szórásától is.

[6] Komplex függvények színes ábrázolása is igen kedves téma számomra. Idevágó mondanivalómat a `https://locsi.web.elte.hu/complexdemo/` honlapon gyűjtöttem össze.



8. ábra. Különböző amplitúdójú szinuszoidok összegének Gábor-transzformáltjai, változó amplitúdóarányokkal

Ezt a jelenséget kívánja alátámasztani a 8. ábra, ahol ismét az $\omega_1 = 1$ és $\omega_2 = 2$ frekvenciák jelennek meg, viszont ezúttal egyikük amplitúdója többszöröse a másikénak (méghozzá a felső sorban 4-szerese, az alsó sorban 16-szorosa). A bal oldalon a magasabb frekvencia erősebb, a jobb oldalon pedig az alacsonyabb. Ez tükröződik a sötétebb sávok árnyalataiban, továbbá megfigyelhető a zérushelyek eltolódása is. A jobb átláthatóság kedvéért segédvonalakkal jelöltük a függvényben szereplő frekvenciákat, valamint a zérushelyek frekvenciáját[7] is, utóbbit szaggatott vonallal.

A diplomamunkában szerepelnek számítások más ablakfüggvények, komplex helyett valós jelek esetére is. A főbb eredményeket tervezzük

[7] Nevezhetjük Czirkos-frekvenciának.

bemutatni a *Strobl24: More on Harmonic Analysis* konferencián egy poszteren, valamint folyóiratcikk is előkészületben van.

Ezen a területen is nevesíthetünk további kutatási irányokat. Érdekes lehet vizsgálni még több (három, négy stb.) konstans frekvencia összegét, a zérushelyek elhelyezkedését más típusú jelek esetén, zaj perturbációs hatását, vagy akár más transzformációk kapcsolódó kérdéseit.

Összefoglalás

Ebben az írásban egy tömör összefoglalását adtuk adjunktusi kutatómunkánknak a numerikus analízis, a digitális jelfeldolgozás néhány témaköréhez kapcsolódóan. Immár témavezetésünkkel készült diplomamunkák eredményeiről is beszámolhattunk.

Bemutattuk a Radon-transzformáció működésének interaktív szemléltetésére létrehozott programunkat. Ezután ismertettük mátrixok indukciós halmazaival (indukciós görbékkel, indukciós felületekkel) és p-sajátvektoraival kapcsolatos matematikai eredményeinket, megemlítve ezek identifikációjának problémáját numerikus közelítő eljárásokkal, valamint kilátásba helyezve alkalmazási területeket is. Végül Gábor-transzformáltak zérushelyeivel foglalkoztunk, megnyugtatóan tisztázva két különböző amplitúdójú komplex szinuszos jel esetét.

Természetesen az említett témakörökhöz kapcsolódóan az egyes fejezetek végén megemlíthettünk további lehetséges kutatási irányokat akár matematikusi, akár programozói, akár mérnöki szemmel nézve. Több linket is kiemeltünk, amelyeken az érdeklődő Olvasót további kapcsolódó anyagok fogadják.

Hivatkozások

[1] A. Czirkos, *Investigating zeros of the STFT of multi-sinusoidal signals*, MSc thesis (supervisor: L. Lócsi), ELTE Faculty of Informatics, Department of Numerical Analysis, Budapest, 2024.

[2] Zs. Gál, *Theory, visualization and applications of the Radon transform* (in Hungarian), MSc thesis (supervisor: L. Lócsi), ELTE Faculty of Informatics, Department of Numerical Analysis, Budapest, 2020.
`https://edit.elte.hu/xmlui/handle/10831/56156`

[3] L. Lócsi, Introducing p-eigenvectors, exact solutions for some simple matrices, *Ann. Univ. Sci. Budapest. Sect. Comput.,* 49 (2019), pp. 325–345.
`http://ac.inf.elte.hu/Vol_049_2019/325_49.pdf`

[4] L. Lócsi, Zs. Németh, On the construction of p-eigenvectors, *Ann. Univ. Sci. Budapest. Sect. Comput.,* 50 (2020), pp. 231–247.
`http://ac.inf.elte.hu/Vol_050_2020/231_50.pdf`

[5] L. Lócsi, Identification of induction curves, *Proc. 14th Joint Conf. on Math. and Comp. Sci. (MaCS 2022), Stud. Univ. Babeş–Bolyai Math.,* 68(3) (2023), pp. 467 –480.
`https://doi.org/10.24193/subbmath.2023.3.01`

[6] L. Lócsi, Zs. Gál, Real-time web-based visualization of the Radon transform, *Proc. 13th Joint Conf. on Math. and Comp. Sci. (MaCS 2020), Ann. Univ. Sci. Budapest. Sect. Comput.,* submitted.

[7] L. Lócsi, Application perspectives of induction surfaces in CAD, smooth connection of concentric spherical surfaces, *Proc. CAD 2024, Computer-Aided Design and Applications,* submitted.

Az egyszerű típuselmélet algebrai reprezentációi[‡]

Luksa Norbert[*]

Eötvös József Collegium[**]

luksan@inf.elte.hu

Témavezető: Kaposi Ambrus
ELTE IK Programozási Nyelvek és Fordítóprogramok Tanszék

Áttekintés

Hagyományosan a programozási nyelveket szintaxissal, operációs szemantikával, típusrendszerrel szokás definiálni. Az algebrai leírás egy magasabb szintű megadási mód, mely mindegyiket egyszerre tartalmazza a felsoroltak közül. Egy programozási nyelv ilyen úton történő definícióján csak olyan konstrukciók adhatók meg, amelyek megtartják a típusokat és az operációs szemantika egyenlőségeit. A programozási nyelv hagyományos leírása esetén a definiált nyelv helyességbizonyításának egy fontos része a típusmegőrzés (tárgyredukció) tétele. Ez a tétel az algebrai leírásban triviálisan teljesül.

A legtöbb programozási nyelv típusrendszerének alapja az egyszerű típuselmélet (simple type theory, egyszerűen típusozott lambda-kalkulus). Az egyszerű típuselméletet algebrai struktúraként adta meg Castellan, Clairambault és Dybjer az egyszerűen típusozott családos kategória (simply typed category with families) fogalmával. Jelen írás

[‡] A szerző 2021. évi OTDK-ra benyújtott dolgozatának rövidített változata.
[*] ELTE Informatikai Kar
[**] 2015–2021

TDK dolgozatom [6] kivonata, melyben megvizsgáljuk az egyszerű típuselmélet egy alternatív algebrai leírását, melyet teleszkopikusnak nevezünk. A teleszkopikus leírás párhuzamos helyettesítések helyett csak egyszeri helyettesítést és gyengítést tartalmaz. Ez a leírás közelebb van a lambda-kalkulus szokásos informális leírásához. A dolgozatban megmutatjuk a kategorikus és a teleszkopikus leírás ekvivalenciáját, és használatukat néhány modellen keresztül.

1. Bevezető

A programozási nyelvek formális leírására számos eszköz áll a rendelkezésünkre. Leggyakrabban először definiáljuk a nyelv szintaktikáját, majd megadjuk a szemantikát átírási szabályok segítségével (operációs szemantika). Ezen felül további szabályokkal szigorítja a nyelv működését a típusrendszer bevezetése.

A fenti szabályok lépésenkénti alkalmazása egy egyre szűkebb halmazt definiál. Egyrészt érvénytelennek minősít néhány programot, továbbá az előző lépésben még különböző kifejezések között egyenlőséget definiál. A programok helyességének ellenőrzése ezeken a halmazokon halad végig (lásd 1. ábra).

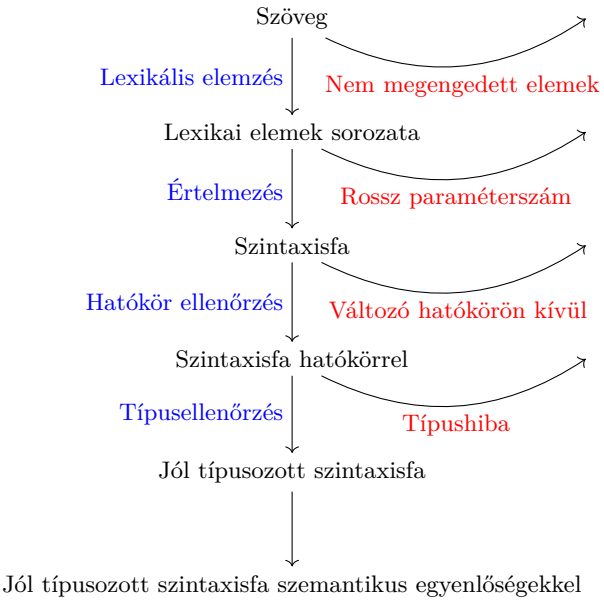
Elsőként a lexikális elemzéssel kiszűrjük a nem megengedett lexikai elemeket, szövegből ilyen elemek sorozatát definiáljuk. Ugyanakkor ezzel egyenlőséget is teszünk a különböző számú szóközökkel („whitespace”-ekkel) elválasztott kifejezések közé.

Az értelmezés során a programot összevetjük a szintaktikai szabályokkal. Elvetjük a hibás kulcsszavakat, rosszul felparaméterezett függvényeket, hiányos zárójelpárokat, stb. Ezek mellett egyenlőség kerül például a feleslegesen zárójelezett kifejezések közé.

A hatókörök ellenőrzésével kiszűrjük a nem deklarált változókat, míg az egy hatókörben található változók közötti átnevezések (α-konverziók) nem változtatnak a programon.

Típusellenőrzés után a hibásan típusozott kifejezések is kívül esnek a helyes programok halmazán.

Vegyük észre, hogy az eddig felsorolt ellenőrzések ugyan nagy jelentőséggel bírnak a fordító számára, szemantikai tartalmat azonban nem adnak az elmélethez. Ezekkel az eszközökkel meg tudunk adni egy szintaktikailag korrekt, típushelyes programot, ugyanakkor a legfontosabb



1. ábra. Programok ellenőrzése

szempont még nem került számításba: a program jelentése. Szemantikailag ugyanazt a programot tehát le tudjuk írni többféleképp, úgy hogy a fenti procedúra során számos ekvivalenciaosztályt kapjunk eredményül.

Ideális esetben számunkra ugyanazt az eredményt (avagy normálformát) adó program pontosan egy ekvivalenciaosztályba esik, így ezek a programok egyenlőségek mentén egymásba vihetőek. A programozási nyelvnek egy ilyen jellegű definícióját kísérelhetünk meg megadni algebrai leírást használva.

Az algebrai leírásban egyszerre adhatjuk meg a szintaktikát, a típusrendszert és a szemantikát is, így az azonos jelentésű programok között egyenlőségeket definiálhatunk, így ezek a programok ebben a felírásban egy ekvivalenciaosztályba eshetnek.

A *függő* típuselmélet [7] egy olyan formális rendszer, melyre felépíthető a matematika, kiváltva az elsőrendű logikát és a halmazelméletet. Az *egyszerű* típuselmélet, és annak bővített változatai az erősen típu-

sozott programozási nyelvek elméleti alapját képezik. Alapvetése, hogy minden kifejezése rendelkezik egy típussal. Előnye programozási nyelvek tárgyalása során mutatkozik igazán, hisz az iménti tulajdonság a legtöbb programozási nyelvben szintén nagy hangsúllyal jelen van.

Az egyszerű típuselmélet leggyakoribb reprezentációja az egyszerűen típusos λ-kalkulus. Jelen dolgozatban az egyszerű típuselmélet kategorikus, illetve teleszkopikus leírásával fogunk foglalkozni. Mivel ezeket algebrai struktúraként adjuk meg, így a reprezentációk között az egyik legmagasabb szintű leírást használjuk.

2. Algebrai leírás

A matematikában algebrai struktúra alatt egyszerű esetben egy, összetettben több alaphalmazból, az alaphalmaz(ok)on értelmezett véges aritású műveletek halmazából, illetve az ezekre a műveletekre vonatkozó véges számú egyenlőségeket (avagy axiómákat) tartalmazó halmazból álló struktúrát értünk. Összetett esetben n alaphalmaz esetén azt mondjuk, hogy a struktúra n-szortú.

2.1. *Példa* (Csoport). A matematikában az asszociatív, invertálható grupoidokat csoportoknak nevezzük. A csoportot definiálhatjuk algebrai struktúraként is:

$$\begin{array}{lll} \mathsf{C} & : \mathsf{Set} & \mathsf{ass} \;\; : (a \otimes b) \otimes c \equiv a \otimes (b \otimes c) \\ - \otimes - & : \mathsf{C} \to \mathsf{C} \to \mathsf{C} & \mathsf{idl} \;\; : \mathsf{u} \otimes a \equiv a \\ \mathsf{u} & : \mathsf{C} & \mathsf{idr} \;\; : a \otimes \mathsf{u} \equiv a \\ -^{-1} & : \mathsf{C} \to \mathsf{C} & \mathsf{invl} \; : a^{-1} \otimes a \equiv \mathsf{u} \\ & & \mathsf{invr} \; : a \otimes a^{-1} \equiv \mathsf{u} \end{array}$$

A továbbiakban Set elemeként tüntetjük ki az alaphalmazokat, így a fenti példában C lesz a tetszőleges alaphalmaz, $- \otimes -$, u, és $-^{-1}$ a műveletek, míg ass, idl, idr, invl, és invr az axiómák.

Definiáljuk a struktúra modelljét, a modellek morfizmusát:

2.1. Definíció (Modell[1])**.** Egy algebrai struktúra modellje alatt az alaphalmaz(ok) és az operátorok megfeleltetésével kapott struktúrát értjük, melyek teljesítik a struktúra egyenlőségeit.

[1] Amit itt *csoportmodellnek* nevezünk, azt a matematikában általában egyszerűen csoportnak nevezik.

2.2. Definíció (Modellmorfizmus)**.** Modellek közötti művelettartó leképezést modellmorfizmusnak nevezzük.

2.2. *Példa* (Csoport modellmorfizmusa). A csoporton értelmezett modellmorfizmus alatt az alábbi CsopMor leképezést értjük:

$$\begin{aligned}
\mathsf{CsopMor} : \quad & \mathsf{CsopMod} \to \mathsf{CsopMod} \to \mathsf{Set} \\
\mathsf{CsopMor}\mathcal{MN} := (\mathsf{C} \ & : \mathsf{C}_{\mathcal{M}} \to \mathsf{C}_{\mathcal{N}}) \\
\times \ (\otimes \ & : \mathsf{C}(a \otimes_{\mathcal{M}} b) \equiv \mathsf{C}a \otimes_{\mathcal{N}} \mathsf{C}b) \\
\times \ (\mathsf{u} \ & : \mathsf{C}(\mathsf{u}_{\mathcal{M}}) \equiv \mathsf{u}_{\mathcal{N}}) \\
\times \ (^{-1} \ & : \mathsf{C}(a^{-1_{\mathcal{M}}}) \equiv (\mathsf{C}a)^{-1_{\mathcal{N}}})
\end{aligned}$$

ahol CsopMod a csoportmodellek halmaza. Az adott modell elemét az alsóindexben feltüntetett névvel jelöljük.

2.3. Definíció (Szintaxis)**.** A szintaxis az iniciális modell, azaz az a modell, melyből pontosan egy morfizmus megy a leírás minden más modelljébe.

Matematikai fogalmak mellett akár programokat is definiálhatunk algebrai struktúraként. Konkrét példán keresztül tekintsük egy egyszerűbb nyelv (nevezzük NatBool-nak) szintaxisát és operációs szemantikáját, lásd 2. ábra.

Jól látszik, hogy teljesen elkülönül a szintaktika és a szemantika megadása. Először definiáljuk a Ty típusok, illetve a Tm termek lehetséges elemeit, majd megadjuk, hogyan tudjuk ezeket az elemeket létrehozni, illetve átalakítani különböző átmeneti szabályok segítségével. Ugyancsak fontos észrevétel, hogy a kifejezések típusai szintén csak a szemantikai szabályokban jelennek meg. Észrevehetjük még, hogy a szabályokkal pontosan definiálhatóak a kiértékelés egyes lépései, beleértve a paraméterek kiértékelési sorrendjét is. (Számos különböző fajta operációs szemantika létezik, számunkra ez a különbség most nincs jelentőséggel.)

Ennél magasabb szintű felírást biztosít számunkra az algebrai struktúra, lásd 3. ábra. Mint láthatjuk, itt is ugyanazt a nyelvet formalizáltuk, mint a fenti ábrán. Összehasonlítva a két reprezentációt szembeötlő lehet, hogy ezúttal nem különül el a szintaxis és a szemantika, egyszerre kerülnek megadásra. Ezúttal a Tm és a Ty nem csupán a szintaktikai felírást megkönnyítő jelölések, hanem a nyelv részét képező kifejezések. Ennek következményeképp a többi kifejezéshez hasonlóan a Tm és a

$$\mathsf{Ty} \;::= \mathsf{Bool} \mid \mathsf{Nat}$$
$$\mathsf{Tm} ::= \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\, t\, \mathsf{then}\, t'\, \mathsf{else}\, t'' \mid \mathsf{num}\, n \mid \mathsf{isZero}\, t$$

$$(- : -) \subseteq \mathsf{Tm} \times \mathsf{Ty} \qquad (- \mapsto -) \subseteq \mathsf{Tm} \times \mathsf{Tm}$$

$$\frac{}{\mathsf{true} : \mathsf{Bool}} \qquad \frac{}{\mathsf{false} : \mathsf{Bool}} \qquad \frac{t : \mathsf{Bool} \qquad t' : A \qquad t'' : A}{\mathsf{if}\, t\, \mathsf{then}\, t' \mathsf{else}\, t'' : A}$$

$$\frac{n \in \mathbb{N}}{\mathsf{num}\, n : \mathsf{Nat}} \qquad \frac{t : \mathsf{Nat}}{\mathsf{isZero}\, t : \mathsf{Bool}}$$

$$\frac{}{\mathsf{if}\, \mathsf{true}\, \mathsf{then}\, t\, \mathsf{else}\, t' \mapsto t} \qquad \frac{}{\mathsf{if}\, \mathsf{false}\, \mathsf{then}\, t\, \mathsf{else}\, t' \mapsto t'}$$

$$\frac{t \mapsto t_1}{\mathsf{if}\, t\, \mathsf{then}\, t'\, \mathsf{else}\, t'' \mapsto \mathsf{if}\, t_1\, \mathsf{then}\, t'\, \mathsf{else}\, t''} \qquad \frac{}{\mathsf{isZero}\,(\mathsf{num}\, 0) \mapsto \mathsf{true}}$$

$$\frac{}{\mathsf{isZero}\,(\mathsf{num}\,(1+n)) \mapsto \mathsf{false}} \qquad \frac{t \mapsto t'}{\mathsf{isZero}\, t \mapsto \mathsf{isZero}\, t'}$$

2. ábra. A NatBool nyelv felírása szintaxis és operációs szemantika segítségével.

Ty kifejezések is rendelkeznek saját típussal: Ty egy Set (alaphalmaz) típusú kifejezés, míg Tm egy Ty-től függő Set, azaz egy típusok felett értelmezett alaphalmaz.

Az átmeneti szabályok helyett itt egyenlőségekkel adjuk meg a kifejezések közötti relációt, melyről tudjuk, hogy reflexív, szimmetrikus és tranzitív. Fontos különbség azonban, hogy itt nem szerepelnek a kiértékelés sorrendjére vonatkozó szabályok megfelelői. Korábban ezekre azért volt szükség, hogy biztosíthassuk, hogy az if–then–else– feltételében megjelenik a true vagy false, így tudjuk alkalmazni a megfelelő elimináló szabályt. Az algebrai felírásban erre nincs szükség, hisz itt egyenlőségek formájában adjuk meg az átmeneteket:

2.3. *Példa.* Tekintsük az $u \equiv \mathsf{if}\, \mathsf{isZero}\,(\mathsf{num}\, 0)\, \mathsf{then}\, t\, \mathsf{else}\, t'$ programot. Ekkor $\mathsf{isZero}\beta_1$ miatt $u = \mathsf{if}\, \mathsf{true}\, \mathsf{then}\, t\, \mathsf{else}\, t'$, ahol $\mathsf{if}\beta_1$ és az egyenlőség tranzitivitása miatt következik, hogy $u = t'$.

Ezzel tehát láthatjuk, hogy az algebrai leírásban a kiértékelés sorrendjére vonatkozó szabályokat elhagyhatjuk, és így egyszerűsödik a rendszerünk (úgymond kevesebb a „boilerplate”).

$$
\begin{array}{ll}
\mathsf{Ty} & : \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} \\
\mathsf{Bool} & : \mathsf{Ty} \\
\mathsf{Nat} & : \mathsf{Ty} \\
\mathsf{true} & : \mathsf{Tm}\,\mathsf{Bool} \\
\mathsf{false} & : \mathsf{Tm}\,\mathsf{Bool} \\
\mathsf{if\_then\_else\_} & : \mathsf{Tm}\,\mathsf{Bool} \to \mathsf{Tm}\,A \to \mathsf{Tm}\,A \to \mathsf{Tm}\,A \\
\mathsf{num} & : \mathbb{N} \to \mathsf{Tm}\,\mathsf{Nat} \\
\mathsf{isZero} & : \mathsf{Tm}\,\mathsf{Nat} \to \mathsf{Tm}\,\mathsf{Bool} \\
\mathsf{if}\beta_1 & : \mathsf{if}\,\mathsf{true}\,\mathsf{then}\,t\,\mathsf{else}\,t' = t \\
\mathsf{if}\beta_2 & : \mathsf{if}\,\mathsf{false}\,\mathsf{then}\,t\,\mathsf{else}\,t' = t' \\
\mathsf{isZero}\beta_1 & : \mathsf{isZero}\,(\mathsf{num}\,0) = \mathsf{true} \\
\mathsf{isZero}\beta_2 & : \mathsf{isZero}\,(\mathsf{num}\,(1+n)) = \mathsf{false}
\end{array}
$$

3. ábra. Az NatBool nyelv algebrai modellje.

További előnye az algebrai leírásnak, hogy mivel a szintaxis, típusrendszer és szemantika egyszerre van definiálva, ezért egy kifejezés megadásával egyben típushelyes, szemantikailag is korrekt kifejezést kapunk. Ennek köszönhetően számos ezekre a tulajdonságokra vonatkozó tétel, mint például a típusmegőrzés tétele triviálisan teljesül, nem kell külön belátni.

A fentiek alapján megadhatjuk a NatBool nyelv morfizmusát is, továbbá megfeleltethetünk egyéb programozásból ismerős kifejezéseket is (például interpreter).

3. Egyszerű típuselmélet

Ebben a fejezetben bevezetjük a típuselmélet fogalmait az egyszerűen típusos λ-kalkulus segítségével [2]. Problémafelvetésekkel hívjuk fel a figyelmet a nyelv nehézségeire, melyekre későbbi fejezetekben mutatunk megoldásokat.

3.1. Egyszerűen típusos λ-kalkulus

Elsőként megadjuk a nyelv kifejezéseit.

3.1. Definíció (λ-kifejezés)**.**

$$\begin{aligned}\langle \lambda\text{-}\textit{kifejezés}\rangle \equiv\ & \langle \textit{változó}\rangle \\ \mid\ & \lambda\langle \textit{változó}\rangle.\langle \lambda\text{-}\textit{kifejezés}\rangle \\ \mid\ & \langle \lambda\text{-}\textit{kifejezés}\rangle\langle \lambda\text{-}\textit{kifejezés}\rangle\end{aligned}$$

Egy λ-*kifejezés* lehet egy változó, egy absztrakció, illetve egy applikáció (a fenti definíció sorrendjében), ezeket t, u, v betűkkel fogjuk jelölni. A változók a programozásban szokásos konstrukciót jelölik, ezeket mi x, y, z-vel fogjuk jelölni.

Az absztrakciókkal függvényeket tudunk leírni, ahol egy $\lambda x.t$ kifejezésben t függhet az x változótól. A t-t nevezzük az absztrakció törzsének. Azt mondjuk, hogy az absztrakció megköti az x változót t-ben, ekkor x kötött változó. A nem kötött változót szabadnak nevezzük. Azokat a λ-*kifejezés*eket, amelyekben szerepel szabad változó, nyíltnak nevezünk, amikben nem, azokat zártnak. Egymás utáni absztrakciók esetén az alábbi rövidítést alkalmazzuk: $\lambda x.\lambda y.t \equiv \lambda x.(\lambda y.t) \equiv \lambda xy.t$. Mint láthatjuk, az absztrakció jobbasszociatív művelet.

Az applikációkkal van lehetőségünk a függvények kiértékelésére, amennyiben az applikáció első λ-*kifejezés*e egy absztrakció. Akkor az absztrakció törzsébe behelyettesítjük x összes előfordulási helyére a második λ-*kifejezés*t.

3.2. Definíció (Helyettesítés)**.** A helyettesítést strukturális indukcióval definiáljuk:

$$y[x := u] \equiv \begin{cases} u & \text{ha } x \equiv y, \\ y & \text{egyébként;} \end{cases}$$

$$\lambda y.t[x := u] \equiv \begin{cases} \lambda y.t & \text{ha } x \equiv y \text{ vagy } y \text{ szabad } t\text{-ben}, \\ \lambda y.(t[x := u]) & \text{egyébként;} \end{cases}$$

$$(tv)[x := u] \equiv (t[x := u])(v[x := u]).$$

Jól látszik, hogy a helyettesítéssel egy változónak az összes szabad előfordulását lecseréljük egy paraméterül adott kifejezésre.

Az absztrakciónál számos kényelmetlen feltételt láthatunk. Fontos, hogy csak a szabad változókat cserélhetjük le, hiszen ha egy kötött

változóba helyettesítenénk be egy kifejezést, azzal teljesen megváltoztathatjuk annak értelmét. A másik irányból hasonlóan: szabad változó se válhat kötötté, hisz ekkor is megváltozna a kifejezés jelentése.

A helyettesítés segítségével már definiálhatjuk a λ-kalkulus redukciós műveletét:

3.3. Definíció (β-redukció)**.** Legyen $t \equiv (\lambda x.u)v$. Ekkor ha a v szabad változói nem válnak u kötött változóivá:

$$(\lambda x.u)v \to_\beta u[x := v].$$

Azt mondjuk, hogy t redukálódik $u[x := v]$-re.

3.1. *Példa* (β-redukció)*.*

$$(\lambda x.x)t \quad \to_\beta \quad t$$

A redukciót a β-redukció reflexív, tranzitív lezártjával definiáljuk és $\to_\beta^*$-gal jelöljük. Szokás még definiálni az η-redukciót/expanziót, azonban hogy ezt helyesen megadhassuk, szükségünk van a típus fogalmára.

A program során létrehozott változókat egy környezetben tároljuk, illetve azokat ebből a környezetből kérdezhetjük le. A $\lambda x.t$ absztrakció kiegészíti t környezetét az x változóval. Így egy kifejezésben a szabad változók értékét a környezet segítségével tudjuk elérni. A környezeteket általában görög nagybetűkkel jelöljük: Γ, Δ, Θ, stb.

Típusrendszer

A típusrendszer bevezetésével szűkítjük az elfogadott kifejezések halmazát. Azokat a kifejezéseket, amelyek nem felelnek meg a típusszabályoknak, szokás pretermnek nevezni.

Az egyszerűen típusos λ-kalkulusban kétféle típust különböztetünk meg.

3.4. Definíció (Típus)**.**

$$\begin{aligned} \langle \textit{típus} \rangle \equiv\ & \langle \textit{alaptípus} \rangle \\ \mid\ & \langle \textit{típus} \rangle \Rightarrow \langle \textit{típus} \rangle \end{aligned}$$

Egy alaptípust definiálunk, ez legyen ι. A típusokat A, B, C, stb. betűkkel jelöljük. A kifejezések lehetnek az alaptípus elemei, vagy egy

függvény típus elemei. Ezt úgy írjuk, hogy $\iota : A$. A környezet a változók mellett azok típusát is tartalmazza.

Csupán a fenti definícióval a típus önmagában nem integrálódik a nyelvbe. Hogy értelmet adjunk nekik, össze kell kötnünk a már meglévő kifejezésekkel, melyhez számos új szabályt kell bevezetnünk.

3.5. Definíció (Következtetés)**.** A $\Gamma \vdash t : A$ következtetéssel azt mondjuk, hogy Γ-ban t kifejezés A típusú.

A következtetés segítségével tudunk szabályokat alkotni a kifejezések típusát illetően. A λ-kalkulus kifejezéseit pretermeknek nevezzük, míg a termek olyan t kifejezések, hogy valamely Γ környezetre és A típusra $\Gamma \vdash t : A$ levezethető. Más szóval csak a típushelyes kifejezéseket tekintjük termeknek. A λ-kalkulus típusszabályait a 4. ábra tartalmazza.

$$\frac{}{\Gamma, x : A, \Gamma' \vdash x : A} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \Rightarrow B} \qquad \frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

4. ábra. Kifejezésekre vonatkozó típusszabályok

A típusok segítségével most már le tudjuk írni a korábban említett η-redukciót:

3.6. Definíció (η-redukció)**.** Amennyiben $\Gamma \vdash t : A \Rightarrow B$, valamint x nem szabad t-ben, akkor a

$$\begin{aligned} \lambda x.tx &\quad \to_{\eta} \quad t \\ t &\quad \to_{\eta^{-1}} \quad \lambda x.tx \end{aligned}$$

műveleteket nevezzük rendre η-redukciónak és expanziónak.

Bizonyítható, hogy a β- és η-redukciók megtartják a kifejezések típusát (5. ábra).

Végezetül pedig meg kell adnunk, hogy a redukciós műveleteinkre milyen szabályok érvényesek, azokat hogy tudjunk használni. Ezekhez további szabályokkal kell a nyelvet kiegészíteni, lásd 6. ábra.

Komoly problémát jelenthet a szabályhalmaz növekedése, ami minden egyes új kifejezés vagy művelet bevezetésével bekövetkezik.

$$\frac{\Gamma \vdash t : A \quad t \to^*_\beta t'}{\Gamma \vdash t' : A} \qquad \frac{\Gamma \vdash t : A \Rightarrow B}{\Gamma \vdash \lambda x.tx : A \Rightarrow B} \qquad \frac{\Gamma \vdash \lambda x.tx : A \Rightarrow B}{\Gamma \vdash t : A \Rightarrow B}$$

5. ábra. Az β- és η-redukciók típustartóak

$$\frac{t \to^*_\beta t'}{\lambda x.t \to^*_\beta \lambda x.t'} \qquad \frac{t \to^*_\beta t'}{tu \to^*_\beta t'u} \qquad \frac{u \to^*_\beta u'}{tu \to^*_\beta tu'}$$

6. ábra. Átalakítási szabályok

A változónevek problémája

Előfordulhat, hogy a λ-*kifejezés*ben a változónevek nem egyértelműek:

3.2. *Példa.* Vegyük a $x\lambda\underline{x}.\lambda\underline{\underline{x}}.\underline{\underline{x}}$ kifejezést. Ebben a példában három különböző kifejezést is jelölünk x-el. Az első x egy szabad változó (aláhúzatlan), a második ($\underline{x}$) csak az absztrakció változója, de nem köt semmit, míg a harmadik előfordulás ($\underline{\underline{x}}$) a belső absztrakcióban szerepel.

Hogy egyértelművé tegyük a változóinkat, több megoldás is rendelkezésünkre áll, ezek lehet például a változók átnevezése (α-konverzió), vagy a de Bruijn-indexek használata (a változók kötésétől számított távolság alapján történő indexelés).

4. Kategorikus leírás

A függő típuselméletet algebrai struktúraként adta meg Castellan, Clairambault és Dybjer az egyszerűen típusozott családos kategória (simply typed category with families) fogalmával [1, 4]. Ez egy magasabb szintű elmélet, mint amivel jelen dolgozatban foglalkozunk, számunkra ennek egy speciális, egyszerűsített változata is elegendő lesz, amivel megadhatjuk az egyszerű típuselmélet kategorikus definícióját.

A családos kategóriák definíciójában [4, 6] a típusok függtek a környezettől, így ezzel a függő típuselmélethez juthatunk el. Egyszerű eset-

ben elhagyjuk ezt a függőséget, a típusok a környezetektől függetlenek, így kapjuk az egyszerű típuselméletet (lásd 7. és 8. ábra). Megjegyezzük, hogy most nem célunk egy minimális méretű kategorikus modell megadása. A megadott leírás az egyszerű típuselmélet egy általánosan használt definíciója [5].

$$
\begin{array}{ll}
\mathsf{Ty} & : \mathsf{Set} \\
\iota & : \mathsf{Ty} \\
- \Rightarrow - & : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty}
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{Con} & : \mathsf{Set} \\
\cdot & : \mathsf{Con} \\
- \triangleright - & : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Con}
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{Tm}-- & : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Set} \\
\mathsf{q} & : \mathsf{Tm}\,(\,\Gamma \triangleright \mathsf{A}\,)\,\mathsf{A} \\
-[-] & : \mathsf{Tm}\,\Gamma\,\mathsf{A} \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta\,\mathsf{A} \\
\mathsf{lam}- & : \mathsf{Tm}(\,\Gamma \triangleright \mathsf{A})\,\mathsf{B} \to \mathsf{Tm}\,\Gamma(\,\mathsf{A} \Rightarrow \mathsf{B}) \\
\mathsf{app}- & : \mathsf{Tm}\,\Gamma(\,\mathsf{A} \Rightarrow \mathsf{B}) \to \mathsf{Tm}(\,\Gamma \triangleright \mathsf{A})\,\mathsf{B}
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{Sub}-- & : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \\
\mathsf{id} & : \mathsf{Sub}\,\Gamma\,\Gamma \\
- \circ - & : \mathsf{Sub}\,\Theta\,\Gamma \to \mathsf{Sub}\,\Delta\,\Theta \to \mathsf{Sub}\,\Delta\,\Gamma \\
\varepsilon & : \mathsf{Sub}\,\Gamma\,\cdot \\
-, - & : \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Tm}\,\Gamma\,\mathsf{A} \to \mathsf{Sub}\,\Gamma(\,\Delta \triangleright \mathsf{A}) \\
\mathsf{p} & : \mathsf{Sub}\,(\,\Gamma \triangleright \mathsf{A}\,)\,\Gamma
\end{array}
$$

7. ábra. Az egyszerű típuselmélet kategorikus reprezentációja (egyenlőségek nélkül)

Négy-szortú algebrai struktúrát használunk a definícióban, ahol az alaphalmazok a $\mathsf{Con}, \mathsf{Ty}, \mathsf{Tm}, \mathsf{Sub}$. Mivel egyszerű és nem függő eset-

ről tárgyalunk, $\mathsf{Ty}\,\Gamma$ helyett egyszerűen Ty-t írunk, továbbá emiatt elhagyhatjuk a típusokon vett helyettesítést is. Az alaptípust ι jelöli. A típusokat ezen felül kiegészítjük a függvénytérrel, bevezetve a „nyíl" típust, mely a típusból típusba képező típus. Ennek megfelelően megadjuk a függvényeken értelmezett operátorokat, a lam (a λ-kalkulus λ-absztrakciójából), illetve az app, mint applikáció műveleteket.

A modellben megadott környezet a családos kategória definíciójának megfelelően van megadva, $\cdot$ jelöli az üres, $\triangleright$ a kiegészített környezetet.

A $\mathsf{q}, \mathsf{p}, -[-]$ és $-, -$ operátorok a családos kategória definíciójában megadott kifejezéseknek felel meg, míg id (identitás) és $- \circ -$ (kompozíció) a kategória definíciójából származnak. Végezetül pedig ϵ a kategória terminális objektumába képező morfizmust reprezentálja.

$$
\begin{aligned}
\mathsf{ass} &: (\sigma \circ \delta) \circ \nu \equiv \sigma \circ (\delta \circ \nu) \\
\mathsf{idl} &: \mathsf{id} \circ \sigma \equiv \sigma \\
\mathsf{idr} &: \sigma \circ \mathsf{id} \equiv \sigma \\
\cdot\eta &: \sigma : \mathsf{Sub}\,\Gamma\cdot \to \sigma \equiv \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\triangleright\beta_1 &: \mathsf{p} \circ (\sigma,\, \mathsf{t}) \equiv \sigma \\
\triangleright\beta_2 &: \mathsf{q}[\sigma,\, \mathsf{t}] = \mathsf{t} \\
\triangleright\eta &: (\mathsf{p}, \mathsf{q}) \equiv \mathsf{id} \\
,\circ &: (\sigma,\, \mathsf{t}) \circ \delta \equiv (\sigma \circ \delta,\, \mathsf{t}[\delta]) \\
[\mathsf{id}] &: \mathsf{t}[\mathsf{id}] \equiv \mathsf{t} \\
[\circ] &: \mathsf{t}[\sigma \circ \delta] \equiv \mathsf{t}[\sigma][\delta]
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow\beta &: \mathsf{app}(\mathsf{lam}\,\mathsf{t}) \equiv \mathsf{t} \\
\Rightarrow\eta &: \mathsf{lam}(\mathsf{app}\,\mathsf{t}) \equiv \mathsf{t} \\
\mathsf{lam}[] &: (\mathsf{lam}\,\mathsf{t})[\sigma] \equiv \mathsf{lam}(\,\mathsf{t}[\sigma \circ \mathsf{p}, \mathsf{q}])
\end{aligned}
$$

8. ábra. A kategorikus leírás egyenlőségei

A struktúra egyenlőségei (8. ábra) közül $\mathsf{ass}, \mathsf{idl}, \mathsf{idr}$ a kategória egyenlőségeiből adódik (asszociativitás és identitás), míg $\cdot\eta$ a terminá-

lis objektumot jelöli ki. A családos kategória definíciójából származik $\triangleright\beta_1$ és $\triangleright\beta_2$, míg $\triangleright\eta, , \circ, [\mathsf{id}], [\circ]$ az identitásra és a helyettesítésre vonatkozó tulajdonságokat adja meg. Végezetül pedig $\Rightarrow \beta, \Rightarrow \eta$ és $\mathsf{lam}[]$ a függvénytér tulajdonságai.

Összehasonlítás a λ-kalkulussal

A λ-kalkulus következtetéseit $\Gamma \vdash t : A$ alakban írtuk. Ezzel különböztettük meg a pretermeket a termektől, amely preterm típusozható volt, arra volt igaz a fenti következtetés. A kategorikus modellben nincs szükség ilyen szabályokra, hisz minden $\mathsf{t} : \mathsf{Tm}\,\Gamma\,\mathsf{A}$ term definíció szerint jól típusozott, Γ környezetben A típussal. A modell kifejezőerejéről árulkodik az is, hogy ugyanezen tulajdonsága miatt nincs szükség a 4. ábra szabályainak bizonyítására sem.

Míg a λ-kalkulusban a helyettesítés egy szintaktai átalakítás volt, a kategorikus leírásban a modell részét képezi, explicit adhatjuk meg őket.

A modellben nem használunk változóneveket, a korábban említett de Bruijn-indexet használjuk. A változókat itt is a környezetben elfoglalt pozíciójuk azonosítja. A $\mathsf{q} : \mathsf{Tm}\,(\,\Gamma \triangleright \mathsf{A}\,)\,\mathsf{A}$ term segítségével tudjuk a környezet utolsó elemét lekérdezni. Kezdeti esetben $(\,\Gamma \equiv \cdot)$ a $\mathsf{q} : \mathsf{Tm}\,(\cdot \triangleright \mathsf{A}\,)\,\mathsf{A}$ a környezet utolsó és egyben egyetlen elemét adja vissza. Kétváltozós környezet esetén $\mathsf{q} : \mathsf{Tm}\,(\cdot \triangleright \mathsf{B} \triangleright \mathsf{A}\,)\,\mathsf{A}$ ismét az utolsó elem. Ahhoz hogy megkapjuk a B-t, használnunk kell a $\mathsf{p} : \mathsf{Sub}(\,\Gamma \triangleright \mathsf{A})\,\Gamma$ helyettesítést, ahol most $\Gamma \equiv \cdot \triangleright \mathsf{B}$, így $\mathsf{p} : \mathsf{Sub}(\cdot \triangleright \mathsf{B} \triangleright \mathsf{A})(\cdot \triangleright \mathsf{B})$. Ezt behelyettesítve az első $\mathsf{q} : \mathsf{Tm}\,(\cdot \triangleright \mathsf{B}\,)\,\mathsf{B}$ termbe, $\mathsf{q}[\mathsf{p}] : \mathsf{Tm}(\cdot \triangleright \mathsf{B} \triangleright \mathsf{A})\,\mathsf{B}$ megadja a második elemét a környezetnek. Analóg módon $\mathsf{q}[\mathsf{p}][\mathsf{p}]$ felel meg a harmadik, $\mathsf{q}[\mathsf{p}][\mathsf{p}][\mathsf{p}]$ a negyedik, stb. változónak. Természetesen a behelyettesített p-k mind más környezetbe képeznek.

5. Teleszkopikus leírás

A dolgozatban [6] kidolgoztuk a típuselmélet egy alternatív reprezentációját, melyet teleszkopikusnak nevezünk [3]. A leírás célja a gyakorlatban gyakran használt kifejezések és a formális elmélet közelebb hozása az intuitívabb leírás és a könnyebb bizonyítások reményében.

A 9. ábra tartalmazza a teleszkopikus szintaxis operátorait. Azonnal szembeötlő, hogy a kategorikussal ellentétben itt nincs külön konstruk-

ciója a szubsztitúcióknak. Láthatjuk, hogy a típusok változatlanok, a környezetek pedig csupán egy új operátorral egészülnek ki, a jelentősebb változást a termek szintjén láthatjuk. Az alábbiakban ezeket tárgyaljuk részletesen.

5.1. Környezetek konkatenációja

A különbséget a korábban bevezetett kategorikus leírással az jelenti, hogy míg korábban a helyettesítést az egész környezeten értelmezett párhuzamos leképezés formájában adtuk meg, a teleszkopikus szintaxis egyszeri helyettesítést biztosít.

Hogy ezt bevezethessük, szükségünk van egy új operátorra a környezetre nézve, hogy annak ne csak a végéről, hanem annak bármely pontjáról tudjunk tárgyalni:

$$- \mathbin{+\!\!+} - : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Con}$$

A $+\!\!+$ operátorral környezetek konkatenációját tudjuk megvalósítani, így az alábbi módon egy környezet tetszőleges elemét elérjük: $\Gamma \triangleright \mathsf{A} + + \Delta$, anélkül hogy a Δ-t is rekurzívan kifejtenénk.

Vegyük észre, hogy ha a környezetet induktívan adtuk volna meg, $\Gamma + + \Delta$ definiálható lenne a korábbi konstruktorainkkal Δ szerinti indukcióval:

$$\begin{aligned} \Delta = \cdot \quad &: \quad \Gamma \mathbin{+\!\!+} \cdot \quad &:= \Gamma \\ \Delta = \Delta' \triangleright \mathsf{A} \, &: \quad \Gamma \mathbin{+\!\!+} (\Delta' \triangleright \mathsf{A}) &:= (\Gamma \mathbin{+\!\!+} \Delta') \triangleright \mathsf{A} \end{aligned}$$

Könnyen belátható az is, hogy a $(\mathsf{Con}, +\!\!+)$ struktúra félcsoportot alkot a $\cdot$ egységelemmel.

De Bruijn a fenti konstrukcióra a teleszkopikus nevet az alábbi asszociációból adta [3]. Függő esetben a $- \mathbin{+\!\!+} -$ operátor második környezete függhet az előzőtől. Többször egymásba ágyazva egy egyre inkább megszorított környezetet kapunk, ami emlékeztethet minket a hagyományos teleszkópokra, amelyben a szegmensek egyre szűkölő szélességgel vannak jelen és átmozgathatóak egymásba.

5.2. Gyengítés és egyszeri helyettesítés

A fenti konstrukció segítségével most már definiálhatjuk a teleszkopikus reprezentáció két új termkonstruáló műveletét, melyek segítségé-

$$
\begin{array}{ll}
\mathsf{Ty} & : \mathsf{Set} \\
\iota & : \mathsf{Ty} \\
- \Rightarrow - & : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty}
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{Con} & : \mathsf{Set} \\
\cdot & : \mathsf{Con} \\
- \triangleright - & : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Con} \\
- \mathbin{+\!\!+} - & : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Con}
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{Tm}-- & : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Set} \\
\mathsf{q} & : \mathsf{Tm}\,(\Gamma \triangleright \mathsf{A})\,\mathsf{A} \\
-[\Gamma, -, \Delta] & : \mathsf{Tm}(\Gamma \triangleright \mathsf{B} \mathbin{+\!\!+} \Delta)\,\mathsf{A} \to \mathsf{Tm}\,\Gamma\,\mathsf{B} \to \mathsf{Tm}(\Gamma \mathbin{+\!\!+} \Delta)\,\mathsf{A} \\
-\langle \Gamma, \mathsf{B}, \Delta \rangle & : \mathsf{Tm}(\Gamma \mathbin{+\!\!+} \Delta)\,\mathsf{A} \to \mathsf{Tm}(\Gamma \triangleright \mathsf{B} \mathbin{+\!\!+} \Delta)\,\mathsf{A} \\
\mathsf{lam}- & : \mathsf{Tm}(\Gamma \triangleright \mathsf{A})\,\mathsf{B} \to \mathsf{Tm}\,\Gamma(\mathsf{A} \Rightarrow \mathsf{B}) \\
\mathsf{app}- & : \mathsf{Tm}\,\Gamma(\mathsf{A} \Rightarrow \mathsf{B}) \to \mathsf{Tm}(\Gamma \triangleright \mathsf{A})\,\mathsf{B}
\end{array}
$$

9. ábra. Az egyszerű típuselmélet teleszkopikus reprezentációja (egyenlőségek nélkül)

vel a Sub konstrukciót teljes egészében elhagyhatjuk:

$$
\begin{array}{ll}
-[\Gamma, -, \Delta] & : \mathsf{Tm}(\Gamma \triangleright \mathsf{B} \mathbin{+\!\!+} \Delta)\,\mathsf{A} \to \mathsf{Tm}\,\Gamma\,\mathsf{B} \to \mathsf{Tm}(\Gamma \mathbin{+\!\!+} \Delta)\,\mathsf{A} \\
-\langle \Gamma, \mathsf{B}, \Delta \rangle & : \mathsf{Tm}(\Gamma \mathbin{+\!\!+} \Delta)\,\mathsf{A} \to \mathsf{Tm}(\Gamma \triangleright \mathsf{B} \mathbin{+\!\!+} \Delta)\,\mathsf{A}
\end{array}
$$

Az első operátor, $\mathsf{t}[\Gamma, \mathsf{u}, \Delta]$ reprezentálja a helyettesítést egy pontban. Szemléletesen, ezzel tudjuk a $\mathsf{t} : \mathsf{Tm}(\Gamma \triangleright \mathsf{B} \mathbin{+\!\!+} \Delta)$ termben a Γ és Δ környezetek által közrezárt pontba behelyettesíteni az $\mathsf{u} : \mathsf{Tm}\,\Gamma\,\mathsf{B}$ termet. Fontos, hogy u csak Γ-tól függhet, szemléletesen így ahova behelyettesítjük, már minden változó pontosan ugyanott szerepel t-ben, mint u-ban.

A második operátor, $\mathsf{t}\langle \Gamma, \mathsf{B}, \Delta \rangle$ a gyengítés művelete, ezzel tudjuk a környezetet egy Γ és Δ által közrezárt pontban kiegészíteni.

$$
\begin{array}{ll}
\mathsf{q}[\cdot] & : \mathsf{q}[\,\Gamma, \mathsf{u}, \cdot] \equiv \mathsf{u} \\
\mathsf{q}[\mathsf{mid}] & : \mathsf{q}[\,\Gamma, u,\, \Delta \triangleright \mathsf{A}] \equiv \mathsf{q} \\
\mathsf{q}\langle\mathsf{mid}\rangle & : \mathsf{q}\langle\,\Gamma,\, \mathsf{B},\, \Delta \triangleright \mathsf{A}\rangle \equiv \mathsf{q} \\
\langle\rangle\langle\rangle & : \mathsf{t}\langle\,\Gamma,\, \mathsf{A},\, \Delta +\!\!+ \Theta\rangle\langle\,\Gamma \triangleright \mathsf{A} +\!\!+ \Delta,\, \mathsf{B},\, \Theta\rangle \equiv \\
 & \quad \mathsf{t}\langle\,\Gamma +\!\!+ \Delta,\, \mathsf{B},\, \Theta\rangle\langle\,\Gamma,\, \mathsf{A},\, \Delta \triangleright \mathsf{B} +\!\!+ \Theta\rangle \\
\langle\rangle[]^r & : \mathsf{t}\langle\,\Gamma,\, \mathsf{A},\, \Delta \triangleright \mathsf{B} +\!\!+ \Theta\rangle[\,\Gamma \triangleright \mathsf{A} +\!\!+ \Delta, \mathsf{u}\langle\,\Gamma,\, \mathsf{A},\, \Delta\rangle,\, \Theta] \equiv \\
 & \quad \mathsf{t}[\,\Gamma +\!\!+ \Delta, \mathsf{u},\, \Theta]\langle\,\Gamma,\, \mathsf{A},\, \Delta +\!\!+ \Theta\rangle \\
\langle\rangle[]^l & : \mathsf{t}\langle\,\Gamma \triangleright \mathsf{A} +\!\!+ \Delta,\, \mathsf{B},\, \Theta\rangle[\,\Gamma, \mathsf{u},\, \Delta \triangleright \mathsf{B} +\!\!+ \Theta] \equiv \\
 & \quad \mathsf{t}[\,\Gamma, \mathsf{u},\, \Delta +\!\!+ \Theta]\langle\,\Gamma +\!\!+ \Delta,\, \mathsf{B},\, \Theta\rangle \\
[][] & : \mathsf{t}[\,\Gamma, \mathsf{u},\, \Delta \triangleright \mathsf{B} +\!\!+ \Theta][\,\Gamma +\!\!+ \Delta,\, \mathsf{v}[\,\Gamma, \mathsf{u},\, \Delta],\, \Theta] \equiv \\
 & \quad \mathsf{t}[\,\Gamma \triangleright \mathsf{A} +\!\!+ \Delta,\, \mathsf{v},\, \Theta][\,\Gamma, \mathsf{u},\, \Delta +\!\!+ \Theta] \\
\langle] & : \mathsf{t}\langle\,\Gamma,\, \mathsf{A},\, \Delta\rangle[\,\Gamma, \mathsf{u},\, \Delta] \equiv \mathsf{t} \\
\langle,] & : \mathsf{t}\langle\,\Gamma,\, \mathsf{A}, \cdot \triangleright \mathsf{A} +\!\!+ \Delta\rangle[\,\Gamma \triangleright \mathsf{A}, \mathsf{q},\, \Delta] \equiv \mathsf{t} \\
\mathsf{lam}\langle\rangle & : \mathsf{lam}\,\mathsf{t}\langle\,\Gamma, \mathsf{C},\, \Delta\rangle \equiv \mathsf{lam}(\,\mathsf{t}\langle\,\Gamma, \mathsf{C},\, \Delta \triangleright \mathsf{A}\rangle) \\
\mathsf{lam}[] & : \mathsf{lam}\,\mathsf{t}[\,\Gamma, \mathsf{u},\, \Delta] \equiv \mathsf{lam}(\,\mathsf{t}[\,\Gamma, \mathsf{u},\, \Delta \triangleright \mathsf{A}]) \\
\mathsf{app}\langle\rangle & : \mathsf{app}\,\mathsf{t}\langle\,\Gamma, \mathsf{C},\, \Delta \triangleright \mathsf{A}\rangle \equiv \mathsf{app}(\,\mathsf{t}\langle\,\Gamma, \mathsf{C},\, \Delta\rangle) \\
\mathsf{app}[] & : \mathsf{app}\,\mathsf{t}[\,\Gamma, \mathsf{u},\, \Delta \triangleright \mathsf{A}] \equiv \mathsf{app}(\,\mathsf{t}[\,\Gamma, \mathsf{u},\, \Delta])
\end{array}
$$

10. ábra. A teleszkopikus reprezentáció egyenlőségei

5.3. Egyenlőségek

A modell egyenlőségeit a 10. ábra tartalmazza.

Az első három axióma a q-ra vonatkozó helyettesítés és gyengítés tulajdonságait írja le. $\mathsf{q}[\cdot]$ esetén felhasználva hogy q típusa $\mathsf{Tm}(\,\Gamma \triangleright \mathsf{A})\,\mathsf{A}$, behelyettesítünk egy $\mathsf{Tm}\,\Gamma A$ típusú termet, u-t a környezet végére. A behelyettesített term típusa pont $\mathsf{Tm}\,\Gamma A$ (lásd helyettesítés definíciója), így az egyenlőség típushelyes lesz.

5.1. *Példa.* $\mathsf{q}[\cdot]$-hoz hasonló egyenlőséget nem tudunk megadni a gyengítésre:
$\mathsf{q}\langle\,\Gamma \triangleright \mathsf{A},\, \mathsf{A}, \cdot\rangle \not\equiv \mathsf{q} : \mathsf{Tm}(\,\Gamma \triangleright \mathsf{A} \triangleright \mathsf{A})\,\mathsf{A}$, mert például ahogy azt a korábbi fejezetben láthattuk, változónevekkel szemléltetve $x \neq y$ esetén egy

$\mathsf{Tm}(\,\Gamma, x : \mathbb{N}, y : \mathbb{N})\mathbb{N}$ termnél nem mindegy, hogy melyik változó határozza meg a kifejezés értékét: $\lambda xy.x \neq \lambda xy.y$. Hasonlóan a modellnél maradva, $\mathsf{lam}(\mathsf{lam}(\mathsf{q}\langle\,\Gamma \triangleright \mathsf{A},\, \mathsf{A}, \cdot\rangle)) \not\equiv \mathsf{lam}(\mathsf{lam}(\mathsf{q}))$.

A másik két q-ra vonatkozó axióma a környezet közepére illeszt, illetve helyettesít be egy elemet, így azok nem befolyásolják a környezet utolsó elemét, ami meghatározza a kifejezés értékét. Ezen felül természetesen az egyenlőségek két oldalán szereplő q termek típusa más: $\mathsf{q}[\mathsf{mid}]$ esetében a bal $\mathsf{q} : \mathsf{Tm}(\,\Gamma \triangleright \mathsf{B} \mathbin{+\!\!+} \Delta \triangleright \mathsf{A})\,\mathsf{A}$, míg a jobb $\mathsf{q} : \mathsf{Tm}(\,\Gamma \mathbin{+\!\!+} \Delta \triangleright \mathsf{A})\,\mathsf{A}$, illetve $\mathsf{q}\langle\mathsf{mid}\rangle$ esetén a bal $\mathsf{q} : \mathsf{Tm}(\,\Gamma \mathbin{+\!\!+} \Delta \triangleright \mathsf{B})\,\mathsf{B}$, míg a jobb $\mathsf{q} : \mathsf{Tm}(\,\Gamma \triangleright \mathsf{A} \mathbin{+\!\!+} \Delta \triangleright \mathsf{B})\,\mathsf{B}$.

A $\langle\rangle\langle\rangle, \langle\rangle[]^r, \langle\rangle[]^l, [][]$ egyenlőségek a helyettesítések és gyengítések felcserélhetőségét definiálják. A $\langle\rangle[]^l$ axiómánál az l azt jelöli, hogy a helyettesítés a gyengítéstől balra, míg $\langle\rangle[]^r$ esetén jobbra történik. Továbbá $\langle\rangle[]^r$-nél mivel a behelyettesítés pontjától balra gyengítünk, ezért itt a behelyettesítendő termet is gyengíteni kell. Hasonló gondolatmenettel $[][]$-ot tekintve is helyettesítenünk kell a behelyettesítendő kifejezésben is.

A $\langle]$ egyelőség a gyengített helyre történő helyettesítést írja le. Vegyük észre, hogy a fordítottjának, $\mathsf{t}[\,\Gamma, \mathsf{u},\, \Delta]\langle\,\Gamma,\, \mathsf{A},\, \Delta\rangle$ ez nem lesz egyenlő t-vel, hisz egyrészt u nem feltétlen A típusú, de ha ez fennállna, akkor se lenne igaz az állítás.

5.2. *Példa.* Vizsgáljuk $\mathsf{t}[\,\Gamma, \mathsf{u},\, \Delta]\langle\,\Gamma,\, \mathsf{A},\, \Delta\rangle$ kifejezést. Ha $\Delta \equiv \cdot$, $\mathsf{t} \equiv \lambda x.x$, és $\mathsf{u} \equiv 1$, akkor $\mathsf{t}[\,\Gamma, \mathsf{u},\, \Delta]$-nak megfelel az 1 λ-*kifejezés*, amit ha gyengítünk: $1\langle\,\Gamma,\, \mathsf{A},\, \Delta\rangle$ az eredetitől különböző kifejezést kapunk: $\lambda x.1$

A $\langle,]$ egyenlőség a fenti helyettesítés egy speciális esete, amikor nem pont abba az elembe helyettesítünk, hanem a mellette álló, azonos típusú elembe. A helyes működés q tulajdonságából következik, miszerint az a környezet utolsó elemét adja vissza.

A további egyenlőségek a helyettesítés és gyengítés kapcsolatát írja le a függvénytérrel. Megjegyezzük, hogy $\mathsf{app}\langle\rangle$ és $\mathsf{app}[]$ feltehetőleg kifejezhető $\mathsf{lam}\langle\rangle$, illetve $\mathsf{lam}[]$ segítségével is.

5.4. Teleszkopikus modellek

Hasonlóan a kategorikus leíráshoz, a teleszkopikus leírásban is definiáljuk a modellek közti TelMor morfizmust: 11. ábra.

Továbbá a korábban bemutatott módon megadhatjuk a teleszkopikus kontextuális modell, illetve a modellmorfizmust a $\mathsf{TelModel}$ modell-

ből, illetve a TelMor morfizmusból a Con, ·, ▷, ++ mezők elhagyásával.

A teleszkopikus kanonicitásmodell megadása nem ígérkezik egyszerű feladatnak, a Sub helyettesítések hiánya miatt, melyek alapvető részét képezik a kategorikus kanonicitásmodellnek. Mi ahelyett, hogy ezt definiálnánk, megmutatjuk a két leírás közötti kapcsolatot, aminek következményeképp elegendő a kategorikus kanonicitásmodell használata.

$$
\begin{aligned}
\mathsf{TelMor} : \quad & \mathsf{TelModel} \to \mathsf{TelModel} \to \mathsf{Set} \\
\mathsf{TelMor}\mathcal{MN} := (\mathsf{Ty} \; & : \mathsf{Ty}_{\mathcal{M}} \to \mathsf{Ty}_{\mathcal{N}}) \\
\times \; (\mathsf{Con} & : \mathsf{Con}_{\mathcal{M}} \to \mathsf{Con}_{\mathcal{N}}) \\
\times \; (\mathsf{Tm} \; & : \mathsf{Tm}_{\mathcal{M}}\,\Gamma\,\mathsf{A} \to \mathsf{Tm}_{\mathcal{N}}(\mathsf{Con}\,\Gamma)(\mathsf{Ty}\,\mathsf{A})) \\
\times \; (\iota \quad & : \mathsf{Ty}\iota_{\mathcal{M}} \equiv \iota_{\mathcal{N}}) \\
\times \; (\Rightarrow \; & : \mathsf{Ty}(\mathsf{A} \Rightarrow_{\mathcal{M}} \mathsf{B}) \equiv \mathsf{Ty}\,\mathsf{A} \Rightarrow_{\mathcal{N}} \mathsf{B}) \\
\times \; (\cdot \quad & : \mathsf{Con}\cdot_{\mathcal{M}} \equiv \cdot_{\mathcal{N}}) \\
\times \; (\triangleright \; & : \mathsf{Con}(\Gamma \triangleright_{\mathcal{M}} \mathsf{A}) \equiv \mathsf{Con}\,\Gamma \triangleright_{\mathcal{N}} \mathsf{Ty}\,\mathsf{A}) \\
\times \; (++ \; & : \mathsf{Con}(\Gamma +\!+_{\mathcal{M}} \Delta) \equiv \mathsf{Con}\,\Gamma +\!+_{\mathcal{N}} \mathsf{Con}\,\Delta) \\
\times \; (\mathsf{q} \quad & : \mathsf{Tm}\,\mathsf{q}_{\mathcal{M}} \equiv \mathsf{q}_{\mathcal{N}}) \\
\times \; (\langle\rangle \; & : \mathsf{Tm}\,(\mathsf{t}\langle\Gamma, \mathsf{A}, \Delta\rangle_{\mathcal{M}}) \equiv \\
& \qquad \mathsf{Tm}\;\mathsf{t}\langle\mathsf{Con}\;\Gamma, \mathsf{Ty}\;\mathsf{A}, \mathsf{Con}\;\Delta\rangle_{\mathcal{N}}) \\
\times \; ([] \quad & : \mathsf{Tm}(\mathsf{t}[\Gamma, \mathsf{u}, \Delta]_{\mathcal{M}}) \equiv \mathsf{Tm}\;\mathsf{t}[\mathsf{Con}\;\Gamma, \mathsf{Tm}\,\mathsf{u}, \mathsf{Con}\;\Delta]_{\mathcal{N}}) \\
\times \; (\mathsf{lam} \; & : \mathsf{Tm}(\mathsf{lam}_{\mathcal{M}}\,\mathsf{t} \equiv \mathsf{lam}_{\mathcal{N}}(\mathsf{Tm}\,\mathsf{t})) \\
\times \; (\mathsf{app} \; & : \mathsf{Tm}(\mathsf{app}_{\mathcal{M}}\,\mathsf{t}) \equiv \mathsf{app}_{\mathcal{N}}(\mathsf{Tm}\,\mathsf{t}))
\end{aligned}
$$

11. ábra. A teleszkopikus modellek közötti morfizmus

5.5. Teleszkopikus modellek ekvivalenciája

A dolgozatban [6] bebizonyítjuk, hogy a két tárgyalt leírás megfeleltethető egymásnak, így belátjuk, hogy a megadott teleszkopikus leírás helyes és teljes. A dolgozatban definiáltuk az $f : \mathsf{CatConModel} \to \mathsf{TelConModel}$, illetve $g : \mathsf{TelConModel} \to \mathsf{CatConModel}$ leképezéseket. Ezeket úgy adtuk meg, hogy a kategorikus, illetve a teleszkopikus kon-

textuális modell operátoraihoz hozzárendeltük a másik modell operátorait, illetve bebizonyítottuk a modellek egyenlőségeit. Ezzel beláttuk, hogy f és g a másik modellbe képező homomorfizmusok. Kimondtuk továbbá a teleszkopikus kontextuális modell helyességét és teljességét.

Ezt követően beláttuk, hogy az $f \circ g$, illetve a $g \circ f$ leképezések megtartják a kifejezések jelentését. Végezetül pedig megadtuk a Sub-okon értelmezett izomorfizmust. Ezek segítségével kimondhatjuk az alábbi tételt:

5.1. Tétel (Ekvivalencia)**.** *Tetszőleges kategorikus kontextuális, illetve teleszkopikus kontextuális modellek ekvivalensek.*

A tétel következménye, hogy a kategorikus és teleszkopikus szintaxisok is ekvivalensek, hisz a szintaxis is egy kontextuális modell.

6. Konklúzió

Kutatásunk során megadtuk a típuselmélet egy programozásközeli leírását, a teleszkopikus leírást. A modellben megadott helyettesítés megfelel a λ-kalkulus egyszeri helyettesítésének, használatával elkerülhetőek a kategorikus leírás párhuzamos helyettesítése, nem szükséges teljes környezetek közötti leképezést megadni. Ugyanakkor az egyszerű típuselméletet algebrai struktúraként definiáltuk, így a λ-kalkulusnál magasabb módszert használtunk, elkerülve a λ-kalkulus bemutatott hátrányait.

Fő kontribúciónkként bebizonyítottuk, hogy a két leírás kontextuális modelljei ekvivalensek. Ezt a két modell közti, mindkét irányba vezető leképezések segítségével adtuk meg. A bizonyítás során jól tudtuk általánosítani a teleszkopikus modell általánosított egyenlőségeit, azok használata intuitívnak bizonyult. További kutatási irány a bizonyítás megadása függő típuselméletre, nyitott kérdés, hogy ott is be lehet-e bizonyítani az ekvivalenciát. Másik irány a teleszkopikus leírás használata egy csak normálformákból álló szintaxis leírására.

A dolgozat során néhány példán keresztül mutattuk be a leírások használatát, azok előnyeit. Az ekvivalencia bizonyításával láttuk, hogy a kategorikus és teleszkopikus kontextuális modellek megfeleltethetőek egymásnak, így érdemes lehet más teleszkopikus modellek vizsgálata is.

Hivatkozások

[1] Simon Castellan, Pierre Clairambault, Peter Dybjer, Categories with families: Unityped, simply typed, and dependently typed, *CoRR,* abs/1904.00827, 2019.

[2] Zoltán Csörnyei, *Bevezetés a típusrendszerek elméletébe,* ELTE Eötvös Kiadó, 2012.

[3] N.G. de Bruijn, Telescopic mappings in typed lambda calculus, *Information and Computation,* 91(2) (1991), pp. 189–204.

[4] Peter Dybjer, Internal type theory, *Selected Papers from the International Workshop on Types for Proofs and Programs, TYPES '95,* Springer-Verlag, Berlin, Heidelberg, 1995, pp. 120–134.

[5] Ambrus Kaposi, Type theory in a type theory with quotient inductive types, 2017.03.

[6] Norbert Luksa, Az egyszerű típuselmélet algebrai reprezentációi, *ELTE IK Tudományos Diákköri Konferencia,* 2020.

[7] Per Martin-Löf, Constructive mathematics and computer programming, *Proc. Of a Discussion Meeting of the Royal Society of London on Mathematical Logic and Programming Languages,* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985, pp. 167–184.

CodeCompass

Kódmegértést támogató keretrendszer

Porkoláb Zoltán*

ELTE Bolyai Kollégium**

gsd@inf.elte.hu

A szoftverfejlesztés során, akár új funkcionalitást fejlesztünk, akár hibát javítunk, vagy éppen valamely meglévő működést módosítunk, elsődleges, hogy a tervezett változtatások minden következményét előre lássuk. Ez nem történhet meg anélkül, hogy teljesen megértenénk az adott programrendszert. Sajnos a hosszú élettartamú, nagy méretű szoftverrendszerek esetében gyakori, hogy a fejlesztőgárda kicserélődése miatt az évtizedek során az eredeti szándékok, döntési szempontok a múlt homályába vesznek. A dokumentáció gyakran hiányos, pontatlan – ha egyáltalán létezik. Mindez azt jelenti, hogy az egyedüli megbízható információ, amely a szoftverről a rendelkezésünkre áll, az maga a forráskód. Az ilyen, ipari méretű, gyakran sok millió forrássorból álló rendszerek megértése rendkívüli kihívást jelent még gyakorlott fejlesztők számára is. Természetes igény, hogy ezt a megértési folyamatot valamilyen módon eszközökkel támogassuk. Egy ilyen eszköz a CodeCompass, egy nyílt forráskódú, kódmegértést támogató, bővíthető keretrendszer, amely a statikus elemzés módszerét felhasználva segíti a fejlesztőket ipari méretű kódok hatékonyabb megértéséhez. A CodeCompass beépülő modulok segítségével több programozási nyelvet támogat, és számos szöveges és grafikus vizuális elemet használ a jobb megértés érdekében.

* ELTE Informatikai Kar

** A Bolyai Kollégium Informatika Szemináriumának vezető tanára 2007 óta

1. Bevezetés

A nagy, hosszú élettartamra tervezett szoftverrendszerek karbantartása közismerten problémás, és elviheti a szoftver teljes élettartamára számolt költségének igen nagy részét. Az ilyen, akár több évtizedeken keresztül is fejlesztett, használt és karbantartott rendszereken nemegyszer több száz fejlesztő dolgozik [3,6], akik időben cserélődnek. Minden, a projektet elhagyó fejlesztővel elvész a rendszert illető tudás egy része, és minden új belépő számára komoly idő- és munkabefektetés a szükséges ismeretek megszerzése. Márpedig sem eredményes hibajavítás, sem pedig megbízható új fejlesztés nem valósulhat meg anélkül, hogy a feladaton dolgozó informatikusok biztonsággal átlássák a rendszer adott részleteit, kapcsolatait más modulokkal.

Ezen okok miatt a mai átlagos szoftverfejlesztő általában sokkal több időt tölt meglévő rendszerek megértésével, mint új kód írásával. Az egyes kutatások eltérő számokat mutatnak, de abban általános az egyetértés, hogy a kódmegértés jellegű feladatok ma a fejlesztők idejének akár 80%-át is elérik. Különösen kritikus a helyzet, amennyiben egy szoftver fejlesztését egy teljesen új team veszi át, illetve megfigyelhető az a jelenség is, hogy a fiatalabb, kevésbé tapasztalt fejlesztőknek több időre van szükségük a rendszer megértéséhez.

Természetszerű, hogy egy ilyen léptékű feladat esetén felmerül a szoftveres támogatás. Az elmúlt évtizedekben a grafikus fejlesztő környezetek (ún. IDE-k) fokozatosan egészültek ki intelligens támogató elemekkel. Ilyen a kódkiegészítés, amely a kódkörnyezet alapján felajánlja a forráskód értelmes folytatását, ilyen az egyes kódrészletek becsukását-kinyitását lehetővé tevő funkcionalitás, ilyen a verziókezelő rendszerek integrációja, és a projekt inkrementális fordítása/szerkesztése is. Ugyanakkor a grafikus fejlesztő környezetek elsődleges használata az új forráskód írása, és az eszköztámogatásuk is elsősorban ezt a célt támogatja. Az új kód írása azonban lényegesen más jellegű, mint amikor egy meglévő forráskódot próbálunk megérteni.

Új forráskód írásakor tipikusan sok időt töltünk egy adott absztrakciós szinten. C és C++ programok esetében megtervezzük az interfészt, és ez alapján létrehozunk egy-két fejlécállományt. Ezek után, esetleg evvel párhuzamosan elkezdjük az implementációs részletek kidolgozását, amit egy forrásfájlban írunk meg. Teszt-alapú fejlesztés esetében ezt még kiegészítheti az aktuális egységtesztek írása, ami még egy forrás-

fájlt jelenthet. Tipikusan egyszerre nem több, mint 4–5 állományunk van megnyitva, köztünk váltogatunk, miközben szerkesztjük az állományokat. Ehhez a megközelítéshez tökéletesen megfelel a jelenlegi fejlesztő eszközök ablakosztása, ahol az egyes ablakokban, esetleg külön megnyitható fülekben találjuk az egyes állományokat. Ezt a gyakori vizuális megközelítést hívjuk a japán ételesdoboz neve alapján „bento-box" stílusnak. A bento-box hátránya, hogy egyszerre nem lehet látni a képernyőn 3–4 fontosabb összetartozó elemnél többet, nem látjuk az állományok közötti kapcsolatokat, és az egyes ablakokban gyakran kell időrabló navigálással töltenünk az időt, mire megtaláljuk a nekünk érdekes elemeket. Mindazonáltal ez a formátum tökéletesen megfelel az új kód fejlesztésének, ahol szekvenciálisan dolgozunk, több időt töltve ugyanazon pár állomány szerkesztésével.

A kódmegértés folyamata azonban jelentősen eltér ettől a sémától. Egy ismeretlen, vagy csak részleteiben ismert szoftver megismerése során számos állományt kell megnyitnunk, a bennünk definiált osztályokat, függvényeket megvizsgálnunk, a köztük levő kapcsolatokat felfedeznünk. Eközben folyamatosan váltakozik az alkalmazott absztrakciós szint: egyszer egy függvény implementációját vizsgáljuk meg, utána megtekinthetjük az ezt tartalmazó osztályt, onnan pedig az osztály-kapcsolatokat kell felmérnünk. Sokszor 10–15 lényeges (és gyakran különböző állományokban elhelyezkedő) programelemet és kapcsolataikat kell egyszerre megjelenítenünk. Ez nem megy a bento-box vizualizációval. A másik jelentős eltérés, hogy amíg az új kód írása tipikusan textuálisan folyik, a megértéshez lehet és érdemes is grafikus megjelenítést alkalmazni, miután az nagyobb sávszélességet biztosít a megértés folyamata során, mint a szöveges.

Ezen különbségek alapján jöttek létre a specifikusan a kódmegértést támogató eszközök. Ezek az eszközök gyors kereséseket, precíz kódnavigációt és széleskörű grafikai jelölésrendszert használnak a fejlesztőknek. Ilyen eszköz a CodeCompass, egy nyílt forráskódú, szabad felhasználású keretrendszer [17]. A CodeCompass beépülő modulokkal bővíthető, a statikus elemzésen alapuló rendszer, amely a szoftverről megszerezhető információk teljes skáláját felhasználja; a forráskódon kívül pl. a fordítási és szerkesztési parancsokat vagy a git verziókezelő adatait is. Az így megszerzett információt egy adatbázisban tárolja, majd egy webböngésző vagy REST API segítségével továbbítja a felhasználó felé. A

CodeCompass mindig a szoftver egy pillanatképét ismeri, módosulás esetén gyors inkrementális parszolásra van lehetőség.

2. A kódmegértés folyamata

A kódmegértés régóta fontos, alaposan vizsgált része a szoftverfejlesztés folyamatának. Bár minden szoftver más, és mindegyik programozó kialakítja a saját stratégiáját a nagy rendszerek megértéséhez, mégis vizsgálhatjuk ezt a folyamatot tudományos igényességgel. Többek között von Mayrhauser [16], Storey [15] és O'Brien [10] gyűjtötte össze, vizsgálta és kategorizálta a megértési modelleket.

Ahogy von Mayrhauser és társai leírják [16], számos olyan közös elem van a különböző stratégiákban, amely független a megismerendő rendszertől. Ezek részben statikusak, mint pl. tervek, amelyeket a fejlesztők követnek; programelemek, amelyek megértését a többi elemtől függetlenül próbálják végrehajtani; hipotézisek, melyeket felállítanak a kóddal kapcsolatban és amelyeket igazolnak vagy elvetnek. Másrészt léteznek dinamikus elemek is, mint a kapcsolatkeresés más programrészekkel. Mindamellett a legtöbb esetben a fejlesztők vagy a program nagyobb léptékű struktúrájától haladnak a kisebb részletek felé (top-down módszer) vagy pedig kisebb programelemek megértése után próbálják szintetizálni az ismereteket a teljes rendszer megértése céljából (bottom-up módszer).

Brooks modellje szerint a fejlesztők a top-down módszer szerint haladnak, a szakterületi ismereteikre alapozva mikro-hipotéziseket állítanak fel a kódról, és ezeket belátják és tovább finomítják, vagy elvetik és alternatív hipotéziseket állítanak fel a megismert részletek alapján [1].

Soloway, Adelson és Ehrlich cikkükben egy másik top-down módszert ismertetnek [14]. Modelljük szerint a megértés során a fejlesztők hierarchikus megértési célokat állítanak fel a teljes rendszerrel szembeni elvárásokkal kezdve. Innen indulva a részletek felé próbálják iteratív módon finomítani a megismert kód hatására az elképzeléseiket. Úgy vélik, a top-down módszereket elsősorban akkor alkalmazzák a fejlesztők, ha ismert kódmintákkal és programstruktúrákkal találkoznak.

Vannak azonban, akik elsősorban a bottom-up módszereket elemezték. Pennington [11] két mentális modellt különböztet meg, a program modellt és a szituációs modellt. Az előbbi a program vezérlési szerkezetét veszi alapul, és így építkezik alulról felfelé. A szituációs modellben

a program (rész)céljait és az adatáramlást vesszük tekintetbe. Itt is a részletek felől közelítünk, de figyelembe vesszük a szakmaspecifikus tudást, és hierarchikus modellt építünk.

Shneiderman és Mayer a megértés folyamatának irányát vizsgálták a memória szempontjából, miközben a programkód és a már meglévő magasabb szintű elképzelések feldolgozásával szerzett tudás a rövid távú memóriából a hosszú távú memóriába jut [13]. Modelljük megkülönbözteti a szintaktikai és a szemantikai tudást. Míg az előbbi pl. az alkalmazott programozási nyelv elemeinek felismerésén alapul, az utóbbi a kód alacsony szintű gyakorlati részeitől a programabsztrakciókon át a szoftver magasabb céljai felé terjednek.

Levy [7] megjegyzi, hogy a felülről lefelé történő megértési folyamatot általában akkor alkalmazzák a fejlesztők, ha céljuk a teljes rendszer architektúra vagy egyes nagyobb modulok szerepének megértése. Az alulról felfelé történő megközelítésre pedig elsősorban akkor kerül sor, amikor hibajavítás vagy egy új funkció beépítése történik.

A kódmegértés gyakorlati alkalmazása során fontos koncepció a funkcionalitás lokalizációja [8]. Amikor egy hibát javítunk, vagy egy új funkcionalitással bővítünk egy rendszert, először meg kell keresni azt a részét a szoftvernek, amely a leginkább vonatkozik az adott feladatra. Ez a terület néha közvetlen módon beazonosítható: létezik rá vonatkozó osztály, függvény vagy valami más hasonló programelem. Néha azonban csak implicite találjuk meg a megfelelő helyet, valamely hasonló absztrakció, előfeltétel alapján [12].

A funkcionalitás lokalizációja nem kapcsolódik egyetlen specifikus megértési modellhez sem, inkább az összes modell kezdőlépésének tekinthetjük. Első lépésben megpróbáljuk megkeresni a bennünket érdeklő kódrészletet, és onnan kiindulva (top-down vagy bottom-up módszerrel) bővítjük a rendszerről alkotott ismereteinket.

3. Kódmegértést támogató eszközök

Számos szoftvereszköz létezik a kódmegértés direkt vagy indirekt támogatására. Egyeseket a kódfejlesztés céljára fejlesztettek ki, mint a grafikus fejlesztői környezeteket (IDE), másokat célirányosan a kód megértése szempontjai szerint alakítottak ki.

A legtöbb grafikus fejlesztői környezet, mint a Viusal Studio, IntelliJ, Netbeans, Eclipse, CodeBlocks, QtCreator, rendelkeznek vala-

mennyi kódnavigációs funkcionalitással. Legtöbbször ez azonban kimerül a definíció helyére ugrással (GoTo definition) és egy név összes előfordulásának megkeresésével (All references). Még ezekben a rendszerekben sem egységes, hogy csupán név szerinti keresés történik ilyen esetekben, vagy pedig a szemantikusan helyes szimbólum feloldása történik meg. Ezek a lehetőségek elsősorban a kód fejlesztésekor kényelmesek, nem feltétlenül alkalmasak a funkcionalitás lokalizálására (ha pl. nem ismerjük pontosan a keresett elem nevét), illetve a találatok megjelenítése és bejárása sem kényelmes. Másrészt nagy projektek esetén gyakran nem képesek a teljes rendszert beindexelni, ezért a találati eredmények hiányosak lehetnek. Mindazonáltal el kell ismerni, hogy az IDE-k komoly fejlődésen mentek keresztül, és várhatóan egyre több megértést támogató elem épül be eszközkészletükbe.

A célirányosan a kódmegértés támogatására létrehozott eszközök között meg kell különböztetnünk a gyors, kényelmes, de nem feltétlenül pontos kódnavigációt biztosító eszközöket és a mélyebb szemantikus ellenőrzést végző szoftvereket.

Az előző csoportba tartozik pl. az **OpenGrock** [21], ami egy gyors keresztreferencia rendszer. Ezek az eszközök felolvassák a forráskódot, elvégeznek bizonyos lexikai és szintaktikai elemzéseket, de nem végeznek mélyebb szemantikus ellenőrzéseket, név- vagy túlterhelés feloldásokat. Az OpenGrock pl. a ctags [19] könyvtárat használja, amely csak textuálisan értelmezi a forráskódot, indexeli és heurisztikusan próbálja meghatározni az egyes nyelvi elemek típusát. Ez lehetővé teszi, hogy megkülönböztesse függvények, változók vagy típusok neveit, a nevek definícióit és hivatkozásait, de pl. a túlterhelések feloldására már nem alkalmas. A szemantikus elemzés hiánya lehetővé teszi számos programozási nyelv támogatását. Az információ tárolása igen magas szinten optimalizált a keresésekre, így az OpenGrock az esetek többségében gyors válaszidejű. Cserében a felhasználónak gyakran magának kell feloldania a többértelműségeket, ami nagy méretű projektek esetén nem triviális feladat, egy esetleges tévedés pedig komoly félreértésekhez vezethet.

A **Woboc Code Browser** [23] ugyancsak web-alapú rendszer, de ellentétben az OpenGrock-al mély, szemantikus elemzést is végez. A felhasználó gyorsan meg tud találni fájlokat, osztályokat vagy egyéb névvel rendelkező programelemeket egy kódkiegészítést is tartalmazó gyors keresés segítségével. A navigáció az elemzés során legenerált sta-

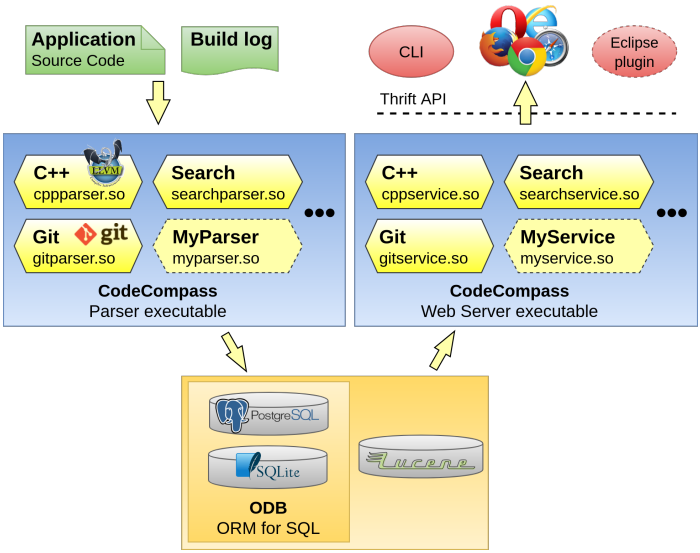
tikus HTML lapok segítségével történik, ami egyértelműen előnyös a működési sebesség szempontjából. Az egyes elemek fölé vitt egér hatására további információk jelennek meg a programelemekről. Függvények esetén pl. láthatjuk a szignatúrát, a definíció és a hívások helyeit; osztályoknál a méretet, az adattagokat és az esetleges öröklési kapcsolatokat; változó esetében a típust és az írási vagy olvasási pontokat. A Woboc jól kezeli a C/C++ makrók kifejtését. Mindezeket a mély információkat a Woboc azáltal éri el, hogy – az OpenGrock-al ellentétben – egy valódi fordítóprogramot, az LLVM/Clang környezetet használja fel az elemzéshez. Mindez persze azt is jelenti, hogy a Woboc használata C és C++ projektekre korlátozódik.

Az **Understand** [22] nem csak kódnézegető és navigációs eszköz, de egy számítógépre telepített komplett IDE is, azaz a fejlesztő módosíthatja is a kódot, és a változások azonnal reflektálódnak az eszközben. Az előzőekben már említett funkcionalitások mellett az Understand számos statisztikát és mérőszámot szolgáltat a vizsgált rendszerről. A szokásos programsor alapú metrikák, mint az összes, átlagos, osztályonkénti programsorok mellett az alap strukturált és objektumelvű metrikákat (mint ciklomatikus bonyolultság, kapcsolódás, kohézió hiánya) is megtaláljuk [5, 9]. A vizuális megjelenítés gyakran használ ún. Treemap-et, ami beágyazott téglalapokkal ábrázolja a hierarchikus kapcsolatokat, és színekkel, illetve a téglalapok méretével a metrika értékeket. Nagy projektekben a rendszer általános architektúrájának vizsgálata is fontos. Az Understand képes megmutatni a különböző függési kapcsolatokat, függvény hívási láncokat és számos más diagramot. Az Understand az elemzés közben adatbázist készít a begyűjtött információkból és mindezeket az adatokat a felhasználó is elérheti egy programozói interfészen keresztül.

A **CodeSurfer** [18] ugyancsak „vastag” kliens, mint az Understand, és C és C++ (valamint x86 alapú gépi kódú) projekteket céloz meg. Hasonlóan a Woboc-hoz és Understand-hez mély szemantikus elemzést is végez, beleértve még bizonyos mutató elemzéseket és hatásvizsgálatokat is. Ez utóbbi például megmutatja, mely utasítások függenek egy kiválasztott program ponttól, ami komoly adatfolyam elemzéseket igényel.

4. A CodeCompass architektúra

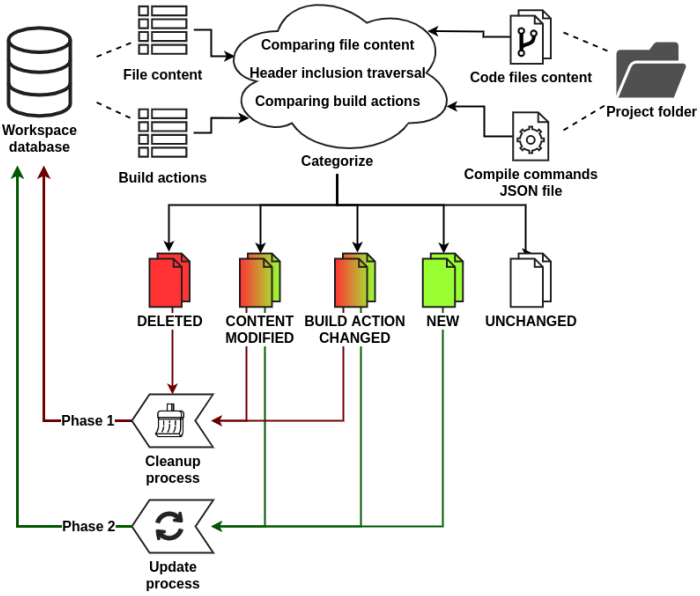
A CodeCompass [2, 17] egy nyílt forráskódú, szabad felhasználású, szoftver megértést támogató keretrendszer, mely az Eötvös Loránd Tudományegyetem és az Ericsson Magyarország Kft. közös fejlesztéseként indult. A keretrendszer általános felépítése az 1. ábrán látható.



1. ábra. A CodeCompass architektúra

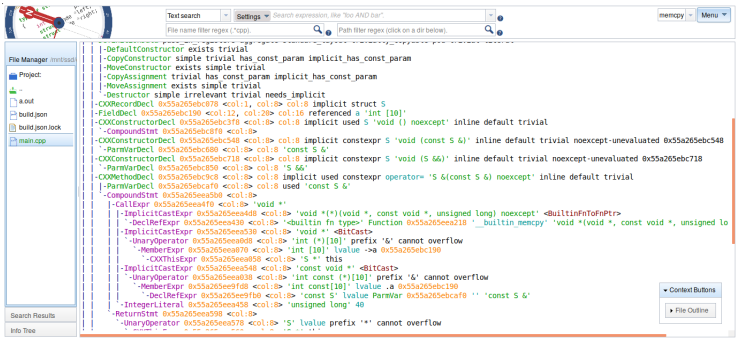
A CodeCompass működése két főbb szakaszból áll: az információ begyűjtése a vizsgált projektről („parszolás”), majd ezen információ publikálása a kliens(ek) felé egy REST API-on keresztül. Ennek megfelelően a két fő bináris komponens a parszer és a webszerver. A parszer által begyűjtött adatokat adatbázisokban tároljuk, ebből dolgozik a webszerver. Mind a parszert, mind a webszervert beépülő modulok segítségével bővíthetjük új funkcionalitással, mely beépülő modulok tartalmazzák a szükséges adatbázis leírásokat is. Az adatbázisok részben relációs alapúak (pl. postgreSQL), részben a gyors keresést támogatják (pl. Lucene), illetve, ha elérhető, a CodeCompass tárolja a projekttel kapcsolatos verzió-kezelő (pl. git) adatbázist is.

A munkamenet a következő. A parszer (a beépülő modulokon keresztül) elemzi a projektet, gyakran a build parancsokkal együtt. A C és C++ projektek esetében ehhez pl. a compile command json állományt használja fel, amelyet cmake alapú projektek esetén a build rendszerből lehet kinyerni, ennek hiányában pedig egy speciális eszköz, a logger tudja generálni a build parancsok végrehajtása közben. Az egyes állományokból absztrakt szintaxisfa (AST) készül, melyeket megvizsgálva a legfontosabb adatokat a parszer eltárolja az adatbázisban.



2. ábra. A CodeCompass inkrementális parszolása

A CodeCompass a szoftverprojekt egy pillanatnyi állapotát képezi le, de lehetőség van több, különböző verzió közötti váltásra is. Amennyiben a kód változik, lehetőség van az adatbázisok inkrementális frissítésére, ahogy azt a 2. ábrán láthatjuk. A parszer detektálja a forrásfájlokban történt változásokat, még akkor is, ha ez implicit módon, egy fejállomány tartalmának megváltoztatásával történt. Ekkor automatikusan meghatározza a módosítás hatását az adatbázisra, és csak a megfelelő tartalmakat frissíti [4].



3. ábra. Egy tipikus képernyőrészlet, forráskód AST nézetének megjelenítésével

A webszerver egy REST API-on keresztül szolgáltat információt a kliensek felé. A kliens lehet egy felhasználó, aki valamely standard böngészőn keresztül éri el a webszervert, lehet valamilyen eszköz, például editor, aki egy kiegészítőn keresztül kommunikál a CodeCompass-szal, vagy lehet egy célirányos szoftver is. Az állapotmentes interfész lehetővé teszi, hogy a szerver jól skálázódjon és egyszerre több szálon szolgálja ki a klienseket. A CodeCompass célkitűzése, hogy a lehető legtöbb lekérdezés 1 másodpercen belül teljesüljön, mivel az a tapasztalat, hogy hosszabb várakozási idő esetében a felhasználó hajlamos félbeszakítani a munkafolyamatot és más tevékenységbe kezdeni.

5. Felhasználói felület, tipikus munkamenet

A CodeCompass felhasználói felülete három főbb részre tagolódik. Ahogy az a 3. ábrán is látszik, legfelül találjuk a kereső felületet és opciókat, bal oldalt a navigációs panelt, a felület legnagyobb részét pedig a központi panel foglalja el, amelyben a kurrens vizsgált kódrészletek vagy diagramok jelennek meg,

A tipikus kódmegértési munkamenet általában a keresett programelem lokalizációjával indul. Erre több lehetőségünk is van. Amennyiben tudjuk, hogy a keresett elem mely fejléc- vagy forrásfájlban található, használhatjuk a megszokott navigációs panelt az állomány betöltéséhez. Amennyiben ez az információ nem áll a rendelkezésünkre, reguláris ki-

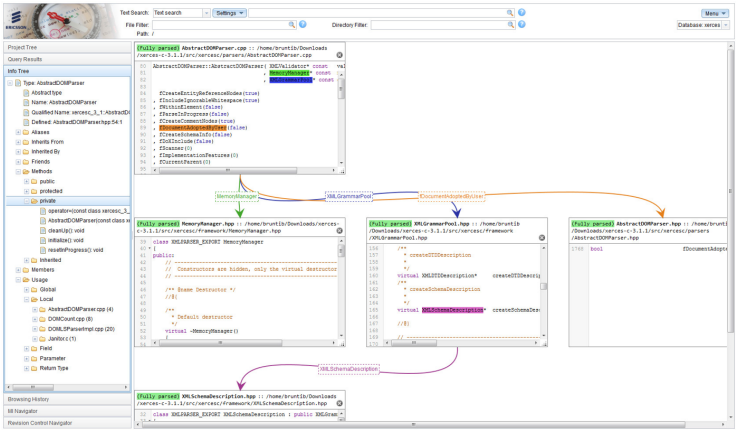
fejezésekkel kereshetünk a fájl, osztály, függvény vagy változó nevére. A keresés a találatokat page-ranking által rangsorolva mutatja. A keresésbe bevonhatjuk a forrásfájlok teljes szövegét és az összes állományt, de kizárhatjuk pl. a kommenteket, és szűrhetünk különböző típusú nyelvi elemekre is.

Az egyik érdekes keresési lehetőség a „log keresés". A programok gyakran bocsájtanak ki üzeneteket, melyeket nem találhatunk meg egy az egyben a forrásfájlban, hiszen bizonyos összetevők (pl. fájlnév, programsor, hibakód) változók értékéből származik. A log keresés meghatározza azokat a pontokat (szintén page-rank alapján megjelenítve), melyek a legvalószínűbb forrásai az ilyen üzeneteknek. A log keresés rendkívül hatékony, ha egy üzenetben kijelzett hiba okát keressük, vagy valamely logfájl alapján derítjük fel a szoftver rendszert.

Amennyiben megtaláltuk a keresett állományt, annak forráskódja megjelenik a központi panelen. Egérrel egy programelemre klikkelve a legfontosabb információk megjelennek a bal oldali ún. információs panelen, illetve jobb-klikk hatására egy, az elem típusától függő menü jelenik meg. A szokásos „ugorjunk az elem definíciójára" lehetőség és az információs panel mellett választhatjuk a verziókezelő információkat, rövid összefoglaló dokumentációt, nyelvspecifikus funkciókat és különböző diagramok megjelenítését.

A diagramok, melyek a vizuális megjelenítés hatékony eszközei, szervesen használhatók a CodeCompass-on belül. A rendszer komponensei közötti kapcsolatokat megmutató architektúra diagram, a függvényhívási lánc megjelenítése, osztálydiagramok, és a gyors kódböngészést lehetővé tevő CodeBites diagram mind interaktívan használható. A CodeBites (lásd 4. ábra) egy különleges megjelenítés, ahol a kódban hivatkozott elemeket (változók, függvények, típusok, makrók) egymás mellé kifejtve jeleníthetünk meg akkor is, ha egyébként különböző állományokban léteznek. A CodeBites a bento-box technika ellentéte; gyors, kötetlen böngészést tesz lehetővé nagyszámú programelem bevonásával, amelyek akár mind-mind különböző állományban szerepelnének.

A CodeCompass továbbfejleszthetőségét a beépülő modulok biztosítják, melyek egyaránt tartalmazhatnak parszer, szerver, web-kliens és adatbázis elemeket is. További bővíthetőségi lehetőséget garantál a Language Server Protocol (LSP) interfész [20], melyen keresztül a CodeCompass bármilyen LSP-képes editorral vagy különálló eszközzel összekapcsolható.



4. ábra. A CodeBites nézet gyors, kötetlen böngészést tesz lehetővé

6. Összefoglalás

A szoftveripar folyamatos hatékonysági kényszerben dolgozik, miközben egyre nő a szoftverek komplexitása, drágul a munkaerő és szűkülnek a határidők. Eközben a fejlesztők idejük jelentős (néha legnagyobb) részét töltik meglévő kódok kiegészítésével vagy javításával, aminek elengedhetetlen előfeltétele a szoftverkörnyezet minél tökéletesebb megértése. A hagyományos fejlesztői környezetek nem a kód megértésére optimalizáltak. A CodeCompass mint egy web interfésszel rendelkező, beépülő modulokkal kiegészíthető, nyílt forráskódú keretrendszer sikeresen bizonyította a kódmegértést támogató célirányos szoftverek létjogosultságát. Az elmúlt évek használata során a felhasználói visszajelzéseken alapuló folyamatos fejlesztések és az új funkcionalitások bevezetése (új programozási nyelvek, szoftvermetrikák stb.) egy kiforrott rendszert eredményeztek, melyet ingyenesen használhat és érdemes is használnia a szoftverfejlesztői közösségnek.

Hivatkozások

[1] R. Brooks, Towards a theory of the cognitive processes in computer programming, *International Journal of Man-Machine Studies,* 9(6) (1997), pp. 737–751.

[2] T. Brunner, M. Cserép, A. Fekete, M. Mészáros, Z. Porkoláb, Towards better tool support for code comprehension, in *Composability, Comprehensibility and Correctness of Working Software,*, Z. Porkoláb and V. Zsók, Eds. Cham: Springer International Publishing, 2023, pp. 165–201.

[3] M. Enderin, D. LeCorney, M. Lindberg, T. Lundqvist, Axe 810—the evolution continues," *Ericsson Review*, 4 (2001), pp. 10–23.

[4] A. Fekete, M. Cserép, Incremental Parsing of Large Legacy C/C++ Software, *21th International Multiconference on Information Society (IS), Collaboration, Software and Services in Information Society (CSS),* Vol. G (2018), pp. 51–54.

[5] B. Henderson-Sellers, *Object-oriented metrics: measures of complexity,* Prentice-Hall, Inc., 1995.

[6] E.-A. Karlsson, L. Taxen, Incremental development for axe 10, *Software Engineering—ESEC/FSE'97,* Springer (1997), pp. 519–520.

[7] O. Levy, D. G. Feitelson, Understanding large-scale software: a hierarchical view, *Proceedings of the 27th International Conference on Program Comprehension,* IEEE Press, 2019, pp. 283–293.

[8] A. Marcus, A. Sergeyev, V. Rajlich, J. I. Maletic, An information retrieval approach to concept location in source code, *11th working conference on reverse engineering,* IEEE, 2004, pp. 214–223.

[9] T. J. McCabe, A complexity measure, *IEEE Transactions on software Engineering,* 4 (1976), pp. 308–320.

[10] M. P. O'brien, Software comprehension–a review & research direction, *Department of Computer Science & Information Systems University of Limerick, Ireland, Technical Report*, 2003.

[11] N. Pennington, Comprehension strategies in programming, *Empirical studies of programmers: second workshop*, Ablex Publishing Corp., 1987, pp. 100–113.

[12] V. Rajlich and N. Wilde, The role of concepts in program comprehension, *Proceedings 10th International Workshop on Program Comprehension,* IEEE, 2002, pp. 271–278.

[13] B. Schneiderman, R. Mayer, Syntactic/semantic interactions in programmer behavior: A model and experimental results, *International Journal of Computer & Information Sciences,* 8(3) (1979), pp. 219–238.

[14] E. Soloway, B. Adelson, K. Ehrlich, Knowledge and processes in the comprehension of computer programs, *The nature of expertise,* pp. 129–152, 1988.

[15] M.-A. Storey, Theories, methods and tools in program comprehension: Past, present and future, *13th International Workshop on Program Comprehension (IWPC'05),* IEEE (2005), pp. 181–191.

[16] A. Von Mayrhauser, A. M. Vans, Program comprehension during software maintenance and evolution, *Computer,* 28(8) (1995), pp. 44–55.

[17] Codecompass. `https://github.com/Ericsson/CodeCompass`

[18] Codesurfer. `https://www.grammatech.com/products/codesurfer`

[19] Doxygen. `http://ctags.sourceforge.net`

[20] Language Server Protocol Specification, Microsoft Corporation Tech. Rep. 3.14.0, 12 2018. `https://microsoft.github.io/language-server-protocol/specification`

[21] Opengrok. `https://opengrok.github.io/OpenGrok`

[22] Ctags. `http://www.stack.nl/~dimitri/doxygen/`

[23] Woboq. `https://woboq.com/codebrowser.html`

Problémák és eredmények

Képes beszámoló a Numerikus Analízis Tanszék múltjáról[‡]

Schipp Ferenc[*]

ELTE Informatikai Kar, Numerikus Analízis Tanszék[**]

schipp@inf.elte.hu

Az informatika (régebben használt elnevezéssel, a számítástechnika) hazai történetében meghatározó szerepet játszott az 1968-ban létrehozott *Numerikus és Gépi Matematika Tanszék* és annak vezetője, *Kátai Imre* professzor. A természettudományi karok együttműködésében beindították az egységes programozó-, majd később a programtervező matematikus képzést, a tudományos kutatásokat, és erőteljes növekedésnek indultak az alkalmazások a számítástechnika területén. Az ún. „matematikus rendezés" eredményeként szétválasztották a tanszéket, és állományából *Számítástechnika* és *Numerikus Analízis Tanszék* néven, *Varga László* és *Schipp Ferenc* professzorok vezetésével két új egység jött létre.

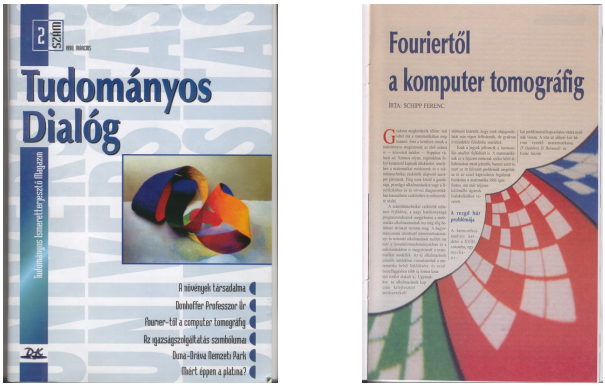
A több mint 60 éves felsőoktatásban töltött idő során számos oktatást és kutatást érintő vitát éltem át. Visszatérő problémaként jelentkezett a *diszkrét és folytonos* módszerek, a *matematika és az informatika* viszonyának kérdése. Ezzel összefüggésben szélsőséges nézetek is napvilágot láttak. Többek között ilyen kérdések kerültek terítékre: gráfelméletet vagy differenciálegyenleteket oktassunk, árthat-e egy informatikai

[‡] A professzor úrnak az Informatikai Műhely felkérésére elkészített előadásának írásos anyaga. Az előadás elhangzott 2023. december 11-én, online, szélesebb körű hallgatósággal.

[*] professor emeritus

[**] Tanszékvezető: 1984–2004.

tárgyú disszertációnak, ha benne matematikát is használnak? Előadásomban ezeket a kérdéseket is fogom érinti. Példák sora bizonyítja, hogy matematikai felfedezések, gyakran jóval születésük után, milyen fontos szerepet játszottak a műszaki- és természettudományokban. A diszkretizáció kérdése, az említettől eltérő felfogásban, a numerikus matematikának is egyik alapvető problémája. Az alábbi dolgozatban (1. kép) [3], amelyet felhasználtak a Wikipedia Fourier-analízisről szóló rész ismertetésében, összegyűjtöttem néhány ilyen példát. Remélem, hogy a bemutatott példák jól illusztrálják véleményemet a vitatott kérdésekben.



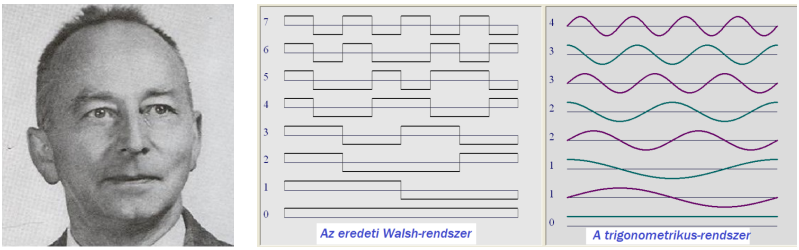
1. kép. Tudományos Dialóg: Fouriertől a komputer tomográfig

Ebben a visszatekintésben a 40 éve alakult Numerikus Analízis Tanszéken folyó kutatásoknak néhány mozzanatát elevenítem fel, közel sem teljességre törekedve.

1. Digitalizáció, diadikus analízis

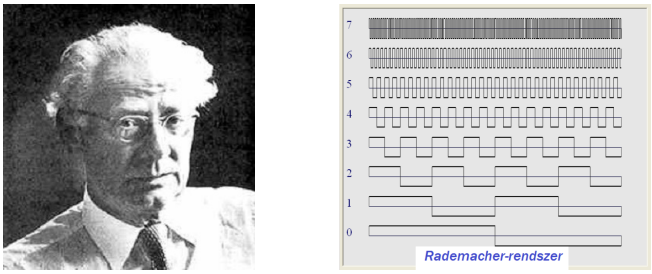
Walsh egy 100 éve írt dolgozatában bevezetett egy, azóta róla elnevezett rendszert, amely a ma divatos szóhasználattal élve a *trigonometrikus rendszer digitális változatának* tekinthető. A $V = (v_n, n \in \mathbb{N})$ Walsh-függvények csak az $1, -1$ értékeket veszik fel, és az előjelváltások száma megegyezik a trigonometrikus rendszerével (2. kép).

Walsh rendszerét nehezen kezelhető, bonyolult rekurzióval definiálta. A Walsh-rendszernek a *Rademacher* által 1920-ban bevezetett



2. kép. J.L. Walsh (1895–1973), a Walsh-rendszer és a trigonometrikus rendszer

$R = (r_n, n \in \mathbb{N})$ rendszer egy részrendszere: $v_{2^n} = r_n$ $(n \in \mathbb{N})$ (3. kép).



3. kép. H. Rademacher (1892–1969) és a Rademacher-rendszer

A R rendszer fontos tulajdonsága, hogy

(i) az r_n függvények r_0-ból dilatációval származtathatók:

$$r_n(x) = r_0(2^n x) \quad (x \in [0, \infty), n \in \mathbb{N}),$$

(ii) az r_n függvény értékei kifejezhetők az $x \in [0, 1)$ szám bináris jegyeivel:

$$r_n(x) = (-1)^{x_n} \quad (x = \sum_{n=0}^{\infty} x_n/2^{n+1} \in [0, 1), n \in \mathbb{N}),$$

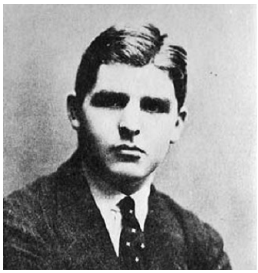
(iii) a $\sum_{n\in\mathbb{N}} a_n r_n$ Rademacher-sor akkor és csak akkor m.m. (majdnem

mindenütt) konvergens, ha

$$\sum_{n\in\mathbb{N}} |a_n|^2 < \infty.$$

Az R rendszer függvényei felfoghatók független valószínűségi változóknak, és a (iii) állítás a Kolmogorov-féle háromsor tétel speciális esete.

Paley egy 1932-ben írt alapvető dolgozatában megmutatta, hogy a Walsh- és a Rademacher-rendszer között szoros kapcsolat van.



4. kép. R.E.A.C. Paley (1907–1933)

Nevezetesen Paley bevezette a Rademacher-függyvényekből alkotott összes lehetséges véges szorzatot. Ezeket az $n = \sum_{k=0}^{\infty} n_k 2^k \in \mathbb{N}$ szám $n_k \in \{0, 1\}$ bináris jegyeit felhasználva

$$w_n = r_0^{n_0} \cdot r_1^{n_1} \cdots r_k^{n_k} \cdots$$

alakban írhatjuk fel. Az azóta elterjedt szóhasználattal élve azt mondjuk, hogy a $W = (w_n, n \in \mathbb{N})$ (ún. Walsh–Paley-féle rendszer) az R *rendszer szorzatrendszere.* Megmutatható, hogy a V előállítható az $(r_k r_{k-1}, k \in \mathbb{N})$ $(r_{-1} := 1)$ rendszer szorzatrendszereként. Ebből már következik, hogy a két rendszer ugyanazokból a függvényekből áll, egymástól különböző sorrendben felírva.

A kapcsolat digitális jellege még szembetűnőbb, ha a bitsorozatok halmazából indulunk ki:

$$\mathbb{D} := \{n = (n_k, k \in \mathbb{N}) : n_k = 0, 1\},$$
$$\mathbb{D}_0 := \{n = (n_k, k \in \mathbb{N}) : n_k = 0, 1, \lim_{k\to\infty} n_k = 0\}.$$

Számok $x = \sum_{n\in\mathbb{N}} x_n 2^{-n-1} \in \mathbb{I} := [0,1)$ $(x_n = 0,1)$ bináris előállítását felhasználva vezessük be az $\mathbb{I} \ni x \mapsto \breve{x} = (x_n, n \in \mathbb{N}) \in \mathbb{D}$ leképezést, ahol diadikus racionális számok esetén a lehetséges két előállítás közül azt választjuk, amely nullákkal végződik. Ekkor az $x \mapsto \breve{x}$ leképezés injektív és a $\mathbb{D} \setminus \breve{\mathbb{I}}$ halmaz megszámlálható. Ennek alapján $\mathbb{D}_0$ a természetes számok $\mathbb{N}$ halmazával, $\mathbb{D}$ az $\mathbb{I}$ intervallummal azonosítható. A Walsh–Paley-rendszer kifejezhető a bináris jegyekkel:

$$w_n(x) = (-1)^{\lfloor n,x\rfloor}, \quad \lfloor n,x\rfloor = \sum_{k\in\mathbb{N}} n_k x_k \quad (n \in \mathbb{N} \equiv \mathbb{D}_0, x \in \mathbb{I} \equiv \mathbb{D}).$$

Jelölje $\mathbb{Z}_2 = (\{0,1\}, \dot{+}, \cdot)$ a kételemű véges testet. A

$$\mathbb{D} = \mathbb{Z}_2 \times \mathbb{Z}_2 \times \cdots \times \mathbb{Z}_2 \times \cdots$$

direkt szorzat az indukált

$$x\dot{+}y = (x_n\dot{+}y_n, n \in \mathbb{N}) \quad (x, y \in \mathbb{D})$$

művelettel csoportot alkot. A $(\mathbb{D}, \dot{+})$ csoportot *diadikus* (vagy Cantor-féle) *csoportnak* nevezzük. A $\dot{+}$ művelet azonos a számítógépekbe beépített, a bytok között értelmezett *EOR* logikai művelettel. Ezen keresztül kapcsolódik ez a terület a logikához és a Bool-függvények elméletéhez.

Néha célszerű a $\mathbb{D}$ halmazt a $\mathbb{Z}_2$ test feletti vektortérnek tekinteni. Ennek $\mathbb{D}_0$ egy altere, és az

$$\|x\| := \sum_{k\in\mathbb{N}} x_k 2^{-(k+1)} \in [0,1]$$

leképezés (egy ún. nem-archimédeszi) *norma* a $\mathbb{D}$ halmazon:

$$\|x\dot{+}y\| \le \max\{\|x\|, \|y\|\} \le \|x\| + \|y\| \quad (x, y \in \mathbb{D}).$$

Minden $\mathbb{Z}_2$-lineáris $T : \mathbb{D}_0 \to \mathbb{D}_0$, $T(x) = y$ leképezés

$$\dot{T} = (t_{ij}, i, j \in \mathbb{N}) \quad (t_{ij} \in \mathbb{Z}_2)$$

típusú végtelen mátrixszal felírható

$$y_i = \dot{\sum_{j\in\mathbb{N}}} t_{ij} x_j \quad (i \in \mathbb{N})$$

alakban. Speciálisan az eredeti Walsh-rendszer a Walsh–Paley-rendszerből a

$$\dot{\mathcal{T}} := \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots \\ 1 & 1 & 0 & 0 & \cdots \\ 0 & 1 & 1 & 0 & \cdots \\ 0 & 0 & 1 & 1 & \cdots \\ \vdots & & & & \end{pmatrix} \tag{1}$$

$\mathbb{Z}_2$-lineáris transzformációval származtatható:

$$v_n = w_{\mathcal{T}(n)} \quad (n \in \mathbb{N}).$$

A szakirodalomban más $\mathbb{Z}_2$-lineáris leképezést is használnak. Például az

$$\begin{aligned} \mathcal{F}_k : (n_0, n_1, \dots, n_{k-1}, n_k, n_{k+1}, \dots) \to \\ \to (n_{k-1}, n_{k-2}, \dots, n_0, n_k, n_{k+1}, \dots) \quad (k \in \mathbb{N}) \end{aligned}$$

bitfordító transzformációkkal definiálhatók a *Hadamard-mátrixok* és a *Kaczmarz-féle átrendezések.* Megjegyezzük, hogy ezek a transzformációk az FFT algoritmusokban is fontos szerepet játszanak.

Az $x \to \check{x}$ leképezésben véve az $\mathbb{I}$ intervallum Lebesgue-mérhető halmazainak képét, egy mértéket vezethetünk be $\check{\mathbb{I}}$ halmazon. A $\mathbb{D} \setminus \check{\mathbb{I}}$ halmaz pontjainak mértékét nullának választva a mértéket kiterjeszthetjük a diadikus csoportra. Ez a mérték *transzláció invariáns,* azaz $|E|$-vel jelölve az E halmaz mértékét

$$|E| = |\check{E}| = |x \dot{+} \check{E}| \quad (x \in \mathbb{D}). \tag{2}$$

Minden $T : \mathbb{D}_0 \to \mathbb{D}_0$ $\mathbb{Z}_2$-lineáris leképezéshez (egyértelműen) létezik olyan $T^* : \mathbb{D} \to \mathbb{D}$ $\mathbb{Z}_2$-lineáris leképezés, hogy

$$\lfloor T(n), x \rfloor = \lfloor n, T^*(x) \rfloor \quad (n \in \mathbb{D}_0, x \in \mathbb{D}).$$

Innen következik, a szóban forgó rendszerek átrendezése argumentumtranszformációval is megvalósítható:

$$v_n(x) = w_{\mathcal{T}(n)}(x) = (-1)^{\lfloor \mathcal{T}(n), x \rfloor} = (-1)^{\lfloor n, \mathcal{T}^*(x) \rfloor} = w_n(\mathcal{T}^*(x)) \\ (n \in \mathbb{N}, x \in \mathbb{D}).$$

Bebizonyítható, hogy ha T leképezés bijekció, akkor T^* *mértéktartó bijekció.*

Több olyan függvényrendszert ismerünk, amelyek *argumentum-transzformációval* származtathatók a trigonometrikus rendszerből. Az arccos transzformáció alkalmazásával eljuthatunk a Csebisev polinomokhoz, Blaschke-transzformációval a diszkrét Laguerre-rendszerhez. Esetünkben a T^* argumentum-transzformáció nem egy új rendszerhez, hanem a Walsh-rendszer átrendezéséhez vezet.

Megjegyezzük, hogy a műszaki alkalmazásokban a szinusz és koszinusz rendszer mintájára használják a

$$\mathrm{sal}_n(x) := v_{2n+1}, \quad \mathrm{cal}_n(x) := v_{2n} \quad (n \in \mathbb{N}, x \in \mathbb{I})$$

rendszereket. A továbbiakban csak a Walsh–Paley-rendszerrel foglalkozunk. A technikai alkalmazásokban felhasznált tulajdonságok az ismertetett kapcsolatok alapján átvihetők a V rendszerre.

A W rendszer egyik legfontosabb tulajdonságát *N.J. Fine* fogalmazta meg 1949-ben: A Walsh-függvények folytonosak a $\mathbb{D}$ diadikus csoporton, és eleget tesznek a

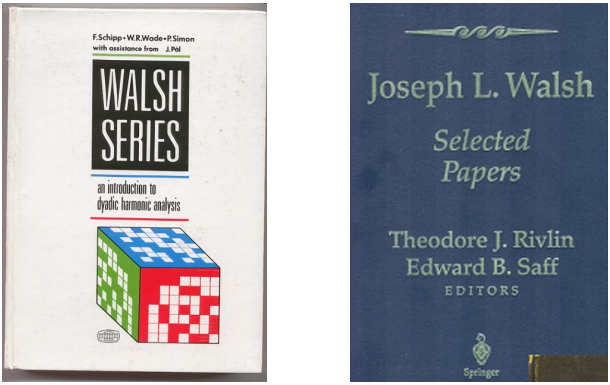
$$w(x\dot{+}y) = w(x)w(y) \quad (x, y \in \mathbb{D}) \tag{3}$$

függvényegyenletnek.

Megjegyezzük, hogy *N. Ya. Vilenkin* orosz matematikus, kiindulva ciklikus csoportok direkt szorzatából, két évvel korábban, 1947-ben függvényrendszereknek egy tág osztályát vezette be, amely a Walsh-rendszert is tartalmazza. A Vilenkin-rendszerekkel kapcsolatban napjainkban is intenzív kutatások folynak. *Simon Péter* és *Weisz Ferenc* professzorok jelentős mértékben hozzájárultak a terület eredményeihez. *Gát György* és *Toledo Rodolfo* professzorok véges, nem-kommutatív csoportok direkt szorzatán végzett vizsgálatai értékes eredményekkel gazdagították a harmonikus analízist.

A (3) függvényegyenlet úgy interpretálható, hogy a $w \in W$ függvények folytonos homomorfizmusok a $(\mathbb{D}, \dot{+})$ diadikus csoportról az egység abszolút értékű komplex számok $(\mathbb{T}, \cdot)$ multiplikatív csoportjába. A harmonikus analízis szóhasználatával: *a Walsh rendszer a diadikus csoport karakter-rendszere.* A W maga is (az $\mathbb{D}_0$ csoporttal izomorf) csoportot alkot a függvényszorzás műveletével. Ennek alapján a Walsh-sorokkal kapcsolatos vizsgálatok az absztrakt harmonikus analízis részének tekinthetők. A diadikus analízis elnevezéssel erre a kapcsolatra utalunk.

1990-ben jelent meg *Walsh series: an introduction to dyadic harmonic analysis* c. monográfiánk [2], amely azóta a szakterület egyik legtöbbet idézett forrása (lásd pl. Enciklopedia of Mathematics). Könyvünk címlapján az említett W, V és a Walsh–Kaczmarz-féle rendszer első 8 tagját szemléltetjük (5. kép).



5. kép. Walsh-sorokról szóló könyvünk, J.L. Walsh válogatott írásai

Walsh születésének 100-adik évfordulóján tanitványai egy válogatott kiadást jelentettek meg dolgozataiból [10]. Érdekességként megjegyzem, hogy hírnevét megalapozó rendszeréről összesen két cikket publikált. Munkásságának a zöme a komplex függvénytan területére esik. A szerkesztők felkérésére írt, a Walsh-rendszer matematikában játszott szerepét bemutató dolgozatom ebben a könyvben jelent meg (5. kép).

Könyvünk *W.R. Wade* amerikai professzorral együttműködésben, az NSF és az MTA támogatásával készült (mintegy 10 évi munkával). Az írás során Wade professzor többször járt Magyarországon, és a tanszék oktatói közül többen töltöttek hosszabb-rövidebb időt Knoxville-ben, a Tennessee egyetemen. Az együttműködésről szól a Természet Világa folyóiratban megjelent interjú [9], amely már címében is felhívja a figyelmet Wade professzornak a matematikáról alkotott szemléletére[1] (6. kép).

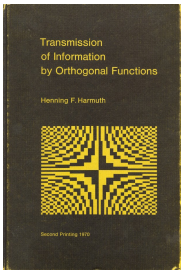
[1] Idézet az előadásból: „Abban az időben – hogy is mondjam – nem ezek voltak a szokványos címek."

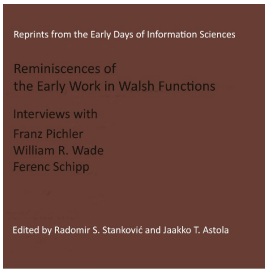
6. kép. Természet Világa, W.R. Wade (1943–2016)

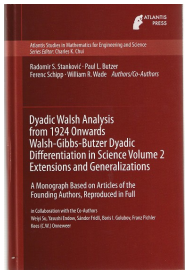
A matematikai analízis fogalmainak többsége átvihető a diadikus analízisbe. A derivált és integrál fogalmának diadikus változatát *J.E. Gibbs* angol fizikus egy ötlete nyomán *P.L. Butzer* és *H.J. Wagner* vezette be 1973-ban. Egyetemi pályájukat ekkor kezdő *Simon Péter* és *Pál Jenő* bevonásával bekapcsolódtunk az ezzel kapcsolatos kutatásokba. Többek között bebizonyítottuk a monoton- és az integrál függvény m.m. differenciálhatóságára vonatkozó klasszikus Lebesgue-tétel analogonját, ezzel megteremtve a diadikus analízis műveléséhez szükséges elméleti hátteret. Számos alapvető fogalom és módszer tanszékünk tagjainak nevével köthető össze. Többek között az approximációelmélet diadikus változatának kidolgozásában *Fridli Sándor* professzor, a diadikus Hardy- és BMO-terek általánosításában, ezek atomos jellemzésében *Weisz Ferenc* professzor szerzett el nem évülő érdemeket.

Az 1970-es évektől kezdődően több átviteltechnikával foglalkozó műszaki szakember látott lehetőséget a Walsh-rendszer alkalmazásában. Többen (pl. Harmuth és Pichler) azon a véleményen voltak, hogy az átviteltechnikában elterjedt trigonometrikus rendszert a Walsh-rendszerrel helyettesítve számos előny származik. Ennek szellemében több konferenciát rendeztek az USA-ban és több könyv jelent meg az elképzelésekkel kapcsolatban.

Az alábbiakban (7. kép) *Harmuth* egy ilyen témájú könyvére [1] és *Stanković* professzornak egy interjú dolgozatára [7] hívjuk fel a figyelmet. Ezekből tájékoztatást kaphatunk az említett alkalmazásokról. A harmadik kötet [8] a diadikus deriváltról nyújt áttekintést.







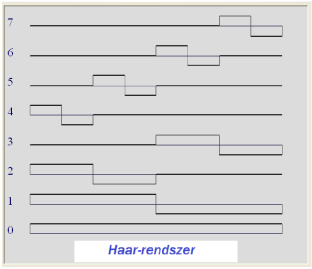
7. kép. H.F. Harmuth, R.S. Stanković, Stanković–Butzer–Schipp–Wade kötetei

2. A Haar-rendszer, waveletek

A matematikai analízisben előforduló feladatokkal összefüggésben függvénysorok konvergenciájának különböző típusait, többek között a pontonkénti-, a m.m., az egyenletes, a mértékben- és a normában való konvergenciát célszerű használni. A trigonometrikus Fourier-sorok konvergenciája attól is függ, hogy a sorba fejtett függvény milyen osztályhoz tartozik. Leggyakrabban a monoton, a folytonos, a folytonosan differenciálható, az integrálható vagy a négyzetesen integrálható függvényekből indulunk ki. Az egyik korai eredmény a trigonometrikus Fourier-sorok (TFS) pontonkénti konvergenciájáról *Dirichlet*-től származik: szakaszonként monoton függvények TFS-a minden pontban konvergens. Ugyanakkor a folytonos, 2π szerint periodikus függvények $C_{2\pi}$ osztályára ez az állítás nem igaz. *Du Bois Reymond* mutatott példát olyan $C_{2\pi}$-beli függvényre, amelynek TFS-a egy pontban divergál. Ezzel összefüggésben *Hilbert* felvetette, létezik-e olyan ortonormált rendszer, hogy bármely folytonos függvénynek az e szerint vett Fourier sora mindenütt konvergens. *Haar Alfréd,* aki ekkor Hilbert tanársegéde volt, 1910-ben doktori disszertációjában pozitív választ adott a kérdésre: olyan (azóta róla elnevezett) $h_n : \mathbb{I} \to \mathbb{R}$ $(n \in \mathbb{N})$ ortonormált rendszert (ONR-t) szerkesztett (8. kép), amelyre minden $f \in C[0,1]$ folytonos függvényre az

$$S^H f \;\sim\; \sum_{n\in\mathbb{N}} \langle f, h_n\rangle\, h_n(x) \quad (f \in L^1 := L^1(\mathbb{I}), x \in \mathbb{I})$$

Haar–Fourier-sor (HFS) egyenletesen konvergens az $\mathbb{I}$ intervallumon.



8. kép. Haar Alfréd (1885–1933) és a Haar-rendszer

A Haar-rendszer függvényei a

$$h(x) := \begin{cases} 1 & (0 \le x < 1/2), \\ -1 & (1/2 \le x < 1), \\ 0 & (x > 1) \end{cases}$$

függvényből transzlációval és dilatációval származtathatók:

$$h_{2^n+k}(x) := 2^{n/2} h(2^n x - k) \quad (0 \le k < 2^n, n \in \mathbb{N}, x \in \mathbb{I}).$$

A Haar-függvényeket célszerű tartójukkal indexelni. Vezessük be a diadikus intervallumok

$$\mathcal{I} := \{I = [k2^{-n}, (k+1)2^{-n}) : 0 \le k < 2^n, n \in \mathbb{N}\} \cup \{\emptyset\}$$

osztályát és legyen

$$h_I := h_{2^n+k} \quad (I = [k2^{-n}, (k+1)2^{-n}), 0 \le k < 2^n, n \in \mathbb{N}), \quad h_\emptyset = h_0.$$

Ezzel összhangban az Haar–Fourier-együtthatókra és az integrálközepekre a

$$\widehat{f}_I = \langle f, h_I \rangle \quad (I \in \mathcal{I}),$$
$$(E_n f)(x) := f_I := \frac{1}{|I|} \int_I f(t)\,\mathrm{d}t \quad (x \in I, I \in \mathcal{I}, |I| = 2^{-n})$$

jelölést fogjuk használni.

Az $\mathcal{I}$ a $\subseteq$ tartalmazási relációval bináris fa gráfot alkot, következésképpen az $(\widehat{f}_I, I \in \mathcal{I})$ Haar–Fourier-együtthatókat és az $(f_I, I \in \mathcal{I})$ átlagokat a bináris fán értelmezett függvénynek célszerű tekinteni. A Haar–Fourier-sorok jó konvergencia tulajdonsága abból következik, hogy részletösszegeik kifejezhetők integrálközepekkel, nevezetesen

$$S_{2^n}^H f = E_n f \quad (f \in L^1, n \in \mathbb{N}). \tag{4}$$

Ebből már egyszerűen következik, hogy bármely $f \in L^p := L^p(\mathbb{I})$ $(1 \leq p < \infty)$ függvényre a Haar–Fourier-sor L^p-normában tart a függvényhez. Ha f folytonos, akkor a konvergencia egyenletes. Ez más szóval azt jelenti, hogy a Haar-rendszer bázis a szóban forgó terekben.

A (4) egyenlőségre valószínűségelméleti interpretáció adható. Jelölje $\mathcal{A}_n := \sigma\{I \in \mathcal{I} : |I| = 2^{-n}\}$ a 2^{-n} hosszúságú diadikus intervallumok által generált σ-algebrát. Az $(\mathcal{A}_n, n \in \mathbb{N})$ sorozatot diadikus sztochasztikus bázisnak nevezzük. A (4) egyenlőség martingálelméleti terminológiát használva azt jelenti, hogy az $(S_{2^n}^H f, n \in \mathbb{N})$ (diadikus) martingált alkot erre a sztochasztikus bázisra nézve.

A Haar- és Walsh-rendszer szoros kapcsolatban áll egymással. Egyszerűen igazolható, hogy a két rendszer szerinti Fourier-sorok 2^n indexű részletösszegei egyenlők: $S_{2^n}^H f = S_{2^n}^W f$ $(f \in L^1, n \in \mathbb{N})$. Paley a már idézett dolgozatában bevezette a

$$Qf := \left(\sum_{n \in \mathbb{N}} |S_{2^n}^W f - S_{2^{n-1}}^W f|^2 \right)^{1/2} \qquad (f \in L^1, S_{2^{-1}}^W f := 0)$$

kvadratikus variációt, és bebizonyította, hogy az f és Qf függvények L^p-normái ekvivalensek $(\|f\|_p \sim \|Qf\|_p)$, azaz léteznek olyan $0 < c_1 < c_2 < \infty$ konstansok, hogy minden $f \in L^p$ függvényre

$$c_1 \|f\|_p \leq \|Qf\|_p \leq c_2 \|f\|_p \quad (1 < p < \infty). \tag{5}$$

Ebből következik, hogy $1 < p < \infty$ esetén L^p-beli függvények Haar–Fourier-sorának bármely átrendezése normában konvergens. Más szóhasználattal a Haar-rendszer ezekben a terekben feltétlen bázis. Ismeretes, hogy ez az állítás $p = 1$ esetben nem teljesül.

A múlt század közepén jelentősen nőtt az érdeklődés az első látásra mesterkéltnek tűnő rendszerek iránt. Kiderült, hogy a Haar-rendszer kitüntetett szerepet játszik a bázisok között. Megmutatták, hogy Banach-

terek fontos osztályai a következő tulajdonsággal rendelkeznek: ha ezekhez tartozó terekben a Haar-rendszer nem feltétlen bázis, akkor itt ilyen nem is létezik.

Az E_n operátorokat felhasználva értelmezhető a Hardy-terek diadikus analogonja. Az $f^*(x) := \sup_{n\in\mathbb{N}} |E_n f(x)|$ $(x \in \mathbb{I})$ függvényt az f diadikus maximál függvényének nevezzük. Ismeretes, hogy $\|f\|_p \sim \|f^*\|_p$ $(1 < p \le \infty)$ és $p = 1$ esetben $\|Qf\|_1 \sim \|f^*\|_1$. Innen következik, hogy ha az (5) állításban az f függvényt felcseréljük az f^*-ra, akkor $p = 1$ esetén is fennáll.

A

$$H^p = H^p(\mathbb{I}) := \left\{ f \in L^1 : \|f^*\|_p < \infty \right\} \quad (1 \le p \le \infty)$$

tereket diadikus Hardy-tereknek nevezzük. A fent mondottak alapján

$$H^p = L^p \quad (1 < p \le \infty), \quad H^1 \subset L^1$$

és H^1 az $\|f\|_H := \|f^*\|_1$ normával Banach-tér. A H^1 tér duálisa (a tér korlátos lineáris funkcionáljainak halmaza) a diadikus BMO-tér:

$$\mathrm{BMO} := \left\{ f \in L^1 : \|f\|_{\mathrm{BMO}} := \sup_{I\in\mathcal{I}} \frac{1}{|I|} \int_I |f(t) - f_I| \,\mathrm{d}t < \infty \right\}.$$

A Riesz–Fischer-tétel szerint (szeparábilis) Hilbert-terekben a tér elemeit jellemezhetjük a tér bármely $\phi_n \in \mathcal{H}$ $(n \in \mathbb{N})$ ortonormált bázisában vett Fourier-együtthatókkal. Részletesebben szólva az

$$f \mapsto \widehat{f}_\Phi = (\langle f, \phi_n \rangle, n \in \mathbb{N})$$

leképezés lineáris, normatartó bijekció a $\mathcal{H}$ és az ℓ^2 tér között. A

$$\|f\|_{\mathcal{H}} = \|\widehat{f}_\Phi\|_{\ell^2} = \Big(\sum_{n\in\mathbb{N}} |\langle f, \phi_n \rangle|^2 \Big)^{1/2}$$

Parseval-formula konkrét alkalmazásokban energiamegmaradásként interpretálható. Megjegyezzük, hogy az L^p terek elemei a trigonometrikus bázisban nem jellemezhetők hasonló formulával, ha $p \ne 2$. Ugyanakkor a $\widehat{f}_H = (\widehat{f}_I, I \in \mathcal{I})$ együtthatókkal, az $(a_I, I \in \mathcal{I})$ alakú sorozatok terében (5) alapján bevezetve a diadikus H^p és BMO normákat, már jellemezhetjük a szóban forgó tereket. Megjegyezzük, hogy

a klasszikus Hardy-térre a Franklin- és (gondolom) a wavelet Fourier-együtthatókkal hasonló jellemzés adható. A diadikus modell kiindulópontja lehet a gráfokon művelt harmonikus analízisnek. Ezzel kapcsolatban az [4] összeállításra utalunk.

A Haar-rendszer tulajdonságai számos fogalomnak és tételnek lettek a kiinduló pontjai a martingálelméletben. Weisz professzor könyvei [11, 12] (9. kép) a klasszikus eredményeken túlmenően olyan martingálokkal is foglalkoznak, amelyek indexhalmazát a sík rácspontjai, illetve a bináris fa csúcsai alkotják. Az ezzel kapcsolatos eredmények zöme tanszékünk oktatóinak nevéhez fűződik. A könyv a harmonikus analízis mély problémáinak megoldásához nyújt módszereket, rámutatva a terület martingálelméleti hátterére.

9. kép. Weisz Ferenc két monográfiája

A jelfeldolgozás egyik alapvető módszere, hogy a jeleket valamely jeltérben vett bázis szerinti koordináta-sorozatokkal jellemezzük. Hilbert-térben ortonormált bázisokat vehetünk fel. Banach-terekben, ahol nincs skaláris szorzat, a következő fogalmat használjuk. Akkor mondjuk, hogy az elemek $e_n \in X$ $(n \in \mathbb{N})$ sorozata bázis az $(X, \|\cdot\|)$ Banach térben, ha minden $x \in X$ elem egyértelműen írható fel

$$x = \sum_{n \in \mathbb{N}} x_n e_n \tag{6}$$

alakban. Ilyenkor az $\widehat{x} = (x_n, n \in \mathbb{N})$ együttható sorozattal adhatjuk meg a tér elemeit. Nyilvánvaló, hogy minden Hilbert-térben a teljes

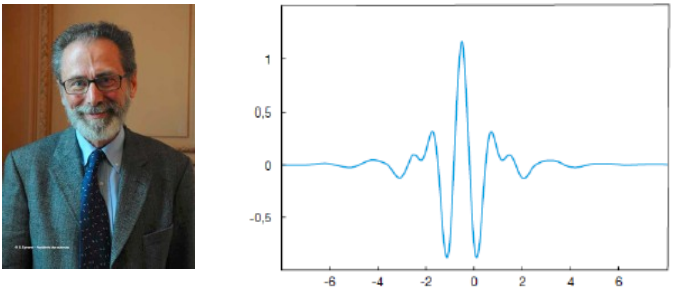
ortonormált rendszerek bázist alkotnak. Ha valamely Banach-térnek van bázisa, akkor véve a (6) alakú sorfejtések részletösszegeit racionális együtthatókkal, egy X-ben mindenütt sűrű, megszámlálható halmazt kapunk. Ez azt jelenti, hogy az ilyen terek szeparábilisek.

10. kép. S. Banach (1892–1945)

Banach a funkcionálanalízis alapjait lefektető híres könyvében (1932-ben) felvetette, hogy vajon minden szeparábilis Banach-térnek van-e bázisa. Ez az ún. *bázis probléma* sokáig a funkcionálanalízis egyik legfontosabb kérdése volt. A probléma mélységét az is mutatta, hogy sokáig nem volt ismert, hogy néhány fontos Banach-térnek (mint pl. a zárt diszken folytonos analitikus függvények maximum normával vett terének) van-e bázisa. Ez utóbbi kérdésre *Sz. V. Bockarjev* orosz matematikus adott pozitív választ 1974-ben, nevezetesen megmutatta, hogy a *Franklin-rendszerből* kiindulva ebben a térben bázis szerkeszthető. Az eredeti kérdésre *P. Enflo* svéd matematikus 1973-ban adott (sokak által nem várt) negatív választ: Létezik olyan szeparábilis Banach-tér, amelynek nincs bázisa. Konstrukciójában a Walsh-rendszer fontos szerepet játszik. Úgy gondoltuk, hogy könyvünkben helye van ennek a problémának. Az Enflo-féle konstrukcióban kapott Banach-teret (egy sztochasztikus bázis által indukált) BMO-típusú térrel helyettesítettük. Érdekes feladat lehet azoknak a sztochasztikus bázisoknak a jellemzése, amelyek ilyen terekhez vezetnek.

Mivel a Haar-függvények nem folytonosak, a HFS részletösszegeivel a sima függvények nem jól approximálhatók. Az 1980-as évektől kezdődően felvetődött a kérdés, hogy a Haar-féle konstrukcióban, a h függvényt sima függvénnyel (ún. anyawavelettel helyettesítve) szerkeszthetünk-e ortonormált rendszereket. Addig amíg a Haar-féle

konstrukció nagyon egyszerű, simasági feltételeknek eleget tevő waveletek konstrukció nehéz feladatnak bizonyult. A bonyolult konstrukciók ellenére napjainkban a waveleteken alapuló módszerek a jel- és képfeldolgozásnak egyik leggyakrabban használt eszközévé vállt. Az első (analitikus) wavelet konstrukció *Y. Meyer* nevéhez fűződik (11. kép). Ezen a területen kifejtett munkásságát Abel-díjjal jutalmazták. Az Érintő internetes folyóiratban írt dolgozatomban a kitüntetett munkásságának ismertetésén túlmenően bemutattam a témakör hazai vonatkozásait is [5].



11. kép. Y. Meyer és a Meyer-wavelet

A waveletek egy alapfüggvényből, az anyawaveletből affin argumentum-transzformációval származtathatók. Transzformációknak egy másik fontos típusa *Gábor Dénes* egy ún. ablakos Fourier-transzformált alkalmazásáról szóló dolgozatára vezethető vissza. Ezzel képzett függvényrendszereket napjainkban *Gábor-waveleteknek* nevezik. Ezek hátterében a Heisenberg-féle csoport áll. Néhány évvel ezelőtt ezekhez a konstrukciókhoz csatlakozva, az affin transzformációkat a hiperbolikus sík egybevágósági transzformációival helyettesítve bevezettük a hiperbolikus waveleteket. Ezzel kapcsolatos eredményeinkről az Alkalmazott Matematikai Lapokban megjelent dolgozatomban számoltam be [6].

Hivatkozások

[1] H.F. Harmuth, *Transmission of Information by Orthogonal Functions,* Springer-Verlag, Berlin, Heidelberg, 1969.

[2] F. Schipp F., W.R. Wade, P. Simon, *Walsh series: An introduction to dyadic harmonic analysis,* Akadémiai Kiadó, Budapest, Adam Hilger, Bristol and New York, 1990.

[3] Schipp F., Fouriertől a komputer tomográfig, *Tudományos Dialóg,* (1998), 59–65.

[4] Schipp F., Néhány gondolat a mély tanulásról.

[5] Schipp F., Waveletek – A 2017. évi Abel-díjjal kitüntetett Yves Meyer munkásságáról, *Érintő, Elektronikus Matematikai Lapok,* 6, (2017).

[6] Schipp F., Hiperbolikus waveletek, *Alkalmazott Matematikai Lapok,* 32, (2015), pp. 1–40.

[7] R.S. Stanković, J.T. Astola (Eds.), *Reminiscences of the Early Work in Walsh Functions: Interviews with Franz Pichler, William R. Wade, Ferenc Schipp,* Tampere International Center for Signal Processing, 2011.

[8] R.S. Stanković, P.L. Butzer, F. Schipp, W.R. Wade, *Dyadic Walsh Analysis from 1924 Onwards, Walsh–Gibbs–Butzer Dyadic Differentiation in Science Volume 1 Foundations,* Atlantic Press, 2015.

[9] W.R. Wade, Hit, zene, matematika (Interjú W.R. Wade professzorral), *Természet Világa,* 11, (1986).

[10] J.L. Walsh, *Selected Papers,* (Eds. T.J. Rivlin, E.B. Saff), Springer-Verlag, New York, Berlin, Heidelberg, (1999).

[11] F. Weisz, *Martingale Hardy Spaces and their Applications in Fourier Analysis,* Lecture Notes in Mathematics, vol. 1568, Springer-Verlag, New York, Berlin, Heidelberg, 1994.

[12] F. Weisz, *Summability of Multi-Dimensional Fourier Series and Hardy Spaces,* Mathematics and Its Applications, vol. 541, Springer-Science+Business Media, B.V., 2002.

Veszélydetektálás nyers 3D LiDAR adatok segítségével[‡]

Szabó Barbara Noémi[*]

Eötvös József Collegium[**]

szabo.barbara.noemi@gmail.com

Témavezető: Istenes Zoltán
ELTE IK Akadémiai-ipari Együttműködési Központ

1. Bevezetés az önvezető rendszerek világába

A robotok, autonóm járművek és különböző vezetést segítő rendszerek egyre nagyobb térnyerése következtében megnőtt az igény újabb és újabb módszerek, algoritmusok kialakítására, amelyek segítik a különböző szenzorokból nyert adatok valós idejű feldolgozását.

Olyan kérdések kerülnek előtérbe, amelyek az ego-jármű és a környezete mozgásának vizsgálatát tárgyalják. Kulcsfontosságú például a veszélyt jelentő objektumok meghatározása. Egy jármű számára a legnagyobb veszélyt az ütközés jelenti, hiszen a vezető nélküli rendszereket is érheti hatalmas anyagi kár. Egy baleset azonban nem csak ránk nézve jelent kockázatot. Például egy gyalogos testi épségének megőrzése önmagában is alapvető elvárás a vezetővel rendelkező járművek számára is, de az autonóm rendszerek esetében még inkább lényeges, hiszen az

[‡] A szerző 2023. júniusi Kari TDK Konferenciára benyújtott dolgozatának rövidített, átdolgozott változata.

[*] ELTE Informatikai Kar

[**] 2020–

embereknek az ilyen rendszerekbe vetett bizalma nem áll stabil lábakon, így nem szabad hagyni, hogy olyan incidens történhessen, amitől még inkább meginogna benne a hitük.

Mivel a biztonságos közlekedés az alapja ezeknek a járműveknek, így az ütközést elkerülő rendszerek fejlesztése egy kifejezetten aktuális és hangsúlyos feladat.

Ahhoz, hogy képesek legyünk a veszély észlelésére, azonosítanunk kell a környezetünkben a dinamikus, mozgó objektumokat, és meg kell figyelnünk ezek pályáját, valamint a statikus objektumokat, hiszen ezekkel is könnyen összeütközhetünk az ego-jármű mozgásából kifolyólag. Az idők során egyre olcsóbbá és így népszerűbbé, elterjedtebbé vált a 3D LiDAR alkalmazása, ami az önvezető járművek szerves részét képezi.

1.1. LiDAR

A Light Detection and Ranging, röviden LiDAR, egy speciális érzékelési technológia, amely kiemelt szerepet játszik a távolságmérésben és az önvezető járművek, robotok orientációjában. A rendszer lényege a lézerfény használata és annak visszaverődésének érzékelése a környezetben. Az eszköz egy lézersugarat bocsát ki, amely a környezetben található objektumokra ütközve visszaverődik. Az eszköz időmérési elve alapján képes pontosan meghatározni az objektumok távolságát, és a visszaverődési idő alapján térképet készít a környezetről.

Ennek segítségével információval rendelkezünk a minket körülvevő tárgyak helyzetéről egy 3D-s pontfelhő képében. Másodpercenként átlagosan 10 ilyen ponthalmaz áll a rendelkezésünkre, hogy az ezek közötti változások vizsgálatával adatokat nyerhessünk ki a saját és a környező objektumok alakjával és mozgásával kapcsolatban.

A mérések során egy Velodyne VLP-16 típusú 3D LiDAR-t volt segítségemre, amely dokumentációja [1] forrásban érhető el. A szenzor 16 függőleges irány mentén végez méréseket, 600 RPM forgási sebességgel, amely során 0.2 fokonként lövi ki a lézersugarakat. A LiDAR használható egyszeres és többszörös visszatérési módban is, a módszerhez a legközelebbi veszély érzékeléséhez az egyszeres visszatérés elegendő információt nyújt.

1.2. Jelölésrendszer

Jelölje $s_t^{i,j}$ a t-edik időpillanatban az i-edik ($i \in 0, 1, ..., 15$) sugár j-edik ($j \in 0, 1, ..., n_t$) mérését a 360°-os körön, n_t pedig a t-edik időpillanatban a mérések számát egy sugár mentén (ROS keretrendszerrel rögzített adatoknál a legtöbb esetben $\frac{28\,800}{16} = 1\,800$).

2. Mi is az a veszély?

A veszélyt azok a pontok jelentik, amelyek közelednek felénk. Ha a LiDAR egy helyben áll, akkor csak azok a pontok kerülhettek hozzánk közelebb két egymást követő mérés során, amelyek olyan objektumról verődtek vissza, amelyik közeledő mozgást végzett. A pontokat azonban nem tudjuk megfelelő biztossággal megfeleltetni egymásnak két pontfelhő között. Ugyanis egy, a térben lévő és mozgó tárgy egy pontja a LiDAR különböző lézersugarait különböző távolságokban fogja visszaverni, így egy mozgó tárgypontot kapunk.

A bemutatott módszer nem azonosít vagy feleltet meg egymásnak pontokat, mindegy, hogy egy pont merre, hogyan ment, melyik sugárirányban metszette a lézersugarat, a lényeg, hogy adott irányból volt-e közeledés, vagy sem. Lehet, sőt valószínű, hogy egy lézersugarat más és más fizikai tárgy, pont fog visszaverni.

Az elmozdulás vizsgálata emiatt úgy történik, mintha a középpontból indítanánk minden irányba egy sugarat, és a két különböző pontfelhőben vizsgáljuk a távolságát (r_1, r_2) az első pontnak, ami abban az irányban (a sugár mentén) helyezkedik el.

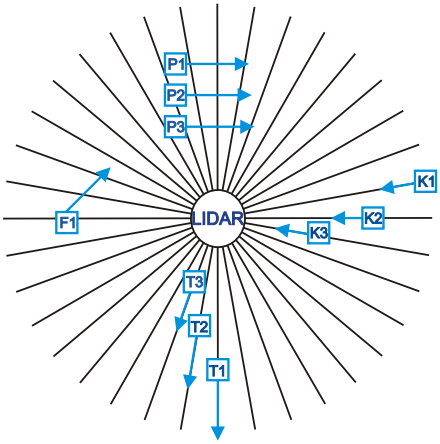
A pontfelhő felépítéséből kifolyólag az origóban a LiDAR található. Tehát, ha az első időpillanatban mért r_1 távolság nagyobb, mint a következő időpillanatban kapott r_2, akkor annak a sugárnak a mentén közeledés történt. Ha pedig r_2 értéke nagyobb, mint r_1-é, akkor távolodásra következtethetünk. Szóval az $r_1 - r_2$ különbség egy előjeles érték az elmozdulás mértékére, amit összevetve a többi sugáron mért elmozdulással egy heat mapet – hőtérképet (3. ábra) – készíthetünk, mivel azok az elemek, amelyek a nagyobb sebességgel közelednek felénk, veszélyesebbek, más szóval ugyanakkora időegység alatt nagyobb távolságot tettek meg az irányunkba, azaz $r_1 - r_2$ értéke nagyobb.

Ennek a megvalósítása a gyakorlatban a szenzor tökéletlenségéből kifolyólag kicsit másképp zajlik. Ténylegesen minden irányban nem tu-

dunk elmozdulást vizsgálni, de még ha tudnánk is, felesleges lenne, mivel a LiDAR véges sok mérést végez, így a térnek csupán véges sok (ROS felvétel esetén körülbelül 28 800) pontjáról rendelkezünk információval. Ezenkívül nem feltétlen – sőt nagy valószínűséggel nem – pontosan ugyanazokban az irányokban történt a mérés a két pontfelhőben, így ha egy sugár irányában az első időpillanatban találunk is pontot, az még nem feltétlenül jelenti azt, hogy a következő időpillanatban készült pontfelhőben is rendelkezünk majd információval ugyanazon egyenes mentén.

A legegyszerűbb tehát az, hogyha magukra a pontokra úgy gondolunk, mint a lézersugár, amely a LiDAR-ból kiindult, majd onnan visszaverődött. Ennek a segítségével egy pontfelhő minden sugarához keressük a hozzá legközelebbit a másik pontfelhőből.

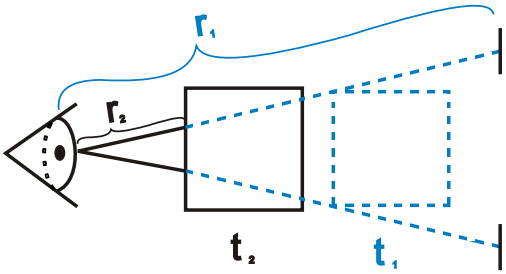
2.1. Eltérő mozgásirányok



1. ábra. Különböző mozgásirányok egy LiDAR mérési sugarainak a kontextusában. T: távolodó, K: közeledő, P: párhuzamos, F: ferde irányú mozgás. Forrás: saját ábra

A sugaras módszer jellegéből adódóan a különböző irányú mozgásokat különböző intenzitással és formával detektálhatjuk. Az 1. ábrán

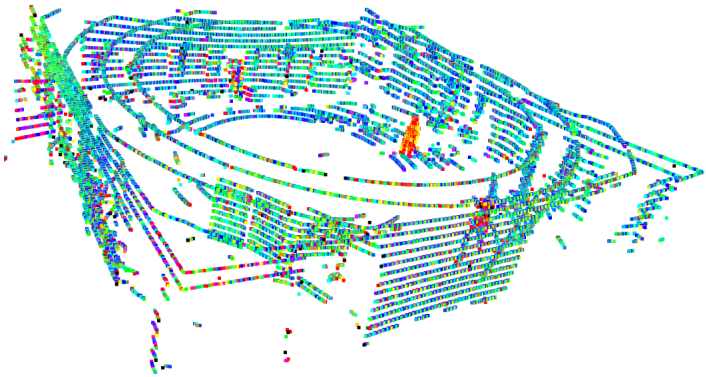
látható 4 típusú mozgásirányt különböztetjük meg. A problémát az jelenti, hogy mivel a valóságban nem pontszerű objektumok mozgását szeretnénk vizsgálni, így a sugarak mentén nem feltétlen kaphatunk helyes értéket a mozgás intenzitásáról egyszerre csupán egy sugár mentén vizsgálva az elmozdulást.



2. ábra. Közeledő test, ahogy kitakarja a hátteret olyan mérési sugár irányában, ahol korábban nem tette. Korábbi időpillanat: kék, szaggatott, későbbi: fekete jelzés. Forrás: saját ábra

Például a 2. ábra a közeledő mozgást szemlélteti. A mérések vizsgálata során nagy valószínűséggel találkozhatunk olyan sugarakkal, amelyek az első időpillanatban elhaladnak a test mellett, és a háttér valamilyen eleméről visszaverődve, annak a távolságáról nyújtanak adatot (r_1), míg – közel – ugyanabba az irányba, a második időpillanatban indított sugár már eltalálja a közelebb került objektumot, és r_2 már a test távolságát jelöli. Ekkor az $r_1 - r_2$ érték, habár helyesen jelzi, hogy valamilyen mozgás történt, mégsem a test mozgásának a sebességéről ad információt, nem úgy, mint egy, az objektum közepe felé irányuló lézersugár. Ez a gyakorlatban azt jelenti, hogy a LiDAR adatokon vizsgált mozgások esetében a testek széle, mintha gyorsabban közeledne a belsejénél, ez figyelhető meg a 3. ábrán látható hőtérkép elszíneződésében is, ahol az ember szélén lévő pontok világosabbak, sárgásak. Távolodó tárgyak esetében ennek a fordítottja játszódik le, tehát a távolodó test mögül hirtelen felbukkanó háttérelemeket fogja úgy érzékelni, mintha hirtelen eltávolodtak volna.

Párhuzamos mozgás esetén ugyanez úgy jelenik meg, hogy az objektum mozgásirány szerinti eleje hirtelen kitakarja a hátteret, így olyan,



3. ábra. Hőtérkép. A statikus pontok színe zöldes-kékes, és minél nagyobb egy pont elmozdulása egy korábbi pontfelhőhöz képest pozitív irányba, tehát minél gyorsabb, annál pirosabb. Forrás: saját ábra

mintha gyorsan közeledne, a hátulja mögül felbukkanó elemek pedig hirtelen távolodást fognak mutatni, annak ellenére, hogy nem feltétlenül került hozzánk közelebb a test.

A ferde mozgások az előző két eset kombinációjából kerülnek ki.

Ezeket a jelenségeket különböző módszerekkel lehetne kezelni. Egy lehetséges módszer a pontok csoportosítása valamilyen klaszterező algoritmus segítségével – lehetőleg már az elmozdulásértékek figyelembevételével –, majd egy ponthalmazra kiszámolni az elmozdulás mértékét, akár valamilyen statisztikai jellemző, (például medián, módusz) segítségével, akár az objektum középpontjának elmozdulásértékét tulajdonítva a teljes test mozgásának. Ezek a megoldások azonban rengeteget lassítanának a módszeren, és különben sem biztos, hogy jó lenne teljesen kiszűrni azokat a mozgásokat, amelyek nem felénk irányulnak. (Ez történne például az oldalirányú mozgásokkal, hiszen a test pontjaihoz tartozó elmozdulási értékeknek például a módusza nagy valószínűséggel 0 lenne, kivéve az egészen keskeny objektumok esetén.) Ezért inkább egy simítást szeretnénk végrehajtani a kiugró értékeken.

3. A feldolgozás lépései

3.1. LiDAR-ból érkező adatok beolvasása

Mivel a gyorsaság kulcsfontosságú, ezért fontos a nyers, szenzorból érkező adatokkal dolgozni. A bináris adatokat `pcap` fájlformátumból olvashatjuk be, és alakíthatjuk át hexadecimális számokká, amelyeket később decimális értékekké tudunk konvertálni.

3.2. Azimut (elfordulás), távolság értékek kinyerése

A térbeli pontokat a három (r, θ, φ) gömbi koordináta és egy, a visszaverődés intenzitását jelző érték reprezentálja.

A LiDAR-ból nyert nyers adatcsomagok alapvetően a következő információkat tartalmazzák [1, 9. fejezet]:

- Időbélyeg: az idő, amikor a mérés történt;
- Azimut: a horizontális szög, ahol a lézersugarat kibocsátotta (φ);
- Távolságérték: az objektum távolsága a LiDAR-tól, amiről a sugár visszaverődött, minden sugárcsatornához;
- Intenzitás: a visszavert jel intenzitása.

Ezeket az adatokat bináris formában tárolja, ahol minden egyes információt (minden egyes pontot) egy byte-sorozat reprezentál.

VLP-16-os LiDAR esetén mindez abban a formában valósul meg, hogy 12 blokkból álló sorozatokban érkezik az információ, amely a mérési adatokon kívül tartalmaz egy UDP fejlécet, időbélyeget és egy factory részt, amely a LiDAR típusáról és a visszaverődés mérésének egyéb módjáról nyújt információt. Egy blokkban megtalálható a blokk kezdetét jelző („`ff ee`") flag, az azimut értéke és 2 kibocsátási szekvencia távolságértékei, a 16 sugár mindig ugyanabban, a dokumentációban leírt sorrendben követik egymást [1, 54–55. oldal].

A sugarak pozicionális struktúrája jelentősen megkönnyíti és felgyorsítja a sugarak azonosítását, így csupán az azimutértékek vizsgálatával beazonosítható egyszerre 32 sugárhoz is a hozzájuk legközelebbi egy másik pontfelhőben.

A blokkok felépítéséről és a byte-sorozat dekódolásáról a dokumentációban [1] találhatunk bővebb leírást.

Az azonosítás során a 360°-ot rekeszekre osztjuk fel, az azonos rekeszbe kerülő sugarak távolságértékeit pedig átlagoljuk.

Így, amikor egy s_1 sugárhoz keressük azt, hogy az előző mérési körből, melyik helyezkedett el hozzá a legközelebb, akkor csupán a megfelelő rekeszben található távolságértékekre van szükség. A kérdéses rekesz j indexe pedig könnyen kiszámolható a következő képlettel:

$$j = \left\lfloor \frac{\varphi_1}{\frac{360}{n_{pac}}} \right\rfloor$$

ahol, n_{pac} a rekeszek számát jelöli. A rekeszek száma a módszer egyik paramétere.

3.3. Elmozdulások kiszámítása a pontfelhőre

Ha beazonosítottuk az $s_{t_2}^{i,k}$ sugárhoz a megfelelő j indexet, akkor még figyelembe kell venni, hogy a sugarak esetében, ha túl távol vannak egymástól, azaz a két szög különbsége túl nagy, akkor könnyen elképzelhető, hogy a két mérési pont túl messze helyezkedik el egymástól, emiatt nem túl valószínű, hogy ugyanarról az objektumról verődtek volna vissza. A LiDAR-tól távolodva pedig ez egyre nagyobb eltérést jelent. Ez 0.5° különbségnél már 40 méter távolságban is – ami még a ROI (region of interest), magyarul a releváns tartományba tartozik – igen jelentős, fél méteres távolságot jelent.

A sugarak beazonosítása esetén ennek a vizsgálata utólag történne, a rekeszek bevezetésével azonban ez a probléma szinte magától megoldódik, csupán arra kell figyelni, hogy n_{pac}, azaz a rekeszszám megválasztásakor a rekeszek hossza ne legyen túl nagy, azaz ne legyen túl kicsi a rekeszszám. Így két egymástól túl távoli sugár értékei nem eshetnek ugyanabba a rekeszbe. Kevés rekesznél a sugárra merőleges mozgásokat nem feltétlen lehet azonosítani, csak a sugárirányút.

3.4. Time To Collision

Az eredeti problémafelvetés a veszély detektálása, tehát az ütközés előrejelzése volt. Nyilvánvalóan erre a kérdésre nem adhatunk száz százalékig biztos megoldást, hiszen például egy, a közelünkben közlekedő gyalogos bármikor irányt változtathat, ha úgy tartja kedve, megfordulhat, vagy elénk ugorhat hirtelen. A TTC (Time To Collision [3]) érték

azt mutatja meg, hogy várhatóan mennyi idő múlva fog ütközés bekövetkezni, ha az objektumok az eddigi pályán és sebességgel haladnak tovább, ami egy elég jó becslést nyújt a baleset várható idejére.

Ennek a kiszámítása a következő képlet szerint történik:

$$\mathrm{TTC} = \frac{r}{\frac{\Delta r}{\Delta t}}$$

Tehát az objektum távolságát el kell osztani a pillanatnyi sebességével, ami az előzőekben megkapott, részletesen kifejtett elmozdulási adatok osztva a két pontfelhő rögzítése között eltelt idővel, ami default beállítások esetén 0.1 másodperc.

Mivel a távolságértékek nem kizárólag két pontfelhőben fellelhető különbségekből kerülnek ki, hanem egy időablak vizsgálatával, így a módszer nem pontosan konstans sebességgel tekint a pontok mozgására.

Mindezek után veszélyesnek jelöljük azokat a pontokat, amelyeknél ez a TTC érték egy kritikusnak jelölt tartományba esik. A mérések pontatlansága és az átlagolás következtében ugyanis a statikus objektumoknak sem lesz 0 az elmozdulása, így, habár az ő TTC értékük lényegesen nagyobb, mégsem biztos, hogy egy végtelen közeli szám.

3.5. Tér- és időbeli simítás

Gyakorlati példák vizsgálata során derült ki, hogy az, hogy nem minden mérési kör pontosan 0 foknál kezdődik, és lépésközök sem feltétlen egyeznek meg, egy olyan problémához vezet, hogy a tárgyaknak a széléhez legközelebbi sugár vagy eltalálja éppen a testet vagy nem. Emiatt a távolságértékek folyamatosan ugrálnak ezeken a területeken a statikus objektumok esetén is, amik a különbségek vizsgálatakor kiugró értékekként jelennek meg.

A rekeszeken belüli távolságok átlagolása már önmagában egyfajta horizontális simítást eredményez. A vertikális simításhoz, az ugyanakkora azimut (φ elfordulás) szögeknél lévő, (tehát az egyforma rekeszindexekkel rendelkező) sugarakhoz tartozó távolságértékeknek vesszük a súlyozott átlagát, olyan módon, hogy a vízszintes szöghöz közeli sugarak kapják a legnagyobb súlyokat, ettől távolodva pedig egyre kisebbeket. Negatív irányba (lefelé) lassabban csökken a súlyok értéke, mint felfelé, ugyanis a testek általában a földfelszínen helyezkednek el. Ezáltal meg tudjuk határozni, hogy a veszélyforrás melyik irányból közeledik. Az

algoritmus enélkül is pontos eredményeket tud nyújtani, hogy itt miért döntöttem mégis mellette, arról a 4-os számú fejezetben lesz bővebben szó.

De a még pontosabb eredmények elérése érdekében javasolt egy időablak használata. Ez azért fontos, mert a kiugró értékek – attól függően, hogy a háttér milyen távol helyezkedik el mögöttük – igen nagyok is lehetnek, és az előfordulásuk sem ritka. Ezeknek az úgymond eltüntetéséhez, de legalábbis csillapításához egy elég nagy időablakot kell vizsgálni, amin belül szintén átlagoljuk a távolságértékeket minden rekeszre.

Az időablaknál nem számít, hogy az átlagolt távolságok különbségét, vagy a különbségek átlagát vesszük-e.

4. Eredmények kiértékelése

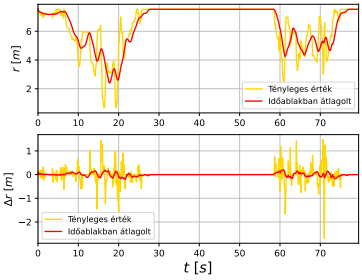
Az alábbi fejezetben a zaj mértékének változását figyelhetjük meg a rekeszek számának változtatásával, vertikális átlagolással és az időablak alkalmazásával. Az eredmények saját mérési adatokon alapulnak, ahol a LiDAR előtt különböző számú, sebességű és irányú testek mozgását vizsgáltam, VELODYNE VLP-16 LiDAR-ral az ELTE épületében.

A különböző simításokra, átlagolásokra azért van szükség, mert a mérési értékek gyakran zajosak lehetnek a LiDAR pontatlanságából kifolyólag. Ezek a kiugró értékek jól láthatóak, és nem lehet őket figyelmen kívül hagyni. A 4. ábra segítségével két egymás melletti rekesz értékeit hasonlíthatjuk össze.

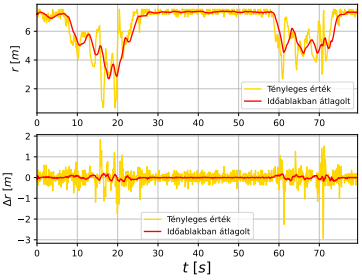
A sárga tényleges értékek nagyon különböznek egymástól, azonban a végső, piros görbe sokkal jobban tükrözi a valóságot a zajos képen is, hiszen szinte teljesen ugyanazt mutatja, mint a mellette lévő rekesz képe, időben alig észrevehetően elcsúsztatva.

4.1. Rekeszszám

Az 5. ábrán a rekeszszám azért ilyen nagy (1080), hogy meg tudjuk vizsgálni a zaj mennyiségét egy körülbelül 1°-os körszeletben. Látszódik, hogy a kiugró értékek elsősorban 0°–0.5° között jellemzőek, de később (0.5°–1°) környékén is találkozhatunk vele, főleg a 30. és 60. másodperc között. Az is látható, hogy mivel egy rekesz hossza kicsi



(a) A rekesz: 32.05°–32.55°



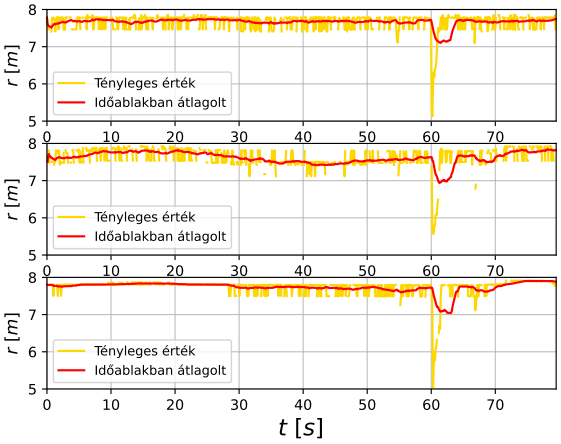
(b) A rekesz: 32.55°–33.05°

4. ábra. Felső diagramok: távolságértékek. Alsó diagramok: egymást követő távolságértékek különbségei. Sárgával a megjelenő rekesz értékei a 16 függőleges sugár mentén vannak csak átlagolva, pirossal ezenkívül az időablakon belül is átlagolt értékek jelennek meg. Időablak: 20, a rekesz intervalluma ábránként különböző, a rekeszek mérete: 0.5° (rekeszszám: 720). Forrás: saját ábra

(0.33°), így sok olyan van, amelyikben nem található érték, ezért ilyen foghíjas néhol a sárga vonal.

Mivel a zaj mennyisége és nagysága 0° és 1° közötti tartományban, a 30–60 másodperces intervallumban közel azonos, így vizsgálhatjuk, hogy ezen a szakaszon hogyan változik a nagysága a rekeszszám változtatásával. A rekeszek bevezetését, tehát az egymáshoz közeli sugárértékek átlagolását az a feltételezés indokolja, hogy egymáshoz közel elhelyezkedő sugarak nagy valószínűséggel hasonlóan viselkednek. Tehát, ha egy sugár egy mozgó objektumról verődött vissza, akkor a mellette lévő sugár is nagyobb valószínűséggel fog olyan tulajdonságokat mutatni, amelyekből mozgásra következtethetünk (azaz a hossza változik az idő mentén), mint egy olyan sugár melletti másik sugár, amelyik hossza adott időn belül közel konstans.

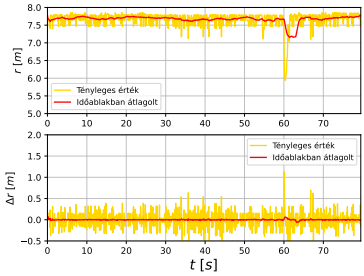
A 6. ábrán látható diagramok segítségével összehasonlíthatjuk a zaj nagyságát különböző rekesznagyságokra. A távolságértékeket mutató diagramon talán kevésbé látványosan jelenik meg, de ott is megfigyelhető még a 30–60. másodperces időintervallumban is, hogy a kiugró értékek nagysága kisebb. A távolságkülönbség értékeknél azonban jól látszik az eltérés, a zaj mennyisége szemmel láthatóan kisebb az 1°-os rekesz méretekkel rendelkező ábrán a 0.5°-oshoz képest.



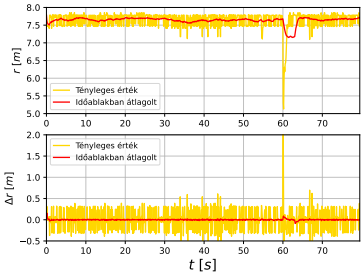
5. ábra. A diagramokon a távolságértékek jelennek meg. Sárgával a megjelenő rekesz értékei a 16 függőleges sugár mentén vannak csak átlagolva, pirossal ezen kívül az időablakon belül is átlagolt értékek jelennek meg. Időablak: 30, a rekesz intervalluma ábránként különböző, a rekeszek mérete: 0.33° (rekeszszám: 1080). A legfelső ábra rekeszének intervalluma 0°–0.33°, az alatta lévőé 0.33°–0.67°, az alsóé 0.67°–1°. Forrás: saját ábra

Ami még szépen leolvasható, hogy a 60. másodperc körüli kimagasló az érték, amely valamilyen mozgásra enged következtetni (valószínűleg párhuzamos elhaladásra), ami az átlagolás miatt csillapított értékeket vesz fel. Mivel csak az időben megelőző adatokat tudjuk átlagolni, így a görbe jobb oldala lesz csak lankásabb. Ez egyértelmű, hiszen a változás bekövetkezte előtt ezekből az adatokból nem jósolhatjuk meg előre a mozgást. Számunkra azonban még így is előnyös, hiszen ez azt jelenti, hogyha valamilyen irányból közeledést észleltünk, akkor az még az elkövetkezendő pillanatokban is fókuszban marad, tehát továbbra is fokozott figyelmet fordíthatunk rá, ami a valóságban egy jó gyakorlat, az emberi figyelem megoszlása is hasonlóan működik.

Amire viszont figyelni kell, hogy nem választhatunk túl kicsi rekeszszámot. Az 5. és 6. jelzésű ábrán látható diagramok beltéri felvételek eredményei, ezért lehetséges, hogy az átlagos távolságértékek 8 méter körül mozognak. Kültéren azonban ennél sokkal nagyobb távolságok



(a) A rekesz intervalluma: 0°–1°, mérete: 1° (rekeszszám: 360)



(b) A rekesz intervalluma: 0°–0.5°, mérete: 0.5° (rekeszszám: 720)

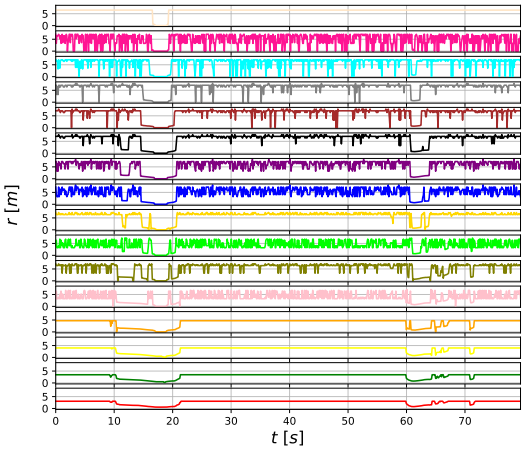
6. ábra. Felső diagramok: távolságértékek. Alsó diagramok: egymást követő távolságértékek különbségei. Sárgával a megjelenő rekesz értékei a 16 függőleges sugár mentén vannak csak átlagolva, pirossal ezenkívül az időablakon belül is átlagolt értékek jelennek meg. Időablak: 30, a rekeszek intervalluma és mérete ábránként különböző. Forrás: saját ábra

és sebességek a jellemzőek. Így a korábban kifejtett okok miatt ennek a paraméternek a beállításakor érdemes figyelembe venni a környezetet, valamint azt, hogy a szenzor pontatlansága milyen arányban nő a LiDAR-tól való eltávolodással, és hogy az alkalmazás során mekkora ROI-val dolgozunk.

4.2. Vertikális átlagolás

A 16 függőleges sugár menti átlagolás mellett azért döntöttem, mert néhány irányban folyamatosan zajos lehet bizonyos sugarak értéke. Például a 7. ábrán a 16 sugárból 10 értékein kisebb vagy nagyobb, de minden esetben szemmel látható zaj jelenik meg.

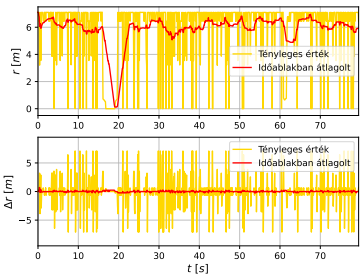
Az átlagolás eredményei jól láthatóak a 8. ábrán. Ha összehasonlítjuk egyetlen sugár képét a vertikális átlagolás utáni eredményekkel, akkor megfigyelhetjük, hogy a zaj mértéke számottevően csökken a 16 sugár értékei átlagolásának köszönhetően. A különbség nemcsak a tényleges, csupán a rekeszekre átlagolt (sárga) vonalon, de az időablak használatával kisimított piros görbén is észrevehető.



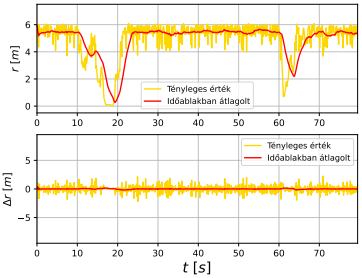
7. ábra. Egy rekesz távolságértékei, időablak menti átlagolás nélkül a 16 sugár mérései különböző színnel, külön aldiagramban jelennek meg. Az ábrán fentről lefelé megjelenő sugarak vízszintessel bezárt szöge a következő: 15°, 13°, 11°, 9°, 7°, 5°, 3°, 1°, −1°, −3°, −5°, −7°, −9°, −11°, 13°, −15°. A rekesz intervalluma: 21.03°–21.53°, a rekeszek mérete: 0.5° (rekeszszám: 720). Forrás: saját ábra

Azt is megfigyelhetjük, hogy habár van olyan mozgás, amelyik nem észlelhető mindegyik mentén, csupán 6 esetében, a 9. ábrán észrevehetjük, hogy mégsem tűnik el. A 7. ábrán olíva színnel jelölt sugáron 70 másodperc környékén látható egy kiugró érték, amiből mozgásra következtethetünk már csak azért is, mert ugyanakkor az alatta lévő másik 5 sugáron is észrevehetünk valami hasonló változást. A 9. ábrán ezt a sugarat vizsgálhatjuk közelebbről. A távolságértékeknél csak a sárga görbéket összehasonlítva azt vehetjük észre, hogy az elváltozás a 70. másodperc környékén még tisztábban is látszik, mint azt az egyetlenegy sugarat nézve.

Érdemes azonban megjegyezni, hogy ez a probléma csak statikus érzékelő esetén lép fel. Ugyanis a folyamatos kiugró értékeket az érzékelőnek az a tulajdonsága okozza, hogy nem mindig pontosan ugyanott (ugyanakkora szögnél) kezdi el a mérési kört és a két mérés közötti elfordulás mértéke is változhat. Emiatt bizonyos irányokban a legközelebbi sugár néha eltalálja a tárgy szélét, néha pedig nem.



(a) Sárgával a vízszintessel 11°-ot bezáró sugár rekeszértékei, pirossal ugyanezen sugár időablakon belül átlagolt értékei jelennek meg



(b) Sárgával a megjelenő rekesz értékei a 16 függőleges sugár mentén vannak csak átlagolva, pirossal ezenkívül az időablakon belül is átlagolt értékek jelennek meg

8. ábra. Felső diagramok: távolságértékek. Alsó diagramok: egymást követő távolságértékek különbségei. Időablak: 30, a rekesz intervalluma: 21.03°−21.53°, a rekeszek mérete: 0.5° (rekeszszám: 720). Forrás: saját ábra

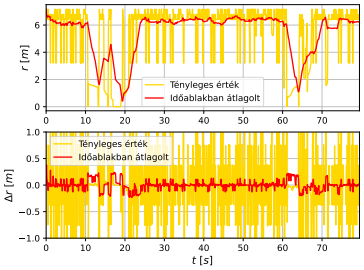
4.3. Időablak

A 10. ábrán látható, hogy az időablak változtatásával hogyan módosul a zaj és a részletesség mértéke. Az időablak növelésével az apró részletek természetesen elvesznek, mint például a 65. másodperc körül egy rövid közeledést követően ismét a távolodás folytatódik. Ez következhet mérési hibából vagy tényleges mozgásból is. Ilyen szintű részletességre általában azonban nincs is szükségünk. Megfigyelhető még, hogy a 10-es és 20-as időablakméretek adta eredmények közötti jelentős különbség már nem figyelhető meg a 20-as és 30-as időablakok értékei között. Ebből arra lehet következtetni, hogy a 30-as időablak nyújtotta simításokra már nincsen szükségünk.

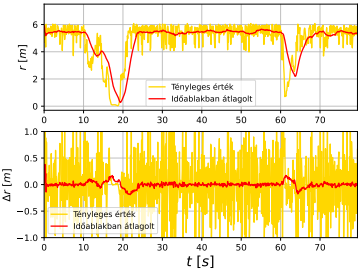
4.4. Limitációk

A módszer hátránya, hogy a nagyon lassú mozgásokat nem feltétlen tudja detektálni, vagy megfelelő súllyal jelezni.

Bonyodalmat jelenthet még a szenzor nyers adatainak sajátos felépítésének használata, ugyanis más szenzorra más csomagfelépítés lehet



(a) Sárgával a vízszintessel 5°-ot bezáró sugár rekeszértékei, pirossal ugyanezen sugáridőablakon belül átlagolt értékek jelennek meg



(b) Sárgával a megjelenő rekesz értékei a 16 függőleges sugár mentén vannak csak átlagolva, pirossal ezenkívül az időablakon belül is átlagolt értékek jelennek meg

9. ábra. Felső diagramok: távolságértékek. Alsó diagramok: egymást követő távolságértékek különbségei. Időablak: 30, a rekesz intervalluma: 21.03°−21.53°, a rekeszek mérete: 0.5° (rekeszszám: 720). Forrás: saját ábra

jellemző. OLEI LR-16F 3D LiDAR-nál például ugyanolyan a kommunikációs protokoll felépítése [2].

4.5. Összesített eredmények TTC ábrázolása

Az összesített eredmények szemléltetésére a 11. ábra szolgál. A diagramon a TTC értékek reciproka jelenik meg a szemléletes ábrázolás érdekében. A TTC értékek ugyanis végtelenhez közeli számok, amennyiben abban az irányban csupán statikus objektumokkal helyezkednek el, és minél veszélyesebb valami, annál inkább a 0-hoz közelít az ütközésig hátralévő idő. A reciprokát megjelenítve a statikus elemek 0 körül értéket vesznek fel, a gyorsan közelítő testek irányában egyre nagyobb pozitív, a távolodást pedig negatív értékek jelzik. Ennek megfelelően az (a) diagramon egy közeledést figyelhetünk meg 30° környékén, míg a (b)-n ugyanott távolodást láthatunk a (c)-n pedig a közeledésen kívül egy oldalirányú mozgás is megjelenik. A jellegzetes formából azt is megállapíthatjuk, hogy a mozgás iránya balról (0° felől) jobbra (360° felé) történik. Az értékek néhol még így is erősen zajjal terheltek, de a tényleges mozgások jól elkülönülnek tőle.

A 12. ábrán pedig egy mozgásban lévő jármű értékei láthatóak. Középen az látszódik, ahogy a jármű távolodik a faltól. Kétoldalt pedig azt lehet látni, ahogy a jármű elhalad különböző objektumok mellett, némelyik éppen feltűnik pár sugár mentén, némelyik pedig el.

5. Összegzés

Az eljárás a LiDAR felépítését és működését kihasználva, lézersugármodell segítségével becsüli meg az ütközés bekövetkezéséig hátralevő időt, ezzel nyújtva információt a környezetben fellelhető veszélyes zónákról. Nem alakítja át a beérkező adatokat tényleges pontfelhővé, csupán az elfordulási szögekből (azimutértékek) és a távolságadatokból képes meghatározni a szükséges információt. A számításokhoz TTC becslést alkalmaz a sugár hosszváltozásának (a sugár mentén történt változás, elmozdulás) mérésével.

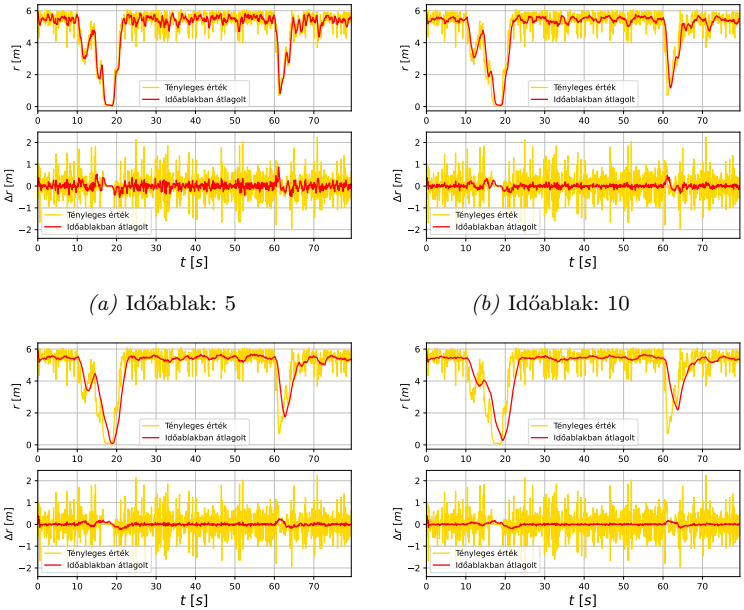
A módszernek egy egyszerű, mégis gyors megközelítést mutattam be a potenciális veszélyzónák valós idejű felismerésére, amely az önvezető technológia fejlődésével egyre fontosabbá válik. Ez a technika tehát képes az objektumok detektálása nélkül bármilyen környezetben kiszámíthatóan, ugyanolyan teljesítménnyel működni, így sokoldalú és praktikus megoldást jelent a valós alkalmazások számára.

Hivatkozások

[1] Velodyne, Inc., *VLP-16 User Manual*.
`https://velodynelidar.com/wp-content/uploads/2019/12/63-9243-Rev-E-VLP-16-User-Manual.pdf`

[2] OLEI, *OLEI LR-16F 3D LiDAR Sensor Communication Data Protocol User Manual*.
`https://device.report/manual/6825101`

[3] R. van der Horst, H., J. Hogema, Time-To-Collision and collision avoidance systems, *Proc. 6th ICTCT Workshop, Salzburg,* 1994.
`https://www.researchgate.net/publication/237807114_TIME-TO-COLLISION_AND_COLLISION_AVOIDANCE_SYSTEMS`



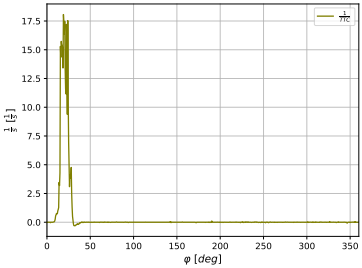
(a) Időablak: 5

(b) Időablak: 10

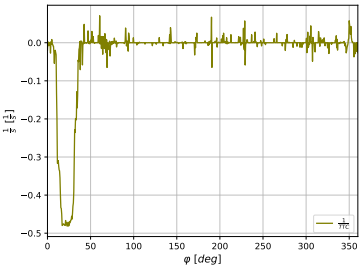
(c) Időablak: 20

(d) Időablak: 30

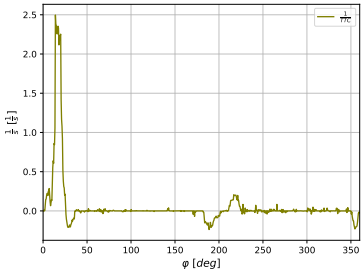
10. ábra. Felső diagramok: távolságértékek. Alsó diagramok: egymást követő távolságértékek különbségei. Sárgával a megjelenő rekesz értékei a 16 függőleges sugár mentén vannak csak átlagolva, pirossal ezenkívül az időablakon belül is átlagolt értékek jelennek meg. Az időablak ábránként különböző, a rekeszek mérete: 0.5° (rekeszszám: 720). A rekeszek intervalluma: 21.03°–21.53°. Forrás: saját ábra



(a) 181. időpillanat: közeledés

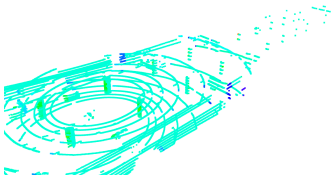


(b) 211. időpillanat: távolodás

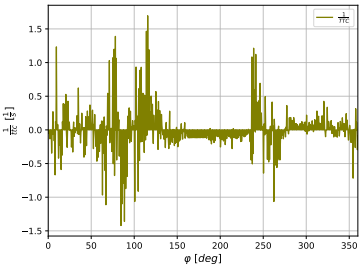


(c) 618. időpillanat: közeledés és oldalirányú elhaladás

11. ábra. A diagramokon a TTC értékek reciproka jelenik meg egy időpillanatban a 360°-os kör mentén. Forrás: saját ábra



(a) Az ábrán annak a pontfelhő hőtérképe látható, amely adatok alapján készült a (b) ábra diagramja



(b) Az diagramokon a TTC értékek reciproka jelenik meg egy időpillanatban a 360°-os kör mentén.

12. ábra. Mozgó jármű. Forrás: saját ábra

Neurális kódvisszafejtés másoló mechanizmussal[‡]

Szalay Gergő, Poór Máté Bálint[*]

Eötvös József Collegium[**]

gergo.szalay@gmail.com

Témavezetők: Pintér Balázs és Gregorics Tibor
ELTE IK Programozáselmélet és Szoftvertechnológiai Tanszék

A kódvisszafejtés (decompilation) célja, hogy egy gépközeli nyelvű kódrészletet magasabb absztrakciós szintű nyelvű kóddá alakítsunk, ami könnyebben értelmezhető emberek számára.

A hagyományos kódvisszafejtő eszközök tárgyszakértők által megállapított, kézzel írt szabályokon alapulnak, így fejlesztésük, karbantartásuk és bővítésük lassú, időigényes. Ezen problémák megoldására az utóbbi évtizedben több, mélytanuláson alapuló kezdeményezés is született, melyek assembly kódot próbáltak meg C programozási nyelven írt kódra visszaalakítani. Az általuk készített architektúrák nem voltak képesek maradéktalanul kezelni a literálokat és az azonosítókat, így különféle utólagos hibajavítási technikákat kellett alkalmazniuk.

Az utólagos hibajavítás elkerülésére előálltunk egy end-to-end, single-pass kódvisszafejtő eszközzel, mely a másoló mechanizmus (copying mechanism) felhasználásával alakítja vissza az alacsony absztrakciós szintű kódot. A másoló mechanizmus lehetővé teszi tokenek

[‡] A szerzők 2022. decemberi Kari TDK Konferenciára benyújtott dolgozatának rövidített, átdolgozott változata. A dolgozat a 36. OTDK-n II. helyezést kapott.

[*] ELTE Informatikai Kar

[**] Sz.G.: 2022–

(szavak) közvetlen másolását a bemeneti adatokból. A módszer alkalmazásával az utólagos hibajavítások elkerülhetők, hiszen már elsőre a megfelelő tokeneket tudja generálni a rendszer.

Az architektúránk hatékonyan alakította át a bemenetet idiomatikus, C programozási nyelven írt kódra a regiszterek imitálása és goto utasítások használata nélkül. Korábbi munkákhoz képest három új nyelvi elemmel bővítettük a modellünk által támogatott nyelvi elemek listáját. A modell teljesítményét egy 10 szintből álló, fokozatosan nehezedő szintrendszeren mértük. Emellett külön is megvizsgáltuk, hogy milyen hatékonyan tudja az egyes nyelvi elemeket visszafejteni. Az eredményeink bizonyítják, hogy utólagos hibajavítás nélkül is lehet készíteni neurális kódvisszafejtő rendszert, mely hatékonyan képes helyes eredményt adni.

1. Bevezetés

A kódvisszafejtés lényege, hogy alacsony absztrakciós szintű kódot alakítunk vissza magas absztrakciós szintű, ember számára is könnyen értelmezhető nyelven írt kódra úgy, hogy a két kód jelentése, működése megegyezzen. Ez a folyamat a fordítóprogram munkájának (fordítás) inverz folyamataként is elképzelhető. A kódvisszafejtésnek több, a való életben is jelentős felhasználási módja van. Legfontosabbak ezek közül a rosszindulatú szoftverek beazonosítása, sebezhetőségek felismerése, sérült vagy elvesztett programkód visszaszerzése.

A hagyományos (mélytanulási technikákat nem alkalmazó) kódvisszafejtő eszközöket tapasztalt tárgyszakértők fejlesztik. Ezen programok mintákat keresve törekednek a legnagyobb pontosság elérésére egy adott fordító esetén, melynek következménye rengeteg kézzel írt szabály alkalmazása. A sok szabály nagy mértékben lerontja a kódvisszafejtő alkalmazás implementációjának átláthatóságát, karbantarthatóságát és bővíthetőségét, emiatt fejlesztésük rendkívül költséges és lassú. Néhány a legfontosabb hagyományos kódvisszafejtő rendszerek közül: IDA Pro[1], RetDec[2], Radare2[3]. Ezek a kódvisszafejtő szoftverek bizonyítottan jól teljesítenek [9], viszont általános hiányosságaik is vannak:

[1] `https://hex-rays.com/ida-pro/` (2022)
[2] `https://retdec.com/` (2022)
[3] `https://rada.re/n/` (2022)

- Csak azok a kódsémák alakíthatóak vissza, melyekhez visszaalakító szabályok készültek.
- A generált kód gyakran nem idiomatikus, az alacsony szintű kódban előforduló részletek jelennek meg benne, mint például regiszterek és ugrások használata.
- A hagyományos kódvisszafejtők általában csak egy-egy fordítóprogramhoz (azoknak is gyakran csak bizonyos verzióihoz) lettek elkészítve, így nem képesek más fordítóprogramokkal vagy verziókkal dolgozni, mert ezekben az esetekben gyakran hibás kimenetet adnak vissza.

Ezen limitációk kiküszöbölésére a kutatók egy csoportja a mélytanuláshoz és az általa nyújtott lehetőségekhez fordult.

1.1. Neurális kódvisszafejtés

A mély neurális hálók (seq2seq) nagyon hatékonynak bizonyultak a gépi fordítás [13] terén. A kódvisszafejtés két nyelv között történő fordításnak tekinthető, így jogosan vetődik fel az a gondolat, hogy az eddig megalkotott gépi fordító eszközöket alkalmazzuk a problémán. A leglényegesebb különbség a kódvisszafejtés és a gépi fordítás hagyományos alkalmazásai között, hogy a kódvisszafejtés esetében a nyelvek formálisak, szemben a gépi fordításban általában alkalmazott természetes nyelvekkel. A formális nyelvek kötöttsége, szabályrendszere bizonyos szempontból könnyebbé is teheti a fordítást, hiszen a rendszernek elegendő a szabályokat megtanulnia. Cserébe viszont egy kis hiba is teljesen elronthatja az utána következő részeket a kimenetben. A modell által vétett hibákat a kutatók eddig különféle utólagos hibajavítási technikák alkalmazásával próbálták orvosolni.

A legkomolyabb kihívást az ezen a területen végzett kutatások esetében az jelentette, hogy megfelelően prediktálják az azonosítókat és a literálokat. Mivel a gépi kódban lévő számok tetszőlegesen változatosak lehetnek, viszont a legtöbbnek nagyon kicsi az előfordulási gyakorisága, így egy egyszerű seq2seq architektúrának esélytelen minden bemenetben megjelenő számliterálhoz a vele megegyező kimeneti számliterált megtanulnia.

A hagyományos gépi fordítási feladatot ellátó modellek, mint az RNN-eket alkalmazó seq2seq mélytanulási architektúra, nem képesek

pontos predikcióra a fenti esetekben. D. S. Katz et al. [5] voltak az elsők, akik megpróbálták megoldani a neurális kódvisszafejtés problémáját mélytanulás segítségével. Az ő munkájukat tekintjük alapozó cikknek. Ők egy egyszerű seq2seq hálót [10] használtak. Ezzel az architektúrával képesek voltak viszonylag alacsony Levenshtein-távolságot (a hálózat hibájának egy mérőszáma) elérni, de a helyesen visszafordított programok aránya meglehetősen alacsony maradt. Ők csak strukturális hibákat javítottak utólagosan, így a hibásan prediktált változók és literálok nem lettek korrigálva. Az ő munkájuk tekinthető az első eredménynek a területen.

O. Katz et al. [6] javítottak az alapozó cikk eredményein a saját keretrendszerükkel, melynek neve TraFix. Nyolc különböző nehézségi szintet definiáltak a modell hatásfokának mérésére. A hibásan prediktált változók és literálok javítására egy utólagos kereső algoritmust készítettek, mely minden potenciális javítást behelyettesített a programba, az így keletkező forráskódot lefordította, és összehasonlította a bemenettel. Amennyiben ez a kettő nem egyezett meg, akkor tovább próbálgatta a javítási lehetőségek behelyettesítését. Elmondásuk szerint az alapozó cikk sikertelensége az utólagos hibajavítási technikák hiányából fakadt, véleményük szerint a probléma ezek nélkül nem megoldható.

Fu et al. [3] létrehoztak egy két részből álló kódvisszafejtő keretrendszert, mely a Coda névre hallgat. Az első rész egy kódvázat hoz létre, melyet a második rész javít egy neurális, iteratív hibajavító rendszer segítségével, amely két almodellből áll. A hibadetektálásokat és -javításokat egymás után, kis lépésekben tették meg az iteratív hibajavító rendszerrel.

1.2. Saját ötletünk

Ahogy azt fent láttuk, a korábbi munkák egy közös problémába ütköztek: a helyes változók és literálok prediktálása meglehetősen nehéz egyetlen end-to-end és single-pass modellel. Egy modell akkor single-pass, ha a bemenet csupán egyszer halad át a rendszeren, azaz nem használ iteratív módszereket. Az end-to-end pedig azt jelenti, hogy az egész problémát egy mélytanuló rendszer oldja meg az elejétől a végéig, hagyományos utófeldolgozás, javítás nélkül. A feladat jelentőségét fejezi ki, hogy a TraFix szerzői szerint ez lehetetlen. A problémát a

kutatók különböző kiegészítő komponensek és modellek alkalmazásával próbálták megoldani, melyek utólagos hibajavítási módszereket biztosítanak. Ez alól egyedül az alapozó cikk a kivétel, ami csak strukturális javítási technikákat alkalmazott, az azonosítók és literálok javításával nem foglalkozott.

Egy single-pass, end-to-end modell sokkal hatékonyabb lenne a fentiekhez képest: ha egy modell elsőre is képes lenne helyesen prediktálni a megfelelő változókat és literálokat, akkor nem lenne szükség a különböző javítási technikák alkalmazására. Ez azt is jelentené, hogy ezen megoldás sokkal skálázhatóbb lenne, ugyanis nem függne a visszafordítás sebessége az utólagos hibajavítás sebességétől, melynek jelentős időbe is beletelhet a megfelelő megoldás előállítása. Például a gráfizomorfizmusra épülő javítási módszer [6] vagy az iteratív javítási módszer [3] sebessége nagyban függhet az adatok hosszától.

A feladatot mi egy end-to-end és single-pass modellel közelítjük meg, mely képes hatékonyan gépi kódot visszafejteni, ezzel cáfolva a TraFix [6] szerzőinek korábbi kijelentését. Ennek alapja egy seq2seq háló [10] kiegészítve másoló mechanizmussal (copying mechanism, copy) [4] és attention mechanizmussal [1].

A copy egy olyan technika, mely lehetővé teszi egy token másolását közvetlenül a bemenetről a kimenetre, így a ritkán előforduló azonosítókra vagy literálokra nem a seq2seq-nek kell megpróbálnia visszaemlékezni. Ez azt is jelenti, hogy képes helyes eredményt generálni még akkor is, ha abban egy korábban még nem tanított token szerepel, amennyiben a token másolható és a token kontextusából arra következtet, hogy a tokent másolni kell. Modellünk a copy-t a bemenetben és az elvárt kimenetben is megjelenő literálok, változók és függvénynevek másolására használja.

Emellett alkalmazzuk az attention mechanizmust is. Ez egy olyan módszer, ami képes a seq2seq memóriáján javítani, hogy időben még korábbi dolgokra is képes legyen emlékezni. Ezt úgy éri el, hogy képes kiválasztani azokat a részeket a bemenetből, amikre oda kell figyelni. Az így kiválasztott tokenek ezután nagyobb mértékben lesznek figyelembe véve a következő token generálásakor. Így azokat a tokeneket is pontosabban tudja értelmezni a rendszer, amelyeket nem másolunk a copy segítségével. Ezeknek köszönhetően a single-pass modellünk mellé semmilyen hibajavítási technika sem szükséges (azaz end-to-end is), ez egy hatalmas előrelépést jelent a korábbi architektúrákkal szemben.

A modell eredményességének mérésére a TraFix [6] által bevezetett szintrendszer általunk bővített változatát alkalmazzuk. A 8 nehézségi szintjükhöz képest mi nehezebb nyelvi elemeken is tanítjuk a modellünket, ezáltal még 2 újabb nehézségi szintet hozzáadva az eddigiekhez. Az új nyelvi elemek mind olyanok, amiket a programozók rendszeresen használnak a gyakorlatban, azonban a korábbi megoldások még nem próbálták betanítani. Adathalmazaink a 6. szinttől tartalmaznak ternáris operátorokat, a 9. szinttől switch utasításokat (case és default ágakkal), a 10. szinttől függvénydefiníciókat és függvényhívásokat.

Munkánk fő kontribúciói az alábbiak:

- Egy end-to-end, single-pass neurális háló elkészítése kódvisszafejtéshez. Modellünk felhasználja a másoló mechanizmust [4] egy seq2seq architektúra keretén belül, attention mechanizmussal [1] kiegészítve.

- Két nehezebb szint definiálása, melyben lévő nyelvi elemek gyakran előfordulnak a programozók által írt kódokban. Az új nyelvi elemek a switch utasítások, a ternáris operátorok, a függvényhívások és a függvénydefiníciók.

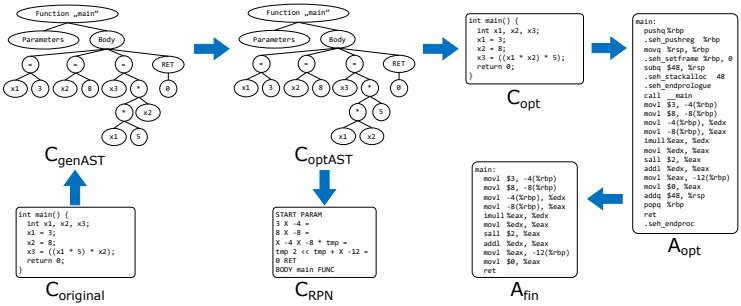
- Modellünk szignifikánsan jobb eredményeket ért el a sikeres kódvisszafejtésben, mint a TraFix [6] a 7. (while ciklusok) és a 8. (egymásba ágyazott elágazások és while ciklusok) szinten anélkül, hogy bármilyen hibajavítási technikát is alkalmaztunk volna.

2. Módszerek

A következőkben bemutatjuk az adatgenerálásunkat, az előfeldolgozási módszereinket, a modellünk belső működését és a C programozási kód előállítását a modellünk kimenetéből.

2.1. Adathalmazok

Az interneten rengeteg nyilvános forráskódot lehet találni, melyeket fel lehet használni egy neurális kódvisszafejtő tanítására. Például D. S. Katz et al. [5] egy nyílt forráskódú programtároló kódbázisát használták fel (Fedora csomagok).



1. ábra. Egy tanítási példa előállítása

Az emberek által írt kódokban viszont bizonyos nyelvi elemek előfordulása sokkal ritkább, mint a generált kódok esetében, így az ember által írt kódok nem számítanak jó minőségű tanítási példának egy mélyháló számára. Például a nyílt forráskódú adathalmazai a Codának [3] és a Neutronnak [8] (Karel[4], Hacker's Delight ciklusmentes programok [12]) nem tartalmaztak olyan nyelvi elemeket, melyeket a modellünknek tanítottunk, mivel a legtöbb ilyen példa elágazás- és ciklusmentes vagy csak egyszerű függvényhívásokat tartalmaz. Mindez azt jelenti, hogy az említett példák csak az első 4 bonyolultsági szintet fedik le az általunk használt 10 szintes rendszerben.

A fentebb említettek miatt, hasonlóan a TraFixhoz [6], egy adatgenerálót hoztunk létre, mely C kódokat generál és fordít le assemblyre ekvivalens módon.

Mivel a modellünk teljesítményét valós kódokon is szerettük volna leméri, létrehoztunk egy emberek által írt programokat tartalmazó adathalmazt (Codeforces[5] versenyprogramozási weboldalon található beküldött megoldások). A realisztikus adathalmaznak a részletes leírását és a modellünk ezen elért eredményeit a 3.5. részben fejtjük ki.

2.2. Előfeldolgozás

Az assembly kód C kódból való előállításához GCC fordítót használtunk[6], mely az AT&T x86-os assembly szabványt alkalmazta. A GCC

[4] `https://www.cs.mtsu.edu/~untch/karel/` (2022)

[5] `http://codeforces.com/` (2022)

[6] `https://github.com/gcc-mirror/gcc/tree/releases/gcc-7`

több száz optimalizációs szabályt alkalmaz fordítás közben, még a `-O0` (optimalizációk kikapcsolása) kapcsoló megadása mellett is. Az előfeldolgozás során arra kellett törekednünk, hogy a C kódot ekvivalens transzformációk sorozatával olyan alakra hozzuk, mely a lehető legjobban hasonlít az assembly kódra. Annak érdekében, hogy ezt elérjük, utánoznunk kellett a fordító által használt optimalizációkat. Ezek alapján az optimalizációk három típusát különböztettük meg.

Az első típusú optimalizáció az utasításokon van végrehajtva, ahol a tisztán konstansokkal végzett műveletek kiértékelődnek, illetve olyan alakra transzformáljuk az utasításokat, hogy azok későbbi optimalizációkat lehetővé tegyenek, vagy csökkentsék a futási időt. Ezen optimalizációkra található egy példa a 2. ábrán: a zárójelek átrendeződnek, a kommutatív operátorok kiértékelésének sorrendje megváltozik, majd a csak konstansokat tartalmazó rész kiértékelődik.

A második típusa az optimalizációknak a változók olyan szorzására, osztására, maradékszámítására vonatkozik, melyeknek egyik operandusa konstans érték. Ezek egy jelentős része biteltolásokká alakul át, ezzel növelve a számítások numerikus stabilitását, és a kiértékelés sebességét.

A harmadik optimalizációtípus egyes logikai kifejezések negálása. Speciális esetekben a GCC a logikai kifejezések operátorának az ellentétét használja. A későbbiekben kifejtjük az említett transzformációk szükségességét.

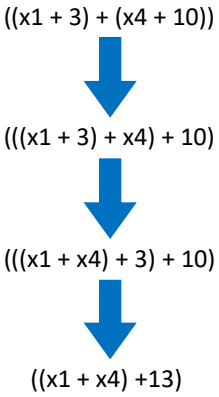
Az előfeldolgozás első lépéseként a kezdeti C kódot ($C_{original}$) átalakítjuk egy absztrakt szintaxisfává (AST) egy parser[7] segítségével (C_{genAST}).

Garantálnunk kellett, hogy ugyanazok a literálok és változók jelennek meg az assembly kódban és a C kódban. Ennek érdekében a GCC fordítás során használt optimalizációs szabályainak jelentős részét implementálnunk kellett. A változók esetében kihasználtuk, hogy a memóriában sorfolytonosan helyezkednek el, és mindegyikhez tartozik egy regiszter offset, mely a változó első bájtját indexeli. Tehát minden lokális változó a C kódban megfeleltethető egy regiszter offsetnek az assemblyben, mellyel az elvárt kimenetben a változókat azonosítjuk.

[7] `https://github.com/eliben/pycparser`

Első típusú optimalizációk

Az első típusú optimalizációk szabályai aritmetikai optimalizációkat hajtanak végre az AST struktúráján és tartalmán. AST alakban ezek a módosítások sokkal egyszerűbben végrehajthatóak a példákon, mivel a fa alakja jelzi a kapcsolatot az egyes csúcsok között, szemben a C nyelvű kóddal, ahol a lineáris alakban ezek az információk csak nehezen elérhetőek, habár a szabályok esetében kulcsfontosságúnak bizonyulnak. Fontos megemlíteni, hogy habár ezek az átalakítások mind ekvivalensek, mégsem lehet őket egyértelműen visszaalakítani az eredeti formájukra. Példának okáért a „(9)" kifejezés lehet egy optimalizáció utáni alakja az (5 + 4)-nek, a (3 * 3)-nak, stb.



2. ábra. Egy példa, amin látható pár GCC által használt optimalizáció, melyeket mi is alkalmazunk

A szerkezeti optimalizációs algoritmus (első típusú optimalizációk) akkor ér véget, ha már nem tud több optimalizációt végrehajtani egyetlen részfán se. Miután az AST elérte végleges alakját (C_{optAST}), visszaalakítjuk C programozási nyelven írt kóddá (C_{opt}).

C_{opt}-ot ezt követően lefordítjuk assembly kódra (A_{opt}), melyet ezután kanonikalizálunk. Kanonikalizáció alatt azt értjük, hogy az A_{opt}-ot tokenizáljuk, és eltávolítjuk belőle a vesszőket. A kanonikalizáció végén a kódvisszafejtés szempontjából szükségtelen részeit az assemblynek (például a program fejlécet, illetve a lábfejet) eltávolít-

juk. Az így kapott kanonikalizált assembly kód lesz a végső formája a modellünk bemenetének (A_{fin}).

Második típusú optimalizációk

Az előfeldolgozás célja, hogy olyan adatokat hozzunk létre, amelyekben az A_{fin} és az elvárt C kód megegyezik az utasítások jellegében és tartalmában (literálok és változók) amennyire csak lehetséges. Ennek érdekében a C_{optAST}-t egy posztorder bejárással alakítjuk át, így elérve a módosított lengyel jelölés formát (C_{RPN}), ahogy a TraFix [6] is tette. Ez a módosított RPN megkönnyíti a modell számára, hogy összefüggéseket találjon az assembly és C nyelvű utasítások között, mivel az assembly szerkezete nagy mértékben hasonlít az RPN-re. Egy ilyen RPN-re történő átalakítás látható a 3. ábrán. A módosított RPN ekvivalens módon visszaalakítható a kezdeti C alakra (C_{optAST}, majd C_{opt}), így nem veszítünk információt a transzformációval.

A posztorder bejárás közben végrehajtjuk a második típusú optimalizációkat. Ekvivalens transzformációkkal a speciális konstansokkal való szorzást, osztást és maradékszámítást átalakítjuk, hogy az assemblyvel megegyező formát vegyen fel a C kód. Ez azt jelenti, hogy biteltolások és túlcsordulások jelennek meg a kódban, hasonlóan az assemblyben történő számításokhoz.

Harmadik típusú optimalizációk

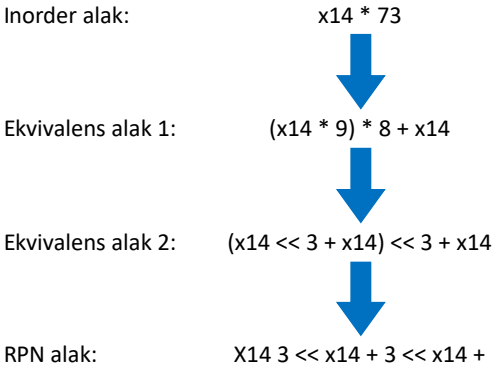
Nem csak aritmetikailag ekvivalens transzformációkat hajtunk végre a posztorder bejárás során. Bizonyos logikai kifejezések operátorát negáljuk annak érdekében, hogy az assemblyben lévő feltételes ugrások típusával megegyezzenek. Például egy ($<$) reláció átalakulhat ($>=$)-re, ha az assemblyben is `jge` (*jump if greater than or equal,* azaz ugrás, ha nagyobb vagy egyenlő) utasítás jelenik meg a `jl` (*jump if less than,* azaz ugrás, ha kisebb) helyett az optimalizáció következtében.

Az assemblyben megjelenő ugrások jelentése nem mindig konzisztens. Ennek oka, hogy különböző módokon kell kezelni a feltételeit egy elágazásnak és egy ciklusnak. Elágazás esetén az assemblyben csak akkor következik be a feltételes ugrás, ha a feltétel hamis, különben a kód a következő sorral folytatódik. Ezzel szemben a ciklusoknál a feltételes ugrás pont fordítva működik, és csak akkor következik be, ha a

feltétel igaznak bizonyult. Így a modell tanítása érdekében a ciklusok feltételeiben lévő operátorokat negáltuk az elvárt kimenetben.

Ez a transzformáció egyszerűen visszaalakítható az eredeti alakra, miután a modell végzett a kimenetének prediktálásával. Az átalakítás lehetővé teszi, hogy ezekben az esetekben is kimásolhassuk az operátort a másoló mechanizmus segítségével a bemenetből.

Bár a harmadik típusú optimalizációk egyidejűleg történnek a második típusú optimalizációkkal, mégis megkülönböztetjük őket, mivel a kódon végzett módosítások jellege különbözik bennük.



3. ábra. Egy példa a literállal való szorzás ekvivalens módon történő átalakítására biteltolások által

A generált példák szűrése

Az adathalmazba tevés előtt minden generált példának át kell mennie egy szűrőrendszeren. Amennyiben egy generált adat fennakad a szűrőn, úgy el lesz vetve, és nem használjuk fel a későbbiekben. Az alábbi szabályoknak kell megfelelnie a generált adatoknak:

- A literáloknak és darabszámuknak meg kell egyeznie C_{RPN}-ben és A_{fin}-ben. Csak minimális eltérést engedünk meg, ellenkező esetben a példát elvetjük: ha egy adott literál n-szer szerepel az optimalizált C kódban, akkor legalább n-szer kell szerepelnie a végső assembly kódban (A_{fin}) is. Ez a szűrés elengedhetetlen a copy

működése szempontjából, ugyanis ha egy literál nem jelenik meg az assembly kódban, viszont az elvárt C kódban megtalálható, úgy az nem tud mit lemásolni, és a modell képtelen a literált prediktálni.

- A logikai kifejezéseket is teszteljük. Ha C_{opt}-ban a különböző relációk (>=, >, stb.) és az assemblyben nekik megfeleltethető feltételes ugrások darabszáma eltér, akkor a példa fennakad a szűrőn és elvetjük.

- Korlátoztuk A_{fin} maximális hosszát 750 tokenre. Ez az érték szignifikánsan magasabb, mint amivel a korábbi munkák dolgoztak, és véleményünk szerint kellően magas, hogy valós programok is beleférjenek. Fontosnak tartjuk megemlíteni, hogy az előfeldolgozás során töröltük a vesszőket és a a visszaalakításhoz szükségtelen részeket az assembly kódból, így az eredeti bemeneti alacsony absztrakciós szintű kód sokkal hosszabb volt. A leghosszabb sikeresen visszafordított assembly állomány (előfeldolgozási átalakítások nélkül) 808 token hosszú volt. Ez összevetve a TraFix [6] eredményével, ahol 668 token volt a leghosszabb sikeresen visszaalakított kód tokenszáma, 20%-os javulást jelent.

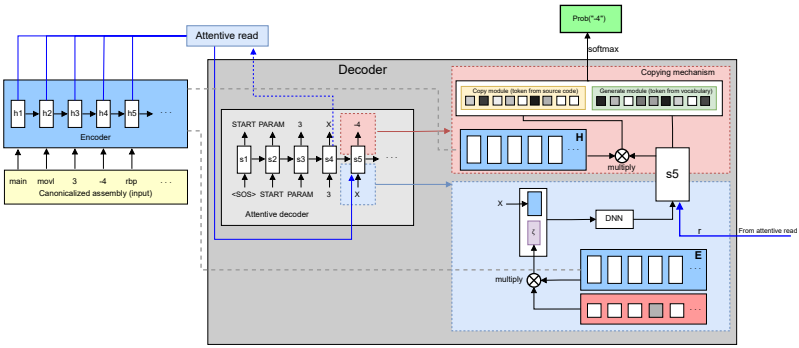
Minden adathalmazunk ($C_{original}$, C_{RPN}, A_{fin}) hármasokból épül fel (beleértve a valódi kódokat tartalmazó adathalmazunkat is). A kezdeti kódot, $C_{original}$, eltároltuk, mivel C_{RPN}-t csak C_{opt} alakra lehet ekvivalens módon egyértelműen visszaalakítani a korábban tárgyalt okok miatt.

A példákat addig generáltuk, amíg a sikeresen generált példák száma el nem érte a kezdetben kijelölt értéket. Amikor ezt a darabszámot elértük, az adathalmazt késznek nyilvánítottuk.

Fontos megemlíteni, hogy a main függvény a példáinkban az összes lehetséges változó definíciójával kezdődik (egész számok (int) X0-től X19-ig, feltételezésünk szerint az általános kódok nem használnak 20-nál több lokális változót egyetlen függvényben). A változókat mind a volatile kulcsszó segítségével definiáltuk, mely garantálta, hogy az első változó (X0) kapja meg a -4-es regiszter offsetet az assembly kódban, míg a második változó (X1) kapja meg a -8-as regiszter offsetet, stb. Ez biztosította a kapcsolatot a változók nevei és a regiszterek offsetei között, ezáltal lehetővé téve a másolást.

Az egész folyamat az 1. ábrán látható.

2.3. A használt modell



4. ábra. A modell, ahogy egy kanonikalizált assembly kódot (bemeneti kódot) dolgoz fel.

A modellünk (4. ábra) egy seq2seq háló Bahdanau-attentionnel [1] és másoló mechanizmussal [4] kiegészítve. A decoder a modell teljesítményének növelése céljából tartalmaz két különböző attentiont: Bahdanau-attention [1] és a másoló mechanizmus [4] saját attentionjét. A decoder egy lépésében átadja az aktuális rejtett állapotot (hidden) a másoló mechanizmus [4] két moduljának. Ezek a modulok a logitokat prediktálják a másolható és a nem másolható részéből a szókészletnek. Softmax nemlineáris függvényt alkalmazva a két logit vektor összefűzésén, megkapjuk a valószínűségeit a következő generált tokennek.

Encoder

Az encoder összetömöríti a bemeneti sorozatban ($X = [x_1, ..., x_L]$) lévő információt egy beágyazási (embedding) réteg és egy rekurrens réteg (GRU [2]) felhasználásával egy rejtett vektor reprezentációba ($\mathbf{H}$).

A **beágyazási réteg** minden bemenethez egy adott méretű vektort rendel hozzá, a vektorban lévő értékeket a modell tanulja.

$$\mathbf{h_t}, \mathbf{e_t} = \mathrm{GRU}(\mathrm{Emb}(x_t), \mathbf{h_{t-1}}), \tag{1}$$

$$\mathbf{H} = [\mathbf{h_1}, \ldots, \mathbf{h_L}], \tag{2}$$

$$\mathbf{E} = [\mathbf{e_1}, \ldots, \mathbf{e_L}], \tag{3}$$

ahol $\mathbf{h_t}$ a rejtett állapot, $\mathbf{e_t}$ pedig az encoder GRU-jának a kimenete a t időpillanatban (1).

Decoder

A decoder felhasználja az encoder által összetömörített információt, és ez alapján prediktálja a modell kimenetét. A decoderünk tartalmaz egy beágyazási réteget, egy dropout réteget (csak tanításkor használt) és egy rekurrens réteget (GRU [2]), melyeken felül még kiegészítettük Bahdanau-attentionnel [1], másoló mechanizmussal [4] (generate modul és copy modul) és az ahhoz tartozó külön attentionnel. Az említett mechanizmusoknak a lényege, hogy lehetővé teszik a modell számára, hogy a megfelelő tokeneket prediktálja elsőre, ezzel mindenféle utólagos hibajavítási technikát szükségtelenné téve.

A **dropout rétegben** egy előre meghatározott p valószínűséggel nullázzuk ki a paraméterként kapott tömb (vektor, tenzor, mátrix) elemeit, a változatlanokat pedig megszorozzuk $\frac{1}{1-p}$-vel, ezzel az átlagos összérték egy soron/oszlopon belül nem változik.

A Bahdanau-attention és a selective read (copy mechanizmushoz külön kifejlesztett attention) az alábbi módon működik:

$$\zeta = \mathrm{Sel}(\mathbf{y_{t-1}}, X, \mathbf{l_{t-1}}, \mathbf{E}), \tag{4}$$

$$\mathbf{r} = \mathrm{Attn}(\mathbf{s_{t-1}}, \mathbf{E}), \tag{5}$$

ahol ζ a selective read (a copy cikkjében olvasható módon [4]) eredménye, r az attentive read eredménye (Bahdanau-attention [1]), $\mathbf{y_t}$ a prediktált token t időpillanatban (9), $\mathbf{s_t}$ a GRU (6) rejtett állapota t időpillanatban, $\mathbf{l_t}$ tartalmazza a másolási logitokat a bemenetre vonatkoztatva, $\mathbf{l_{t}}_{(i)}$ reprezentálja a logitját az i-edik tokennek a bemenetből (8). A decoder többi része az alábbiak szerint működik:

$$\mathbf{s_t} = \mathrm{GRU}(\mathrm{Emb}(\mathbf{y_{t-1}}), \zeta, \mathbf{r}), \tag{6}$$

$$\mathbf{g} = G(\mathbf{s_t}), \tag{7}$$

$$\mathbf{c},\ \mathbf{l_t} = C(X, \mathbf{s_t}, \mathbf{H}), \tag{8}$$

$$\mathbf{y_t} = \mathrm{Softmax}(\mathbf{g}\ \oplus\ \mathbf{c}), \tag{9}$$

ahol ζ a selective read (4), $\mathbf{r}$ az attentive read (5), G és C a generate modul és a copy modul, továbbá $\oplus$ jelenti az összefűzését a két eredménynek, melyekből a Softmax függvény használatával kapjuk meg az eloszlását a következő tokennek.

A **generate modul** (G) felelős a szókészlet nem másolható szavainak prediktálásáért. Egyetlen DNN-t tartalmaz, így a működése az alábbiak szerint írható le:

$$\mathbf{g} = \mathrm{DNN}(\mathbf{s_t}), \tag{10}$$

ahol $\mathbf{s_t}$ a decoder hidden állapota, mely megegyezik a (6) egyenletben lévővel, és a (7) egyenletben paraméterként van átadva.

A **copy modul** (C) felelős a szókészlet másolható tokenjeinek prediktálásáért. Egyetlen DNN-t tartalmaz, a működése pedig a következők szerint zajlik:

$$\mathbf{D} = \mathrm{DNN}(\mathbf{H}), \tag{11}$$

$$\mathbf{l_t} = \mathbf{D} \cdot \mathbf{s_t}, \tag{12}$$

$$\mathbf{c}_{(i)} = \sum_{j:X_{(j)}=V_{(i)}} \mathbf{l_t}_{(j)} \qquad (\forall i \in [\,1, \ldots, \|V\|\,]), \tag{13}$$

ahol $\mathbf{H}$ az encoder hidden kimeneteit jelöli a (2) egyenletből, $\mathbf{s_t}$ a decoder rejtett állapota a (6) egyenletből, mely a (8) egyenletben lett paraméterként átadva, valamint V az X szókészlet azon részhalmaza, mely tartalmazza a másolható tokeneket.

A hibafüggvény és a hiperparaméterek

Hibafüggvényként a cross-entropy losst használtuk, ennek eredményén futtattuk a backpropagationt is.

A tanítás közben az ADAM [7] adaptív optimalizációs algoritmust használtuk, melynek segítségével a tanulási ráta paraméterenként külön-külön szabályozható, ezzel még hatékonyabbá téve a tanulás folyamatát. A használt hiperparaméterek az 1. táblázatban találhatóak.

2.4. C kód előállítása

Miután a modell feldolgozta a bemenetet, és befejezte a kimenet generálását, a kapott RPN formát még vissza kell alakítani C programozási nyelven írt kódra. Ennek érdekében az előfeldolgozási lépéseket fordított sorrendben kellett implementálni, ellentétes működéssel.

Első lépésként a biteltolásokat szűrjük ki, és alakítjuk vissza, melyek eredetileg számliterálokkal való szorzások, osztások, maradékszámítások lehettek. Ez az átalakítás egyértelműen végezhető el, hiszen az

Név	Érték
Batch méret	32
Hidden méret	512
Begyazási méret	128
Tanulási ráta	0.0001
Dropout ráta	0.4

1. táblázat. A modell hiperparaméterei

RPN-ben a megfelelő literálok találhatóak meg, melyeket az assembly kódból másoltunk ki. Egy mintaillesztő algoritmust használunk a megfelelő biteltolási kifejezések felismerésére. Az osztások fordítása során az assemblyben utasítások hosszabb sorozata jelenik meg, melyek nagy, látszólag véletlenszerű számokkal dolgoznak. Ezen jellegű osztásokat visszaalakítjuk eredeti alakra az eltolások visszaalakítása előtt.

Miután a konvertálható bitműveleteket visszaalakítottuk, a keletkezett RPN-t egy AST-vé konvertáljuk. Ezalatt a konverzió alatt a módosított logikai kifejezéseket (ismételten negálva a jelentésüket) visszatranszformáljuk eredeti jelentésükre egyértelmű módon.

Az RPN tokenjeit egyesével dolgozzuk fel az AST-vé alakítás során, mindegyik egy új csúcsot eredményez. Ezen csúcsok belekerülnek egy verem adatszerkezetbe. Egyes speciális tokenek (mint például IF vagy WHILE), esetén a korábbi csúcsokat ki kell venni a veremből, és összefogni egy új részfává, melyet ezután a verem tetejére teszünk. Miután minden tokent feldolgoztunk, a verem egyetlen tokent tartalmaz: az AST reprezentációját az egész programnak.

A pycparsert[8] használva a keletkezett AST-t visszaalakítjuk C programozási nyelven írt kóddá, melyet ezután meg lehet mutatni a felhasználónak.

3. Kiértékelés

A következő szakaszban bemutatjuk a kiértékelési módszereinket és a modellünk eredményét: a teljesítményét a 10 bonyolultsági szinten, a különböző nyelvi elemeken, valamint az egymásba ágyazott bináris

[8] `https://github.com/eliben/pycparser`

kifejezések nélkül, amik a legnehezebben visszafejthető nyelvi elemek. A szakaszt azzal zárjuk, hogy bemutatjuk a modell teljesítményét a valós adathalmazon.

3.1. Módszerek a teljesítmény mérésére

Több mód is van arra, hogy a kódvisszafejtés sikerességét mérni tudjuk. Ebben a dolgozatban két fő kiértékelési módszert fogunk alkalmazni annak eldöntésére, hogy milyen jól teljesít a modellünk.

Az első a tokenenkénti pontosság. Ez a technika azt méri, hogy hány token egyezik meg az elvárt RPN kimenetben és a modell prediktált kimenetében, osztva az elvárt kimenet hosszával. Egyezésnek csak azt tekintjük, ha ugyanazon a pozíción szerepel a két token. Tehát egy megegyező, de eltolt szakasza a kimeneti tokeneknek nem tekinthető egyezésnek. A tokenenkénti pontosság képlete alább látható:

$$\frac{\sum_{i=1}^{|\mathbf{y}|} \mathrm{ind}(\mathbf{y}_{(i)} = \hat{\mathbf{y}}_{(i)})}{|\mathbf{y}|}.$$

A második módszer a tökéletes visszafejtéseket méri, ahol az elvárt és a tényleges kimenet tokenjei megegyeznek minden indexen, és a hosszuk is azonos.

3.2. Teljesítmény a bonyolultsági szinteken

A kódvisszafejtés problémáját 10 bonyolultsági szintre osztottuk. Ezek a szintek hasonlóak ahhoz, amiket a TraFix [6] szerzői bevezettek, de van egy pár kisebb eltérés, valamint mi két új célkitűzést is hozzáadtunk (9. és 10. szint).

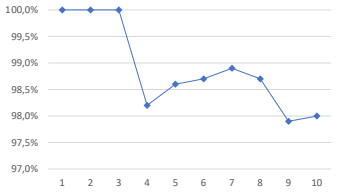
1. Számliterálok értékül adása;
2. 1. szint + Változók értékül adása;
3. 2. szint + Unáris operátorok ($++$, $--$, $\sim$), mint önálló utasítások, és értékadásokon belül;
4. 2. szint + Értékadások, amiknek a jobb oldala tartalmazhat bináris kifejezéseket ($+$, $-$, $*$, $/$, $\%$, $|$, $\&$, $\hat{}$, $<<$, $>>$, ahol az első 5 operátor egymásba ágyazva is szerepelhet);

5. A 3. és a 4. szint együttese;
6. 5. szint + If és If-else elágazások (egymásba ágyazhatóság nélkül) és ternáris operátorok (egymásba ágyazhatóság nélkül);
7. 6. szint + While ciklusok (egymásba ágyazhatóság nélkül);
8. 7. szint + Egymásba ágyazhatóság;
9. 8. szint + Switch utasítások (egymásba ágyazhatóság nélkül, és más utasításoknak sem lehetnek részei);
10. 9. szint + Függvények paraméterekkel, visszatérési értékkel és függvényhívások.

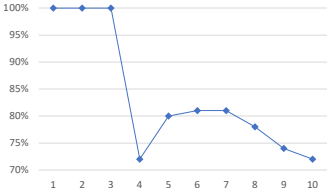
A különbség az első 8 szint meghatározása és a TraFix [6] által definiált szintek között a következő: a 6. szinten, valamint afelett lehetnek ternáris operátorok is.

A generált adathalmazunk tartalmaz kódokat minden bonyolultsági szintről. Összességében az adathalmazunk 260161 tanítási példát tartalmaz. Az egyes bonyolultsági szinteket reprezentáló példák mellett az adathalmazban szerepelnek olyan példák is, amik főként egy adott nyelvi elemet használnak, vagy valamilyen speciális kódot tartalmaznak, amit a GCC különleges módon optimalizál. Ez utóbbiból 161 darab található az adathalmazban.

A modell által elért tokenenkénti pontosság és a tökéletesen visszafejtett kódok aránya a különböző szinteken az 5. ábrán láthatók.



(a) A pontosság a 10 bonyolultsági szinten



(b) A tökéletes kódvisszafejtés mértéke a 10 bonyolultsági szinten

5. ábra. A modellünk pontossága a bonyolultsági szinteken. A teljesítményt a tokenenkénti pontossággal és a tökéletes kódvisszafejtéssel ábrázoljuk (az y tengelyek nem 0-tól kezdődnek)

3.3. Teljesítmény az egyes nyelvi elemeken

A modellünk pontosságát megmértük olyan adathalmazokon is, amik csak egy-egy nyelvi elemre koncentrálnak. Ezek az adathalmazok az adott típusú nyelvi elemen kívül a többit sokkal kisebb mértékben tartalmazzák (csak értékadásokat literálokkal vagy változókkal).

Ezek az eredmények a 2. táblázatban láthatók: BINARY egymásba ágyazható kétoperandusú operátorokat, BITWISE értékadásokat bitműveletekkel, IF egymásba ágyazható elágazásokat, TERNARY értékadásokat ternáris operátorokkal, WHILE egymásba ágyazható while ciklusokat, SWITCH egymásba nem ágyazható switch utasításokat és FUNCTION értékadásokat tartalmaz függvényhívásokkal, valamint függvénydefiníciókat paraméterekkel és visszatérési értékkel.

Szint	Tokenenkénti pontosság	Tökéletes kódvisszafejtés
BINARY	97.45%	62%
BITWISE	100.00%	100%
IF	99.08%	90%
TERNARY	99.04%	86%
WHILE	99.70%	97%
SWITCH	96.34%	75%
FUNCTION	99.95%	98%

2. táblázat. A modellünk teljesítménye olyan adathalmazokon, amik csak egyes nyelvi elemeket tartalmaznak

3.4. Teljesítmény egymásba ágyazott bináris operátorok nélkül

A modellünk teljesítményét olyan adatokon is megmértük, amik nem tartalmaznak egymásba ágyazott bináris operátorokat. Ezen adathalmaz elkészítésének oka az, hogy az előzetes méréseink alapján ez a nyelvi elem bizonyult a legnehezebbnek a modell számára. Az elért eredmények a 3. táblázatban olvashatók.

Szint	Tokenenkénti pontosság	Tökéletes kódvisszafejtés
4	99.81%	99%
5	99.76%	97%
6	99.82%	98%
7	99.81%	98%
8	99.56%	96%
9	98.82%	91%
10	98.88%	88%

3. táblázat. A modellünk eredménye az egymásba ágyazott bináris operátorokat nem tartalmazó adathalmazon

3.5. Eredmények valós adatpéldákon

A kódpéldák a realisztikus adathalmazunkban Codeforces-ról[9] származnak, egy weboldalról, ami online programozási versenyeket szervez. 2000 különböző feladatból 10582 darab beadott, C-ben írt fájlt gyűjtöttük össze. Minden feladatnál a 10 legrövidebb, helyes megoldást töltöttük le (amennyiben nem volt 10 ilyen, akkor annyit, amennyi volt).

Mielőtt még hozzáadtuk volna az adathalmazhoz őket, a beadott fájlokat megszűrtük. Minden fájl, ami a következő szabályok bármelyikét teljesítette, elvetettük:

- tartalmaz olyan operátorokat, amiket nem láttunk tanítás közben (pl. +=, &&, &=);
- tartalmaz nem tanított beépített típusokat (pl. struct, enum, float, tömbök);
- tartalmaz nem tanított beépített kulcsszavakat (pl. continue, goto, exit);
- tartalmaz nem tanított beépített függvényeket (pl. strcpy, abs);
- használ olyan könyvtárakat, amik nem tanított beépített függvényeket tartalmaznak (pl. math.h, string.h, stdlib.h);

[9] `http://codeforces.com/` (2022)

- egy másik beadott megoldás duplikációja;
- üres fájl;
- tartalmaz scanf függvényhívást egy ciklus feltételében (ennek az az oka, hogy a többi scanf hívást lecseréljük értékadásokra, ahogy azt később el is magyarázzuk);
- tartalmaz értékadásokat ternáris operátorokon belül vagy láncolt értékadásokat;
- tartalmaz több függvénydefiníciót is vagy globális változót;
- tartalmaz void kulcsszót.

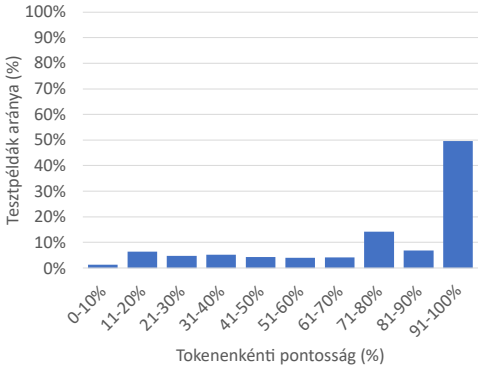
A szűrés után megmaradt fájlok még mindig tartalmazhattak olyan ciklusokat, amiket tanítás közben nem ismert meg a modell (for ciklus, do-while ciklus). A for ciklusok esetében ekvivalensen át tudjuk őket alakítani while ciklusokká, amiket tanult a modell. Ezzel szemben a do-while ciklusok legalább egyszer lefuttatják a ciklus belsejét. Ezt a problémát úgy oldottuk meg, hogy a ciklus belsejének egy másolatát helyeztük a ciklus elé, és a ciklust egy egyszerű while ciklussá alakítottuk. Fontos megjegyezni, hogy ez nem ugyanazt az assembly kódot generálja, mintha közvetlenül egy do-while ciklust fordítanánk le GCC-vel. Ezek mellett a változók és a függvények nevét is megváltoztattuk, hogy konzisztensek legyenek a tanítópéldákkal.

Végül az átalakított fájlok szintén átestek azokon a teszteken, mint amiken a tanítópéldák (ld. a Módszerekben szereplő szűrésről szóló szakaszt). A fájlokat, amik sikeresen átmentek ezen a szűrőn, hozzáadtuk az adathalmazhoz. A végleges adathalmaz 669 darab példát tartalmaz.

Habár a modellünk generált adatokon volt tanítva, ami a különböző nyelvi elemeket (pl. while ciklusok, ternáris operátorok) sokkal nagyobb gyakorisággal tartalmazta, mint az emberek által írt kód, így is jól teljesít a realisztikus adathalmazon. 76%-os tokenenkénti pontosságot és 35%-os tökéletes kódvisszafejtést ért el. A visszafejtés pontosságának eloszlása a 6. ábrán látható.

4. Diszkusszió

A modellünk felülmúlta azokat az eredményeket, amiket az alapozó cikk [5], a TraFix [6], és a Coda [3] elértek (Coda esetében csak



6. ábra. Az elért pontosságok eloszlása a realisztikus adathalmazon

az ott mutatott hibajavító technikájuk alkalmazása nélkül). A tökéletes visszafordítási arányban 8%-ot javítottunk a TraFix-hoz képest a számukra legnehezebb 8. komplexitási szinten (5. ábra), ezzel cáfolva a TraFix szerzőinek korábbi kijelentését, miszerint a feladat nem megoldható utólagos hibajavítások nélkül. Emellett eredményesen növeltük a sikeresen visszafordított kód maximális hosszát, mely esetünkben 808 token hosszú volt az előfeldolgozást megelőzően, ez 21%-os növekedést jelent a TraFix korábbi eredményével szemben, az alapozó cikkhez képest pedig 621%-os javulást jelent (az alapozó cikk leghosszabb kódja 112 tokenből állt). Habár a Coda tökéletes visszafordítási arányban 90% felett teljesített, az ő példáikban a függvényhívások domináltak, így a számliterálok (és az ezeken értelmezett operátorok), valamint a ciklusok elenyésző mértékben jelentek meg a példáikban. Hasonló példákon architektúránk hasonlóan jó visszafordítási arányt ért el utólagos iteratív hibajavítások nélkül, ez látható a 2. táblázat IF, FUNCTION soraiban.

Az eredményeink azt mutatják, hogy a modell számára a visszafejtés során a legnehezebbek az egymásba ágyazott bináris operátorok, amik szorzást, osztást vagy maradékszámítást írnak le egy számliterállal a jobb oldalukon, valamint a switch utasítások (2. táblázat). A bináris kifejezések azért okoznak problémát, mert gyakran bitenkénti eltolásokra konvertálódnak fordítás során, még a GCC `-O0`-s beállításának használatával is, és a keletkező hosszú kódban nehéz számontartani a

szabályokat. A switch utasítások pedig azért nehezek, mert viszonylag magas az absztrakciós szintjük, és a fordításkor keletkező kód hossza nagyobb, mint a többi nyelvi elem esetében.

Az 5. ábrán látható, hogy a 4. szint nehéz a modell számára, mivel ebben a szintben jelennek meg a bináris operátorok. Úgy gondoljuk, hogy a modell azért teljesít jobban a későbbi szinteken, mert ott kisebb a gyakorisága az egymásba ágyazott bináris operátoroknak, mivel több nyelvi elem típus van a példákban.

5. Összefoglalás

Ebben a dolgozatban bemutattuk az end-to-end, single-pass kódvisszafejtő eszközünket, amely másoló mechanizmust alkalmaz. Megmutattuk, hogy egy neurális kódvisszafejtő iteratív hibajavító algoritmusok használata nélkül is képes megtanulni helyesen prediktálni a literálokat és a változókat.

A modellünk nagy pontosságot ért el a különböző kódvisszafejtési feladatokon, miközben idiomatikus kódot generált anélkül, hogy imitálta volna a regisztereket vagy goto utasításokat használt volna. A bonyolultsági szintrendszerünk a TraFix [6] hasonló szintrendszerén alapult, de kiegészítettük két új szinttel, hogy újabb nyelvi elemeket is megtanuljon a modell (ternáris operátorok, switch utasítások, függvények), amiket a programozók nap mint nap használnak.

Azt is megmutattuk, hogy egy end-to-end, single-pass modell képes hatékonyan visszafejteni alacsony absztrakciós szintű kódot anélkül, hogy iteratív hibajavítási módszereket alkalmazna. A másoló mechanizmus hozzáadása a modellünkhöz lehetővé tette, hogy helyesen prediktálja a literálokat és a változókat, ami nehéz feladat a tudományterületen.

Kutatásunk közben azt tapasztaltuk, hogy az egymásba ágyazott bináris kifejezések visszafejtése a legnehezebb feladat. Ezek a kifejezések olyan köztes értékeket használnak fel az alacsony szintű számításokban, amik lényegesen meghosszabbítják a kódot.

Úgy gondoljuk, hogy a jövőben érdemes lenne megnézni, hogy egy tree LSTM [11] copy-val hogyan kezelné a kódvisszafejtés problémáját. Ahelyett, hogy lineáris szövegként tárolná az adatokat, ezek a hálók egy faszerkezetet alkalmaznak a program leírásához, ami sokkal egyszerűbbé teszi a nyelvi elemek közti kapcsolatok felismerését.

Későbbi TDK munkánk során (Teleki Sándor, Kovács Gergely Zsolt, Szalay Gergő: Neurális kódvisszafejtés rekurzív szegmentálással) az assembly kód rekurzív szegmentálásának módszerét vezettük be. A rekurzív szegmentáció, mely megvalósításaként egy külön modellt tanítottunk be, megoldást nyújtott a hosszabb bemenetek kezelésére.

Az eredményeinket összefoglaló nemzetközi publikációt küldtünk be a Springer Nature: Neural Computing and Applications folyóiratába.

Hivatkozások

[1] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, Neural machine translation by jointly learning to align and translate, *CoRR,* abs/1409.0473 (2014).

[2] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, Yoshua Bengio, On the properties of neural machine translation: Encoder–decoder approaches, *Proceedings of SSST-8,* abs/1409.1259 (2014), pp. 103–111.

[3] Cheng Fu, Huili Chen, Haolan Liu, Xinyun Chen, Yuandong Tian, Farinaz Koushanfar, Jishen Zhao, Coda: An end-to-end neural program decompiler, *Advances in Neural Information Processing Systems (NeurIPS),* 32 (2019).

[4] Jiatao Gu, Zhengdong Lu, Hang Li, Victor O. K. Li., Incorporating copying mechanism in sequence-to-sequence learning, *Proc. 54th Annual Meeting of the Association for Computational Linguistics,* 1, abs/1603.06393 (2016), pp. 1631–1640.

[5] Deborah S. Katz, Jason Ruchti, Eric Schulte, Using recurrent neural networks for decompilation, *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER),* (2018), pp. 346–356.

[6] Omer Katz, Yuval Olshaker, Yoav Goldberg, Eran Yahav, Towards neural decompilation, *CoRR*, abs/1905.08325 (2019).

[7] Diederik P. Kingma, Jimmy Ba, Adam: A method for stochastic optimization, *Proc. 3rd ICLR,* (2015).

[8] Ruigang Liang, Ying Cao, Peiwei Hu, Kai Chen, Neutron: an attention-based neural decompiler, *Cybersecurity,* 4(1) (2021), pp. 1–13.

[9] Zhibo Liu, Shuai Wang, How far we have come: Testing decompilation correctness of c decompilers, *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, New York, NY, USA (ISSTA 2020),* pp. 475–487.

[10] Ilya Sutskever, Oriol Vinyals, Quoc V Le, Sequence to sequence learning with neural networks, *Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K.Q. Weinberger (Eds.) Advances in Neural Information Processing Systems,* 27, 2014.

[11] Kai Sheng Tai, Richard Socher, Christopher D. Manning, Improved semantic representations from tree-structured long short-term memory networks, *CoRR*, abs/1503.00075, 2015.

[12] H. S. Warren, *Hacker's delight,* Pearson Education, Addison-Wesley, 2013.

[13] Jiajun Zhang, Chengqing Zong, Deep neural networks in machine translation: An overview, *IEEE Intelligent Systems*, 30(5) (2015), pp. 16–25.

Parametrikus felületek távolságmezőjének generálása és megjelenítése[‡]

Szente Péter[*]

Eötvös József Collegium[**]

szentep.hu@gmail.com

Témavezető: Bán Róbert
ELTE IK Algoritmusok és Alkalmazásaik Tanszék

1. Bevezetés

A számítógépes modellezés egyik meghatározó eszköze testek felületének különböző parametrikus felületekkel való leírása. Egyre szélesebb körben elterjedő implicit felületreprezentációk az előjeles távolságfüggvények és diszkretizált változatuk, a távolságmezők. A távolságmezővel reprezentált felület egy speciális sugárkövető eljárással, a sphere tracinggel jeleníthető meg.

A cél parametrikus felületekkel ábrázolt objektumok távolságmezőjének generálása GPU segítségével. A létrejövő távolságmezők megjeleníthetők GPU-val gyorsított módon, az említett sphere tracing módszerrel. Ezt a módszert összehasonlítom a parametrikus felület háromszögekkel tesszellált megjelenítésével.

A távolságmezőt felhasználom még geometriai lekérdezések elvégzésére, mint a felületi normális meghatározása, ami a felület árnyalásához

[‡] A szerző 2023. júniusi Kari TDK Konferenciára benyújtott dolgozatának rövidített, átdolgozott változata.

[*] ELTE Informatikai Kar

[**] 2020–

elengedhetetlen. Meghatározhatók még láthatósági viszonyok, melyet vetett árnyékok számításánál használok.

A készített programmal kétdimenziós függvények grafikonjai és Bézier-felületek jeleníthetők meg. A Bézier-felületekre azért esett a választás, mert a kontrollponthálójukon keresztül intuitívan manipulálhatók, és a folytonosság megtartása mellett összeilleszthetők, így különösen alkalmasak felületek modellezésére. Harmadfokú Bézier-felületek távolságmezőjének generálására speciális algoritmust is adok, melyet összehasonlítok a korábban tárgyalt módszerekkel.

2. Szakirodalmi áttekintés

A téma Hart cikkével [9] kezdődik, melyben bemutatja, hogyan használható a sphere trace algoritmus felületek megjelenítésére. A módszer a sugárkövetést gyorsítja fel azzal, hogy a felülettől vett távolsággal lép a sugáron. Ez különösen a felülettől távol eredményez gyorsítást. A módszer előfeltétele, hogy a távolságfüggvényt viszonylag gyorsan ki tudjuk számítani. Ez könnyen megtehető egyszerűbb testek esetében. A módszert általában a GPU pixel shaderében implementálják, így a teljes színtér eltárolható GPU kódként. Emiatt a Demoscene-ekben nagyon sok példát látunk rá. Ilyeneket gyűjt össze például a Shadertoy weboldal.

Ha a felületek távolságfüggvénye nehezen számítható, vagy a jelenet túl összetett, akkor közelítésekre lehet szükség. Ha a távolságfüggvény becslés nem elég jó, akkor a sugárkövetés lelassul. Az alap sugárkövető algoritmus felgyorsítására több módszer is született. [2, 3, 5, 10] A sugárkövetés gyorsítható, ha a távolságfüggvényt egy diszkrét rácson kiértékeljük, és az eredményt eltároljuk. Ennek a műveletnek nem kell valós idejűnek lennie, minden objektumra előszámítható. A távolságmezőből a távolságfüggvény becslését úgy kapjuk, hogy a szomszédos 8 pontot trilineárisan interpoláljuk. Ha a távolságmezőben pontos értékek vannak, akkor a 8 távolság által reprezentált harmadfokú felület pontosan rekonstruálható. [8]

3. Elméleti háttér

A sugárkövetéssel történő képszintézisről részletes összefoglaló olvasható Bálint Csaba OTDK dolgozatában [6, 11–16. o.], így azt itt nem részletezem.

3.1. Sphere tracing

Adott egy $d : \mathbb{R}^n \to \mathbb{R}$ előjeles távolságfüggvény, mely minden bemenetre a pont a felület határától vett előjeles távolságát adja. Ez az érték a felület által határolt térrész belsejében negatív, kívül pedig pozitív. Formálisan d akkor távolságfüggvény, ha az alábbi teljesül [9]:

$$f(p) = d(p, f^{-1}(0)) \quad (p \in \mathbb{R}^n).$$

A sugárkövetés során adott egy kiindulópont és egy sugárirány. A cél a félegyenes–felület metszéspont megtalálása. Míg a sugármetszés egyszerűbb matematikai objektumok esetén (pl. gömb, sík) analitikusan elvégezhető, bonyolultabb testek esetén csak óvatosabban közelíthetünk a felülethez a sugáron (ray marching [14]).

A sphere trace algoritmusról Hart cikkében [9] részletesen olvashatunk. Röviden összefoglalva a sphere trace egy sugárkövető algoritmus, mely minden lépésben kiértékeli a távolságfüggvényt. Tudjuk, hogy a távolságfüggvénnyel megegyező méretű üres tér van a pont körül, hiszen az a felülethez vett távolság minimumát, vagy annak *alsó közelítését* adja. Ekkor a távolsággal megegyező méretűt léphetünk a sugár mentén. A sugárkövetés a felület közelében lelassul. Ha a távolság epszilonnál kisebb, vagy fix lépésszám után leállítjuk az iterációt.

3.2. Előjeles távolságfüggvények

Egyszerűbb alakzatok

Inigo Quilez oldalán [15] felsorolja sok egyszerűbb alakzat analitikus távolságfüggvényét. Például egy o középpontú és r sugarú gömb előjeles távolságfüggvénye:

$$d_g(p) = \|p - o\|_2 - r.$$

A weboldalon további testek távolságfüggvényei is láthatók. Ezekből aztán könnyen építhetünk színteret a tömörtest-modellezésben is hasz-

nált műveletekkel: unió, kivonás, metszet, nagyítás, lekerekítés, forgatás, tükrözés, stb.

Parametrikus felületek távolságfüggvénye

Háromdimenziós euklideszi térben parametrikus egyenlettel megadott felületeket hívjuk így. $f : \mathbb{R}^2 \to \mathbb{R}^3$ Azért felület, mert a vektorértékű függvény értelmezési tartomány beli pontjaihoz háromdimenziós pontokat rendelünk. Modellezéskor így a test felszínét adjuk meg. Parametrikus egyenlettel megadhatók függvények grafikonjai is. Legyen $h(u, v)$ a függvény, ekkor a grafikon parametrikus egyenlete:

$$f(u, v) = (u, v, h(u, v)).$$

Ezen felületek távolságfüggvénye analitikusan sok esetben nem meghatározható. Vegyünk egy kétdimenziós példát, az $y = \sin(x)$ függvényt és egy (e, f) mintavételezési pontot, amihez a legközelebbi pontot keressük a görbén. ekkor a minimalizálandó függvény:

$$d(x) = \sqrt{(x - e)^2 + (\sin(x) - f)^2}.$$

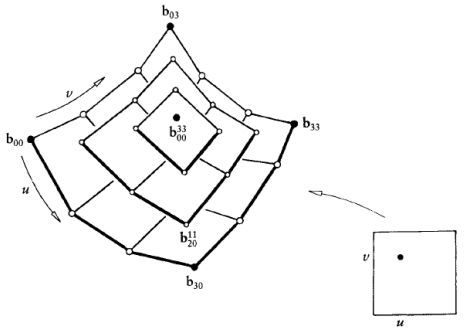
Ezt csak néhány speciális esetben tudjuk analitikusan megoldani. Polinomokkal sem boldogulunk könnyen analitikus módszerekkel. Például ha a polinom fokszáma harmadfokú, akkor a megoldás során egy ötödfokú polinom gyökeit kellene meghatározni.

3.3. Bézier-görbék

Ennek a résznek alapul G. Farin: Curves and Surfaces for CAGD című könyve szolgált. [7] A de Casteljau-algoritmusról és Bézier-görbékről a könyv 45–47., a Bernstein-polinomokról a könyv 57. oldalán olvashatunk. Ezeket itt nem részletezem, de ajánlom az Olvasó figyelmébe.

3.4. Bézier-felületek

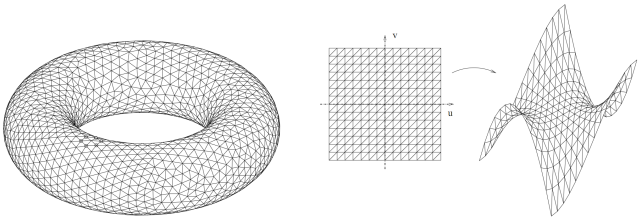
A matematikai háttérhez ismételten ajánlom Farin könyvének [7] megfelelő fejezeteit. Farin a Bézier-felületeket a tenzorszorzat felületek bevezetéseként írja le. Egy egyik irányban m, másik irányban n-edfokú felületet $(m+1) \times (n+1)$ kontrollpont definiál. A felület pontjai kiszámíthatók a kontrollponthálóó ismételt bilineáris interpolációjával. Ezt a felület pontbeli kiértékelésének nevezzük.



1. ábra. Bézier-felület kiértékelése [7, 248. o.]

3.5. Tesszelláció

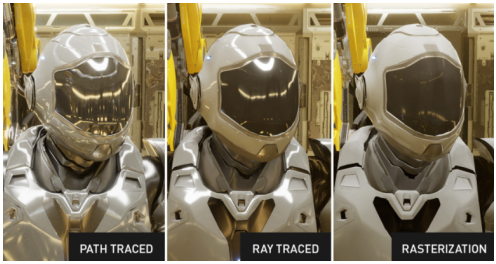
A legelső megjelenítési mód a felület háromszögekkel való közelítése. A tesszelláció vagy háromszögelés során a cél úgy lefedni háromszögekkel a felületet, hogy annak részletei ne vesszenek el. Ez egyben a leggyakrabban használt modellezési módszer is. A videókártyák hardveresen támogatják háromszögek raszterizációját, így ez a módszer nagyon gyors. A többi módszer helyességét a háromszögekkel tesszellált közelítéssel fogom ellenőrizni. Ha függvények grafikonjait háromszögeljük, általában egyenletes felosztást veszünk a paramétertérben. A függvényértéket a harmadik koordináta reprezentálja.



2. ábra. Tórusz és függvénygrafikon háromszögelése. (wikiwand.com – Surface_triangulation)

3.6. A sugárkövetés motivációja

A sugárkövetés költségesebb művelet, mint a raszterizáció, hiszen visszaverődéseket, törést és árnyéksugarakat is számítunk a jobb eredmény érdekében. Ha a fotorealisztikus eredmény a cél, akkor viszont mindenképpen sugárkövetést használunk, és a cél az extra számítási költségek csökkentése, a sugárkövetés felgyorsítása.



3. ábra. Megjelenítési módok összehasonlítása. (blogs.nvidia.com – What is path tracing?)

3.7. Fénymodell

Fénymodellnek egy egyszerű Blinn–Phong-árnyalást használtam [1] árnyéksugarakkal. Az árnyéksugár-számítás ugyanazt az algoritmust használja, mint a sugárkövetés. Ha elmetsszük a felületet a felületi pontból a (pont)fényforrás felé indított sugárral, akkor a felületi pont árnyékban van.

Fontos megemlíteni, hogy a távolságfüggvények segítségével puha árnyékszámító algoritmus is adható. [4]

3.8. Távolságfüggvényrács számítása

A távolságfüggvényt egy diszkrét rácson értékelem ki, melyet egy háromdimenziós textúrában tárolok. Mivel csak a mintavételezési pont változik, maga a kiértékelendő (vagy meghatározandó) távolságfüggvény nem, így ki tudom használni a GPU masszívan párhuzamos architektúráját. A textúrát saját GPU compute shaderek segítségével számítom ki. A számítás elvégzésére később több módszert is mutatok.

Fontos megjegyezni, hogy ennek a műveletnek nem kell valós idejűnek lennie. A program indításakor a textúra több frame alatt kiszámítható. Ezután elég a textúrát a fragment shader számára feltölteni a GPU-ra a sugárkövetés előtt.

3.9. Sugárkövetés távolságmezőn

A sugárkövetés során adott a kiinduló pont P, és a sugár iránya d. Emellett adott egy F függvény, mely az előbb részletezett textúraolvasást és bilineáris interpolációt elvégzi, majd visszaadja a távolságfüggvény becslését. A távolságfüggvényrácsból az értékeket a textúra bilineáris interpolációjával nyerem ki. Ez a fajta textúra-mintavételezés egy hardveresen támogatott művelet a GPU-n. Az alábbi függvény (1. algoritmus) a kiindulási pont és a sugár felülettel vett első metszéspontjának távolságát adja meg.

1. algoritmus. Sugárkövetés távolságmezőn

Funct DSFBoxTrace(P, d, F)

```
1: if P a dobozon kívül van then
2:     return −1                                  ▷ Távolság nem definiált
3: end if
4: if F(P) ≤ 0 then
5:     return 0          ▷ A doboz oldalán már a felületben vagyunk
6: end if
7: depth := 0
8: for i ← 1 to maxSteps do                       ▷ Maximum lépésszám
9:     dist := F(P + depth ∗ d)
10:    if abs(dist) < ε then
11:        return depth                           ▷ Eltaláltuk a felületet
12:    end if
13:    depth += dist
14:    currPos := P + depth ∗ d
15:    if currPos a dobozon kívül van then
16:        return −1;        ▷ Távolság nem definiált, nincs metszés
17:    end if
18: end for
19: return −1;               ▷ Távolság nem definiált, nincs metszés
```

3.10. Egyszerűbb távolságfüggvény-számító algoritmusok

Ebben a részben két egyszerűbb megközelítést mutatok be, melyek általánosan használhatók előjeles távolságfüggvények generálására.

Lipschitz-módszer

A módszer elméleti hátterét Bálint Csaba dolgozatából [6, 18. o.] idézem: „ha $f \in \mathbb{R}^n \to \mathbb{R}$ függvény Lipschitz-folytonos, akkor $|f|/\mathrm{Lip}(f)$ távolságfüggvény becslés. Általában az abszolút érték elhagyható, hogy előjeles távolságfüggvényt kapjunk. Ezzel egy módszert kaptunk arra, hogy egy implicit függvényhez hogyan gyártsunk távolságfüggvényt. Sokszor a $\mathrm{Lip}(f)$-et csak felülről tudjuk becsülni, vagy nem éri meg kiszámolni. Ilyenkor természetesen f-et a $\mathrm{Lip}(f)$ felső becslésével osztva egy rosszabb becslést kapunk, amivel sokszor lényegesen lassabb a számolás."

Brute force

A felületet valamilyen felbontáson kiértékeljük. A legközelebbi pont távolságát eltároljuk a távolságmezőben. A módszer hátránya, hogy a számítási igény négyzetesen nő a felbontás növelésével. (Amellett, hogy a háromdimenziós rács minden pontjára külön ki kell számolni.)

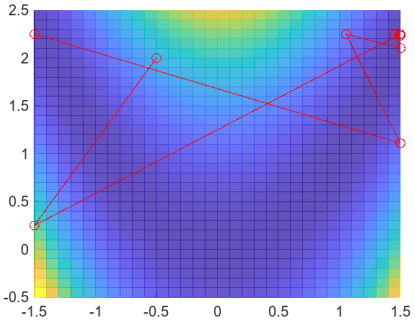
A módszer előnye, hogy a távolságmező (a felbontás limitációja mellett) pontos és maximális lesz. Ezen tulajdonság miatt a sugárkövetésnek sokkal kevesebb lépésre van szüksége a felület megtalálására. A bemutatott két módszer közül a brute force a preferált, hiszen a távolságmező előszámítható.

3.11. Gradiens módszerek

A felülettől vett minimális távolság valójában egy optimalizációs probléma. Optimalizációs problémák megoldásának nagyon széles szakirodalma van. Ilyen problémát kell megoldani többek között neuronhálók tanításánál is. 2020-ban távolságfüggvényekkel való ütközésdetektálás javítására is használtak lokális optimalizációt. [13] A távolságfüggvényekhez két, a gépi tanulásban elterjedt algoritmust is implementáltam.

Gradiens módszer

A gradiens módszer egy elsőrendű iteratív optimalizációs algoritmus. A módszer alapötletét egészen Cauchy-ig vezetik vissza [12], így egyáltalán nem újszerű. Az iteráció egy lépése: $a_{n+1} = a_n - \alpha \nabla F(a_n)$ Ahol F a minimalizálandó függvény, ∇ a gradiens-operátor és α a tanulási ráta. α egy kis konstans, ami a módszer konvergenciáját biztosítja. Ha túl nagyra állítjuk, akkor a módszer nem konvergál (4. ábra).



4. ábra. Gradiens módszer divergenciája túl nagy tanulási ráta esetén

Adam sztochasztikus gradiens módszer

Az Adam [11] egy sztochasztikus gradiens módszer. Minden lépésben adaptívan változtatja a tanulás paramétereit. Ezek közül m_t egy tapasztalati momentum, melyet a korábbi gradiensek exponenciálisan csökkenő súlyozásával kapunk. Hasonlóan számítandó v_t, mely a gradiensek négyzetének súlyozott átlaga, avagy a tapasztalati szórás. Legyen g_t a gradiens, ekkor

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2.$$

Ahol β_1 és β_2 a módszer hiperparaméterei. A paraméterek a gyakorlatban a kezdeti értékük felé (általában 0) statisztikai ferdeséget (bias) mutatnak, ezért korrigálni kell őket:

$$\hat{m_t} = \frac{m_t}{1-\beta_1^t}, \qquad \hat{v_t} = \frac{v_t}{1-\beta_2^t}.$$

Ezután egy lépés szabálya:

$$a_{n+1} = a_n - \frac{\alpha}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t.$$

A hiperparaméterekre a cikk szerzői ezeket az értékeket javasolják: $\alpha = 0.002$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ és $\varepsilon = 10^{-6}$. Ezeket szimulációs eredmények alapján állítottam be a saját alkalmazáshoz. Mivel a GPU-n a dupla pontosságú lebegőpontos értékek számításigényesebbek, így shader környezetben nagyobb ε-t kell használni a numerikus stabilitás érdekében.

AdaMax variáns

Ezt a módosítást a [11] cikk szerzői a cikk végén írják le. A lényege, hogy a szórás számításánál a kettes norma kicserélhető végtelen normára. Ezután a frissítés szabálya átalakítható max függvénnyé: (Itt v_t-t u_t-re cseréljük a megkülönböztethetőség kedvéért.)

$$u_t = \beta_2 u_{t-1} + (1 - \beta_2) \left\| g_t \right\|_\infty ,$$
$$u_t = \max(\beta_2 u_{t-1}, |g_t|).$$

Az AdaMax frissítési szabálya ennek felhasználásával:

$$a_{n+1} = a_n - \frac{\alpha}{u_t} \hat{m}_t.$$

Végül ezt a variánst implementáltam GPU-n, mivel a számítása egyszerűbb és numerikusan stabilabb, illetve mert néhány iterációval jobb eredményt ért el a szimuláció során.

Vetített gradiens módszer

Előfordul, hogy az optimalizációs problémát a paramétertér csak egy kis részén kívánjuk megoldani. Például Bézier-felületeknél általában a $[0, 1]$ intervallumon. Ebben az esetben nem elég az optimalizáció végén levetíteni az eredményt, ugyanis az a legtöbb esetben nem egyezik meg a tartományon vett minimumhellyel. [17] Az sem jó, ha a lépés megtétele után vetítjük le a pontot a tartományra, ekkor ugyanis a gradiens módszer helytelen adatokkal fog számolni.

Már a gradiens számításakor figyelembe kell venni a korlátokat. Legyen g a gradiens, $\pi_C()$ pedig egy függvény, ami minden pontot a hozzá legközelebbi C tartománybeli pontba visz. A vetített gradiens így számítható:

$$g_p = (a_n - \pi_C(a_n - \varepsilon g))/\varepsilon,$$

ahol ε egy kis szám. Ez úgy értelmezhető, hogy a gradiens irányába megteszünk egy kis lépést, majd azt levetítjük a tartományra. A vetített pont és az előző pont különbségéből megkapjuk a vetített gradienst irányát. Ezt odaadjuk a gradiens módszerünknek, majd a lépés után ismét levetítjük.

4. Szimulációs eredmények

GPU környezetben nehéz algoritmusokat tesztelni és hibákat javítani a masszívan párhuzamos környezet miatt, ezért az algoritmusokat egy szálon, Matlabban is implementáltam. A vizsgálat középpontjában az algoritmusok stabilitása és konvergenciája állt.

4.1. Gradiens módszerek stabil paraméterezése

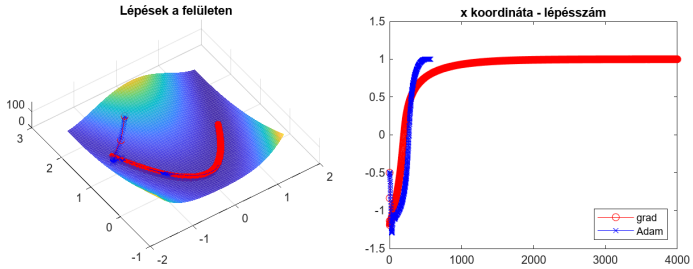
A gradiens módszer egyszerűsége ellenére meglepően jól teljesít Bézier-felületeken abban az esetben, ha a tanulási rátát (α) a lehető legnagyobb értékre állítjuk. Ekkor azonban nem tudunk garantálni semmiféle konvergenciát. Emiatt olyan hiperparaméter-beállítást keresünk, mely szélsőséges esetben is a lokális minimumhoz konvergál.

4.2. Rosenbrock-függvény

A Rosenbrock-függvény egy klasszikus „nehéz” példa, melyet optimalizációs algoritmusok tesztelésére szoktak alkalmazni. Definíciója:

$$f(x, y) = (a - x)^2 + b(y - x^2)^2.$$

Ennek a globális minimuma az (a, a^2) helyen van. Az a paraméter értéke általában 1. A b paraméterrel a probléma „nehézségét” lehet állítani. Minél nagyobb a b érték, annál nagyobb lesz a gradiensvektor hossza. Én 20-ra állítottam.



5. ábra. Stabil paraméterezés a Rosenbrock-függvényen

A gradiens módszer paramétere: $\alpha = 0,005$. Az AdaMax módszer paraméterei: $\alpha = 0,005$, $\beta_1 = 0,9$, $\beta_2 = 0,99$. Az 5. ábrán pirossal a gradiens módszer lépéseit, kékkel az AdaMax módszer lépéseit jelöltem. Az AdaMax módszer 569 lépésben 10^{-6} nagyságrendű hibával konvergál. A gradiens módszer hibája 1000 lépés után 1/10-nél nagyobb, és még 5000 lépés után is 10^{-4} nagyságrendű. Ebből a példából jól látszik a bonyolultabb módszer előnye, ha megköveteljük a konvergenciát nehezebb példákra is.

4.3. Globális minimum harmadfokú Bézier-felületen

A módszereket Bézier-felületeken is összehasonlítottam. A cél alapvetően a Bézier-felület távolságfüggvényének generálása. Ehhez a globális minimumot kell meghatározni. Ezt úgy kívánjuk elérni, hogy a lokális optimumkereső algoritmust több pontból elindítjuk.

10 pont módszer

Az első 9 indítási pont egy egyenletes 3×3-as felosztás a paramétertérben. Ezzel a felület nagyobb vonásait lefedjük. Azért, hogy a módszer a felület közelében is gyorsan konvergáljon, a mintavételezési pont alatti felületi pontból is elindítom a keresést. Mivel grafikonon vagyunk, ez megegyezik az első két koordinátával. Ennél sokkal kevesebb pont a tesztek alapján nem elég.

AdaMax algoritmus Bézier-felületeken

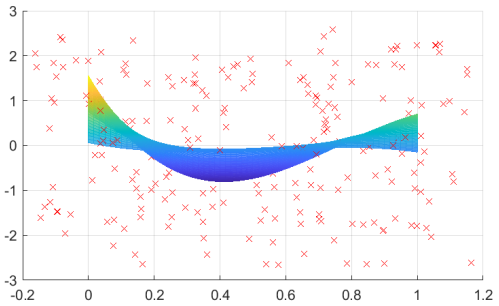
A 10 pont módszer helyességét szimulációval kívánom igazolni. Ehhez nagyszámú mintát generáltam, majd ellenőriztem, hogy a javasolt módszer minden alkalommal megtalálta-e a globális minimumot.

Felületgenerálás

A harmadfokú Bézier-felületek generálásához elég a 4×4 kontrollpontot megadni. Ehhez egyenletes eloszlással véletlen pontokat vettem a $[-1, 1]$ intervallumból. Az így keletkező felületek a közepükön „laposak" voltak. Ennek oka, hogy amíg a 0. és a 3. harmadfokú Bernstein-polinom maximuma az 1 értéket veszi fel, addig a középsők maximuma csak $4/9$. Ez különösen látványos a felület közepén, ott ugyanis két Bernstein-polinom szorzata áll, melyek maximuma $(4/9)^2$. Ezen tulajdonság miatt a középső kontrollpontoknak sokkal kisebb hatása van, mint a szélsőknek, ahol ráadásul interpolál is a felület. Emiatt minden kontrollpont harmadik koordinátáját leosztottam a hozzá tartozó Bernstein-polinom maximumával.

Mintavételezési pont generálás

A mintavételezési pontokat a felület bennfoglaló dobozából és annak környezetéből vettem. Ehhez egy egyenletes felosztású térrács pontjait random vektorokkal eltoltam. A 6. ábrán egy példa merőleges vetülete látható.



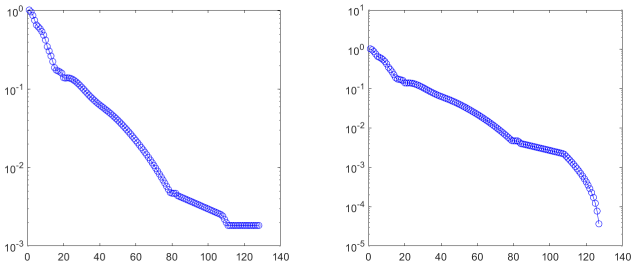
6. ábra. Felület és ponthalmaz képe

Referencia értékek

Referencia értékként kiszámítottam minden mintavételezési pontra a „brute force” módszer eredményét, azaz a távolságot egy $n \times n$-es rácson kiértékeltem. A tesztek alapján a hiba a műveletigénnyel arányosan csökken, pontosabban a felbontás (n) duplázásakor a hiba közel a negyedére csökken. A továbbiakban $n = 256$.

Validáció

A 10 pontos algoritmust 1000 különböző példára futtattam, majd összehasonlítottam a kapott minimumot a referencia értékkel. Mivel azt szeretnénk, hogy az algoritmus a *legrosszabb* esetben is megadja az optimumot, azért a grafikonon az összes futtatás közül a relatív hiba maximumát ábrázoltam minden egyes lépésszámra (7. ábra).



7. ábra. Abszolút hiba maximuma logaritmikus skálán. Balra a lépésszám függvényében, jobbra az iteratív módszerhez viszonyítva

A módszer minden futtatás esetén megtalálta az optimumot legalább a referencia érték pontosságával, kevesebb mint 120 iteráció alatt. A grafikon 110 lépés után azért laposodik el, mert az algoritmus eléri a referencia érték pontosságát. Ha referenciának az utolsó iteráció által adott távolságot vesszük, akkor látható, hogy a módszer tovább konvergál.

5. Implementáció

Ebben a részben néhány fontos implementációs részletre térek ki. Ennek motivációja, hogy a CPU implementáció profilozásakor kiderült,

az algoritmus az idő 60%-át a felület kiértékelésével és a gradiens kiszámításával tölti.

5.1. de Casteljau-algoritmus

Iterációval

Az algoritmus CPU implementációja két egymásba ágyazott ciklussal működik. Ez az 1. forráskódban látható.

Swizzle operátorokkal

Részben azért esett a választás a köbös Bézier-felületekre, mert a GPU kód támogat műveleteket legfeljebb 4 hosszú vektorokkal. Ez azt jelenti, hogy a felület kontrollponthálójának egy sora belefér egy ilyen vektorba. Az elméleti áttekintésnél láttuk, hogy a felület kiértékelése visszavezethető az egydimenziós problémára, így ezt a függvényt kell vektorizálni. Ezt a szintén hardveresen támogatott ún. swizzle operátorokkal tesszük. Ezek lényege, hogy a művelet elvégzése előtt a bemeneti vektorok elemeit tetszőlegesen átrendezhetjük, akár duplikálhatjuk is. A javított implementáció a 2. forráskódban látható.

A swizzle implementáció ciklusát kézzel is kibontottam. Ez nem okozott jelentős javulást. Végül lecseréltem minden sort egy lerp (lineáris interpoláció) utasításra. Az implementáció a 3. forráskódban látható.

1. forráskód. de Casteljau iterációval

```
float deCasteljau_iter(float4 pts, float t) {
    for (int j = 1; j <= 3; j++) {
        for (int i = 0; i <= 3 - j; i++)
        {
            pts[i] = (1-t) * pts[i] + t * pts[i+1];
        }
    }
    return pts[0];
}
```

2. forráskód. de Casteljau swizzle operátorokkal

```
float deCasteljau_swizzle(float4 pts, float t) {
    for (int j = 1; j <= 3; j++)
    {
        pts.xyz = (1-t) * pts.xyz + t * pts.yzw;
    }
    return pts[0];
}
```

3. forráskód. de Casteljau lerp függvénnyel

```
float deCasteljau_lerp(float4 pts, float t)
{
    pts.xyz = lerp(pts.xyz, pts.yzw, t);
    pts.xy = lerp(pts.xy, pts.yz, t);
    return lerp(pts.x, pts.y, t);
}
```

Mérési eredmények

Az algoritmusokat a „Brute force" távolságmező generáló módszerrel hasonlítottam össze, mert ez főként felületkiértékelést végez. Minden esetben egy 32^3 méretű textúrát generáltam, és kiátlagoltam 512 futtatást. Az 1. és 2. táblázatok a számításhoz szükséges GPU-időt foglalják össze milliszekundumban. Először a Bézier-felületből készített magasságtérképet textúrába írtam, majd a távolságmező generálásakor onnan kiolvastam. Az 1. táblázat második oszlopában ezek az értékek szerepelnek. A harmadik oszlopban a távolságmező generálásakor számítottam ki a függvényértékeket.

Referenciának kimockoltam a de Casteljau függvényt azzal, hogy csak az utolsó sort hagytam meg. A GPU Analyzer-ből kiderül, hogy ekkor teljesen eltűnik a függvény, mert a driver mindenhol inline-olja. Ezt az értéket kivontam a harmadik oszlopból, így megkaptam, mekkora része a futási időnek a felület kiértékelése.

A mérésekből kiderül, hogy a ciklusos implementáció nagyjából *50-szer lassabb* GPU-n, mint a többi számításos verzió. A második osz-

(ms)	Memóriából	Számítva	ebből de Cast.
két ciklusos	51,60	76,97	75,76
swizzle	51,35	2,74	1,53
kibontott	51,33	**2,63**	1,42
üres	51,41	1,21	0,00

1. táblázat. de Casteljau-algoritmus mérése egy 32^3 méretű textúrán

lopban az értékek nagyjából megegyeznek, a futási idők pedig jóval a táblázatban szereplő legjobb idő felett vannak. Ebből arra következtethetünk, hogy ha memóriából olvassuk ki a felületértékeket, akkor a limitáció a memóriaelérés által okozott késleltetés lesz. A legjobb megoldásokat összehasonlítottam egy nagyobb, 64^3 méretű textúrán is. Ennek eredményeit a 2. táblázat tartalmazza. A kézzel kibontott ciklus nem okoz lényeges javulást, viszont a lineáris interpoláció nagyjából 25%-kal gyorsabb.

(ms)	Számítva	ebből de Casteljau
swizzle	64,09	49,38
kibontott	62,13	47,42
lerp	**52,85**	38,14
üres	14,71	0,00

2. táblázat. de Casteljau-algoritmus mérése egy 64^3 méretű textúrán

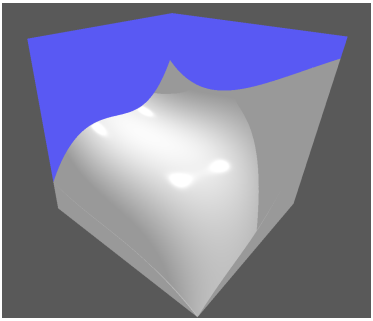
5.2. AdaMax

A AdaMax algoritmus kódja a pszeudokóddal teljesen ekvivalens. Kizárólag a numerikus stabilitásra kell figyelni. A vetített gradiens számításánál az ε érték nem lehet túl kicsi. $\varepsilon = 10^{-6}$ érték például már a végeredményben is látható numerikus hibákat okoz. Én $\varepsilon = 10^{-4}$ értéket használtam.

Szintén numerikus hibák és a zéróosztás elkerülése végett a tapasztalati szórás minimumát is 10^{-4}-re állítottam. Ez alapján u frissítési szabálya: $u_t = \max(\beta_2 u_{t-1}, |g_t|, 10^{-4})$.

6. Eredmények

Az alábbi mérési eredményeket ugyanazon tesztpéldákon végeztem el mindhárom módszerre. A használt videókártya egy Nvidia GTX 1060 Ti notebook GPU. A felbontás minden esetben Full HD.



8. ábra. Bézier-felület

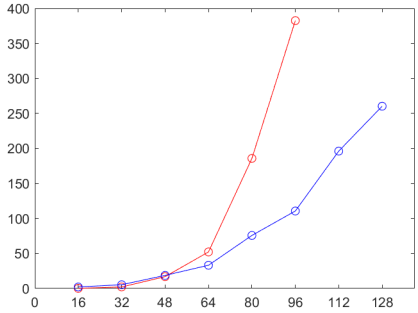
6.1. Távolságmező generálása

Mivel a Lipschitz-módszer csak egy gyenge alsó közelítést ad a távolságmezőre, így a Brute force algoritmust hasonlítottam össze az AdaMax algoritmust használó módszerrel. A 3. táblázatban az előbb bemutatott tesztpélda távolságmezőjének generálásához szükséges átlagos GPU-idők szerepelnek a felbontás függvényében, milliszekundumban. Az értékeket a 9. grafikonon meg is jelenítettem.

Felbontás	16	32	48	64	80	96	112	128
Brute Force	0.17	2.18	16.9	52.2	186	383	-	-
AdaMax	1.09	4.36	15.8	28.0	63.5	92.7	163	219

3. táblázat. A generálás ideje a felbontás függvényében, ms-ban

A 3. táblázat eredményeiből látszik, hogy a két módszer algoritmikus komplexitásából adódó különbségek már kis felbontás esetén (64^3) is jelentősek. A komplexitásbeli különbség oka, hogy amíg a Brute force

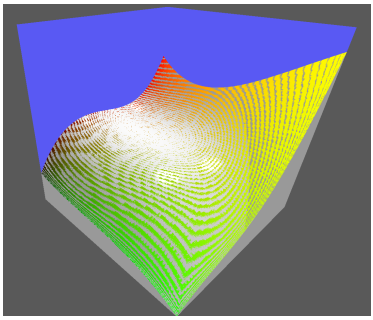


9. ábra. A generálás ideje a felbontás függvényében, ms-ban

megoldás a függvényt egyre növekvő felbontáson értékeli ki, addig az új módszer fix lépést végez 10 pontból.

6.2. Összehasonlítás a tesszellációval

A felületet a sugárkövetés mellett háromszögekkel közelítve is megjeleníthetjük. A két módszer eltéréseit úgy tudjuk vizualizálni, ha sugárkövetés után a fragment shaderben beleírjuk a mélységadatot a mélység bufferbe, és az egyik felület árnyalását valami egyszerűre cseréljük. A 10. ábrán a sugárkövetett felületet Blinn–Phong-árnyalással, a háromszögekkel tesszelláltat pedig egy egyszerű színskálával jelenítettem meg.



10. ábra. Tesszellált és sugárkövetett felület egyszerre kirajzolva

A 10. ábrán a két módszerrel kirajzolt felület színe váltakozva jelenik meg. A váltakozás oka, hogy amíg felületet tartalmazó voxelekben a sugárkövetés egy bilineáris felületet határoz meg, addig a raszterizációnál a felületet két, a csúcspontokban interpoláló háromszöggel közelítjük.

A raszterizációt felhasználhatjuk arra, hogy az első sugárkövetést megspóroljuk és csak az árnyékokhoz használjunk sugárkövetést. Ezt a technikát használják például az Unreal Engine-ben [16] is puha árnyékok számítására. A 4. táblázatban ezen javítás eredményei láthatók.

Generálási módszer	Átlagos GPU idő (ms)
Lipschitz	0,72
Brute force	0,51
AdaMax	0,50
Háromszögelés + AdaMax	0,24

4. táblázat. Sugárkövetés helyettesítése raszterizációval

7. Összefoglalás

Bemutattam és implementáltam a távolságmezők generálásának alapvető módszereit, majd áttekintettem a parametrikus- és Bézier-felületek elméleti hátterét. Fő eredményként bemutattam, hogyan használhatók gradiens módszerek távolságmezők generálásához és szimulációval validáltam az AdaMax algoritmus alkalmazhatóságát köbös Bézier-felületek távolságmezőjének generálására. Kitértem a módszer GPU implementációjának részleteire, mellyel nagyságrendekkel gyorsítottam a kiértékelést. Végül mérésekkel igazoltam, hogy az új módszer már kis felbontáson is lényegesen gyorsabb, mint a referencia implementáció.

Hivatkozások

[1] James F. Blinn, Models of light reflection for computer synthesized pictures, *ACM SIGGRAPH Computer Graphics,* 11(2) (1997), pp. 192–198.
`https://doi.org/10.1145/965141.563893`

[2] Csaba Bálint, Mátyás Kiglics, Quadric tracing: A geometric method for accelerated sphere tracing of implicit surfaces, *Acta Cybernetica,* 25 (2021), pp. 171–185.
https://doi.org/10.14232/actacyb.290007

[3] Csaba Bálint, Gábor Valasek, Accelerating Sphere Tracing, *Eurographics 2018 – Short Papers,* (2018), pp. 29–32.
https://doi.org/10.2312/egs.20181037

[4] Róbert Bán, Csaba Bálint, Gábor Valasek, Area lights in signed distance function scenes, *Eurographics 2019 – Short Papers,* (2019), pp. 85–88.
https://doi.org/10.2312/egs.20191021

[5] Róbert Bán, Gábor Valasek, Automatic Step Size Relaxation in Sphere Tracing, *Eurographics 2023 – Short Papers,* (2023), pp. 57–60.
https://doi.org/10.2312/egs.20231014

[6] Bálint Csaba, Távolságfüggvényekkel definiált felületek interaktív megjelenítése, *Országos Tudományos Diákköri Konferencia,* 2016.

[7] Gerald Farin, *Curves and Surfaces for CAGD: A Practical Guide,* Computer graphics and geometric modeling, Elsevier Science, 2002.
https://books.google.hu/books?id=D0qGMAwSUkEC

[8] Herman Hansson Söderlund, Alex Evans, Tomas Akenine-Möller, Ray tracing of signed distance function grids, *Journal of Computer Graphics Techniques (JCGT),* 11(3) (2022), pp. 94–113.
http://jcgt.org/published/0011/03/06/

[9] John C. Hart, Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces, *The Visual Computer,* 12(10) (1996), pp. 527–545.
https://doi.org/10.1007/s003710050084

[10] Benjamin Keinert, Henry Schäfer, Johann Korndörfer, Urs Ganse, Marc Stamminger, Enhanced sphere tracing, *Smart Tools and Applications in Graphics – Eurographics Italian Chapter Conference,* (2014), pp. 1–8.
https://doi.org/10.2312/stag.20141233

[11] Diederik P. Kingma, Jimmy Ba, Adam: A method for stochastic optimization, *3rd International Conference on Learning Representations ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings,* 2015.
http://arxiv.org/abs/1412.6980

[12] C. Lemaréchal, Cauchy and the gradient method, *Documenta Mathematica,* (2012), pp. 251–254.
https://www.math.uni-bielefeld.de/documenta/vol-ismp/40_lemarechal-claude.pdf

[13] Miles Macklin, Kenny Erleben, Matthias Müller, Nuttapong Chentanez, Stefan Jeschke, Zach Corse, Local optimization for robust signed distance field collision, *Proceedings of the ACM on Computer Graphics and Interactive Techniques,* 3(1):8 (2020).
https://doi.org/10.1145/3384538

[14] Morgan McGuire, Ray marching.
https://graphicscodex.courses.nvidia.com/app.html?page=_rn_rayMrch

[15] Inigo Quilez, Distance functions.
https://iquilezles.org/articles/distfunctions/

[16] Unreal Engine, Distance field soft shadows.
https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/RayTracedDistanceFieldShadowing/

[17] Niculae Vlad, Optimizing with constraints: reparametrization and geometry.
https://vene.ro/blog/mirror-descent.html

Rászoruló embertársaink megsegítése szociális média felületeken

Sztupák Raven Szilárd Zsolt

Eötvös József Collegium**

zsolt@raven.scot

Bevezetés

Az egyetem elvégzése után számos lehetőség adódik a volt hallgatóknak, hogy mit kezdjenek életükkel. Vannak, akik a szakmán kívül helyezkednek el, mások maradnak az egyetem csábító közelségében. Elég gyakori az is, hogy a korporét élet GDP termelési mutatói veszik le lábukról az embereket, akik emiatt úgy döntenek, hogy a magánszektorban folytatják pályafutásukat. Függetlenül attól, hogy milyen életpályát választunk, a megszerzett tudásunkkal ne legyünk önzők, és ahol tudunk, próbáljunk segíteni felebarátainkon.

Vannak erre kifejezetten szakosodott nemzetközi közhasznú szervezetek. Egyik ilyen a CodeBar [1], ami a technológiai szektorban alulreprezentált embertársainknak segít azzal, hogy keres nekik tapasztaltabb mentorokat a problémáik megoldásában. Sokáig volt Budapesten is aktív közösségük, azonban ez sajnos 2019 körül önkéntesek hiányában elapadt. Segítő szándékú fejlesztők jelentkezését nemzetközi szinten azonban szívesen várják, akár online, távmunkában is lehet nekik havonta 1–2 óra ráfordítással besegíteni. A szerző emellett természetesen nagyon reméli, hogy idővel a budapesti közösség is újra aktív lesz.

** 2004–2009

A másik hasonló nemzetközi kezdeményezés pedig a MigraCode [2], mely menekülteknek, és hátrányos szociális helyzetből származó embereket segít programozni tanítani. Nekik sajnos nincs kirendeltségük Magyarországon, de több országban aktív szervezetük is szívesen várja a távmunkában segíteni szándékozó jelölteket!

Azonban nem ez az egyetlen módja a segédkezésnek. Ha valaki szemfüles, és odafigyel, akkor szociális média felületeken is könnyedén találhat oltalomra szoruló illetőket. Az alábbi cikkben három példán keresztül mutatom be hogyan tudunk embertársaink támogatására sietni, ha szükségük lenne rá!

1. Kossuth Lajos azt üzente

`sztivan` felhasználónevű Tumblr felhasználó egy hűvős februári napon úgy érezte, hogy egy fontos információt szeretne a világról megtudni. Ez az információ pedig nem más, mint amit az 1. ábrán ábrázolt képernyőmentés ábrázol [3].

1. ábra. Problémafelvetés

Ez pontosan az a fajta kérdés, ahol a több évnyi Eötvös Collegiumban szerzett tapasztalatomat alkalmazni tudom. Aktív szociális életet élő lényként a Földrajz–Földtudományi Műhely tagjaival gyakran szerveztünk közös programokat, ami során megismerkedtünk egyik kedvenc[1] foglalkozásukkal, ami nem más, mint térképek digitalizálása. Ez a nagy szintű precizitást és monotóniatűrést igénylő feladat lehetővé teszi, hogy régi, papír alapú térképeken szereplő adatokat digitális eszközökkel kezeljünk, rajtuk különféle lekérdezéseket futtassunk le. Ezúton is szeretném köszönetemet kifejezni azokkal szemben, akik ezeket a tevékenységeket önerőből, gyakran anyagi kompenzáció nélkül elvégezték, és végül munkájuk eredményét ingyen elérhetővé tették a világhálón, főleg mert így ezt már nem nekem kell megtenni.

[1] gy.k.: gyűlölt

Nemzetközi viszonylatban a legismertebb ilyen digitalizációs projekt az EC Informatikai Műhelyével egyidős OpenStreetMap [4], mely célja egy egész bolygót lefedő, ingyenes térképészeti adatbázis létrehozása. Hasonló projekt a szintén 2004 óta létező magyar Turistautak [5] is, mely Magyarország és a környező országok turistaútjainak feltérképezését tűzte ki céljául. A két projekt sokáig egymás konkurenciájaként üzemelt, többek között a különböző licencfeltételeknek köszönhetően: míg a Turistautak térképének böngészése kizárólag non-profit céllal volt engedélyezett, addig az OpenStreetMap adatait kereskedelmi célokra is fel lehetett használni. 2015-ben történt egy változás, amikor a Turistautak szerkesztősége szintén teljesen szabaddá tette az adatokat, ami után a két projekt adatbázisa összefésülésre került. Ennek eredményeképpen az OpenStreetMap Magyarországgal kapcsolatos része is élvezhette a több évnyi, önkéntes digitalizációs munka eredményét.

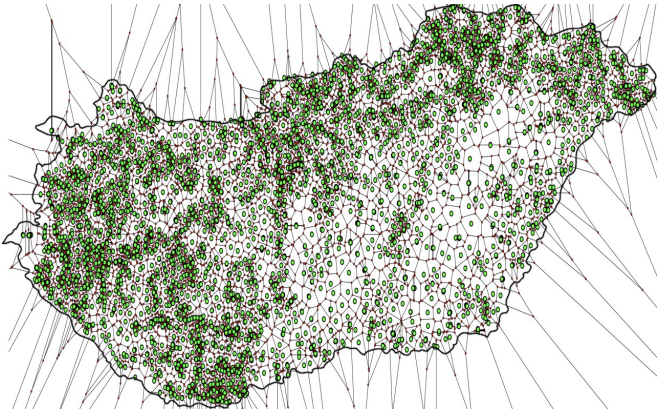
Ez nekünk pont kapóra jön, mert a kérdés megválaszolásához szükségünk lesz Magyarország összes Kossuth utcájának a listájára, amit az OSM adatbázisából fogunk beszerezni. Számos eszköz létezik, hogy az OSM adatait saját számítógépünkön kezeljük, az `osmosis`, valamint az `osmconvert` parancssoros, míg például a `QGis` grafikus feldolgozást tesz lehetővé. Mindegyik említett eszköz ingyenes, nyílt forráskódú. A 2. ábrán látható módon fogjuk őket használni, először, hogy az egész Magyarországot tartalmazó adatsort leszűrjük csak a nevesített utakra (amelyek az OSM rendszerben `highway` jelzéssel vannak ellátva), majd utána hogy ebből a szűrt adatokból egy CSV állományt nyerjünk ki a Kossuth utcák középpontjának GPS koordinátáival.

```
$ osmosis -read-pbf-fast hungary.osm.pbf
    file="hungary.osm.pbf" -way-key keyList=highway
    -way-key keyList=name -used-node -tag-filter
    reject-relations -write-xml file="hungary-roads.osm"
$ osmconvert hungary-roads.osm -all-to-nodes
    -csv="@id @lon @lat name" -csv-headline | grep -i
    kossuth > streets.csv
```

2. ábra. Kossuth utcák adatainak kinyerése az OSM adatbázisából

Amint kinyertük az utcák koordinátáit, keresnünk kell egy algoritmust, ami segít nekünk eldönteni, hogy hol találjuk meg a megoldást. Egy gyors Google használat sejteti, hogy nagy valószínűséggel egy

Voronoj-diagramot érdemes készíteni az utcák adatainak alapján [6]. A Voronoj-diagram ugyanis a teret úgynevezett Voronoj-cellákra osztja, ahol minden egyes kiinduló elemhez (a mintánkban minden egyes Kossuth utcához) pontosan egy cella tartozik. A cellák jellegzetessége, hogy azt a területet határolják körbe, amin belül minden egyes pont a cellához rendelt elemhez (azaz Kossuth utcához) közelebb van, mint akármelyik másik elemhez (Kossuth utcához). Ebből következik, hogy a cellák határain vannak azok a pontok, amelyek egyenlő távolságra vannak több Kossuth utcától is. Továbbgondolva: ezeken a határvonalakon lesznek a Kossuth utcáktól található legtávolabbi pontok, ezen területen kell keresnünk majd azt a pontot, ami a szomszédos Kossuth utcáktól a lehető legtávolabb van. Külön öröm az is, hogy a Voronoj-diagramok általában nagyon szépen néznek ki, feldobva ezzel a róluk szóló cikkek élvezeti értékét.



3. ábra. Kossuth utcák Voronoj-diagramja Magyarország felett

A 3. ábrán látható térképet nézegetve egyből szembetűnik, hogy a Tiszántúlon a ritkább népsűrűséghez ritkább Kossuth-koncentráció is tartozik. Naivan így azt gondolhatjuk, hogy az eredményt is majd ott kell keresni. Ha a fent leírt algoritmust lefuttatjuk, ténylegesen ott is fog nekünk eredményt találni, mégpedig a Hortobágyi Nemzeti Park környékén.[2] Ez a pont azonban nem a legtávolabbi pont, mivel algoritmusunkba egy hiba csúszott.

[2] Az algoritmus első futtatásakor Körösladány környékére tette az eredményt,

Első éves hallgatóként anno megtanultuk, hogy az egyik leggyakoribb programozási hiba nem gondolni arra, mit csinál a programunk, vagy algoritmusunk az adatsorok határán. Bár általában ezt képletesen kell érteni (tömbök esetén például a tömb elején és végén érdemes megvizsgálni jól működik-e a programunk), itt azonban tényleges határról van szó: ha ügyesek vagyunk észrevesszük, hogy nemcsak a Tiszántúlon, hanem a határ menti területeken is ritkábbak a Kossuth utcák. Itt azonban algoritmusunk nem működik kielégítően, hisz az csak a Voronoj-cellák metszéspontját vizsgálta eredetileg. A hiba javításához a határ menti cellákra rá kell illesztenünk Magyarország határvonalát, és a kettő metszésében található pontokat is meg kell vizsgálnunk.

Az algoritmust újra lefuttatva megkapjuk a tényleges nyertest: a szlovén–osztrák–magyar hármashatárpontot. Innen a legközelebbi Kossuth utca úgy 17 km-re található, Szentgotthárdon. Ha egy túra során el szeretnénk zarándokolni ide, a Kossuth utcáktól való félelmünkben, akkor örülhetünk, ugyanis ez a lokáció több okból is híres. Egyrészt hármashatárpont, másrészt ez Magyarország legnyugatibb koordinátája is. A helyszínen így egy kisebb pihenőt, játszóteret, valamint piknikezésre alkalmas padokat is találunk, hogy teljes mértékben kiélvezhessük túránk gyümölcsét. Azonban, ha egy kicsit nagyobb kihívásra vágyunk, a második helyen a Hortobágyi Nemzeti Park területén található ponton leszünk legmesszebb a Kossuth utcáktól, úgy 13 km-re. Itt azonban nem lesz semmi indikátora a helyszín fontosságának, mindenképp érdemes egy jól feltöltött, és precíz GPS lokátorra hagyatkoznunk a célpont megtalálásához.

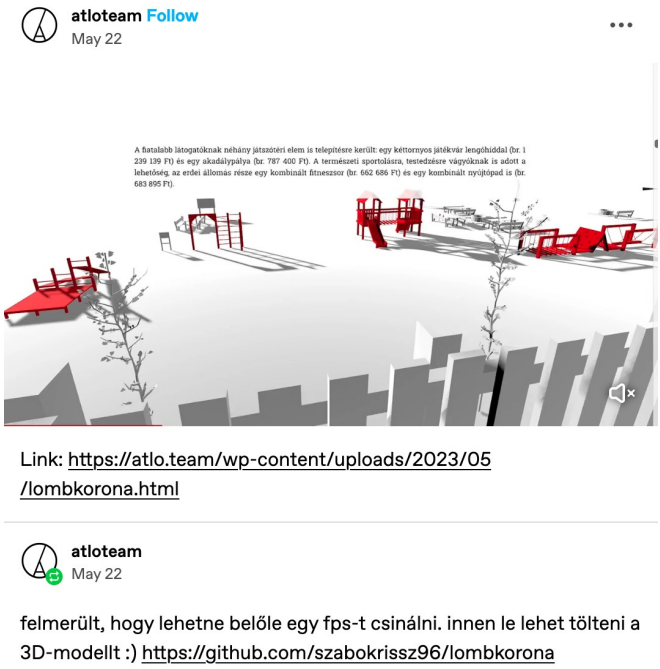
A teljes adatsor, valamint egyéb példák, mint például Petőfik, Rákóczyk vagy József Attilák megtekinthetők a projekt honlapján [7], ahol kiderül, hogy Magyarországon bárhol is vagyunk, lesz 12 km-en belül egy sportlétesítmény, az Egyesült Királyságban meg nagy átlagban – különösen Angliában – ha véletlenszerűen ledobnak egy repülőből[3], akkor 1 km-en belül fogunk találni egy kocsmát. Ami jól jön, ha esetleg egy kis frissítőre vágynánk a landolás után.

azonban alapos vizsgálat kimutatta, hogy a város vasútállomásának bekötőútját a közelmúltban átnevezték Vasút utcáról Kossuth-ra, mely átnevezésről az OpenStreetMap nem tudott. Szerencsére, mivel az OSM adatbázisa bárki által szerkeszthető, ez a hiba azóta már orvosolva lett a szerző által.

[3] remélhetőleg működő ejtőernyővel

2. A lombkoronák réme

2023. év elején nagy port kavart egy hírhedt falétesítmény az ország keleti felén [8]. Számos hírportál írt cikkeket, és vizsgálta az ügyet. Az Átlátszó adatvizualizációs csapata, az ATLO Team is készített a létesítményről egy inforgrafikát, melyet a 4. ábrán látható felhívással tettek közre [9].



4. ábra. A problémafelvetés

Természetesen kaptam az alkalmon, hogy megnézzem mit is lehet egy ilyen modellel tenni! 3D grafikával volt némi kapcsolatom már középiskolai éveim során is, amikor megjelentek a korabeli webes 3D rendszerek, például a VRML. 2D alapú játékfejlesztés mindig is érdekelt, több 2D alkalmazás fejlesztésében is részt vettem, de 3D grafikával

nem túl sokat foglalkoztam. Később, egyetemi éveim alatt bár jártam 3D grafikával foglalkozó extra kreditre, de szerintem csak azért sikerült azt teljesítenem, mert a vizsgán az első tételt húztam, ami bevezetésképpen csak kétdimenziós terekkel foglalkozott.

Mindezt összevetve kíváncsivá tett mennyit fejlődött a technológia az elmúlt 25–30 évben, azóta amióta először találkoztam a VRML nagyon korai példányaival. Úgy rémlik már akkor is sikerült benne egy nagyon-nagyon kezdetleges FPS-t[4] elkészítenem, bár sajnos már nincs ennek nyoma az archívumomban. Persze nem kell túl sokat gondolni: egy lapos szürke sík terep volt a talaj, és pár kockadarab jelezte a falakat rajta. Ellenfél nem volt, csak statikus gömbök (talán sárga színűek), amelyeket fel kellett venned. A falak pedig azt sem érzékelték, ha nekimentél, simán át lehetett siklani rajtuk. Persze ne feledjük el, ez még bőven a múlt évezred végén volt, amikor még az Internet Explorer és a Netscape Navigator volt a két legismertebb böngésző, és sokan gondolták, hogy a honlapokon majd Java alapú alkalmazások fognak futni. Szerencsére azóta már se Internet Explorer, se Netscape Navigator, és a Java alapú alkalmazásokat is csak az APEH – mai nevén NAV – vette komolyan.

No de lássuk, mennyi mindent fejlődött a világ úgy 25 év alatt: egy kis Google keresés kidobott egy Three.JS keretrendszer alatt írt FPS rendszert [10], amiben csak a terepet kellett lecserélni a nyírmártonfalvi lombkoronasétány modelljére, és 5 perccel később már kész is volt egy nagyon kezdetleges, de működő játék. Lehetett benne mozogni, lőszert felvenni a földről, lövöldözni, a tereptárgyaknak neki lehetett menni (és nem siklottál át rajtuk), valamint az ellenfél is próbált némi intelligenciát mutatni azzal, hogy megpróbált üldözni, meg csapkodni. Bár természetesen a grafika nem egy AAA kategóriás játékot idézett, lényegesen jobban nézett ki, mint bármi más, amit pár perc alatt, webes platformra össze lehetett hozni, mondjuk úgy 10 éve.

Az igazán érdekes rész viszont csak ezután következett. Bár egy valamennyire működő játék 5 perc alatt elkészült, azért ez nem volt túl nagy teljesítmény, tekintve hogy csak más emberek által ekészített projekteket hoztam össze egy kalap alá. A Telex is, akinek tudomására jutott a honlapom, elintézte az egész próbálkozást azzal, hogy „(csak) arról van szó, hogy valaki ráhúzta egy már létező kezdetleges, belső nézetes demóra a lombkoronasétány virtuális mását” [11]. Így írjon rólad

[4] belső nézetű lövöldözős játékot (first-person shooter)

szépeket a magyar média ugyebár. Pedig ők már a javított változatot látták, mert hát a projekt nem állt le, sőt most kezdett csak érdekessé válni az egész.

A rendszerben még volt bőven hiba, és mivel a játékmotort nem magam írtam, ezért ezek kijavításához először rá kellett jönni, hogy mi, hogyan, és miért úgy működik, ahogy. Szerencsére az eredeti kód nem volt túl komplex (bár nem is volt túldokumentálva), így sikerült hamar megfejteni a működésének alapjait, és a vizsgálgatás során lehetett tanulni egyet s mást a 3D grafika, valamint a játékfejlesztés alapjairól. A két nap, amit ebbe a projektbe fektettem, mindenképp többet tanított nekem, mint egy fél évnyi 3D grafika az egyetemen. Vagy lehet, csak izgalmasabb volt!

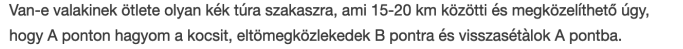
Néhány példa, hogy miket sikerült ez alatt a pár napos fejlesztés alatt tanulnom:

- A navmesh működése – 3D modell ami megmutatja az AI által irányított lényeknek, hogy hova mehetnek, és hogyan jutnak oda a leggyorsabban.
- Különféle textúrák működése webes környezetben, beleértve az égbolt és talaj rendes mintázatát.
- Különféle ütközésvizsgálati módszerek, statikus–dinamikus, illetve dinamikus–dinamikus modellek között.
- Játéktechnikai javítások – jobb AI kezelés, különféle nehézségi szintek, többfajta sebezési pontok támogatása.
- Webes hangkezelés 3D térben, valamint a játék kezelésének támogatása mobil környezetben.

Az elkészült projekt és a forráskód hozzá megtekinthető a szerző honlapján [12].

3. Másfél millió lépés Magyarországon, tömegközlekedve

A 3D-s kitekintő után egy újabb térképészeti feladat az 5. ábrán, ezúton `mindenfoglaltmar` nevű felhasználótól [15].



5. ábra. Problémafelvetés

Megint egy tökéletes feladat, ha esetleg túl sok határidős munkánk lenne, és egy kis agyi felfrissülésre vágynánk – miközben természetesen segítünk valakinek megoldani világraszóló problémáját. A fenti feladat hasonlóan térképészeti megoldást igényel, mint a Kossuth térképes, azonban más forrásokból kell beszerezni hozzá az adatokat. Az ötlet az az, hogy megpróbáljuk megtalálni az összes létező kombinációját az olyan túráknak, amelyek egy buszmegállóból indulnak, eljutnak egy másik buszmegállóba tömegközlekedve, onnan egy rövid sétával továbbvisznek egy kéktúrás pecsételőhelyhez, majd onnan elvezetnek a kiszemelt Kéktúra szakaszon keresztül vissza abba a buszmegállóba, ahonnan eredetileg is indultunk. Nézzük, honnan szerezzük be az adatokat ehhez.

- Szükségünk lesz a Kéktúrák teljes útvonalára. Ez elérhető a Kéktúra hivatalos honlapjáról GPX formátumban. Az egyszerűség kedvéért a három különböző Kéktúra vonalát `QGis` segítségével kézzel összefűztem egyetlen Magyarországot átszelő Kékkörré.

- Szükségünk lesz Magyarország összes buszmenetrendjére – vagy legalábbis azokra, melyek Kéktúra szakaszok közelében helyezkednek el. Mind a BKK, mint a Volánbusz ezt az adatsort ingyenesen elérhetővé teszi bárki számára a honlapján, GTFS formátumban. A MÁV oldalán sajnos ez nem szerepel, de ezt nem tekintjük túl nagy problémának. A Budapest környéki vasútvonalak, például a HÉV menetrendek így is szerepelni fognak a rendszerben, a BKK adatbázisa miatt.

- Végezetül turistautat kell találnunk a buszmegállók és a Kéktúra szakaszok kiinduló és végpontjai között. Erre megint az OSM adatbázisát fogjuk használni, onnan nyerjük ki a potenciális gyalogutakat.

Miután a nyers adatokat beszereztük, fel kell őket dolgozni. Először is minden, a Kéktúra szakaszok végpontjaihoz közel lévő buszmegállót

összegyűjtünk. Az egyszerűség kedvéért az egymástól csak pár száz méterre lévő buszmegállókat egynek tekintjük, mivel általában ezek az út két szemközti oldalán helyezkednek el. Miután kiszűrtük a buszmegállókat, lefuttatunk egy keresést, ami megkeresi a legrövidebb turistautat a buszmegállók és a Kéktúra szakaszok végpontjai között. A szerző több különféle technológiát is kipróbált, végül az adatokat egy PostGIS [13] adatbázisba töltötte be, és a `pg_routing` programcsomag segítségével kereste meg az eredményt, A* algoritmus segítségével.

A következő lépcsőfok a buszútvonalak kiderítése. A GTFS adatsort egy OpenTripPlanner [14] nevű programba töltjük be, ahonnan ezután lekérdezhetjük, hogy A és B pont között milyen útvonalak léteznek. Ezt a keresést szintén lefuttatjuk az összes buszmegállópár között (bizonyos távolsági kereteken belül), majd az eredményeket sorba rendezzük kritériumaink alapján, hogy megkapjuk az optimális buszmenetrendeket. Ilyen kritériumok például, hogy melyik a leggyorsabb, a legsűrűbb, vagy esetleg a legkevesebb átszállást, illetve gyaloglást igénylő útvonal.

Miután minden adat megvan már, csak egy honlapot kell fejleszteni, ahol a három különböző szekciót egy térképen ábrázolunk, kereshető formátumban. Persze az eredeti kérdés csak a 15–20 km-es szakaszokra vonatkozott, ha igazán hasznos honlapot szeretnénk készíteni, gondolunk olyanokra is, akik ennél csak kevesebbet, vagy ellenkezőleg, jóval többet szeretnének túrázni.

Mindenesetre az eredeti kérdésre a válasz: rengeteg ilyen útvonal létezik, Budapest környékén egyből három ilyen szakasz is van: A Dobogókő – Kevély nyereg, a Piliscsaba – Dorog, valamint a Mogyorósbánya – Dorog szakasz mind bejárható egy nap alatt egy rövid kis 15–20 km sétálással, de pár szakaszt leszámítva majdnem az egész Kékkör elvégezhető napi 20–25 km sétálással úgy, hogy mindig visszajutunk a kiindulópontunkra a nap végén.

A Kékkör kereső alkalmazás és forráskódja szintén megtalálható a szerző honlapján [16].

Utószó

Remélem, a fenti pár példa meghozta az olvasók kedvét arra, hogy érdekes problémafelvetések megoldására új technológiákat próbáljanak ki, különösen ha azok amúgy nem tartoznak a nap-mint-nap használt eszköztárukba. Még ha nem is sikerül megoldani a problémát, renge-

6. ábra. Dobogókő–Kevély nyereg túrajavaslat

teget lehet tanulni a sikertelen kísérletekből is. És természetesen ne feledjünk, hogy nem csak furcsa kérésekkel lehet másokon segíteni, a bevezetőben említett szervezetek folyamatosan várják önkéntesek jelentkezését. Tanulni vágyó illetők mindig lesznek, és mindig lesz szükség arra, hogy valaki tanítsa őket. Ötleteket, új technológiákat, érdekes problémákat ezen alkalmak során is bőven lehet gyűjteni.

Ha sokáig csináljuk ezeket a projekteket, előbb-utóbb utolér a hírnév is, így könnyen lehet, hogy amikor valaki megkérdezi tőlünk, hogy „te, meg tudod mondani, hogy melyik az a busz, ami a legtöbb Kossuth utcán halad át?”, és akkor erre már tudod is egyből a választ, hogy „természetesen, amit te keresel az a Nagykanizsa és Zalaegerszeg között közlekedő 6458-as busz, ami rekordmennyiségű, hét különböző Kossuth utcán is megáll, az egyiken kétszer is, a két végén”.

„Hétköznap érdemes felszállni rá reggel 6:25-kor, Nagykanizsán.”

A szerzőnek számos más projektje is van, különösen szeret térképészeti problémákat megoldani, és politikai indíttatású játékokat készíteni. Portfóliója megtalálhatók a honlapján [17].

Hivatkozások

[1] Codebar is a charity that facilitates the growth of a diverse tech community by running free regular programming workshops for minority groups in tech. https://codebar.io

[2] MigraCode Europe A European Network to promote Open Tech Education for Refugees and Migrants. https://migracode.eu

[3] Az észtek litvánok lettek, ID #182822065158
https://sztivan.tumblr.com/post/182822065158

[4] OpenStreetMap, a free, open geographic database updated and maintained by a community of volunteers via open collaboration. https://openstreetmap.org

[5] turistautak.hu, Magyarország leggyorsabban fejlődő nonprofit térkép-portálja
https://turistautak.hu

[6] Wikipédia-szerkesztők, 'Voronoj-cella', Wikipédia (2022.05.22.)
https://hu.wikipedia.org/w/index.php?title=Voronoj-cella

[7] Zsolt Sz. Sz. Raven, Determine furthest points away in a country for a specific set of points
https://sztupy.hu/kossuth-map/

[8] Sokszínű Vidék, Kivágott erdő helyén épült lombkorona-sétány Nyírmártonfalván
https://sokszinuvidek.24.hu/mozaik/2023/03/23/lombkorona-setany-nyirmartonfalva-nyirseg-hajdu-bihar-megye-erdo/

[9] ATLO Team, powered by ATLATSZO. ID #718041277567025152
https://atloteam.tumblr.com/post/718041277567025152

[10] Mohsen Heydari, Three FPS Demo
https://github.com/mohsenheydari/three-fps

[11] Flachner Balázs, Virtuálisan ugyan, de már mutánsokra is lehet lövöldözni a lomb nélküli lombkoronasétánynál
https://telex.hu/zacc/2023/05/24/lombkoronasetany-nyirmartonfalva-videojatek-lovoldozes

[12] Zsolt Sz. Sz. Raven, Canopy FPS
https://sztupy.hu/lombkorona/

[13] PostGIS extends the capabilities of the PostgreSQL relational database by adding support storing, indexing and querying geographic data.
https://postgis.net/

[14] OpenTripPlanner (OTP) is a family of open source software projects that provide passenger information and transportation network analysis services.
https://opentripplanner.org/

[15] SztupY, ID #728193481495003136
https://tumblr.sztupy.hu/post/728193481495003136

[16] Zsolt Sz. Sz. Raven, Kékkör szakasz kereső
https://sztupy.hu/kekkor-kereso/

[17] Zsolt Sz. Sz. Raven, Projects page
https://sztupy.hu/projects/

Szoftverek sérülékenységeinek azonosítása statikus elemzéssel

Tóth Melinda, Bozó István

ELTE Informatikai Kar
Programozási Nyelvek és Fordítóprogramok Tanszék

{toth_m,bozo_i}@inf.elte.hu

1. Előszó

Az *„Eötvös József Collegium Informatikai Műhelyében”* minden évben meghirdetjük a Funkcionális Programozás 2 nevű kurzust, ahol a hallgatók az Erlang programozási nyelvvel ismerkedhetnek meg. A tantárgyhoz szorosan kapcsolódik a RefactorErl kutatás-fejlesztési projekt, melyben Erlang programok statikus elemzésével és transzformálásával foglalkozunk, hallgatók intenzív bevonásával. A projekt kezdeti motivációját egy ipari projekt adta, mely Erlang programokhoz tervezett refaktoráló eszközt. Az utóbbi években a kutatásunk központjában az Erlang programok sérülékenységeinek a vizsgálata áll.

2. Bevezetés

Napjainkban egyre többet foglalkoztatja mind a szakembereket, mind a szoftverek felhasználóit a kiberbiztonság, kibervédelem, biztonság kérdése. Ezt a kutatási kérdést több szempontból is vizsgálhatjuk. Egyik ilyen a biztonságos szoftverfejlesztés kérdésének vizsgálata, amikor a forráskódot vizsgáljuk statikus eszközökkel, és olyan pontokat

próbálunk azonosítani, melyek a szoftvert sérülékennyé teszik, potenciális támadási kiskapukat vezetnek be a kódba.

A széles körben használt programozási nyelvekhez számos statikus elemző [5, 7, 14] biztosít ellenőrzőket. Ezek az ellenőrzők általában a jól ismert biztonságos kódolási szabályok betartását ellenőrzik.

A legismertebb ilyen szabvány az Application Security Verification Standard [10]. Ebben a leggyakrabban előforduló sérülékenységek között listázzák a nem megfelelő inputvalidációt, felhasználóazonosítást, jelszókezelést, adatelérések és hozzáférések kezelését, adatbázisok, fájlok, memória- és rendszerkonfigurációk nem megfelelő kezelését és hozzáférését.

Az Erlang [6] programozási nyelvet nagy rendelkezésre állással bíró, jól skálázható, hibatűrő, masszívan konkurens, elosztott rendszerek létrehozására tervezték. Eredetileg a telekommunikációs rendszerek elvárásainak [1] szerettek volna megfelelni, de azóta sok egyéb területen is bizonyította alkalmazhatóságát (pénzügyi, banki szolgáltatások, játékipar, egészségügy, IoT, web szerverek stb).

Az egyik leginkább emlegetett Erlang tulajdonság, a „Let it crash" filozófia. Ennek lényege, hogy hagyjuk, hogy a rendszerben futási idejű hibák váltódjanak ki, ezzel terminálјon a hibás kifejezést végrehajtó folyamat. Majd ezeknek a folyamatoknak a terminálásának az észlelésére, az úgynevezett processzekhez kapcsolódó hibák kezelésére, felismerésére pedig egy külön réteget biztosít az Erlang ökoszisztéma, így biztosítva a rendszerek stabil életciklusát.

Emellett fontos megjegyezni, hogy Erlangban nem jellemző az erőforrások, adatok megosztása (Shared Nothing), minden folyamat a saját izolált környezetében fut, saját erőforrásokkal rendelkezik. Mindemellett Erlangban a változók nem változnak, a funkcionális magnyelv részeként megírt szekvenciális függvényeink tiszták, a rendszerek modulárisan felépítettek, hibatűrőek. Ezek mind olyan tulajdonságok, melyek nagymértékben hozzájárulnak ahhoz, hogy a rendszerek jól ellen tudjanak állni a biztonsági támadásoknak.

Ennek ellenére sem mondhatjuk azt, hogy az Erlang rendszerekben nem fordulhat elő sérülékenység. Hiába készülünk fel ugyanis a folyamatok terminálásának kezelésére, bizonyos hibák ahhoz vezetnek, hogy az Erlang Virtuális Gép, a BEAM, vagy akár a rendszerünk omlik össze. Ezt pedig nem tudjuk forráskódból már kezelni, így az inputvalidációt sem tudjuk teljesen eliminálni. Illetve nem mehetünk el a változó kör-

nyezet hatásai mellett sem. Az Erlangot eredetileg úgy tervezték, hogy zárt, védett környezetben futó virtuális gépeken fut majd. Ez azonban a mai használatra már nem teljesül. Nem zárhatjuk ki továbbá az emberi tényezőt sem a szoftverfejlesztésből. Tapasztalatlan fejlesztők, vagy esetleg egy nem ismert szoftver nem megfelelő felhasználása is vezethet be biztonsági sérülékenységeket egy rendszerbe. Ezért fontos, hogy Erlang esetén is foglalkozzunk a biztonságos programozás kérdésével.

A RefactorErl projekt keretében Erlang programokhoz azonosítottunk a forráskódban sérülékenységi mintákat, statikus elemzési módszereket adtunk, melyekkel azonosíthatjuk meglévő kódokban a biztonsági réseket és elemzéseinket a RefactorErl [4] statikus elemző keretrendszerben megvalósítottuk.

3. Erlang programok sérülékenységei

Korábbi kutatásainkban [2,3], illetve az Erlang Ecosystem Foundation [9] keretében is egyre fontosabbá vált a biztonságos programozás támogatása az utóbbi években. Utóbbi csoport útmutatásokat definiált, melyeket az Erlang fejlesztőknek be kell tartani, figyelembe kell venni, amennyiben biztonságos programokat akarnak előállítani. A RefactorErl kutatócsoportban a jól ismert sérülékenységi típusokat próbáltuk meg Erlangra leképezni, illetve az Erlang nyelvi sajátosságai, a BEAM által nyújtott könyvtárak lehetőségeit figyelembe véve megadni az Erlangos sérülékenységi típusokat. Öt kategóriába soroltuk a biztonsági problémákat [2]:

1. Együttműködési képességből fakadó sebezhetőségek;
2. Konkurens programozásból eredő hibalehetőségek;
3. Elosztott programozásból adódó sérülékenységek;
4. Beszúrásos támadási típusok;
5. Memória túlterhelést eredményező támadási formák.

Az első csoportba olyan sérülékenységek tartoznak, melyek egy külső (nem Erlangban implementált) folyamat elindításával, kezelésével kapcsolatosak. Itt kell megemlíteni például az ellenőrizetlen NIF-ek

(Natively Implemented Function) Erlang virtuális gépbe való betöltését. Amennyiben ugyanis egy olyan, C-ben implementált kódot töltünk be, melynek a forrása ismeretlen, esetlegesen megváltoztathatta valaki, akkor az fenyegetés a rendszerre nézve.

A konkurens programozáshoz kapcsolódó hibák leginkább a versenyhelyzetek rossz kezelésére vonatkoznak. Például egy közös erőforrás használata esetén – egy ETS (Erlang Term Storage) tábla esetén – a konkurens írás és iteratív olvasások a tábla rögzítése nélkül nem várt eseményekhez vezethetnek.

Az elosztott programozási sérülékenységek a hálózat, kommunikáció paraméterezésének nem megfelelő módjából adódnak.

A beszúrásos támadások nagy része arra vonatkozik, hogy egy ellenőrizetlen, operációs rendszer szintű parancs végrehajtása során a nem ellenőrzött parancsok támadási felületet adnak, de említhetnénk itt az ellenőrizetlen fájlból beolvasott és végrehajtott kifejezéseket is.

A legutolsó kategória, mely az egyik leggyakrabban előforduló probléma, a memória túlterheléséből adódó támadási felület. Ennek egyik legnaivabb, ugyanakkor leggyakoribb formája az atom tábla túlterhelése, amikor is a nem szemétgyűjtött atom táblában létrehozott azonosítókból generálunk túl sokat, ezzel leterhelve és leálláshoz juttatva a virtuális gépet.

Mindegyik kategóriába több sérülékenységi típus lett besorolva. Ezek azonosítására a RefactorErlben külön ellenőrzőket adtunk meg, melyek külön-külön, illetve aggregálva is lekérdezhetőek [17].

4. Egy egyszerű példa

A szemléltetés kedvéért tekintsük az alábbi egyszerű kódrészletet. A `test` modulban található függvények közül a `secure1` és `secure2` függvények is biztonságosnak tekinthetőek, hiszen csak a programozó akaratának megfelelő statikus `cd` parancs hajtódik végre. Függetlenül attól, hogy ez jól láthatóan bele van írva a függvény törzsébe, vagy egy másik függvényt hív meg, ami előállítja ezt a hívást. Azonban az `insecure` függvény a `cd` parancs végrehajtását egy input argumentumként érkezett változóval egészíti ki. Ennek oka, hogy futási időben akarja definiálni, hogy melyik könyvtárba lépjünk be. Azonban semmilyen ellenőrzés nem történik az argumentum értékére, így az potenciális beszúrásos támadási pontot jelent.

```
-module(test).
-export([secure1/0, secure2/0, insecure/1]).

secure1() ->
  os:cmd("cd /tmp").

secure2() ->
  cd("/tmp").

cd(X) ->
  os:cmd("cd " ++ X).

insecure1(X) ->
  os:cmd("cd " ++ X).
```

Célunk, hogy olyan módszereket definiáljunk, melyek az első két esetet biztonságosnak, a harmadikat pedig sérülékenynek azonosítják.

5. Sérülékenységi elemzések a RefactorErlben

A RefactorErl [4] eszköz egy nyílt forrású statikus kódellenőrző és transzformáló eszköz Erlang nyelvhez. A programtranszformációk biztosítása mellett legfőbb feladata, hogy a fejlesztők mindennapos munkáját támogassa különböző funkcionalitásokkal a kódmegértés, kódellenőrzés terén is.

Ehhez az eszköz a forráskódot egy többrétegű gráf-modell segítségével reprezentálja. Ezt a gráfot nevezzük Szemantikus Programgráfnak – Semantic Program Graph (SPG) [11] –, mely egyaránt tartalmazza a szintaxisfát és a forráskód visszaállításához szükséges lexikális réteget, valamint egy szemantikus réteget. Ez utóbbi tartalmaz olyan információkat, mint a változók láthatósága, a függvényhívások azonosítása, közvetlen adatfolyam élek, hivatkozások stb. Elemzéseinket ezen eszközkészletre építve adjuk meg.

5.1. A sérülékenységek detektálásának algoritmusa

A sérülékenységek azonosítása három fő lépésre bontható:

1. A sérülékeny függvények definiálása, és az azokra vett hivatkozások azonosítása a forráskódban;
2. A sérülékeny függvényhívások argumentumai lehetséges értékeinek a kiszámítása;
3. Valós fenyegetések azonosítása, azaz a fals pozitív találatok kiszűrése.

Az első lépés magában foglalja azt, hogy definiáltuk és megadtuk a forráskódban fellelhető potenciális sérülékenységet hordozó függvényeket [2, 3]. Majd ezen azonosítókat és ezekre való referenciákat keresünk az SPG-ben a benne tárolt függvényhívási gráf alapján. Ez a gráf egyaránt tartalmaz statikus és dinamikus hívási információkat is, így a valós végrehajtás egy jó becslését biztosítja.

A második lépés során, a gyanús paraméterek értékének a kiszámítása során egy adatfolyam elérési relációt használunk [16]. Ezzel megbecsüljük egy adott ponton lévő kifejezés értékét, és azokat a pontokat a forráskódban, melyek értéke megegyezik a gyanús paraméter értékével. Erre azért van szükség, mert egy adott konkrét függvényhívás alapján nem tudjuk eldönteni, hogy az adott kifejezés valóban hordoz-e veszélyt, mert a gyanús értékek nem olvashatóak le kifejezés argumentumából. Ott gyakran csak egy változót látunk, mely értékének és eredetének az ismerete szükséges ahhoz, hogy a valós fenyegetést azonosítani tudjuk.

A harmadik lépés lényege, hogy az összegyűjtött információk alapján ezeket a szűréseket elvégezzük. A fals pozitív esetek szűrése során három fő lépést végzünk el:

- Először azonosítjuk azokat az eredet kifejezéseket, melyek ismeretlen helyről származnak, azaz olyan függvény állítja őket elő, melynek nem ismert a törzse. Ezeket a pontokat sérülékenynek kell tekintenünk, mivel az elemzésünk nem tudja az ellenkezőjét belátni.
- Azonosítjuk továbbá azokat a pontokat, ahonnan felhasználó által módosítható, megadható adat kerülhet a gyanús argumentumokba. Ezek például az exportált függvények, azaz azok, amit a modulon kívülről meg lehet hívni.

- A harmadik lépés ezen találatok közül azoknak a tovább szűrése, amit a felhasználó biztonságosnak definiált. Lehetőséget biztosítunk arra, hogy a felhasználó megadhassa az általa biztonságosnak vélt inputokat, ezzel segítve, pontosítva az elemzéseinket.

Ezen kívül, néhány ellenőrzés esetén plusz lépéseket is végrehajtunk. Például, az ETS tábla használatainak ellenőrzésekor megnézzük, hogy volt-e a tábla rögzítve, azaz kértek-e kizárólagos használatot. Ebben az esetben ugyanis nem tekintjük sérülékenynek az iteratív bejárásokat.

5.2. Sérülékenységi elemzések futtatása

A potenciális sérülékenységeket az ún. ellenőrzők (security checkerek) azonosítják a RefactorErlben. Ezek az ellenőrzők a RefactorErl szemantikus lekérdező nyelvén (SQ) keresztül érhetőek el [12]. Az SQ egy XPath-szerű lekérdező nyelv, mellyel a fejlesztők különböző entitásokat érhetnek el a forráskódban úgy, hogy azokra szemantikus kapcsolatokat, feltételeket fogalmaznak meg. Például a `mods.funs[name = foo].refs` lekérdezés megmutatja a foo függvény hivatkozásait.

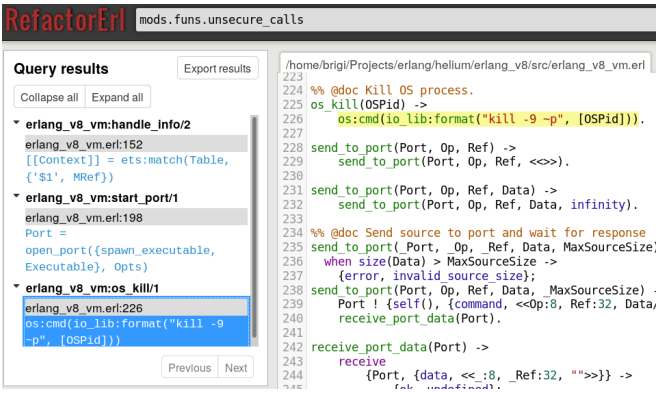
A biztonsági ellenőrzések a `mods.funs.unsecure_calls`, vagy a keresésben optimalizált `unsecure_calls` lekérdezéssel érhetőek el a legegyszerűbben. Amennyiben csak bizonyos típusú sérülékenységre vagyunk kíváncsiak, akkor a megfelelő ellenőrző (lásd 1. táblázat) megadásával tehetjük meg. Ha például az operációs rendszer szintű beszúrásos támadásokra vagyunk kíváncsiak a `foo` modulban, akkor azt a `mods[name=foo].funs.unsecure_os_call` lekérdezéssel tehetjük meg.

Megjelenítés

A RefactorErl számos felhasználói felülettel rendelkezik, ahol az eredményeket meg tudja jeleníteni. Az 1. ábrán láthatjuk, hogy egy lekérdezés eredménye hogyan néz ki az eszköz egyszerű parancssoros felületén. A 2. ábrán látható, hogy az eredmény hogyan jeleníthető meg a rendszer webes felületén. Utóbbi előnye, hogy az eredmény interaktívan nézegethető. Az Eredménylista egy elemére kattintva a felület megnyitja a sérülékenységet tartalmazó modul kódját, és kijelöli benne a gyanús kódrészletet.

```
(refactorerl@localhost)22> ri:q("mods.funs.unsecure_calls").
erlang_v8_vm:handle_info/2
    [[Context]] = ets:match(Table, {'$1', MRef})
erlang_v8_vm:start_port/1
    Port = open_port({spawn_executable, Executable}, Opts)
erlang_v8_vm:os_kill/1
    os:cmd(io_lib:format("kill -9 ~p", [OSPid]))
ok
(refactorerl@localhost)23>
```

1. ábra. Sérülékeny függvényhívások lekérdezése parancssorban



2. ábra. Sérülékeny függvényhívások lekérdezése webes felületen

Ellenőrzők

Jelenleg 15 különböző sérülékenységi típust azonosítottunk. Ezeket mutatja az 1. táblázat. Az ellenőrzőket az alapján soroltuk kategóriákba, hogy milyen jól ismert sérülékenységi típusba tartoznak.

Megjegyeznénk, hogy egy speciális kategóriát is definiáltunk, melyet „Deprecated”-nek neveztünk. Ezek olyan pontok a forráskódban, melyek lecserélendő vagy már lecserélt könyvtári függvényre hivatkoznak. Ennek a nagyon speciális kategóriának az elemei potenciálisan a „Denial-of-Service, DoS” kategóriába is besorolhatóak lennének, hiszen egy ilyen jellegű hiba általában a szolgáltatás kiesését eredményezheti, hiszen hiába indítunk újra egy folyamatot, az az újraindítás után is a nem létező függvényre fog hivatkozni.

Race condition	**Man-in-the-Middle**	**Deprecated**
unsafe_prioritisation	unsafe_crypto	deprecated
unsafe_linking	unsafe_network	removed
unsafe_ets_calls	unsafe_communication	to_be_removed

Injection	**Denial-of-Service**
unsafe_os	unsafe_atom
unsafe_port	unsafe_xml
unsafe_nif	unsafe_file_read
unsafe_port_driver	
unsafe_compile_load	
unsafe_file_eval	

1. táblázat. Ellenőrzők a RefactorErlben

6. Eredmények

Számos nyílt forrású projekten kipróbáltuk már az elkészült ellenőrzőket. Több mint félmillió sornyi Erlang kódot elemeztünk, és több száz sérülékenységet azonosítottunk. A jövőbeni munkáink egy része ezen találatok rangsorolására fókuszál, mellyel szeretnénk elérni, hogy a súlyosabb hibák nagyobb figyelmet kaphassanak.

A talált sérülékenységek túlnyomó része, közel kétharmada sorolható DoS (szolgáltatás elérhetetlenségi támadás) kategóriába sorolható. Azaz elmondhatjuk, hogy az atomok dinamikus generálása jelenti a legnagyobb fenyegetést. Általánosságban elmondható, hogy a korábban említett súlyossági besorolás itt is nagyon fontos, hiszen nem mindegy, hogy egy atomot generálunk, vagy iteratívan tesszük mindezt, egy iterációszámot a felhasználó által könnyen változtatható módon. Jelenleg az utóbbi esetben a sérülékenységet magas (*high* severity) prioritásúként kezelünk, az egyszeri alkalmazást pedig alacsony (*low* severity) prioritású sérülékenységnek jelölünk meg.

A talált támadások következő nagy csoportja a beszúrásos támadások kategóriája volt.

Az elemzéseink során arra jutottunk, hogy a fals pozitív találatok szűrése eredményes. A találatoknak átlagosan a harmadát tudjuk ki-

szűrni. Azonban a találatok elemzése során további pontokat azonosítottunk a szűrések finomhangolására. Ilyen például bizonyos könyvtári függvényekről való információk beépítése az elemzésbe, mely segíti az ismeretlen törzsű függvények további szűrésének pontosítását. Ezenkívül azonosítottunk végrehajtási utak elemzése mentén elvégezhető fals pozitív szűrési lehetőségeket is. Ezek kidolgozásán jelenleg is dolgozunk.

7. Kapcsolódó kutatások

A biztonsági hibák korai azonosítása egy fontos feladat. Ezt a feladatot tudják a különböző statikus ellenőrzők ellátni. A népszerű programozási nyelvekhez számos eszköz létezik. Például a SonarQube [5], a SpotBugs [14] vagy éppen a CodeChecker [7], hogy párat említsünk közülük. Jellemzőjük, hogy már a fejlesztői környezetekben is elérhetőek (IntelliJ, Visual Studio Code, Eclipse stb), illetve a CI folyamatok részeként is beépíthetőek. Erlanghoz nem érhető el hasonló eszköz.

Azonban az Erlang közösségben is jelentős figyelmet kapott a biztonság kérdése az utóbbi években. Cikkek [8,13] és kezdetleges eszközök is megjelentek [15]. A PEST előnye, hogy egy gyors szintaktikus ellenőrzéssel azonosít számos sérülékenységet a forráskódban, de a statikus ellenőrzés hiányában az eredményei sok fals pozitív találatot tartalmaznak. Illetve az általa detektált sérülékenységi típusok nem fedik le a RefactorErl által megtalált sérülékenységeket. Általánosságban elmondhatjuk, hogy a RefactorErl statikus szemantikus elemzés alapú ellenőrzői egy jobb megközelítést adnak.

Említésre méltó még az a kezdeményezés, mely bevezeti a „trust zone” fogalmát [8]. Az inputok sérülékenységeinek definiálását a felhasználóra bízza a biztonságos zónák kijelölése által, és a nem biztonságos irányból érkező adatok terjedése alapján határozza meg azt, hogy hol lehet sérülékenység. Ez a munka szintén a RefactorErl elemzéseit használja.

8. Összegzés

A szoftverek biztonsági, kibervédelmi kérdése napjaink széles körben kutatott területe. A biztonság növelésének egy módja a biztonságos programozás, biztonságos szoftverfejlesztési módszerek, támogató esz-

közök alkalmazása. A statikus elemzési módszerek kiváló eszközkészletet adnak a biztonsági sérülékenységek azonosítására még a szoftverek üzembe helyezése előtt.

A RefactorErl kutatócsoportban Erlang programok statikus elemzésével foglalkozunk, ennek keretében pedig vizsgáljuk, hogyan tudjuk biztonságos Erlang programok fejlesztését támogatni. Ebben a cikkben azt mutattuk be, hogy milyen Erlangos sérülékenységeket azonosítottunk, és hogyan tudjuk ezeket a forráskódban azonosítani. A módszereinket a RefactorErl nyílt forrású statikus elemzőben meg is valósítottuk.

Hivatkozások

[1] Joe L. Armstrong, Making reliable distributed systems in the presence of software errors, 2003.
`https://api.semanticscholar.org/CorpusID:28795665`

[2] Brigitta Baranyai, István Bozó, Melinda Tóth, Supporting Secure Coding with RefactorErl, *Talk at the 19th ACM SIGPLAN International Workshop on Erlang, Virtual Event,* **23** (2020).

[3] Baranyai Brigitta, *Funkcionális nyelvek és a statikus kódelemzéssel támogatott biztonságos szoftverfejlesztés,* TDK dolgozat, ELTE, 2021.

[4] I. Bozó, D. Horpácsi, Z. Horváth, R. Kitlei, J. Köszegi, M. Tejfel, M. Tóth, RefactorErl — Source Code Analysis and Refactoring, *Proceedings of the 12th Symposium on Programming Languages and Software Tools, Tallinn, Estonia,* ISBN 978-9949-23-178-2 (2011), pp. 138–148.

[5] G. Ann Campbell, Patroklos P. Papapetrou, *SonarQube in Action,* Manning Publications, 2013. ISBN-978-161-72-90-95-4

[6] Francesco Cesarini, Simon Thompson, *Erlang programming,* O'Reilly, ISBN-978-0-596-51818-9, 2009.
`http://shop.oreilly.com/product/9780596518189.do`

[7] CodeChecker Documentation, 2024.
`https://codechecker.readthedocs.io/`

[8] Viktória Fördős, Secure Design and Verification of Erlang Systems, *Proceedings of the 19th ACM SIGPLAN International Workshop on Erlang (Virtual Event, USA),* ISBN-978-1450380492 (2020), pp. 31–40.
https://doi.org/10.1145/3406085.3409011

[9] Erlang Ecosystem Foundation, *Secure coding and deployment hardening guidelines,* 2024.
https://erlef.org/wg/security

[10] The OWASP Foundation, *Application Security Verification Standard (4.0),* 2019.
https://owasp.org/www-pdf-archive/OWASP_Application_Security_Verification_Standard_4.0-en.pdf

[11] Zoltán Horváth, László Lövei, Tamás Kozsik, Róbert Kitlei, Anikó Nagyné Víg, Tamás Nagy, Melinda Tóth, Roland Király, Modeling semantic knowledge in Erlang for refactoring, *Knowledge Engineering: Principles and Techniques, Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEPT 2009, Studia Universitatis Babeș-Bolyai, Series Informatica, Cluj-Napoca, Romania* **54** (2009), pp. 7–16.

[12] László Lövei, Lilla Hajós, Melinda Tóth, Erlang Semantic Query Language, *Proceeding of 8th International Conference on Applied Informatics, ICAI 2010, Eger, Hungary,* ISBN 978-963-98-94-72-3 (2010), pp. 165–172.

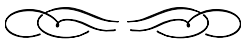
[13] Alexandre Jorge Barbosa Rodrigues, Viktória Fördős, Towards Secure Erlang Systems, *Proceedings of the 17th ACM SIGPLAN International Workshop on Erlang, St. Louis, MO, USA,* ISBN-978-1450358248 (2018), pp. 67–70.
https://doi.org/10.1145/3239332.3242768

[14] SpotBugs Documentation.
https://spotbugs.readthedocs.io/

[15] Michael Truog, *Primitive Erlang Security Tool (PEST),* 2024.
https://github.com/okeuday/pest

[16] Melinda Tóth, István Bozó, Static Analysis of Complex Software Systems Implemented in Erlang, *Central European Functional Programming Summer School – Fourth Summer School, CEFP 2011, Revisited Selected Lectures, Lecture Notes in Computer Science (LNCS),* ISSN: 0302-9743, Springer-Verlag, **7241** (2012), pp. 451–514.

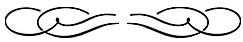
[17] RefactorErl Wiki Page, *Detecting Vulnerabilities,* 2024. `http://pnyf.inf.elte.hu/trac/refactorerl/wiki/howto#Detectingvulnerabilities`



– Mester, ez hazugság! – fordul hozzám a 326-os fiatalabbik informatikusa szobájából kilépve, karikás szemekkel, miközben a folyosó végi fotelok egyikén ülve a reggeli müzlimet majszolom békésen, kezében egy „JavaScript 24 óra alatt" könyvet szorongatva. (A *Tanuljuk meg a . . . -t 24 óra alatt* könyvsorozat szerzői e könyveket 24 leckére, „órára" tagolták, melyekből tipikusan hetente néhányat elvégezve pár hét alatt egész jól elsajátíthatjuk az adott programozási nyelv vagy technológia alapjait.)

– No, mi az, Józsi? Nem működik? Esetleg hibás valamelyik példakód?

– Mi? Ja, nem... Viszont ezt 21 óra alatt is meg lehet csinálni!



Névjegyek

Ambrus Tamás

EJC: 2012–2015

Nekem a Collegium nagy lendületet biztosított a tanulmányaim során. Amikor jelentkeztem tagnak, akkor egyúttal egyéb, nem szakkollégiumokba is beadtam a jelentkezésemet, hiszen vidékről jöttem fel Budapestre tanulni, és egy kollégiumi hely volt a legegyszerűbb módja a lakhatásnak. Egyszer csak eljött az a pont, amikor kiderült, hogy a sima kollégiumok egyikébe sem vesznek fel engem, viszont még folyamatban volt a Collegiumi felvételi folyamat. Ekkor már azon is stresszeltem, hogy „jaj, legalább ide vegyenek fel, ne kelljen albérletbe mennem", de főként azon, hogy itt valószínűleg nincs helyem, ide olyanok jelentkeznek, akik sokkal többet foglalkoztak informatikával korábban, mint én. Nem igazán gondoltam még arra, hogy milyen jót is tenne velem az, ha tag lehetnék.

Aztán csodák csodájára felvettek! Bár sokaknak tényleg jelentősebb múltja volt az informatikával, részemre is találtak helyet a Collegiumban. És ez nagyszerű, mert a kevés tapasztalatommal, de annál nagyobb lelkesedésemmel nekem is lehetőségem nyílt a lehető legjobban fejlődni: az, hogy másokkal együtt tanultunk, jegyzeteltünk, csináltunk leckéket, készültünk vizsgára, versenyeztünk egymással, lazítottunk; tehát éveket tölthettem el velem hasonló mentalitású, jó képességű hallgatókkal, az egy nagyon jó irányba sodort engem, minőségi idő volt.

És bár én a mesterszak elvégzése után már nem maradtam tanulmányi-kutatói pályán, maradandó nyomot hagyott rajtam a Collegium tagjaként szerzett tudás, lelkesedés, igény.

A mesterszak vége óta, azaz közel 5 éve én egy Interactive Brokers nevű pénzügyi cégnél dolgozom Java programozóként. Itt – talán mivel pénzügyi cég – kevésbé van azon a hangsúly, hogy minél újabb, minél változatosabb programozói eszköztárral dolgozzunk; például lambda kifejezések (Java 8) is csak elvétve fordulnak elő a kódbázisunkban. Ellenben nagyon figyelünk arra, hogy minél kevesebb hibát vigyünk be a rendszerbe, azaz kiemelt jelentőségű nálunk a minőség.

Kellemes munkahely, nekem tökéletes; de a figyelmem valójában a családom felé irányult az elmúlt években és irányul most is. Hiszen nemrégiben megházasodtam, és néhány nappal ezelőtt megszületett első gyermekünk is – úgyhogy jelenleg boldog éveimet töltöm!

Bagladi Milán Zsolt

EJC: 2023–

Bagladi Milán Zsolt vagyok, az ELTE Informatikai Karának hallgatója és az Informatika Műhely tagja. Lentiben gyerekeskedtem, és a Lenti Arany János Általános Iskola diákja voltam. A karrierem kezdetét nagyjából felső tagozat elejére teszem, hiszen ekkor kezdtem el az informatikával mélyebben foglalkozni. Kósáné Robb Olga tanárnő délutáni szakkörein örömmel vettem részt, ezeken az alkalmakon nyílt lehetőségem a versenyzésbe is belekóstolni. Elkezdtem versenyekre járni, melyeken rendre szép eredményeket értem el. Legjobban a grafikus programozás (LOGO), illetve az alkalmazói feladatok tetszettek, így ennek megfelelően a legkiemelkedőbb eredményeim a Kozma László Országos Informatikai Versenyhez és a Nemes Tihamér Nemzetközi Informatikai Tanulmányi Verseny alkalmazói kategóriájához fűződnek. Nyolcadik osztály végén az Arany János Alapítvány kuratóriuma az iskola legrangosabb kitüntetésével, az Arany János-díjjal ismerte el munkámat.

A középfokú tanulmányaimat Nagykanizsán, a Batthyány Lajos Gimnáziumban végeztem. Az iskolát Erdősné Németh Ágnes tanárnő ajánlotta nekem, akivel egyébként egy országos verseny döntő fordulóján találkoztam először. A beszélgetés után nem maradt bennem kétség. A Batthyány Lajos Gimnáziumban kiemelkedő lehetőségek nyíltak meg számomra a matematika és az informatika területén. Matematikából a Rátz Tanár Úr Életműdíjas, egykori Eötvös Collegista Erdős Gábor Tanár úr tanított. Egy kisgimnazista osztályba kapcsolódtam be, így komoly munka volt a felzárkózás az első évben. Úgy vélem, ez sikerült. A délutáni matematika és informatika szakkör és az Erdős Iskola foglalkozásai rengeteget segítettek a szakmai fejlődésben. A közös munka gyümölcse a rengeteg szép megyei versenyeredmény mellett 11. osztályra érett be, hiszen a csoportból egyedül én jutottam tovább a matematika II. OKTV második fordulójába.

Középiskolás éveim során sikerrel szerepeltem számtalan számítástechnikai versenyen, különös tekintettel a Nemes Tihamér és az OKTV versenyekre. Úgy vélem, hogy sokrétű tudáshoz jutottam a gimnazista évek alatt, mely kitűnő ugródeszka volt a felsőfokú tanulmányaim megkezdéséhez.

A középiskola végén a Batthyány Lajos Gimnázium vezetősége és nevelőtestülete az iskola legrangosabb kitüntetésével, a Batthyány Emlékláncccal ismerte el munkám. A Nagykanizsa város és a Nagykanizsai Tankerületi Központ által szervezett Városi Tanévzáró rendezvényen a felkészítő tanáraim és jómagam munkáját oklevéllel jutalmazták.

Az ELTE Informatikai Karán programtervező informatikus szakon vagyok jelenleg hallgató. Az egyetemi feladatok mellett közreműködök néhány szervezési, lebonyolítási feladatban is, és lehetőségem nyílt néhány szakmai csoport munkáját is segíteni.

Érdeklődési területem közé tartozik a mesterséges intelligencia, illetve a különféle kódolási módszerek matematikája. Az idei tanévben sikerült egy kutatói pályázat keretében mélyebben foglalkoznom a mesterséges intelligenciával és az emberi karjelzések értelmezésének kérdésével. A kutatási projekt jelenleg is tart, így ennek eredményeiről jelen írásban még nem számolok be.

Az Eötvös Collegiumot évekkel ezelőtt ajánlotta nekem az informatika tanárom, ennek megfelelően jelentkeztem a IX. TermTudTáborba. A jelentkezésem melléklete egy autós oktatójáték volt, amely elnyerte a szervezők tetszését, így a részvételi díjam elengedésre került. A felejthetetlen élmények motiváltak arra, hogy a X. Eötvös József Tehetséggondozó Táborban is részt vegyek. A Collegiumban az említett eseményeknek köszönhetően sikerült számtalan baráti kapcsolatot kialakítanom.

A Collegiumba 2023-ban nyertem felvételt, elsőéves egyetemistaként. A felvételi eljárás során bejáró státuszt kértem, ugyanis nem szerettem volna elvenni más, nehezebb helyzetben lévő testvérünktől a helyet. A bejáró státusz sajnos nehézségeket is hordoz magában, hiszen a közösséggel való kapcsolattartás bejáróként jóval nehezebb, továbbá a Collegium programjain való részvételhez nekem utaznom szükséges, ám mégis úgy vélem, hogy megéri a Collegium tagjának lenni.

A Collegiumban nagy sikerrel működik az ALFONSÓ nyelvoktatási program is. Én angol nyelvet kezdtem el tanulni, hiszen ebből a nyelvből még nem próbálkoztam felsőfokú nyelvvizsgával, és úgy gondoltam,

hogy ez egy megfelelő út erre.

Az Informatika Műhely tagjaként mindig örömmel veszek részt a műhelyprogramokon. A kutatószemináriumokon a felsőbb éves collegisták mutatják be a szakterületüket. Az idei évben hallgattunk már előadást a számítógépes grafikáról, a virtualizáció működéséről, a LiDAR technológia által szolgáltatott adatok feldolgozásáról, és mindezeket úgy, hogy még vége sincs a félévnek. Sokat jelent az, hogy meghallgathatom, mások milyen munkát végeznek a kutatásuk során, így ez nekem is támpontot jelent a saját témám szempontjából.

A szakmai munkát, az immáron 20 éves Informatika Műhely támogatásával, kutatói területen tervezem folyatni, továbbá az egyetemi munka során is törekszem a minél jobb eredmény elérésére.

Balassi Márton

EJC: 2009–2014

mbalassi@apache.org

2009 és 2014 között voltam a Collegium bentlakó tagja, ebben az időszakban végeztem el az ELTE programtervező informatikus BSc és MSc képzését, illetve a BME (azóta megszüntetett) alkalmazott közgazdász képzését. Ezen sorok írásakor az Apple-nél vezetek egy elosztott rendszerekkel foglalkozó nemzetközi mérnök csapatot.

A Collegium szellemisége, tanárai és tagjai meghatározó szerepet játszottak abban, hogy tanulmányaim sikeres teljesítése mellett legyen ambícióm, energiám és időm választott kutatási témámmal foglalkoznom. Collegiumi pályafutásom kezdetén ökoszisztémák szimulációja volt ez a téma, amin keresztül megtanultam a programozás és kutatás szeretetét. Idővel kutatótársammal rá kellett döbbenünk, hogy az ifjúsági versenyeken való jó szereplésen[1] kívül mérsékelt témánk gyakorlati hasznossága.

Harmadéves hallgatóként kezdtem el foglalkozni elosztott, adatintenzív, úgynevezett „Big Data" rendszerekkel, és ennek kapcsán csatlakoztam az MTA SZTAKI Informatikai Kutatólaboratóriumához. Itt az a szerencse ért, hogy az elsők közt dolgozhattam az azóta Apache Flinkként ismertté vált nyílt forráskódú rendszeren. Flinket használ többek között a Netflix ajánlórendszere, az Uber árazása, illetve több ismert pénzügyi intézet csalásmegelőzési rendszere ahhoz, hogy valós időben tudják kiszolgálni a lekérdezéseiket. 2022-ben a Flink kapta az ACM SIGMOD Systems Awardját.

2016-ban döntöttem úgy, hogy a kutatás helyett az ipari pályát választom. Kezdetben „Big Data" tanácsadóként dolgoztam Európaszerte, majd a Cloudera nevű szoftvercégnél vállaltam el a Flink termékük és csapatuk kiépítését, illetve vezetését Magyarországon. Két évvel

[1] Többek között eljutottunk az OTDK-ra és az EU Fiatal Tudósok Versenyére.

ezelőtt kerestek meg az Apple-től, hogy ismételjem meg ezt az ő belső adatplatformjuk számára Ausztriában. A mai napig igyekszem követni témám kutatási eredményeit, többször építek rájuk napi munkámban.

Magánéletem mindennapjait szerető feleségem és örökmozgó kislányunk édesítik meg, jövő hónapra várjuk kisfiunk érkezését.

Benics Balázs

EJC: 2015–2021

Egyetemi éveim alatt hosszan az EJC Informatikai Műhely tagja voltam. Mindvégig a fordítóprogramok foglalkoztattak, szorosan összefonódva a C++-szal. Megismerkedtem Horváth Gáborral (aki szintén kapcsolódik az EJC-hez), majd rajta és Porkoláb Zoltánon keresztül az ELTE Szoftver laborral. A laborban C++ kódok statikus kódelemzésével foglalkoztunk, a Clang C++ frontendre alapozva. A témában írtam a szakdolgozatom és a diplomamunkámat is. Foglalkoztam a „strict-aliasing" szabállyal [1], illetve taint analízissel [2] a tanulmányaim során. Az AST-matching után a szimbolikus végrehajtás foglalkoztatott, melyek különböző kódelemzési technikák. Manapság főleg az utóbbira koncentrálok. A labor alatt az Ericsson CodeChecker csapatát erősítettem, majd az egyetemet követően a genfi SonarSource [3] cégnek dolgozom, továbbra is Budapestről – remote. Nem bántam meg beleásni magam ebbe a témakörbe, mivel rendkívül sok izgalmas, kiaknázatlan potenciál van benne. A munkám mellett, illetve részeként is megbecsült aktív tagja vagyok az llvm-project-nek [4]. A munkám során, többféle feladatkört is betöltök egészen a maintenance-től a research-ig, mint például inkrementálissá tenni a Clang Static Analyzer szimbolikus végrehajtási motorját.

Hivatkozások

[1] `https://edit.elte.hu/xmlui/handle/10831/44652`

[2] `https://edit.elte.hu/xmlui/handle/10831/77321`

[3] `https://www.sonarsource.com`

[4] `https://github.com/llvm/llvm-project/commits?author=steakhal`

Bertalan Dániel László

EJC: 2023–

Bertalan Dániel László vagyok, elsőéves hallgató programtervező informatikus BSc-n. A Collegium bentlakó tagja vagyok, újabban rendszergazda is. A középiskolát a Lovassy László Gimnázium matematika tagozatán végeztem, de már kis korom óta tudtam, hogy az informatikával szeretnék foglalkozni. Kedvenc területeim az operációs rendszerek és a fordítóprogramok; több ezekhez kapcsolódó nyílt forráskódú projekt fejlesztésében közreműködöm szabadidőmben, kutatásomat is ilyen területen szeretném elkezdeni.

Az online oktatás alatt bukkantam rá a SerenityOS projektre, melynek célja egy teljes értékű operációs rendszer létrehozása külső forrásból származó kód használata nélkül. Saját kernellel, grafikus felhasználói programokkal és böngészővel rendelkezik (ez Ladybird néven macOS-en és Linuxon is elérhető). Az elmúlt két és fél évben sok területtel ismerkedhettem meg a fejlesztés során: részt vettem a kernel Raspberry Pi-ra való portolásában, az általunk használt fordítóprogramok hibáinak javításában, valamint dolgoztam a programbetöltőnkön és a JavaScript interpreterünkön. Jelenleg a projekt egyik *maintainere* vagyok.

2022-ben részt vettem a Google Summer of Code programban, mely támogatást nyújt hallgatóknak és pályakezdőknek abban, hogy nyílt forráskódú szoftverek fejlesztésében közreműködjenek. A kezdeményezésnek számtalan olyan neves projekt tagja, mint a Mozilla, a TensorFlow, az Apache és a FreeBSD. A jelentkezés során az általuk javasolt feladatok egyikéről pályázatot kell benyújtani, amely leírja, hogyan kívánjuk felhasználni a rendelkezésre álló három hónapot. A nyertes pályázók a nyári hónapokban mentorok segítségével dolgoznak a kiválasztott feladaton, és sikeres teljesítés esetén pénzügyi támogatásban részesülnek. Én az LLVM Mach-O linkerének (`ld64.lld`) fejlesztésében működtem közre, a Chromium projekt mentoráltjaként. A Google és a Meta (ex-Facebook) több macOS- és iOS-applikációja is már ennek

segítségével készül. Fő munkám a 2022-es WWDC konferencián bejelentett *chained fixups* formátum implementálása volt, mely gyorsabb programbetöltést tesz lehetővé. Ezen felül javítottam a teljesítményén, és felhasználóbarátabbá tettem a hibaüzeneteit. Nagyon élveztem, hogy munkám az iparban is hasznosul, és hogy a mentorommal való kapcsolattartás során sokat tanulhattam a szoftverfejlesztésről.[1]

Tizedik osztálytól kezdve indultam programozási versenyeken, legjobb eredményem a Nemes Tihamér versenyen abban az évben elért 4. helyezés volt. Idén Szente Péterrel és Csertán Andrással csapatban az ICPC-válogatón negyedikek lettünk, így habár a regionális fordulóba pont nem jutottunk be, Marosvásárhelyen, a Sapientia ECN-en képviselhettük az egyetemet, ahol az első helyen végeztünk.

Hallgatóként lehetőséget kaptam, hogy a szervezői munkába is bekapcsolódjak: Zsakó László tanár úr meghívására bekerültem a 2023–24-es Nemes Tihamér és OKTV versenybizottságba. Az első fordulóban egy általam írt feladat is kitűzésre került. Ezen felül az Algo Pro Club egyesület tagjaként középiskolásoknak tartott programozási szakkörökön segédkezem.

[1] `https://gist.github.com/BertalanD/6a11e6e658be7d834870b03fb3e8af7b`

Biborka Ágnes

EJC: 2022–

2022 szeptemberében nyertem felvételt a Collegiumba és kezdtem meg alapszakos tanulmányaimat az ELTE Informatika Karán. Középiskolás koromban sok természettudományi és humán tanulmányi versenyen vettem részt, ezért tetszett meg az Eötvös József Collegium, ahol különböző tudományágak diákjai tanulnak.

Középiskolás éveim végére a versenyek közül kiemelkedett a bridzs és a robotika, ezt találtam a legizgalmasabbnak, és a csapatommal szép eredményeket értünk el. Az egyetemen töltött idő előrehaladtával azonban régebbi elfoglaltságaimat felváltották új, különleges kihívások. Az egyetemen csatlakoztam a Neumann tehetséggondozó körhöz, és minden alkalmat megragadtam arra, hogy valami érdekeset tanuljak.

A kutató műhelytársaim példáját látva alig vártam, hogy én is találjak egy témát, és valami hasznos felfedezéssel járuljak hozzá a kutatóvilág eredményeihez. Az egyetem első félévében a *Kutatószeminárium* keretében sok témába hallgattam bele, és Kvantumszámítás címmel tartottam rövid bemutatást a témában. Második félévem végére kikristályosodott, hogy mesterséges intelligenciával szeretnék foglalkozni a kutatásom során. A 2023 júliusában leadott pályázatommal elnyertem az ÚNKP ösztöndíjat, kutatásom témája: Programhibák detektálása nagy nyelvi modellekkel.

A nagy nyelvi modellek széles körben felhasználhatóak és fejlesztésük nagy népszerűségnek örvend, ezért érdekes lehet azt megvizsgálni, hogy mennyire alkalmasak a programozók életét megnehezítő „bug-ok” felfedezésére. A kutatást két évfolyamtársammal végzem, azt vizsgáljuk, hogy a nagy nyelvi modellek mennyire hatékonyan alkalmazhatók Python nyelven írt programkód szemantikai hibáinak felfedezésére és kijavítására.

A mesterséges intelligencia mellett jelenleg leginkább az algoritmusok és a versenyfeladatok érdekelnek. Elsőéves koromtól kezdve az egyetem által kínált és ajánlott informatika versenyeken is szívesen megmé-

rettettem magam. Az egyetemen tanultak, főleg az idén végzett, a témába illő tárgyak (*Algoritmusok és adatszerkezetek II.*, *Informatikai versenyfeladatok I.* és a collegiumi *Nevezetes algoritmusok és C++ STL*) eredményességének ékes bizonyítéka, hogy mennyit fejlődtem, ugyanis idén sokkal jobb helyezést értem el az egyetemi versenyen, mint tavaly.

Algoritmusok és adatszerkezetek tanárom felkért, hogy következő szemeszterben óratartó demonstrátor legyek *Algoritmusok és adatszerkezetek I.* tárgyból. A felkérést örömmel fogadtam, alig várom, hogy átadhassam a tudásomat és lelkesedésemet az ifjabbaknak.

Boros Attila Péter

EJC: 2015–2020

attila9778@yahoo.com

Elsőéves programtervező informatikus hallgatóként nyertem felvételt az Informatikai Műhelybe 2015 augusztusában, majd 2020 májusáig voltam tagja. Ez idő alatt rengeteg tapasztalatot sikerült szereznem mind akadémiai, mind ipari téren, ami kiváló lehetőség volt arra, hogy több témán keresztül kalandozva belekóstolhassak a különböző szakterületek mélységeibe. Ezekhez a Műhely tagjaként a collegiumi műhelyórák és az egyetemi tevékenységeim is hozzájárultak.

Szakdolgozatomban egy nagyobb garázsprojekt részeként hálózati protokollokat implementáltam egy fejlesztés alatt álló valós idejű operációs rendszer hálózati alrendszeréhez. Ezáltal rengeteg programkódírási gyakorlatot és mélyebb ismereteket szereztem a C++ programozási nyelvben és a számítógépes hálózatokban.

Mesterképzésem alatt egészen más területet próbálhattam ki a Big Data labor keretein belül Lehotay-Kéry Péter doktorandusz hallgató és Kiss Attila tanár úr vezetésével, melynek során az adattudomány világába nyertem betekintést. Tevékenységem során különböző adatelemzésre alkalmas keretrendszer teljesítményét hasonlítottam össze melyből kisebb folyóirat cikkek [1, 2] is születtek.

Párhuzamosan részt vettem a Horpácsi Dániel, Simon Thompson és Német Dávid János tanár urak által vezetett HARP (High Assurance Refactoring Project) projektben, ahol a nagy megbízhatóságú refaktorálások alapjául szolgáló Applicative Matching Logic logikai rendszer operációs szemantikájának formalizálásán dolgoztam a Coq tételbizonyító rendszer segítségével. Az operációs szemantika a programozási nyelvek leírásának strukturális megközelítése, melyben átírási szabályokkal határozunk meg végrehajtási lépéseket. A logikai rendszer szabályrendszerének kiinduló formalizálásával pedig közelebb kerültünk a gyakorlatban is használt programozási nyelvekre (mint az Erlang,

C++) megfogalmazott refaktorálások végrehajtásának kiértékeléséhez és azok matematikai helyességének bizonyításához. A projekt keretén belül végzett munkámból diplomamunka és egyben Tudományos Diákköri Konferencia dolgozat is született.

A mesterképzés befejeztével végül az iparban helyezkedtem el fejlesztői pozícióban az Ericsson Hungary, Expert Analytics projektjén, a mesterképzés alatt megkezdett Data Science gyakornoki program folytatásaként. A projekten egy hálózati adatforgalom elemzéséhez használt üzleti intelligencia szoftver fejlesztésében vettem részt, melynek során infrastruktúra- és termékfejlesztéssel foglalkoztam. Két év után a Continental AI Development Center tagjaként folytattam munkámat. Elsőként itt egy cloud infrastruktúrára épülő belső platform fejlesztésében vettem részt, majd később egy, a mesterséges intelligencia fejlesztőket adattal kiszolgáló, alkalmazás integrációjával foglalkoztam és foglalkozom erre platformra.

Időközben idén nyáron megnősültem és párommal Debrecenbe költöztünk.

Hivatkozások

[1] A. P. Boros, P. Lehotay-Kéry, A. Kiss (2020, January). A Comparative Evaluation of Big Data Frameworks for Log Processing. In ICAI (pp. 57-64).

[2] Boros, Attila & Lehotay-Kéry, Péter & Kiss, Attila. (2021). Performance impact of network encryption on log processing with Spark (13th Joint Conference on Mathematics and Computer Science (the 13th MaCS), on October 1-3, 2020). 10.13140/RG.2.2.32584.85768.

Börzsönyi Réka

EJC: 2023–

Börzsönyi Réka vagyok, másodéves hallgató az ELTE programtervező informatikus szakán. A matematika és a fizika témakörei már gimnáziumban érdekeltek, az informatika, programozás világába egy szakkörnek köszönhetően csöppentem bele. Egyre jobban felkeltette az érdeklődésemet az algoritmikus gondolkodás, így az egyetemi felvételin a matematika mellé az informatikát választottam.

Elsőévesként felvételt nyertem a szak kiemelt csoportjába, a Neumann Körben kezdhettem egyetemi tanulmányaimat. A motiváló környezet mind tanulmányi, mind közösségi szempontból rengeteget számított. Nekik köszönhetően ismertem meg az Eötvös Collegium közösségét, Informatikai Műhelyét, lehetőségeit. Az idei évben már én is aktív tagként vehettem részt előadásokon, programokon, a Collegium bejárós diákja lettem.

Első félévemben a felsőbb éves műhelytagoknak köszönhetően a kutatószeminárium keretei között több témakörbe pillanthattam bele, tágult a látásköröm. Ezeknek köszönhetően alakult ki, hogy szívesen foglalkoznék mélyebben számítógépes grafikával.

Az egyetemen specializációként az elméletibb, modellező szakirányt választottam, ebben a félévben különösen a numerikus módszerek tárgyunk keltette fel az érdeklődésemet.

Csertán András

EJC: 2019–

2019-ben kezdtem meg tanulmányaimat az ELTE Informatikai Karán, és ugyanebben az évben nyertem felvételt a Collegiumba. A Collegiumról már középiskolás éveimben is hallottam, például az OKTV döntők előtti éjszaka volt lehetőségem bentaludni. Az akkori tapasztalatok egyébként némileg elbizonytalanítottak abban, hogy ide akarok-e jönni, a leharcolt infrastruktúra némiképp demotiváló volt. Szerencsére végül jelentkeztem, és első helyen fel is vettek az Informatikai Műhelybe.

Már a kezdetektől fogva igyekeztem a közösség aktív tagja lenni. Amint lehetőség volt rá, jelentkeztem Estike csaposnak, majd az első félév után csatlakoztam a Választmányhoz, rögtön alelnökként, amit aztán két és fél éven keresztül, az alapképzés végéig csináltam. Sajnos a Választmányi tagságom kezdetén ütött be először a Covid, így az azt követő egy–másfél év nem teljesen úgy telt, mint ahogy azt előzetesen elképzeltük. Ezzel együtt a karanténban sem unatkoztunk, például felépítettük Minecraftban a Collegium épületét. Ezen kívül az alapképzés harmadik évétől Estike főcsaposként működtem, működök ezen névjegy írásának pillanatában. Először Katkó Dominikkal ketten vittük a bizniszt, majd mióta ő Koppenhágába emigrált MSc-re, egyedül csinálom, a lelkes csaposok segítségével.

Szakmai tekintetben alapvetően a gépi tanulás állt a kezdetektől fogva érdeklődésem középpontjában, már az alapképzés első félévében is ebben a témában mélyedtem el az ÚNKP ösztöndíj keretein belül. Jelenleg a HUN-REN SZTAKI-nál dolgozom, ahol neurális hálók generalizációs hibájának vizsgálatával foglalkozom. Ebből a témából TDK dolgozatot is írtam, amire a kari TDK-n első díjat kaptam.

Ami (szinte) biztos, hogy a következő félév végén fogom megkapni az MSc-diplomámat, viszont hogy utána mit fogok csinálni, még magam sem tudom, így ennek kitalálását az Olvasóra bízom.

Csimma Viktor

EJC: 2021–

csimmaviktor03@gmail.com

Legelőször is a száraz adatok. 2003-ban születtem Győrben; onnan költöztem ide. Harmadéves BSc-hallgató vagyok; modellező (régi nevén „A”-) szakirányon. Az egyetemen és a Coliban is 2021-ben kezdtem; az első *igazán*[1] COVID utáni évfolyamban. Jelenleg a 231-es szobában lakom; Jánossal (akiről szintén van itt névjegy) és Bérczy Barnával.

Annak idején, középiskolában nagyon vacilláltam a fizika és az informatika között. A nővéremmel[2] beszélgettem, és ő mondta azt, hogy a mérnökség kevésbé való nekem; inkább az infót vagy az elméleti fizikát javasolná. És mérnökként amúgy sem jöhetnék a Coliba. Végül az infó mellett döntöttem; bár bizonytalan voltam, mert a versenyprogramozást valójában annyira nem szerettem. Azért is örültem, amikor láttam, milyen sokszínű ez a tudományág, és mennyi különböző terület van, amik más-más készségeket igényelnek.

A nővérem is collegista volt társtudosként, 2014 és 2017 között. (Ő úgy jutott annak idején ide, hogy felhívták, hogy nincs-e kedve felvételizni, mert szép lett a pontszáma. Szóval közvetve a telemarketing hozott ide.) Szeretett itt lenni; azért volt nekem is már viszonylag régóta tervben, hogy felvételizem ide. Látva, hogy milyen versenyprogramozó-legendák jelentkeztek, nem bíztam annyira magamban; de páran Cambridge-be mentek végül, ha jól emlékszem. Így kerültem ide. És szerintem ez a legjobb hely, ami van az egyetemen.[3]

A kutatási irányaim az Agda programozási nyelv gyakorlati alkalmazhatóságával kapcsolatosak. Ez egy olyan funkcionális nyelv, amiben

[1] Ez azt jelenti, hogy minket már soha nem küldtek hivatalosan haza az épületből.

[2] 1995-ben született, úgyhogy több mint 8 évvel idősebb nálam. Ő a második anyukám.

[3] De tényleg. Ha valakinek, aki ezt olvassa, van gyereke vagy ismerőse, akinek releváns lehet a Coli, mondja neki!

bizonyításokat is lehet írni a programok helyességével vagy úgy bármivel kapcsolatban. Igazából a téma talált meg engem; Kozsik Tamás hívta meg 2022 tavaszán *Kaposi Ambrust*, aki a néhai *Modern elméletek az informatikában I.* kurzus utolsó iterációjában megismertetett minket az Agdával. Én is szerettem a nyelvet, a nyelv is szeretett engem; úgyhogy ezzel kezdtem el ÚNKP-t írni Ambrus témavezetése alatt, és most az első TDK-mat is erről készítettem. Folytatni tervezem a kutatást; azt hallottam, nem csak nekem tetszett, amit csináltam.

A szakmai gyakorlatomat szintén agdázással töltöttem Ambrusnál. Hálás vagyok neki, mert a segítségnyújtás mellett hagyta, hogy azzal foglalkozzak, ami igazán érdekel.

A Coliban első rendszergazda vagyok 2023 augusztusa óta. (2022 nyarán kezdtem el rendszergazdaként dolgozni; ennek körülményeiről a másik írásomban van szó.) Emellett rendezvényeket is sokszor segítek szervezni, főleg Andris alatt. Ezek a legnépszerűbb pozíciók: segítek problémákat megoldani, amiért hálásak az emberek; ráadásul nem kell nagyon veszekedni senkivel, és nem kell emberéletekről dönteni.

Hobbim… az hagyományos értelemben nincs. A szabadidőmet igyekszem beszélgetésekkel és picit az egyetemtől elrugaszkodottabb, akár kreatív munkával tölteni. Ha eszembe jut egy jó ötlet, néha novellákat is írok. Győrbe viszonylag ritkán járok haza; ha igen, akkor anyukámmal beszélgetek és filmezek, illetve röfögtetem a kutyát.[4]

Alapvetően szeretetorientált vagyok; ez motivál a munkámban is. Örülök, ha másnak örömet tudok okozni, és sokszor ölelkezem.

[4] Ő ilyen. Úgy játszik, hogy rágja a kezemet, és közben röfög.

Eichhardt Iván

EJC: 2009–2012

ivan.eichhardt@gmail.com

A Collegiumhoz 2009-ben csatlakoztam, amikor elkezdtem a programtervező informatikus alapképzést. Külsős tagságom ellenére igyekeztem részt venni Csörnyei Zoltán tanár úr és a többiek előadásain, óráin, valamint néhány eseményen.

2012 óta az ELTE IK Algoritmusok és Alkalmazásaik Tanszék oktatói gárdájának tagja vagyok. Sok éven át megbízott oktatóként dolgoztam, majd 2020-tól adjunktusként félállásban folytattam munkámat.

Szintén 2012-től egészen 2020-ig az MTA SZTAKI-ban dolgoztam, először szoftverfejlesztőként, majd kutatóként Csetverikov Dmitrij tanár úr irányítása alatt. Disszertációmat végül 2019-ben védtem meg.

Kutatási témám a számítógépes látás geometriai problémáira összpontosít, különös tekintettel az affin képi jellemzők felhasználására, valamint a vegyes/többszenzoros rendszerek kalibrációjára és fúziójára.

Geometriai feladatok megoldására általában képi régiók jellemzőinek megfeleltetéseit használjuk. A nézetek közötti viszonyt leíró geometria ismeretében pedig lehetséges a környezet ritka rekonstrukciója, felhasználva a régiók helyzetét. A régiók további hasznos információt hordoznak, amelyek megkönnyíthetik a geometriai feladatokat. A pontszerű képi információ mellett a régiók alakja közötti affin leképezés további geometriai megkötéseket kínál, és lehetővé teszi felületi normálisok hozzárendelését a ritka rekonstrukcióhoz. Saját elméleti és algoritmikus eredményeimmel az affin megfeleltetések használatát terjesztettem ki olyan új területekre, mint a többnézetű geometria és a structure-from-motion, általános kameramodellek alkalmazása mellett.

Különös kihívást jelent a nézetek közötti megfeleltetések létrehozása eltérő modalitások esetén (például mélységképen nem jelenik meg textúrázottság). A különböző bemenetek megfelelő fúziója azonban előse-

gítheti a döntéshozó algoritmusok robusztus működését. Eredményeim közé tartozik a színes és mélységképek, valamint videók fúziója, ahol a tipikusan csekély felbontású mélységképeket sikerült HD minőségűvé javítani élethű élekkel.

A legtöbb alkalmazás szempontjából fontos, hogy a különböző érzékelőket egy közös koordináta-rendszerben helyezzük el. Ezt szolgálja a kalibráció folyamata, amely során a szenzorokat modellező paramétereket becsüljük és finomítjuk. Munkám kiterjedt a többnézetű rendszerek kalibrációjára is, kamerákat és LiDAR-okat egyaránt használva. Főbb eredményeim között kiemelhető a kalibráció pontosságának és robusztusságának javítása.

Hivatkozások

[1] Iván Eichhardt, Levente Hajder, Computer Vision Meets Geometric Modeling: Multi-view Reconstruction of Surface Points and Normals using Affine Correspondences, *Proc. Int. Conf. on Computer Vision Workshops,* (2017), pp. 2427–2435.

[2] Iván Eichhardt, Dmitry Chetverikov, Zsolt Jankó, Image-guided ToF depth upsampling: a survey, *Int. Journal of Machine Vision and Applications,* 28(3–4) (2017), pp. 267–282.

[3] Gábor Kátai-Urban, Iván Eichhardt, Vilem Otte, Zoltán Megyesi, Paul Bixel. Reconstructing atmospheric cloud particles from multiple fisheye cameras, *Solar Energy,* 171 (2018), pp. 171–184.

[4] Iván Eichhardt, Dmitry Chetverikov, Affine Correspondences Between Central Cameras for Rapid Relative Pose Estimation, *Proc. European Conf. on Computer Vision,* (2018), pp. 488–503.

[5] Zoltán Pusztai, Iván Eichhardt, Levente Hajder, Accurate Calibration of Multi-LiDAR-Multi-Camera Systems, *Sensors,* 18(7) (2018), 2139.

[6] Daniel Barath, Iván Eichhardt, Levente Hajder, Optimal Multi-View Surface Normal Estimation using Affine Correspondences, *IEEE Trans. Image Processing,* 28(7) (2019), pp. 3301–3311.

Englert Péter

EJC: 2009–2015

2015-ben fejeztem be tanulmányaimat az ELTE programtervező informatikus szakán és költöztem ki a Collegiumból. Először Zürichben, majd New Yorkban dolgoztam a Google-nél, termékkereső-optimalizálás majd reklámszolgáltató backendek fejlesztése után egy nagy gráfok elemzésével foglalkozó kutatócsapatban.

Ezek után egy alkalmazott kutatói állással kerültem az Amazon tokiói irodájába, ahol főleg termékek árazásával foglalkoztam. Jelenleg ajánlórendszereket fejlesztek egy Indeed nevű álláskereső oldalnak, a szoftverfejlesztés és a gépi tanulás határterületein dolgozva.

Fonyó Viktória

EJC: 2014–2019

fonyoviki@gmail.com

Fonyó Viktória vagyok, 27 éves. Középiskolában sokat versenyeztem matekból. Nem szerettem volna kutató vagy mérnök lenni, így kötöttem ki az Eötvös Loránd Tudományegyetemen, programtervező informatikus szakon 2014-ben. Ugyanekkor vettek fel a Eötvös Collegiumba is, ahova bátyám, Dávid javaslatára jelentkeztem, aki szintén bentlakó volt a kollégiumban.

Az egyetem első két évében inkább a tanulásra fókuszáltam, hamar kialakult egy kisebb verseny az évfolyamunkon, hogy ki tud több kreditet elvégezni egy adott félév alatt. A versenyzést matekból nem hagytam abba, többször képviseltem az ELTE-t a Hajós György Matematika versenyen, ahol másodéves koromban 2. helyezést értem el.

Másodéves koromban lettem az Informatikai Műhely titkára amit 3 évvel később adtam tovább amikor kiköltöztem a kollégiumból.

A BSc-szakdolgozatomat „Pszeudovéletlen bitsorozatok konstrukciói" címmel készítettem Tóth Viktória vezetésével, majd MSc-n folytattuk tovább a munkát a konstrukciók által generált sorozatok statisztikai elemzésével. Ezzel a témával részt vettem a kari TDK-n és továbbjutottam vele az OTDK-ra.

Az MSc-s tantárgyaim döntő többségét elvégeztem az első évem alatt. Ezután kiköltöztem a kollégiumból, és miután az első helyezett csapatban végeztem a Graphisoft ITech Challenge programozói versenyén, utolsó egyetemi évemben a tanulás mellett elkezdtem dolgozni a Graphisoftnál. 5 évig dolgoztam a cégnél C++ szoftverfejlesztőként, és ez idő aktív tagja voltam az ITech Challenge szervezői csapatának. A munkámban az első időszakban térbeli geometriai alakzatok helyes kirajzolásáért feleltem, majd részt vettem egy projektben, amelynek keretében a tervrajzok felhő alapú tárolását tettük lehetővé, végül a csapatommal új statikai eszközöket fejlesztettünk a programhoz.

Időközben férjhez mentem Leitereg Andráshoz, aki szintén az Informatikai Műhely tagja volt, vettünk egy lakást Óbudán, a járvány alatt lett egy kiskutyánk, és fél éve költöztünk Ürömre, új családi házunkba. Jelenleg a TNG Technology Consulting német cég magyar leányvállalatánál dolgozom, azon belül pedig egy biztosító cég terjeszkedésében segítek backend fejlesztőként. Nem vagyunk sokan, de így is van olyan kollégám, akit még Kozsik tanár úr és én vettünk fel a műhelybe.

Gansperger István

EJC: 2012–2017

gansperger@gmail.com

2012-ben kezdtem az ELTE programtervező informatikus BSc szakán, és ekkor lettem az Eötvös Collegium bentlakó tagja is. Több mint 10 év után visszagondolva bátran állíthatom, hogy a Collegiumban töltött első félév volt tanulmányaim és karrierem szempontjából is az egyik legmeghatározóbb időszak. Rengeteg nálam sokkal okosabb emberrel ismerkedtem meg, akiktől nagyon sokat tudtam tanulni. Nem is beszélve Csörnyei Zoltán tanár úr Modern elméletek az informatikában I. kurzusáról, amiből akkor nagyjából semmit nem értettem, de visszatekintve alapjaiban változtatta meg a számítástudományhoz való hozzáállásomat.

Egyetemi éveim alatt több dolgot kipróbáltam, hogy eldöntsem, milyen irányba szeretnék továbbmenni. 2014-ben elkezdtem gyakornokként dolgozni egy cégnél. Részt vettem egyetemi projektekben, pl. az akkor népszerű futtatható UML modellekkel kapcsolatos fejlesztésekben. Később ebben a témában írtam a BSc szakdolgozatomat is. Nagyon érdekelt a típuselmélet, úgyhogy gondolkodtam ehhez kapcsolódó akadémiai pályán is. A tanítást is kipróbáltam: tartottam órát *Practical Software Engineering* névvel külföldi hallgatóknak, egy félév Algoritmusok és adatszerkezetek gyakorlatot, illetve egy F# specit több alkalommal. MSc végére eléggé belefáradtam az egyetemi létbe, úgyhogy végül az ipar mellett döntöttem.

Az első munkahelyem egy kis cég volt. 5-en voltunk fejlesztők, és egy WebSharper nevű nyílt forráskódú, *full-stack* web keretrendszer volt a fő termékünk. A segítségével F# nyelven lehetett mind a kliens, mind a szerver oldali logikát definiálni. A kliens oldal JavaScriptre fordult egy saját fejlesztésű fordító segítségével – ami talán a termék legérdekesebb

része volt –, és a böngészőben futott *Single Page Application*ként. Sajnos sem az F# nyelv, sem a WebSharper nem túl népszerű, pedig azóta sem találkoztam ennél elegánsabb UI technológiával.

Ezután kaptam egy jobb ajánlatot a Morgan Stanley-től, úgyhogy ott folytattam a pályafutásomat Java és Scala fejlesztőként a következő 4 évben. Friss diplomásként részt vehettem a cég 15 hetes képzésében, aminek keretében 9 hetet a New York-i, 6 hetet pedig a londoni irodában dolgoztam.

Innen visszatérve a *Global Expiry System* nevű komponensen kezdtem el dolgozni. Ez a cég *Prime brokerage* ügyfeleinek az opcióit és *future*-jeit, azok lejáratását és könyvelését kezelte. Itt megtapasztaltam, milyen egy hatalmas, világszerte 80.000 embert foglalkoztató cégben dolgozni, illetve hogy hogyan kell kezelni egy olyan szoftver fejlesztési életciklusát, amin potenciálisan több százmillió dollár múlik havonta. Ezek mellett egy kis pénzügyi tudás is rám ragadt, ami nem haszontalan a mindennapokban.

Jelenleg a Cloudera Hungary Kft.-nél dolgozom *Staff Software Engineer* minőségben szintén Scala és Java programozóként. A cég *Backup and Disaster Recovery* szoftverét fejlesztem, ami nagy mennyiségű adatok másolásáért felel sok ügyfelünknél. Különböző *big data* technológiákhoz (pl. HDFS, Hive, HBase, Ozone) fejlesztünk olyan megoldásokat, amivel az ügyfeleink könnyen és gyorsan tudnak biztonsági mentést készíteni, és hiba esetén ezt visszaállítani, vagy a forgalmukat átirányítani a *Disaster Recovery* clusterre.

Budapesten élek, dolgozom, és a közeljövőben nem is tervezek ezen változtatni. Sosem tudja az ember, mit hoz a jövő, de szerintem egyelőre maradok még az iparban jó pár évig. Csörnyei tanár úr nyomán próbálom minél jobban népszerűsíteni a funkcionális programozás szépségeit.

Gévay Gábor

EJC: 2010–2015

2010 és 2015 között voltam az Informatikai Műhely tagja. Ezalatt részt vettem sok programozási versenyen, főként csapatban, Englert Péter és Bencs Ferenc collégákkal. Pl. 2011-ben megnyertük a Sapientia ECN programozási versenyt Marosvásárhelyen. 2015-ben Danner Gáborral közösen írt TDK dolgozatunk első díjat nyert az OTDK-n. Témája a Malom játék és variánsaihoz írt mesterséges intelligencia volt. A dolgozatból egy cikk is született [1]. Ezen eredmények alapján Pro Scientia Aranyérmet nyertem.

Az MSc diploma megszerzése után a Berlini Műszaki Egyetemen doktori tanulmányokat folytattam 6 éven keresztül [2]. Kutatási területem az elosztott adatfeldolgozást megkönnyítő, „big data" rendszerek voltak, pl. Apache Flink és Spark. A doktorim meglehetősen elhúzódott, de végül jó eredménnyel zárult: az egyik cikkem [3] megnyerte a konferencia legjobb cikke díjat az ICDE konferencián, és azután még az ACM SIGMOD Research Highlights Awardot is.

A PhD fokozat megszerzése után hazaköltöztem Magyarországra, és most Budapesten lakom. A berlini kaland során arra jutottam, hogy az akadémiai közegben való kutatásnál jobban érdekel valódi rendszerek fejlesztése, ahol közelebb vagyok a gyakorlati felhasználásokhoz. Így a *Materialize* nevű New York-i startup cégnél kezdtem dolgozni távmunkában.

A *Materialize*-nál egy újfajta adatbázisrendszert fejlesztünk, amellyel folyamatosan beérkező (azaz streaming) adatokat lehet feldolgozni, hasonlóan a Flink, Storm, Spark Streaming stb. rendszerekhez. Azonban míg a fentebbi rendszerek használatához általában egy egész csapat specialista programozóra és üzemeltetőre van szükség, a mi rendszerünket sokkal könnyebb használni. Ennek egyik oka, hogy a rendszerünk nem csak feldolgoz, hanem tárol is adatokat. Így a felhasználók közvetlenül, egyszerűen a mi rendszerünkből lekérdezve férhetnek

hozzá a feldolgozott adatokhoz anélkül, hogy még egy másik adatbázis-rendszerrel kéne bonyolítaniuk az architektúrájukat. Rendszerünk így bizonyos esetekben egyedül kiválthat pl. egy Kafka–Flink–HBase architektúrát.

A felhasználóink standard SQL-ben írhatják meg a programjaikat, viszont a hagyományos SQL rendszerekkel szemben a lekérdezéseket nem csak egy adott pillanatban értékeli ki a rendszer, hanem a lekérdezés eredményét folyamatosan frissíti másodpercenként, ahogy újabb bemenő adatok érkeznek. Az SQL egy deklaratív nyelv, azaz a felhasználónak nem kell foglalkoznia a (párhuzamos vagy akár elosztott) végrehajtás számos bonyolult részletével. A magas szintű SQL lekérdezéseket a rendszer fordítja le alacsony szintű *dataflow* programokká. A fordításnak egy fontos lépcsőjét végzi az *optimizer* (amelynek fejlesztésén dolgozom): a sokféle lehetséges végrehajtási terv közül igyekszik minél hatékonyabbat választani. Ehhez sokszor hasznos a matematikus gondolkodásmód (egy apró példa: [4]), amit az Informatikai Műhely tárgyai (pl. a λ-kalkulus) is támogattak.

Hivatkozások

[1] G. E. Gévay, G. Danner, Calculating Ultra-Strong and Extended Solutions for Nine Men's Morris, Morabaraba, and Lasker Morris, *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3), 2015.

[2] G. E. Gévay, Nested Parallelism and Control Flow in Big Data Analytics Systems, *Doktori disszertáció, Technische Universität Berlin*, 2022.

[3] G. E. Gévay, T. Rabl, S. Breß, L. Madai-Tahy, J.-A. Quiané-Ruiz, V. Markl, Efficient Control Flow in Dataflow Systems: When Ease-of-Use Meets High Performance, *IEEE 37th International Conference on Data Engineering (ICDE)*, 2021.

[4] Pushing temporal filters through TopK, `https://github.com/MaterializeInc/materialize/issues/23527`

Gilián Zoltán

EJC: 2010–2016

2007-ben kezdtem meg tanulmányaimat az ELTE Informatikai Karán, az Eötvös Collegium Informatikai Műhelyének 2010-től lettem tagja. 2012-ben szereztem mesterdiplomát Programtervező Informatikus szakon, ezt követően a kar doktori képzésébe vágtam bele a Numerikus Analízis Tanszéken. 2013-ban egy passzív félévet Zürichben töltöttem a Google gyakornokaként mint Site Reliability Engineering Intern. A doktori képzést a Bolyai Kollégium belsős tagjaként folytattam, de rendszeresen jártam az Eötvös Collegiumba többek közt a floorball és a röplabda játékokra, illetve vettem részt a Collegium Szarvasűzők futóversenyen induló csapatában.

Ezalatt az idő alatt világossá vált számomra, hogy a versenyszférában szeretnék elhelyezkedni, így 2016-tól csatlakoztam a Digic Pictures akkor még magyar tulajdonú animációs stúdió fejlesztői csapatához, ahol a filmgyártáshoz használt eszközökön és szoftverinfrastruktúrán dolgoztam. Ebben az időszakban sok időt töltöttem számítógépes grafikával szabadidőmben is, ehhez kapcsolódó projektjeimet a `https://zogi.github.io/graphics` weboldalon gyűjtöttem össze.

2018-ban ismét Zürichbe költöztem, itt a Facebook (ma már Meta) által felvásárolt Oculus svájci csapatához csatlakoztam mint Software Engineer, ahol a Quest márkájú headsetek rendszerszoftverén dolgoztam, többek közt a szenzor adatok felvételét és visszajátszását lehetővé tevő infrastruktúrán.

2022-ben a Headlands Technologies, egy algoritmikus kereskedelemmel foglalkozó cég amszterdami fejlesztőcsapatának tagja lettem, hogy a szoftverinfrastruktúra fejlesztésével és karbantartásával foglalkozzam. 2024-től a feleségemmel Magyarországon lakom és távmunkában dolgozom.

Hapák József

EJC: 2007–2014

hapakj@hotmail.com

Egyetemi tanulmányaimat és collegiumi pályafutásomat 2007 szeptemberében kezdtem meg, s első éves informatikusként az Informatikai Műhely tagja lettem. Műhelytársaim sokat segítettek az egyetemi életben való elindulásban, óráim sikeres elvégzéséhez a legjobb egyetemi gyakorlatvezetők kiválasztásában. Az évek során a műhelytagokkal sok kellemes élményben lehetett részem, és sok szakmai tapasztalatot szerezhettem.

Meghatározó élményem másodévesként, amikor műhelytársaim közül kikerülő rendszergazdáknak segítve felújítottuk a Collegium számítógépes hálózati infrastruktúráját, azt modernizálva és megbízhatóbbá téve. 2010-ben és az elkövetkezendő néhány évben az Eötvös Konferencia weboldalának karbantartásával járultam hozzá az esemény sikeres lebonyolításához. 2011 végén csatlakoztam Cséri Tamás műhelytársamhoz a rendszergazdai feladatok ellátásában. Jelentős mennyiségű karbantartást hajtottunk végre az akkori informatikai rendszeren, hogy biztosíthassuk a frissen bekötött széles sávú internet-hozzáférést a collegistáknak. Rendszergazdaként ezen kívül feladatunk volt a Collegium különböző eseményeinek (versenyek, konferenciák) számára biztosítani a szükséges eszközöket.

Egyetemi éveim után doktoranduszként, ifjú kutatóként helyezkedtem el az MTA SZTAKI keretein belül. Sajnos hamar rá kellett jönnöm, hogy a kutatói pálya nem nekem való, ezért inkább ipari megoldások fejlesztése felé fordultam. Visszatértem eredeti szakterületemhez, a valós idejű számítógépes grafikához, amit 2014-től első jelentősebb munkahelyemen, a Kishonti Kft.-nál kamatoztattam. Itt világszínvonalú grafikus teljesítménymérő szoftvereket fejlesztettünk a dinamikusan fejlődő okostelefon-piacra. Rengeteg tapasztalatot szerezhettem a szakterületemen kívül a végfelhasználói szoftverek fejlesztésével kapcsolatban. Ezen

kívül érdekes volt látni a különböző nagy partnercégek kommunikációját, illetve munkákhoz való hozzáállását. Itt töltött éveim alatt zajlott le a számítógépes grafika területén a *low-level forradalom,* mely radikálisan megváltoztatta annak módját, hogy hogyan szólítjuk meg a számítógépek, tabletek, mobiltelefonok grafikus processzorait. Úgy éreztem, hogy az MSc-tanulmányaim során szerzett széles látókör, főleg párhuzamos, konkurens rendszerekkel kapcsolatban nagy segítséget nyújtottak ezen új megközelítések befogadásában. Közvetlen kollégáimmal szemben nekem hamarabb sikerült a képszintézis/rendering eddigi szokványos szekvenciális megfogalmazása helyett a hatékonyabb gráf- és függőségalapú (DAG) megfogalmazását elsajátítani.

További jelentős szakmai tapasztalat volt itt töltött munkám során egy shaderfordító könyvtár fejlesztése, mely egyedülálló volt abban a tekintetben, hogy egységesítette az akkori modern platformokra való fejlesztést, azok legkorszerűbb képességeit megőrizve. A munka megkönnyítése érdekében olyan validációkat is tartalmazott, amelyekkel még a mai eszközökben sem igazán találkozok. A shaderfordító kifejlesztésében sikeresen kamatoztathattam egykori műhelyvezetőm, *Csörnyei Zoltán* MSc-s *Fordítóprogramok és assemblerek* óráján tanultakat.

Az évek során a cég egyre inkább profilt váltott, és előbb Adasworks, később Aimotive néven folytatta tevékenységét az automotive iparban. Továbbra is szakterületemen dolgozhattam, hiszen feladatunk egy szimulációs szoftver fejlesztése volt, amely különböző scenariok segítségével lehetőséget biztosított a vezetést segítő vagy teljesen önvezető rendszerek validációjában. Az ezen való munka folyamán mélyebb tapasztalatot a szoftverek minőségét biztosító CI rendszerekkel, különböző ipari ISO szabványokkal és azoknak való megfeleléssel, illetve a csapatvezetéssel. Sajnos az ezekkel járó körülményesség és nehézségek miatt a projekten való munka már nem motivált megfelelően, így 2021-ben a váltás mellett döntöttem.

Pályafutásomat a *Gaijin Entertainment* videójáték-fejlesztő stúdiónál folytattam. Kevés az ilyen jellegű cég, amely nagyrészt saját technológia alapokra építkezik. Itt folytatott munkám során ezt a technológiai alapot folyamatosan fejlesztjük a legújabb eljárásokkal és az újabb eszközökben elérhető lehetőségek kiaknázásával, hogy termékeink minden platformon a lehető legjobb élményt nyújtsák mind sebesség, mind képminőség tekintetében.

Hegyi Tünde

EJC: 2022–

tundehegyi24@gmail.com

2019 telén költöztem be a Collegiumba, igaz, akkor még BME-s tanulóként, így csak vendégként lakhattam bent. Két nagyon kedves bölcsész colleginához kerültem, akikkel élmény volt együtt lakni, illetve szépen lassan, az itt töltött félévek alatt egyre inkább a közösség részévé váltam, barátokat szereztem, így nem is akartam már visszaköltözni BME-s kollégiumba. Ez a három év „vendégeskedés" alatt nagyon sokat kaptam a Colitól, mind tanulmányi segítséget, motivációt, mind közösségi élet terén élményeket, emlékeket, melyhez nagyban hozzájárult a bentlakók sokszínűsége.

A matematikus BSc elvégzését, és egy félbehagyott egészségügyi mérnök képzést követően 2022 telén jelentkeztem az ELTE informatikus mesterképzésére. Az első félév jól sikerült, így a kezdeti nehézségekkel induló matek szak ellenére felvételiztem a Collegiumba, és bejáró helyet kaptam. Szerencsémre eddigre már létezett a bentlakó-bejáró státusz, így megtarthattam a szobánkban a helyemet. A korábbi bentlakás egyetlen hátrányaként azt tudnám felhozni, hogy így a góJatábor nem tartogatott akkora meglepetéseket, illetve kevésbé ismerkedtem (ott) a velem együtt felvett góJákkal, hisz a szervezőkkel is jóban voltam már (na meg az átlag góJánál kb. 3 évvel idősebb és egyetem szempontjából tapasztaltabb voltam, ami egy kisebb szakadéknak tűnt).

Az infó mester „leggyengébb" szakirányát (Információs rendszerek) választottam, mivel ez érdekelt leginkább, illetve kevésbé volt kedvem a „tömény matekot" (Modellalkotó) folytatni. Sikerült olyan kurzusokat (Számítási modellek, algoritmusos tárgyak, Térinformatika, Logika, Jel- és képfeldolgozás...) találnom, melyek anyaga számomra érdekes és izgalmas volt, illetve új érdekes területeket felfedeznem, például a térinformatikát. Továbbá, ahogy egy kiemelkedően jó egyetemi tanárom

mondta, nem csak a nehéz, bonyolult tudományos eredmények értékesek, hanem a „leggyengébb szakirányon" végzett érdekes, hasznos, szép és adott esetben szórakoztató kutatások – melyeket talán könnyebb is alkalmazni – is nagyon fontosak.

A felvétellel az infóműhely tagjává váltam, annak minden szépségével és nehézségével. Úgy gondolom, hogy a műhely közössége, egymás támogatása, motiválása nagy szerepet játszik az egyetemi tanulmányaink során. Jó volt látni, hogy az egyes félévek alatt ki, hogyan haladt a kutatásával, vagy éppen annak keresésével, és hogy milyen érdekes területei vannak az informatikának, milyen klassz dolgokat lehet csinálni, mindeközben persze ez milyen nehézségekkel jár. Úgy vélem, az egymásnak tartott előadások és órák (pl. Bölcsészinfó) sokaknak lehetőséget adnak kellemes környezetben egyéb fontos készségeket fejleszteni. Továbbá a közös órák, mint a Típuselmélet, hozzájárulnak a műhely közösségének összekovácsolódásához.

Ebben a félévben végeztem először demonstrátori tevékenységet, melyre személyesen kértek fel, ez számomra nagy megtiszteltetés volt. Mivel ezt a félévemet Erasmusszal külföldön töltöm, így csak házi javítást vállalhattam, azonban nagy lelkesedéssel végeztem ezt is, próbáltam minél jobb visszajelzéseket adni, és motiválni a hallgatókat, továbbá a korlátozott lehetőségeim ellenére átadni a tárgy iránti rajongásomat. Hasznos tapasztalat ezen felül a „másik irányból" látni a beadandókat, így megértőbben tudom én is kezelni a pontlevonásokat, kritikákat a saját házijaimnál.

Mivel Németországban töltöm a külföldi tanulmányaimat, az itteni tanév pedig nyári és téli félévre van felosztva, így végül csúszok a tanulmányaimmal, ugyanakkor úgy gondolom, megéri. A szemeszter során lehetőségem van egy anyagilag jobb helyzetben lévő, kicsit más rendszerrel működő egyetemet megismerni, új perspektívákat és lehetőségeket találni, az itteni modern kutatásokba belelátni. Továbbá az angol nyelvtudásomat is fejlesztem, hiszen végül minden órámat angolul végzem, rengeteg külföldi társaságában. Önmagában elmondható Freiburgról, hogy sokkal több külföldi él itt, kezdett új, színvonalasabb életet, mint otthon. Továbbá mindenképpen említésre méltó a város környezettudatossága! Freiburg egyetemváros, így nagyon sok kikapcsolódási lehetőség van, többek közt a helyi Erasmus csoport és az International Club szervezésében kirándulásokon, kulturális esteken, bulikon lehet új ismeretségeket kötni (akár a világ másik feléről származó emberekkel),

szemlélet tágító beszélgetéseket folytatni, és másik kultúrákat megismerni. Úgy gondolom, hogy az egyetemi oktatás is színvonalas, a félév során két kiemelkedően jó oktatóm is volt. A Prof. Dr. Hannah Bast által tartott Information Retrieval kurzuson, még ha sok minden az alapoktól is lett felépítve, olyannyira használható eszközöket kapunk, hogy a tárgyon megszerzett tudással biztosan sokkal eredményesebben tudtam volna a nyáron a SZTAKI-nál végzett gyakornoki munkámat teljesíteni. A kellő háttérismeret, a megfelelő támogatás, és leginkább is a megfelelő hardver híján sajnos kevésbé pozitívan éltem meg a másodjára ott töltött időt. (A SZTAKI egy másik részlegénél írtam a szakdolgozatomat.)

Az otthoni tavaszi félévbe várhatóan áprilisra sikerül majd visszacsöppennem az itteni késői vizsgák miatt. Így viszont a diplomamunkám lényegi részének elkészítésére elég szűkös lesz az idő, várhatóan egy húzós félév áll előttem. Szerencsére viszont a témakör egyelőre érdekesnek tűnik, gráfokon történő kollaboratív szállítási feladat témakörében szeretnék elmélyülni.

MSc után még nem döntöttem el pontosan, hogyan tovább. Felmerült bennem, hogy doktori képzésbe vágjak, akár Freiburgban, hiszen itt rengeteg érdekes kutatást és jó lehetőséget találtam, azonban jelenleg a dolgozás és gyakorlati tapasztalatgyűjtés felé hajlok. Úgy gondolom, ha pár év múlva még mindig kitartok a PhD gondolata mellett, akkor valószínűbb, hogy be is fejezem a képzést, illetve addigra talán jobban fogom tudni, mi az, ami érdekel, és ezen belül mi az erősségem.

Horváth Gábor

EJC: 2011–2020

xazax.hun@gmail.com

2011-ben kezdtem meg a tanulmányaimat programtervező informatikus szakon, felvételt nyertem az Informatikai Műhelybe is, melynek PhD tanulmányaim végéig, 2020-ig aktív tagja voltam. Csatlakozásomkor Csörnyei Zoltán volt a műhelyvezető, és Cséri Tamás a műhelytitkár. Az egyetemi évek elején hamar kialakult egy csodálat a fordítóprogramok iránt, Cséri Tamás közbenjárására elsőévesként sikerült csatlakozni az egyetemen futó CodeCompass projekthez. A feladatom egy olyan szoftver fejlesztése volt, ami kerülendő kódmintákat, és hibákat keres ipari C++ alkalmazásokban. Ennek kapcsán kezdtem el a nyílt forráskódú Clang fordítóprogramon dolgozni.

Az első TDK dolgozatomhoz Pataki Norbertet kértem fel témavezetőnek, aki a C++ programozás tárgy felelőse volt. A közös munka olyan jól sikerült, hogy további TDK dolgozatok, közös cikkek, szakdolgozat, diplomamunka, majd PhD témavezetés követte mindezt. Demonstrátorként részt vettem a C++ tárgy oktatásában. Tanulmányaim során végig szoros kapcsolatot ápoltam az iparral, alkalmazás közeli területeken dolgoztam. Elsőévesen a Graphisoftnál voltam gyakornok, később az Ericssonhoz csatlakoztam, ahol már fordítóprogramokkal kapcsolatos munkát végeztem. Google Summer of Code ösztöndíjat nyertem el a Clang fordítón való munkámhoz, ezután az Apple cupertinoi főhadiszállásán végeztem nyári gyakornokságot. Ezt később Microsoft és Google gyakornokságok követték, végül a Microsoft Visual Studio csapatában helyezkedtem el főállású fejlesztőként.

Kutatási témám a C++ programok elemzése, hibák, kerülendő kódminták, teljesítményproblémák automatikus megtalálása.

A programelemzés során automatizált módon szeretnénk érvelni a programok működéséről. A statikus elemzés fordítási, míg a dinamikus

elemzés futási időben történik. Én elsősorban hibakereséssel foglalkozom statikus elemzés segítségével. Ezen a területen a legtöbb probléma visszavezethető a megállási problémára, algoritmikusan eldönthetetlenek. Ennek az a következménye, hogy nem lehet tökéletes elemzőszoftvert írni, ami minden hibát megtalál és soha nem ad hamis riasztást. Valójában azonban ezeknek az eszközöknek nem kell tetszőleges programkódra tökéletesen működnie. Elég, ha a fejlesztők által írt programokra (amik általában követnek bizonyos programozási mintákat), kevés hamis riasztással működik. Emellett általában nem cél az összes hiba megtalálása, a statikus elemzés a tesztelés egy kiegészítéseként is funkcionál.

Az elemzés során számos szemantikus tulajdonságot próbálunk megállapítani a programról. Ilyen tulajdonság például, hogy egy változónak egy program ponton milyen értékei lehetnek. A változók értékeiről való érvelés segítségével olyan hibákat találhatunk, mint a nullával való osztás vagy null értékű mutató dereferálása. Gyakori probléma még annak az eldöntése, hogy egy feltétel a rendelkezésre álló információ alapján felvehet-e igaz vagy hamis értéket. Ezekre az eldöntési problémákra az elemző szoftverek gyakran SMT megoldóprogramokat vagy tételbizonyítókat használnak (például Z3).

A pontos elemzés érdekében az elemzőszoftverek sokszor megpróbálnak minél pontosabb invariánsokat kikövetkeztetni a programkód alapján. Ilyen invariánsok az előfeltétel, utófeltétel vagy a ciklusinvariáns. Három gyakori megközelítést alkalmaznak statikus elemzés esetén: absztrakt interpretáció (abstract interpretation), szimbolikus végrehajtás (symbolic execution), és axiomatikus módszerek. Az iparban használt megoldások sokszor kombinálják a kűlönböző megközelítéseket.

Az utóbbi időben, a geopolitikai konfliktusok miatt egyre gyakoribbak a kibertámadások, az esetek jelentős részében memóriakezeléssel kapcsolatos hibák okoznak kihasználható sérülékenységeket. Emiatt sokan dolgoznak olyan elemzőeszközökön, amik garantálni tudják, hogy bizonyos memórakezelési hibák nem történhetnek meg a programban. Ezek az eszközök jelenleg sajnos csak limitált formájú programokról tudják bebizonyítani, hogy biztonságosak, emiatt a támadási felületül szolgáló szolgáltatásokat néha újra kell írni a biztonság érdekében.

Imolai Dávid

EJC: 2023–

david@imol.ai

Imolai Dávid vagyok, a szülővárosom Székesfehérvár, és idén nyertem felvételt az ELTE-re, a Collegiumba, az Informatika Műhelybe.

Már fiatalkoromtól kezdve érdekeltek a reál tudományok, óvodában eltökélt álmom volt mérnöknek menni. Szerencsésnek érzem magam, hogy én még nem okostelefonnal a kezemben születtem, viszont már abba a generációba tartozom, akinek az életében szerves részt képez az informatika, amelyet lehet jó célokra is használni.

Az első igazi találkozásom az informatikával 2017-ben volt, amikor gimnazista lettem, és kaptam egy notebookot. Természetesen az elején még átlagos felhasználói célokra (zenehallgatás, filmnézés stb.) használtam, de idővel kíváncsi lettem, hogy mi hogyan működik, és elkezdtem jobban felfedezni az operációs rendszerek világát.

2020 nyarán megépítettem a saját asztali gépemet, amire már Linuxot telepítettem. Sokféle disztribúciót végigjártam (Fedora, Manjaro, Gentoo stb.), de most jelenleg Arch Linuxot használok. Az azóta eltelt pár év alatt egyre gyorsuló ütemben ástam bele magam a gépek világába, és 11. osztályban 98%-os előrehozott emelt szintű informatika érettségit tettem.

Még nem döntöttem el, hogy a tudomány pontosan melyik ágazatával szeretnék továbbtanulni, de ha tehetném, egyszerre mindegyiket választanám. Vettem már részt több kompetitív programozási versenyen, és a kedvenc versenyfajtámnak tartom a CTF (Capture The Flag) típusú, kiberbiztonsági versenyeket. Kiemelném ezekből a HCSC (Hungarian CyberSecurity Challenge) versenyeket, amelyeken az utolsó két évben junior kategóriában 3. lettem. A tavalyi évben ezután a verseny után kiutaztunk a magyar csapattal (top 5-5 junior-senior) Bécsbe, ahol a 2022-es ECSC-t (European CyberSecurity Challenge-et) tartották,

idén pedig Norvégiába repültünk az éves versenyre. Kifejezetten érdekesnek és izgalmasnak tartom a különböző rendszerekben megtalálni a sebezhetőségeket, és a CTF versenyek az átlag informatikai versenynél úgymond „szaftosabbak". Tehát a kiberbiztonság számomra egy olyan része az informatikának, amely az átlagnál is jobban érdekel, van rá esély hogy ezzel folytatom majd a tanulmányaimat.

Továbbá az egyetemen, és az informatika műhelyben eddig töltött pár hónap alatt még több terület ragadta meg a figyelmemet. Például a matematika eddig nem volt a kedvencem, de ezalatt a 3 hónap alatt úgy érzem, hogy megtaláltam a szépségét. Ez azért is kiváló, mert a kriptográfia egy olyan tudományág, amely mindig is érdekelt, viszont eléggé nehéz megfelelő matematikai háttértudás nélkül tanulni azt. A C nyelvcsaládban és funkcionális nyelvekben is nagyon élvezek programozni, például a Haskell kifejezetten megmozgatja az ember agyát, hogyha direkt egy kicsit „nyakatekerten" szeretne megoldani egy feladatot. Emellett open-source / libre szoftverek felhasználója, és kisebb mértékben fejlesztője vagyok.

Várom, hogy az egyetem óriási tárgykínálata által sok, számomra érdekes témáról tanulhassak, és hogy a már 20 éves Informatikai Műhelyben szerzett tudással kiegészítsem azt.

Katkó Dominik

EJC: 2019–2022

katkodominik@gmail.com

Az Eötvös József Collegiummal először a Természettudományos Táborban találkoztam, gimnazistaként, és már ott elhatároztam, hogy collegista szeretnék lenni.

Pár évvel később, 2019-ben fel is vettek várólistára, végül a Csaposfelvételi Estikéig kellett várnom, hogy bent aludhassak először az épületben. Az Estikétől utána se jutottam messzire, az akkori csapos-évfolyam egy nagyon összetartó apró közösség volt, többek között mi újítottuk fel a Sakk-collegium szobáját, és szereltünk fel új hangszórókat, de a számomra legfontosabb tevékenységünk a rendszeres vasárnapi esti közös főzés és vacsoránk volt.

A Collegiumban a közösségi tevékenységem szépen lassan a különböző tisztségek halmozásából állt. Az első félévem után lettem a Kommunikációs Bizottság elnöke a Választmányban, ahol többek között új szóróanyagokat terveztünk, megújítottuk az Epistola Collegii hírlevelet, és interjúsorozatot indítottunk az alumnival. Szintén az első évben még beszálltam az Urán fejlesztésébe, nem sokkal később pedig rendszergazdaként szereltünk fel új routereket, hogy a Collegium wifi hálózatát frissítsük. Az alapképzésem utolsó évére Csertán Andrással közösen a főcsaposi rangot érdemeltük ki az Estike vonatkozásában, miközben a Kommunikációs Bizottság helyett a Közösségi Bizottságért lettem felelős. A „KözBizzel" több év kihagyás után először rendeztük meg az Eötvös Conviviumot.

A közösségi tevékenységek mellett tanítani kezdtem mind az ELTE-n, mind a Collegiumban. Az egyetemen az Objektumelvű programozás „tervezős" óráit, valamint Programozáselmélet gyakorlatot tartottam két éven keresztül, amit rendkívül élveztem. A Collegiumban többekkel közösen elindítottuk a „Bölcsészinfót", ahol nem informatikus hallgatóknak tanítottunk Python alapokat. Ezen felül Luksa Norberttel az

Informatikai Műhely számára open-source fejlesztést és git alapokat tanítottunk, összekötve az Urán fejlesztésével is.

Habár kutatási tevékenységem kevésbé kiemelkedő, a Collegiumnak köszönhetően kezdtem el közösen dolgozni Horpácsi Dániellel az ELTE-ről és Simon Thompsonnal a Kenti Egyetemről. Velük készítettem el a szakdolgozatomat: az Erlang nyelvhez használt *Wrangler* refaktoráló eszközhöz [1] készítettem egy kiegészítőt, a *Wrangler Language Server*t [2], hogy a refaktoráló funkciókat a mai modern kódszerkesztő alkalmazásokban is lehessen használni. A munkámat később lehetőségem volt előadni a *Code Beam America* [3] konferencián 2022-ben, San Franciscóban, ahol az azóta kiadott eszközöm felhasználóival is személyesen találkozhattam, és hasznos ismeretségekre tettem szert.

A collegiumi tagságom 2022 nyarán szűnt meg, távozásomkor odaítélték nekem az *Eötvös Collegiumért Emlékérmet*. Megtisztelő volt többek között Kozsik Tamással együtt átvenni az érmeket, aki azóta jelenleg az ELTE Informatikai Karának dékánja.

Jelenleg a mesterképzésemet Dániában, a *Technical University of Denmark* intézményében töltöm, fókuszálva a programanalízis, szoftver verifikáció, modellellenőrzési technikák, és szoftverek egyéb formális tulajdonságaira. Mindeközben részállásban dolgozom a Trifork cégnél, ahol főleg webes projekteken dolgozok *React*ben és *Elixir*ben.

A dániai tartózkodásom alatt tagja lettem a *Board or European Students of Technology* (BEST) diákszervezetnek, ahol különböző programokat szervezek. Itt vezetőképző és egyéb tréningeken is részt vettem, és tapasztalataimmal tervezem valamilyen formában támogatni a collegiumi Választmány működését a továbbiakban is.

Hivatkozások

[1] Wrangler, `https://refactoringtools.github.io/docs/wrangler/`

[2] A Wrangler Language Server, `https://refactoringtools.github.io/wrangler/wls/wrangler_language_server`

[3] Code Beam America, `https://codebeamamerica.com`

Kenessei Zsombor

EJC: 2020–2022

Kenessei Zsombor néven láttam meg a napvilágot 2001-ben, Magyarország nyugati felében, Sopronban. Életem első 18 évét ezen a környéken töltöttem, mígnem a nagyváros ígérete és az ELTE elhelyezkedése Budapestre nem csábított. Locsmándi Dániel, középiskolám egyik történelemtanára ajánlotta, hogy jelentkezzem a Collegiumba. 2020 szeptemberében így lettem az Informatikai Műhely bejáró tagja.

A másfél év, amíg „jogilag" collega voltam, ösztönző hatással volt rám. Szakmai szempontból a műhely tehetséges és már-már ijesztően intelligens tagjai arra inspiráltak, hogy meg nem alkuvó fegyelemmel és érdeklődéssel viszonyuljak tanulmányaimhoz. Ez a hozzáállás végigkísért az egyetemi éveim alatt.

A nyitott, sokszínű és sokrétű társaság, amibe csöppentem, újra belobbantotta bennem a művészetek iránti rajongásomat, különösen a színjátszás, tánc és zene területén. Az Eötvös Kísérleti Énektanszék tagjai között meglepően sok infó és matfiz műhelyes collega zenélt és/-vagy énekelt, köztük én is. Miután különváltak útjaim a Collegiumétól, ez a zenekar megmaradt egyfajta horgonypontként számomra, folytatva a közös munkát, ameddig csak tudtam.

2023-ban visszaköltöztem szülővárosomba, munka mellett fejezve be az alapszakos programtervezői diplomámat. Az év nagy részében a Győr–Sopron–Ebenfurti Vasút informatikai csapatát erősítettem, az év vége felé pedig egy prózaírói kurzust végeztem. Jelenleg egy vállalatirányítási szoftver egyik fejlesztője vagyok, szabadidőmben pedig első novellaskötetemen dolgozom, amelyben visszaköszön pár collegiumi élmény.

Hatalmas megtiszteltetésként tekintek arra, hogy az Informatikai Műhely és a Collegium tagja lehettem, még ha rövid időre is. A kedves collegáktól kapott értékek és élmények erősen alakították azt, akiként magamra gondolok és biztos vagyok benne, hogy a negyvenéves kötet-

ben is így fogalmaznék majd! Kívánom, hogy az Informatikai Műhely még sokáig formálja az ifjabb nemzedékek szemléletmódját!

Kiss Ádám

EJC: 2018–2021

Kicsit mindig féltem a pályaválasztásommal kapcsolatban, mert úgy éreztem, hogy több „bölcsész hobbim" van, mint klasszikus „informatikus hobbim". Mind gimiben, mind egyetemen egész kevés szakmai hobbiprojektem volt, és kicsit szégyelltem is magam, hogy a legérdekesebb git repo-m egy egyetemi beadandóm erős továbbfejlesztése volt. Helyette sokkal jobban érdekelt az irodalom és a képzőművészet, így gyakrabban bukkantam fel műtermekben és kiállításokon mint szakmai konferenciákon. Részben emiatt is jelentkeztem az EC-be még 2018-ban, mert úgy éreztem, itt a Collegium erős bölcsész vonala miatt színesebb a közeg.

Egyetemi éveim során megkeresett egy cég, hogy dolgoznék-e náluk, mint fejlesztő. Gondolkozás nélkül mondtam igent, a cég ugyanis műkincselemzéssel, műtárgyakba való befektetéssel, illetve aukciókhoz köthető adatok elemzésével foglalkozó tech cég volt. Ezen az érdekes piacon teljesen megtaláltam magam, mind a bölcsész, mind a reálos hobbijaimmal együtt.

BSc-m utolsó éve óta dolgozok a cégnél, és MSc-m során sem lankadt a lelkesedésem a téma iránt. Olyannyira nem, hogy hamarosan két barátommal – akikkel még az IK-n ismerkedtem meg – saját műkincskereskedéssel foglalkozó projektet készítünk a munkánk mellett. Ez már a műtárgyvásárlási fázisban van, hamarosan elindulnak az eladások is, ha minden jól megy.

MSc tanulmányaimat már a BME-n folytattam. Ott az igazságosság matematikai vizsgálatával foglalkoztam, és készítettem TDK dolgozatot a témában, ahol egy speciális interpolációs eljárást felhasználva számítok ki várható munkaterhet alkalmazottakra, illetve keresek igazságtalan elbánásokat komplex beosztásokban.

Kocsis Ábel

EJC: 2017–2020

2017. augusztus 21. este 11 körül az Estikében döntöttem el igazán, hogy az EC tagja szeretnék lenni. Ezen a napon azontúl, hogy megismertem leendő műhelytársaimat, és jókat beszélgettem a colisokkal, este részem volt megtapasztalni, hogy a szakmaiság mellé milyen közösségiség társul. Az estét sosem felejtem, a döntésemet pedig azóta sem bántam meg.

Elsőéves koromban meglehetősen ijesztőnek és bonyolultnak tűnt bármilyen extra projekt, emiatt eleinte többnyire kapcsolatépítésre és egyéb közösségi elfoglaltságokra koncentráltam. Sok baráttal gazdagodtam, Estike csapos lettem, és elkezdtem részt venni EC-s programok szervezésében.

Első egyetemi évem végére lassan elkezdtek biztatni társaim, hogy vállaljak többet az egyetemen. Így csatlakoztam az ELTE Innovációs laborhoz, ami ezen évem egyik legmotiválóbb része volt, és megismerkedhettem a startupok alapjaival, valamint kipróbálhattam, milyen egy játékos startup elkezdése.

Eötvös collegista és Neumann-körös hallgatóként sok kapu nyílt meg a kari életben. Másodéves koromban Java gyakorlatokat kezdtem el tartani, később pedig Funkcionális programozás (Haskell) gyakorlatokat oktattam. Sok örömmel töltött el látni a hallgatók fejlődését.

Alapképzéses éveim végén kezdtem el a kiberbiztonság témakör felé érdeklődni. Ebben jobban Tóth Melinda és Gera Zoltán témavezetésével mélyedhettem el, akikkel biztonsági fókuszú statikus kódelemző eszközöket hasonlítottam össze, valamint ezek közül egyet továbbfejlesztettem. A kutatás ÚNKP támogatásban részesült, és a 2020-as évben 1. díjat kapott a TDK-n.

Látva motiváló, aktív EC-s műhelytársaimat, sok időt igyekeztem fordítani a Collegium közösségére. Részt vettem olyan események szervezésében, mint az Eötvös Konferencia, GóJatábor, valamint első megszervezője voltam az elsőévesek bevonását célul kitűző kurzusnak, ami

akkoriban a bECezdés nevet viselte. Ezeket a tevékenységeket egy évig mint választmányi alelnök, fél évig mint elnök láttam el. Azon túl, hogy a legtöbb esemény jó indok volt egy-egy hatalmas bulizásra, különösen örültem, amikor az utánunk jövő generációkat sikerült bevonni a collegiumi életbe.

2020 nyarán búcsúztam el a Collegiumtól és a műhelytől. Mestertanulmányaimat az Edinburgh-i Egyetem kiberbiztonság képzésén (Cyber Security, Privacy, and Trust) folytattam Stipendium Peregrinum ösztöndíjjal. Itt a Covid időszak sok kihívást és nehéz időszakot tartogatott, de sok jó élménnyel is gazdagodtam. 2021 decemberében kaptam meg diplomámat, diplomamunkám címe „RSA Accumulators and Simultaneous Broadcast Channel for Decentralised E-voting" volt.

2021 augusztusa óta egy manchesteri kiberbiztonság startupnak dolgozom vezető fejlesztőként. A kis céggel digitális valuták és tokenek védelmét igyekszünk megoldani a világon talán egyedülálló módon: saját fejlesztésű „cold storage" technikával. 2023-ban hazaköltöztem Budapestre, és online munkával igyekszem hozzájárulni a sikereinkhez.

Az utóbbi évek tapasztalatai alapján sokszor elgondolkodom a mögöttem álló collegiumi és egyetemi évekről. Egyrészt nagy örömmel tölt el, hogy az egyetemen kapott tudást a gyakorlatban hasznosítani tudom, és az egyetemi órák olyan tudással láttak el, amitől már az első naptól értékes tagja tudtam lenni a cégnek. Másrészt az Edinburgh-i Egyetemen és a cégemnél is nagy élmény volt látni, hogy a színvonalat megtartva, de kicsit lazábban és szabadabban is jól tud működni egy intézmény.

A Collegiumba a mai napig szívesen megyek vissza akár TermTud-Táboros előadásról, akár DJ-zésről, akár focizásról legyen szó. Az itt és a műhelyben kialakult barátságok pedig a mai napig fontos részét képezik az életemnek.

Kocsis Zoltán Attila

EJC: 2012–2013

z.kocsis@unsw.edu.au

Húsz éves az Eötvös Collegium Informatikai Műhelye. Bár én csak 2012–2013-ban voltam része a Műhely közösségének, az itt megszerzett tudás mégis meghatározó hatást gyakorolt mind a karrierem, mind az életem alakulására.

Középiskolai éveim alatt Csörnyei Zoltán *Lambda-kalkulus* c. könyvén keresztül értettem meg, ismertem fel, hogy a matematika és informatika egymást nagyon is átfedő tudáságak. Ez segített a döntésben, hogy mely pályán induljak el. Amikor 2012-ben felvételt nyertem az egyetemre, nem volt kérdés, hogy az Eötvös Collegiumba, az Informatika Műhelybe fogok jelentkezni, melyet Csörnyei tanár úr vezetett.

Műhelytagként végeztem el Diviánszky Péter kétszemeszteres kurzusát az *Agda* funkcionális programozási nyelvről és tételbizonyító rendszerről. Péter egyedi oktatási stílusa mély benyomást tett rám: valahányszor a funkcionális programozásról vagy bizonyításasszisztensekről adok elő, azon trükkökhöz, vizualizációs technikákhoz nyúlok vissza, melyeket az ő kurzusából lestem el. Hasonló élmény volt Kaposi Ambrus *Kategóriaelmélet* különóra-sorozata is (Ambrus akkor Nottinghamben élt, az órákat 1–2 havonta tartotta, amikor hazalátogatott).

2013-ban külföldre költöztem, a Műhelyt elhagytam. Későbbi egyetemi tanulmányaim során érdeklődésem az informatika felől fokozatosan a „tiszta” matematika irányába fordult. Doktori tanulmányaimat 2016-ban kezdtem Angliában, a Manchesteri Egyetem elméleti matematika programjában. Választott kutatási témám, a csoportelmélet és a nemsztenderd analízis kapcsolatának vizsgálata, az informatikához szorosan nem kapcsolódott.

A fordulópont 2018-ban érkezett el, amikor Imamura Takuma japán matematikus rámutatott egy komoly hibára a Manevitz–Weinberger-

tétel publikált bizonyításában. A legrosszabbtól tarthattam, hiszen ez a tétel volt az alapja készülő disszertációm legfontosabb eredményeinek is. Annak érdekében, hogy saját munkám helyességéről megbizonyosodjam, egy olyan eszközhöz fordultam, melynek használatát a Collegiumban sajátítottam el: egy év alatt az egész disszertációmat [1] formalizáltam Agda-ban.

Ez a kihívás visszavezetett az informatika világába, és egy váratlan ajtót is megnyitott. Az Agda-ban való jártasságom nagyban hozzájárult ahhoz, hogy a doktori fokozat megszerzése után az ausztrál nemzeti tudományos ügynökséghez, a CSIRO-hoz kerültem kutatói állásba. Ott az operációs rendszerek helyességbizonyításában értem el eredményeket: többek között az seL4 kernel új, RISC-V processzorokon futó verziójának fordításhelyességét bizonyítottuk tételbizonyító és SMT-megoldó (SMT: satisfiability modulo theories, vagyis kielégíthetőség modulo elméletek) szoftverek segítségével.

A fordításhelyesség a C forráskód formális szemantikája és a bináris program szemantikája közötti azonosság bizonyítását jelenti: így ellenőrizzük a fordítás helyességét, és egyúttal igazoljuk, hogy a forráskódra bebizonyított tulajdonságok (funkcionális helyesség, adatintegritás, titoktartás) a futtatható, lefordított kernelre is érvényesek. Ehhez nincs szükség bizonyítottan helyes fordítóprogramokra, mint pl. a *CompCert*; a sima, `gcc` által fordított binárist használhatjuk, annak minden előnyével, optimalizációjával együtt.

A projekt végeztével a University of New South Wales informatika tanszékén helyezkedtem el (UNSW Sydney, Department of Computer Science and Engineering), jelenleg is az ausztráliai Új-Dél-Wales államban élek. Fő kutatási teruleteim a kielégíthetőségi fokok elmélete (ez azt vizsgálja, hogy egy matematikai struktúrában az elemek legfeljebb hány százaléka elégíthet ki egy adott logikai kifejezést), és az SMT-megoldó szoftverek további alkalmazása helyességbizonyításokban.

Hivatkozások

[1] Z. A. Kocsis, Development of Group Theory in the Language of Internal Set Theory, *University of Manchester administered thesis, Ph. D.*, (2019)

Komáromi Mátyás

EJC: 2016–2019

matyas@komaro.me

2016-ban kezdtem az egyetemet, és az első félévtől az Eötvös József Collegiumban laktam. Az Informatikai Műhely tagja voltam, de a Matematika–Fizika Műhely óráit is rendszeresen látogattam. Az ELTE-n sok szabadon választható tárgyat elvégeztem (összesen 240 ECTS), mindenkinek szívből ajánlom az Agda, a Funkcionális programozás és a Számítógépes grafika órákat. Versenyeken is részt vettem; ha a matematika érdekli a hallgatót, a Hajós György Matematikaversenyt ajánlom. Programozási versenyek közül pedig a különböző cégek csapatversenyei nagyon hasznosak (pl. Ericsson), mivel a csapatmunkát tanulni hasznos, és valós képességek szükségesek hozzá. A Tudományos Diákköri Konferencián is részt vettem, ahol két évben első díjat kaptam, illetve az Országos Tudományos Diákköri Konferenciára is továbbküldték munkámat. Meg is jelent két folyóiratban cikkem; ez a fajta kutatómunka sok energiát igényel, de mindenképp érdemes kipróbálni, jól modellezi a kutatói munkát, amit az ember az egyetemen maradva tapasztalna.

A Collegium tagjaként Csörnyei tanár úr Lambda kalkulus óráira emlékszem vissza szívesen, a mai napig, mikor funkcionális kódot írok, jó érzéssel jutnak eszembe a pincében töltött esték. Bár nem éltem túlzottan társasági életet akkoriban, a Collegiumnak köszönhetően több közeli barátom is lett; kiemelném itt Bagi Richit és Boros Attilát (remélem, az ő írásaikat is olvashatjuk ezeken a lapokon). 2019-ben diplomáztam le, hatékony gráfmegjelenítéssel foglalkozó dolgozatommal és kitűnő jegyeimmel kitüntetést kaptam.

A BSc után az olasz *Università degli Studi di Trento* és a finn *Aalto Yliopisto* egyetemek közös dupla-képzésére mentem az EIT Digital Master School keretében. Ez egy igazán izgalmas döntés volt, hiszen korábban nem éltem, és nem is jártam sokat külföldön. Ellenben a

program témája, a „Visual Computing and Communication”, közel állt a szívemhez, a minor Innováció és vállalatmenedzsment pedig azzal kecsegtetett, hogy valami igazán újat és mást is tanulhatok. Emellett egy jó ösztöndíjat is ajánlottak, így elfogadtam a lehetőséget. Rendkívül kíváncsi voltam, hogy az olasz és a finn kultúra mennyire különbözik, és hogy melyik fog jobban tetszeni.

Az első év nehezen indult. Az olaszok törik az angolt, én pedig nem tudok olaszul... De alakultak barátságok magyarokkal és külföldiekkel egyaránt, és mire észre vettem, fél év el is röppent. Trento északon, az Alpokban található, és én teljesen beleszerettem a hegymászásba. Rengeteget túráztam és másztam ott. Ezzel párhuzamosan elkezdtem a társasági és sportéletre fókuszálni jobban, a tanulmányaimat persze nem elhanyagolva.

Majd beütött a COVID. Olaszország nagyon szigorú szabályokat vezetett be, és én hazajöttem, hogy otthonról, távoktatásban fejezzem be az évet. Sokat lehetne arról is írni, hogy ezek a lezárások hogyan csorbították meg rengeteg tanuló élményét, de arról is, hogy milyen gyorsan és kevés következménnyel vészeltünk át egy globális járványt... Ez az időszak viszonylag eseménytelenül telt a számomra, azt viszont igazán megtanultam, hogy személyesen jelen lenni egy órán többet jelent, mint hinné az ember. Az a kapcsolat, ami a diáktársakkal kialakul, később felbecsülhetetlen értékké válik.

Eljött a második év, és egyszer csak Finnországban voltam, Helsinkiben. Eleinte minden finn szóra felkaptam a fejem, a hanglejtésből azt hittem, magyarok beszélnek. Ennek ellenére északi testvéreink rendkívül különböző mentalitással élik életüket, mint mi. Mikor hall olyat az ember itthon, hogy maszkot viselni csak javaslat, és mégis mindenki önkéntesen megteszi? A finn nyelv határozottan megtetszett, el is végeztem egy alap finn kurzust. A diákélet Helsinkiben elképesztő. Mikor a COVID lentebb hagyott, igazán megtapasztalhattam, mennyi kreatív esemény, feladat, hagyomány és társasági élet van ott. A szaunakultúra megfogott, és a szauna társulat állandó tagja lettem. De volt sörpong társulat, krikett társulat, falmászó társulat és minden egyéb.

Az olasz és finn oktatási kultúra is rendkívül különböző. Volt olyan órám Trentóban, aminek a nagy beadandójára nem is volt határidő. Ha egy évvel később adta be az ember, akkor is maximális pontszámot kaphatott rá. A finneknél majdnem minden órára rendszeres (nem triviális) házi feladatot kaptunk. Ellenben az olasz vizsgák néha igencsak

nehezek (az anyaghoz nem szorosan kapcsolódóak) voltak, míg északon szinte nem is volt vizsgám.

Minden jó véget ér egyszer, és ez a mesterképzésemre is igaz. 2021-ben a koherens fényhatások valós idejű szimulációjával zártam le az egyetemi tanulmányaimat. Summa cum laude végeztem mindkét egyetemen, ezzel megkoronázva öt év tanulmányait.

A mester után Finnországban találtam munkát, annál a cégnél, ahol a diplomamunkámat írtam (az EIT miatt kötelező volt cégnél írni). 3D-ben látó kamerákat fejlesztő kis startup volt, Ladimo Oy volt a neve. Itt megismerkedtem az ipari fejlesztői munka alapjaival, és az idő elkezdett rohanni. Egy kollégám 2022-ben állást váltott, Norvégiában, Oslóban kezdett dolgozni, egy hasonló, de nagyobb cégnél, és felkért, hogy csatlakozzak én is.

Hát jelentkeztem. Az eredményeim és az interjú is lenyűgözte a céget, így kaptam egy állásajánlatot, és két hónapon belül már Oslóban is laktam. A norvégok és a finnek rendkívül különbözőek. Ezzel is meg lehetne tölteni cikkeket, de a lényeg az, hogy időbe telt, mire beilleszkedtem Norvégiában, és kialakultak újra a hobbijaim. Ezen a ponton a Zivid AS szoftverfejlesztője vagyok, nagyrészt GPU programozással foglalkozom, és nagyon élvezem, mikor algoritmikailag nehéz feladatokon dolgozhatok, hogy a kameráink jobb pontfelhőket állíthassanak elő, gyorsabban. Emellett a Norvég ESN kincstárnoka is vagyok, és igyekszem bejárni az ország legszebb hegyeit.

Több közeli barátra is szert tettem az elmúlt négy évben. Még Thaiföldön is jártam, hogy meglátogassam egyiküket. Szerintem az élet egyik sarokköve a kaland. Aki nem vállal semmilyen kockázatot, az élete végén csak arra fog tudni gondolni, hogy mennyi lehetőséget hagyott ki. Én is várom a következő nagy kalandomat.

Kovács Gergely Zsolt

EJC: 2019–

Az Eötvös József Collegium Informatikai Műhelyéhez 2019 őszén csatlakoztam, egyidejűleg az ELTE Informatikai Karának programtervező informatikus BSc képzésének elkezdésével. Itt később a modellező („A") szakirányát választottam, amit 2022 tavaszán kitüntetéses diploma elnyerésével zártam. Ekkor kezdtek el különösen érdekelni a kiberbiztonság különböző területei, így ezt követően a programtervező informatikus MSc angol nyelvű *Cybersecurity* specializációján folytattam tanulmányaimat, melynek jelenleg is a hallgatója vagyok, és amit 2024 tavaszán tervezek befejezni. Mindeközben természetesen a Collegiumban is számos rendszeres, illetve egyszeri kurzuson vettem részt, utóbbiak közül talán a *A mesterséges intelligencia társadalmi kihívásai* nevű bizonyult a legelőremutatóbbnak a nagy nyelvi modellek későbbi elterjedésével.

Az elmúlt, a 2023/24-es tanév első félévét Erasmus+ ösztöndíjjal a hollandiai *Radboud University*-n töltöttem *Nijmegen*ben, ahol szintén kiberbiztonsággal kapcsolatos kurzusokat hallgattam.

BSc tanulmányaim alatt kezdtem el gépi tanuláshoz kapcsolódó témákban kutatni az ELTE *MLforSE (Machine Learning for Software Engineering)* kutatócsoportjához való csatlakozással. A BSc szakmai gyakorlatot és később a diplomamunkát is ebben a témában folytattam, illetve készítettem, utóbbit Pintér Balázs témavezetésével *Assembly optimalizáció mélytanulás segítségével* címmel. Később, már a mesterképzés alatt ugyanezen csoport tagjaként Szalay Gergővel és Teleki Sándorral részt vettem Pintér Balázs és Gregorics Tibor témavezetésével egy TDK dolgozat elkészítésében *Neurális kódvisszafejtés rekurzív szegmentálással* címmel, amely az ELTE Informatikai Kar intézményi TDK Konferencia sorozatának Informatikatudományi Szekciójában első díjat nyert el, így a jövőben az OTDK-n való prezentálására is lehetőségünk nyílik.

Részt veszek ezen felül az Európai Unió *Digitális Európa* program *EuroQCI* kezdeményezése alá tartozó *QCIHungary*[1] projekt ELTE által végzett részében, amelynek keretében a *Quantum Key Distribution* (kvantum kulcselosztás) technológia alkalmazására irányuló kutatási, illetve programfejlesztési feladatokat végzünk.

Erasmus+ ösztöndíjjal eltöltött félévem előtt volt és jelenlegi collegisták ajánlására a *Cloudera* budapesti irodájában dolgoztam gyakornokként körülbelül másfél évig szoftvertesztéléssel kapcsolatos feladatokon.

Örülök, hogy eddigi egyetemi éveim alatt a Collegium tagja lehettem, és remélem, hogy a jövőben is részese maradhatok.

[1] `https://qcihungary.hu`

Kovács Máté

EJC: 2005–2010

Kovács Máté (Szalacsi) vagyok, az Eötvös József Collegium Informatikai Műhelyének 2005 és 2010 között voltam programtervező matematikus hallgatóként tagja. Annak idején, amikor Lócsi Levente Tanár Úrtól átvettem az Eötvös Collegiumi Teremfoci-bajnokság holnapjának szerkesztését, még nem gondoltam, hogy egyszer majd webes frontend fejlesztőként fogom keresni a kenyeremet.

Pályafutásomat 2010-ben kezdtem Budapesten a Nokia Siemens Networks-nél, ezután 2013 és 2019 között dolgoztam szoftvertesztautomatizáló mérnökként az EPAM Systems-nél, amiből 3 évet Zürichben töltöttem. Akkoriban írtam webes tesztautomatizálás témában a „Tízéves az ELTE Eötvös József Collegium Informatikai Műhelye" kötetbe.

2019-ben csatlakoztam egy játékfejlesztő csapathoz, és velük sikerült 2021-re elkészíteni a sci-fi témájú Five Nations valós idejű stratégiai játékot, amely elérhető a Steamen. Mindezzel egy gyerekkori álmom vált valóra. A játékfejlesztés mindig is közel állt hozzám: Eötvös collegistaként előadtam a TTK-s esten, az Eötvös Konferencián és az Adsumus kötetben is megjelent cikkem ebben a témában.

2020-ban kerültem a Senso Media Zrt.-hez, ahol fullstack fejlesztőként dolgoztam Symfony PHP és React JS technológiával. Ebben az évben ütött be a koronavírus a járvány, így megismerkedhettem a távmunkavégzés előnyeivel.

2023 márciusában a Diligenthez kerültem, ahol React frontend fejlesztőként dolgozom. A tesztautomatizálástól sem távolodtam el, mert része a napi munkánknak. Itt a Selenium után a Cypress keretrendszerrel ismerkedhettem meg. Programozási nyelvként a Typescript-et használjuk, amiben hatványozottan igaz, hogy „a típus csak illúzió".

2021-ben költöztünk feleségemmel, Katával Budapestről Dinnyésre, mely az Eötvös collegisták előszilveszterezésének gyakori, kedvelt helyszíne. Két gyermek boldog apukája vagyok, Adrienn 2021-ben,

Bence pedig 2023-ban érkezett a Kovács családhoz. Kertes házban lakunk, ahol gyümölcsfákat és magaságyásokat gondozunk, így az Eötvös MGTSZ hagyományait is tovább tudom vinni.

Visszatekintve az Eötvös Collegiumi és Informatikai Műhelybeli tagságomból nagyon sokat tudtam profitálni. A szakmai fejlődésen kívül az angol nyelvtudás fejlesztése, illetve a Coliban megszerzett kapcsolati tőkének látom leginkább hasznát, hiszen az ember bárhol jár szerte a világban – legyen szó Zürichről, New Yorkról vagy épp az Antarktiszról – tetszőleges $\varepsilon > 0$ sugarú környezetben jó eséllyel találkozni fog Eötvös collegistákkal.

Kovács Péter

EJC: 2007–2013

peter.kovacs496@gmail.com

2007-ben kezdtem meg tanulmányaimat az ELTE Informatikai Karán. Ezzel párhuzamosan az Eötvös József Collegium Informatikai Műhelyének is tagjává váltam. A szakkollégium kiváló lehetőséget nyújtott az egyetemi tanulmányok kibővítésére és hasonló érdeklődésű emberek megismerésére. 2013-ban az MSc diplomával lezárult életemnek ez a fejezete.

Mindig is szerettem a matematikát, az egyetemi évek alatt is efelé orientálódtam a modellalkotó szakirány választásával. Igyekeztem olyan munkát találni, ahol szintén szükség van ilyen tudásra. Az első munkahelyemen egy kutatás-fejlesztési projektben vettem részt, melynek célja drónok ütközésvédelmének biztosítása volt. Ezt két fő érzékelő, az orrkamera és a radar segítségével értük el. A radarjelek feldolgozása komoly kihívást jelentett, hiszen valós időben kellett megtörténnie. Nyers radarmérési adatból objektumlistát állítottunk elő: a detektált objektumoknak meghatároztuk a távolságát, sebességét és irányát.

Egy másik munkahelyen képfeldolgozási projektekben vettem részt. Volt, hogy logót kellett beilleszteni egy videóra életszerűen. Máskor pedig plázákba bemenő és onnan kijövő embereket detektáltunk a bejáratra felszerelt kamerák mélységtérképe alapján. Érdekes tapasztalás volt, hogy az időjárási körülmények mennyire befolyásolni tudják a helyes paraméterezést.

Dolgoztam egy építészettel foglalkozó cégnél is, ahol ArchiCAD-hez fejlesztettünk kiegészítőket. A legnagyobb kihívást az jelentette, hogy háromdimenziós elemeket adtunk hozzá az épületmodellhez, például álmennyezet esetén kazettákat, tartókat és függesztőket. Ezeknek ki kellett számolni a pozícióját különböző alakú álmennyezetek esetén, melyek nem is feltétlenül voltak vízszintesek, ha tetőre rögzítették őket.

A Collegiummal való kapcsolat nem szakadt meg a kiköltözéssel, hiszen a drámakörbe évekig visszajártam, több előadásban is szerepeltem a Nagyklubban és az Estikében. Szakmailag és emberileg is sokat kaptam a Colitól, jó szívvel gondolok vissza az ott töltött időre.

Köllő Hanna

EJC: 2004–2009

hanna.kollo@gmail.com

Az Informatika Műhely első éveiben, 2004 és 2009 között voltam műhelytag. Csörnyei tanár úr órái, az Eötvös Konferenciák és a collegisták társasága meghatározó élmény volt számomra azokban az években... Az egyetem utolsó éve alatt Hollandiában tanultam egy ösztöndíjprogram keretein belül, és végül oda is költöztem.

Mesterképzés után az iparban helyezkedtem el informatikusként, előbb pénzügyi területen majd egy játékfejlesztő cégnél, végül egy online kereső-óriásnál, a Google-nél próbáltam ki magam, és ez utóbbi munka Svájcba szólított. A Google-nél a nagyméretű webalkalmazások skálázhatóságán és robusztusságán dolgoztam, majd később a „big data" problémakörében mélyedtem el. Az elmúlt hat évben a Google Maps csapatában, annak data analytics platformjának építésével foglalkoztam szoftverfejlesztőként és technikai vezetőként.

Több száz millió felhasználó személyi adatainak kezelése nemcsak technikai, hanem etikai követelményeket is támaszt. Szerencsére ma már mindenhol, de főleg az EU-ban erős törvények védik a felhasználók személyi jogait, így az olyan érzékeny adatokat, mint a felhasználó pontos holléte, csak akkor tároljuk, ha azt a felhasználó explicit módon jóváhagyta, valamint csak olyan célból használjuk, amihez ő hozzájárult, és ez komplex, de elegáns szoftvertechnológiai megoldásokat igényel. A felhasználói adatok anonimizálása is jóval bonyolultabb eljárás, mint egy egyszerű aggregáció, és elméleti statisztikai kihívásokat jelent.

Számomra érdekes egy olyan helyzetben létezni, ahol egy-egy szakmai döntésem több millió ember életét befolyásolja, ha csak egy parányi mértékben is, és meggyőződésem, hogy ezzel a befolyással hatalmas felelősség is jár. Talán emiatt kezdtem el hobbiként az etikával foglalkozni.

A Collegium széleskörű műveltséget, szellemi nyitottságot, és az interdiszciplinaritás szeretetét adta nekem útravalóul, amire azóta is büszke vagyok.

Krupinszki Balázs

EJC: 2017–2020

krupybalu@gmail.com

Még az érettségi utáni eufória állapotában élveztem a nyári szünetet 2017-ben, mikor egy korábbi felsőbb éves ismerősöm megkeresett engem és pár osztálytársam, hogy bemutassa az Eötvös Collegium Informatika Műhelyének munkásságát, illetve ösztönözzön minket a jelentkezésre. A pár órával a határidő előtt leadott jelentkezésem mind magán, mind szakmai életem egyik legmeghatározóbb döntésének tartom, hiszen a Collegium nem csak rengeteg szakmai tudással, lehetőséggel, és életre szóló barátságokkal látott el; mások ambíciói nagyban befolyásolták és formálták saját céljaimat és törekvéseimet is. Az idősebb Collégák külföldi tanulmányainak pozitív beszámolói bennem is elvetették a határon túli tapasztalatszerzés akaratának magját. Így jutottam arra az elhatározásra, hogy az MSc diplomámat Dániában, a DTU-n akarom megszerezni.

Az ehhez vezető úton számtalanszor kaptam segítséget Collégáktól – mind az egyetemi jelentkezési folyamat során, mind a kiköltözés alatt sokan láttak el rendkívül hasznos személyes tapasztalatokkal, melynek köszönhetően roppant zökkenőmentesen sikerült minden. Nagy örömömre szolgált, hogy ezt én is viszonozni tudtam egy évvel később egy fiatalabb Collégának az eredményes felvételi és külföldi pályakezdés érdekében.

Tanulmányi szempontból meglepően gyakorlatorientáltnak éreztem az oktatást Dániában, ami a megannyi csoportos projektmunkán alapuló tárgyban is megnyilvánult. Egy ilyen tárgy keretein belül jómagam is megtapasztalhattam a gyors fejlesztéssel, liftbeszédekkel és networking rendezvényekkel telített startupok világát, amit mellesleg kifejezetten támogatnak és promótálnak Dániában. Ebben a startupban egy olyan telefonos applikáció fejlesztésében vettem részt, mely különböző terápiás eszközökkel segít a social media függőség és túlhasználat legyőzésében.

A szürke mindennapok szemszögéből Koppenhága igen hamar megfogott. A kerékpáros kultúra és infrastruktúra messze felülmúlta elvárásaimat, és bár gyakoriak az esős és szeles napok, a dánokat sokszor ez sem tántorítja el attól, hogy biciklire üljenek. Az ország erőteljesen a bizalomra épül, mely megdöbbentően jól működik és egyben szorosan igazodik a saját nézeteimhez is. Másrészről viszont köztudottan drága országnak minősül, így nem meglepő módon érkezésem után hamar munkakeresésbe kezdtem.

Pár hónap keresgélés és egy sok lépcsős interjú folyamat után a Microsoft koppenhágai irodájában találtam magam gyakornoki szoftverfejlesztő pozícióban. Több Dynamics365 termék fejlesztése mellett a kvantum kutatások egy része is itt zajlik, melynek köszönhetően a tejüveg mögé rejtett kvantum hardver gyakran talál kíváncsiskodó szemekre az előtérben járók közül. A pozitív élményekkel és megannyi új tapasztalattal záruló nyári gyakorlat után nagy örömömre egy részmunkaidős diákállást is megajánlottak, így korábbi csapatomban folytathattam DevOps fejlesztői tevékenykedésem az egyetem végéig.

A diplomamunka-témámra is a Microsoft falain belül találtam rá (habár egy másik termékhez kapcsolódóan). A projekt során a flexible job-shop scheduling probléma osztott rendszereken alapuló, ipari alkalmazását vizsgáltam, annak érdekében, hogy a gyártási folyamatütemezésünket még tovább optimalizálhassuk. Dolgozatom alapvetően is stresszes utolsó pár hónapját egy interjú folyamat is tovább nehezítette, mivel a diák pozícióm ellenére a teljes álláshoz (külsősökhöz hasonlóan) nekem is át kellett esnem több technikai interjún. Így a 6 hónapig tartó munkát nem csak egy sikeres védéssel és 12-es jeggyel zárhattam, hanem egy teljes munkaidős állásajánlattal is a kezemben. Ennek köszönhetően 2023 novemberétől a domain specifikus AL programozási nyelv és hozzá tartozó eszközök fejlesztésével folytathattam – az immáron mesterdiplomás – karrierem.

Kruppai Gábor

EJC: 2016–2020

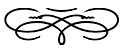
2016-ban kezdtem az egyetemet programtervező informatikus szakon, az akkor először induló Neumann tehetséggondozó csoportban. A Collegiumba másodévesként jelentkeztem és lettem bentlakó tagja a mesterképzésem végéig. Az itt töltött 4 évnek sok szakmailag hasznos lehetőséget, jó emléket és barátságot köszönhetek, melyek közül sok a mai napig megmaradt.

A mesterszak vége felé sokat gondolkodtam a PhD irányon, de végül az „ipar" mellett döntöttem, mondván, hogy az egyetemre később bármikor vissza lehet menni – azóta eltelt 4 év, de még nem tettem.

Mesterképzés óta kutató-fejlesztőként dolgozom pénzügyi területen, ami általában negatív megítélést kap az informatikai megoldásai terén, de szerencsém volt olyan helyre és csapatba kerülni, ahol támogatják a belső kutatásokat is, így néha csak a publikálás és a közeg mássága üt el az egyetemtől, de a tanulás és az irányok szabadsága szinte ugyanaz. A témák középpontjában főként a gépi tanulás áll. Munkám során sokat foglalkoztam biometrikus azonosítással, természetes nyelvfeldolgozással (NLP), saját nagy nyelvi modellekkel (LLMs) és csalásdetektálással. Legújabban a beágyazás/vektorizálás témakörben kutatok, de az utóbbi időben egyre hangsúlyosabb számomra a modelljeink üzemeltetését érintő szervezés (MLOps) is.

A szakmai dolgok mellett viszonylag nagy részt foglal el az életemben az utazás, és már munkám során is egyre inkább törekszem arra, hogy tartózkodási helytől független tudjak működni. Ezek a törekvések még a collegiumi időkből származtathatók, amikor is egy szobatársammal a külföldi tanulmányi programokon túl, minden tanítási szünetet igyekeztünk – egyetemista mércével is megfizethető – külföldi kirándulásokkal tölteni. A szabad időszakok azóta ugyan változtak, de a lendület mindkettőnkben megmaradt. Vagy közösen, vagy külön, de ezalatt a pár év alatt több mint 30 országban sikerült legalább egy hetet tölteni, ami 2023-ban nagyjából 4 hónapot tett ki. A jövőben szeretnék

még több nemzetközi élménnyel gazdagodni, viszont a szakmai munka továbbra is fontos számomra, így egy kicsit talán lassabban, de remélhetőleg biztosabban is haladok a földrajzi függetlenség felé.



Leitereg András

EJC: 2016–2018

andras.leitereg@gmail.com

Leitereg András vagyok, 29 éves, szoftverfejlesztő. A „pályafutásom" már jóval az egyetem előtt elkezdődött, felső tagozatosként és középiskolásként rendszeres résztvevője voltam a Nemes Tihamér versenyeknek, majd az infó OKTV-knek. Matekból is versenyeztem egy darabig, de valami megfogott a feladatmegoldásnak ebben az egyedi formájában, amit algoritmizálásnak szokás nevezni. Annyira beszippantott ez a világ, hogy 12. osztályosként még az IOI-ra, a Nemzetközi Informatikai Diákolimpiára is sikerült kijutnom a magyar csapat tagjaként.

Ma se tudom pontosan megmondani, hogy miért az ELTE-re mentem és nem a BME-re. De nem bántam meg, már elsőévesként belevetettem magam egy kutatócsoport munkájába Lőrincz András vezetése alatt. Emellett szerettem volna visszaadni valamit az utánam jövő generációknak abból a támogatásból, amit én versenyfelkészülés közben kaptam, így elkezdtem infó tehetséggondozással foglalkozni. A Fazekas Gimnáziumban tartottam robotika, majd programozás szakköröket, és besegítettem a Nemes és OKTV feladatsorok kidolgozásába. Zsakó tanár úr már „régi ismerősként" üdvözölt a csapatban. És azért a versenyzést sem hagytam abba teljesen. Különböző csapatfelállásokban egészen elfogadható eredményeket értünk el az ACM-en, és „céges" (NNG, Graphisoft, Loxon, stb. által szervezett) versenyeken.

Az egész BSc-t ilyen állandó pörgésben csináltam végig, egymást érték a projektek, versenyek. Akkor nagyon élveztem, ma már nem biztos. Lényegében (egy tárgy híján) 5 félév alatt letudtam a BSc-t, hogy a 6. félében még elmenjek egy Erasmusra Helsinkibe.

Az MSc-t pedig már az Eötvös Collegium bentlakójaként kezdtem. Ebben nem elhanyagolható szerepe volt annak is, hogy megismerkedtem Fonyó Vikivel, az infóműhely akkori műhelytitkárával, aki azóta

már a feleségem. Addig Szentendréről jártam be, úgyhogy a Coli a műhelyórákon és a baráti-szakmai közösségen kívül a budapesti pezsgést is adta nekem. Na nem mintha törzsvendégei lettünk volna az Estikének.

Az MSc a BSc-hez képest sokkal lazábban telt, elkezdtem komolyabban elmélyülni a „masszívan párhuzamos architektúrák" (GPU-k) programozásában. Az erről írt TDK-val úsztam meg a diplomamunkaírást. Még egy PhD-ba is belekezdtem ebben a témában Kozsik Tamás (akkori műhelyvezetőnk) irányítása mellett, de ez később érdeklődés (az enyém, nem az övé) hiányában félbemaradt.

Időközben Vikivel elhagytuk a Colit és két belvárosi albérlet után Kaszásdűlőn vettünk magunknak egy panellakást. Bővült a családunk Jollyval, a Cavalier King Charles spániellel. Innen 3 nyugalmas, coviddal és home office-szal telt év után még kijjebb költöztünk, ugyanis felépült álmaink családi háza Ürömön. '23 júliusa óta itt próbáljuk feldolgozni az építkezés viszontagságait, és kikászálódni a lakberendezés végtelennek tűnő labirintusából.

Az egyetem óta végig egyéni vállalkozói státuszban, otthonról dolgozom. Először a Vacuumlabs-on keresztül vettem részt külföldi, főleg fintech és crypto projektekben. Majd egyetemi (és collegiumi) ismerősöm, Nagy Vendel invitálására elkezdtem az Eventus nevű folyamatmenedzsment szoftveren dolgozni, immár 2.5 éve töretlen lelkesedéssel. Csapatépítésként pedig rendszeresen járunk falat mászni.

Leskó Dániel

EJC: 2005–2014

ldani@elte.hu

2005-ben kezdtem tanulmányaimat az ELTE-n, az éppen 2 éves Informatika Karon – a kar 3. induló évfolyamán –, illetve a még fiatalabb Informatikai Műhelyben, a Ménesi úton. Mondhatni mindkét intézménnyel együtt nőttem fel a feladathoz, az egyetemista léthez.

Szakmai szempontból az első jelentős lépés 2008-ban történt, amikor csatlakoztam egy egyetemi kutatócsoporthoz, mely egyrészt érdekes szakmai feladatokkal, másrészt plusz ösztöndíj lehetőséggel motivált – talán ezt a pontot lehetne kiemelni, ami elindított a későbbi doktori iskola, illetve oktatási tevékenységek irányába.

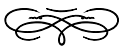
Utólag visszatekintve 2008 más szempontból is meghatározó volt, ugyanis ekkor ismerkedtem meg a felségemmel, illetve indítottam el egy online hobbiprojektet, mely aztán néhány év alatt komoly vállalkozássá nőtte ki magát.

A mintatanterv szerint haladva 2008-ban BSc, 2010 MSc fokozatot szereztem, 2013-ban pedig az Informatika Doktori Iskolában abszolváltam. Ezen évek alatt, és később is folyamatosan részt vettem ELTE-s kutatócsoportok munkájában, jellemzően nyelvtervezés, fordítóprogram készítés, statikus elemzés, véletlenszerű program, illetve tesztadat generálás témakörökben.

2012-ben, az akkor frissen induló ELTE EIT Digital Doctoral Schoolba is felvételt nyertem, melynek célkitűzése az volt, hogy a doktori hallgatók képességeit és attitűdjét üzleti, üzletfejlesztési, marketing és startup ismeretekkel tágítsák, képessé téve őket a saját tudományos eredményeik üzleti környezetben történő értelmezésére, hasznosítására. Ezeken a képzéseken nagyon sokat profitáltam, ekkor lépett komolyabb szintet a korábbi hobbi online vállalkozásom (egy szálláskereső oldal, a SzállásKérés.hu), immár teljes értékű cégként működve tovább.

A doktori iskolából abszolválás után tanársegédként maradtam, illetve vagyok jelenleg is az ELTE Informatikai Kar kötelekében. Ez a munkahely az évek során egyre inkább a hobbimmá vált, helyet cserélve a hobbiként induló, de folyamatosan fejlődő és komolyabb eredményeket elérő online vállalkozásommal.

Személyes fronton sem volt eseménytelen az elmúlt 10 év, egy esküvő, 3 gyerek, 2 költözés és egy építkezés kötötte le a szabad kapacitásaimat (vagy épp csökkentette az alvásidőmet).



Lócsi Levente

EJC: 2003–2011

locsi@inf.elte.hu

Tulajdonképpen engem még a Matematika–Fizika Műhelybe vettek fel 2003-ban, amikor Vas megyéből Budapestre kerültem, az ELTE frissen indult Informatikai Karára, programtervező matematikus szakra, és jelentkeztem az Eötvös József Collegiumba. Viszont így elmondhatom, hogy magam is ott voltam az Informatikai Műhely alapító műhelygyűlésén, 2004. februárjában, és tavasszal már Csörnyei Zoltán tanár úr lelkes előadásait hallgathattam a λ-kalkulus rejtelmeiről. Nem feledem humorát, és gondoskodó, motiváló figyelmét, mely végigkísérte egyetemi tanulmányaimat.

A Műhelynek és a Collegiumnak rengeteg szakmai és személyes élményt köszönhetek. Felemelő volt olyan környezetben lenni, ahol választott hivatását minden ott lakó (és bejáró) szívvel-lélekkel, igényesen, magas színvonalon igyekszik művelni, miközben a társasági élet is pezsgő, üdítő, változatos személyiségekkel teli, valóban kicsit mint egy „nagy család". Megtiszteltetésnek tartom, hogy Csörnyei tanár úr és Kozsik Tamás dékán úr után, Horváth László igazgató úr felkérésére az Informatikai Műhely vezetője lehetek 2022 óta. Remélem, méltónak bizonyulok majd e feladatra, és vissza tudok majd valamit adni mindabból, amit magam is kaptam útravalóul. A stílusban éles váltásra semmiképp sem gondoltam: talán az átmenet nemcsak folytonosra, hanem differenciálhatóra is sikeredett.

Célom volt, hogy öt év alatt elvégezzem a szakot – ez akkoriban még kevésbé volt gyakori –, amit el is értem: 2008-ban vehettem át programtervező matematikus kitüntetéses oklevelemet (tényleg vörös), diplomamunkámat Schipp Ferenc tanár úr témavezetésével írtam „Ortogonális rendszerek szerinti gyors Fourier-transzformációk és alkalmazásaik" címmel. Kicsit korábbi TDK dolgozatom témája pedig „Komp-

lex függvények színes ábrázolása" volt, amiről azóta is szívesen adok elő pl. tehetséggondozó táborokban. Egyébként a XXVIII. OTDK Szakmai Bizottságának Különdíját nyerte el. Doktoranduszként szintén Schipp tanár úr volt a témavezetőm. Tanulmányaim alatt, 2011 tavaszán, egy félévet az Universität Wien NuHAG kutatócsoportjában (Numerical Harmonic Analysis Group) tölthettem Erasmus ösztöndíj keretében. Disszertációm „Racionális függvényrendszerek alkalmazása a jelfeldolgozásban" címmel készült el, és 2015 februárjában védtem meg.

Itt is maradtam a Numerikus Analízis Tanszéken, miután Simon Péter tanszékvezető úr felkínálta a lehetőséget, s azóta rendületlenül oktatom a – főleg programtervező informatikus – hallgatókat Analízis, Numerikus módszerek, Matlab programozás (és rokon) tárgyakból.

Több (legyen három) projektet is megemlítenék azonban, melyekben az oktatói, kutatói munka mellett az utóbbi években szerepem lehetett. (1) Közel 5 évig voltam a Kar angol nyelvű BSc-képzésének koordinátora. Ebben az időben kezdett eléggé felfutni ez a szak, 2–3 helyett a végén már 20–30 külföldi hallgatót vettünk fel egy évben. (Ma 200–300.) „Nem kell külföldre mennem, a külföld jön el hozzám" – szoktam mondogatni abban az időben. (Főleg déli és keleti irányból jött.) A feladatkört akkor adtam át, amikor GYED-re mentem. (2) A „Tehetséggondozás és kutatói utánpótlás fejlesztése autonóm járműirányítási technológiák területén" című EFOP projekt 2017–2022 között futott. Itt a matematikai versenyfeladatos órák mellett szakmai vezető asszisztens lettem Fridli Sándor tanár úr, majd Gede Mátyás mellett. A Kar jó részének publikációs és konferencialátogatási tevékenységének figyelemmel kísérése mellett főleg a hallgatói ösztöndíjszerződésekkel foglalkoztam, gondoskodhattam több mint 200 millió Ft ösztöndíj szétosztásáról tehetséges hallgatók (kb. 200 fő) részére, illetve ennek megfelelő adminisztrálásáról. (3) 2021-ben zárult az ELTE Multimédiás Tananyagfejlesztési Pályázata, amelynek keretében elkészíthettem egy félévnyi Numerikus módszerek előadás szépen tagolt, weben is elérhető videós feldolgozását, mely segédanyag-gyűjtemény a „Magyarázatok Numerikus Módszerekhez" címet kapta.[1]

Ugyancsak a Collegiumban, a kórusban ismerhettem meg feleségemet, Zsófit is; 2012-ben házasodtunk össze. Budapest VII. kerületében, trolibuszban gazdag vidéken élünk (utcánkon bár 4–5 járat is áthalad,

[1] `https://locsi.web.elte.hu/mnm/`

előttünk pont egyik sem). Nagyobbik kisfiunk, Norbert immár második osztályos, a kisebbik, Bálint pedig alig pár hónapos. (Társasházunk sivatagában kis lakásunk egy valódi oáázis lett.) Van továbbá egy „kulturált városi kutya” okleveles kedvtelésből tartott háziállatunk: Szuszi. Hobbiként szívesen foglalkozok zenével. Van olyan kórusművem, amelynek felvétele elérhető az interneten a hágai magyarok kvintettjének előadásában; valamint alakul kisebb gitáreffektpedál gyűjteményem, és halmozom a – tipikusan pengetős, de nem szokványos (ezáltal némi kihívást is kínáló) – hangszereket, így megtalálható nálunk többek között ukulele, kalimba és hanago is. Annak idején gyakran zengett gitárom hangjától a Collegium folyosója...

S végezetül, némi interdiszciplinaritás és egy csipetnyi tréfa kedvéért álljon még itt e kis költeményem.

Lövei Péter

EJC: 2017–2021

2017 őszén, másodéves BSc-s hallgatóként (amikor úgy döntöttem, B szakirányról A-ra váltok az ELTE IK-n) csatlakoztam az EJC Informatikai Műhelyéhez. Az elsőként még felettébb furcsa, Roxfortra emlékeztető collegiumi órák gyorsan átalakultak a funkcionális programozás elméletével megismerkedő lambda- (és egyéb görög betűk) kalkulus foglalkozásokra. Érdekes volt az egyetemi, gyakorlatiasabb óráin túl a lambda függvényekről és a fixpont kombinátorokról is hallani Csörnyei Zoltán gondolataiból.

Ezeket egészítették ki a nyelvi (angol) és egyéb szociális programok és esték Kozsik Tamás, akkori műhelyvezetővel, és persze a többi műhelytaggal. Emlékszem a kis Neumann János Tehetséggondozó Körös csoportunk lényegében mind műhelytag volt, így az egyetemi matekos gyakorlati órák is meghittebbek voltak, több idő maradt „exkluzívabb" tartalmak megismerésére. Itt szeretném kiemelni többek között, de nem kimerítően Horváth Gyulát, Tóth Melindát, Németh Zsoltot, Csörgő Istvánt, Filipp Zoltánt és Lócsi Leventét, akik ezt lehetővé tették, számunkra nagyon hasznos, barátságos és interaktív órák és foglalkozások megtartásával. Ezen túl Zsolttal és egy-két műhelytaggal közösen több programozási (és Hajós, matek) versenyre is mentünk a 2–3. évek során.

A BSc éveim végét egy Gera Zoltán által vezetett, és Porkoláb Zoltán által támogatott Ericssonos projekttel zártam, amelyben a C++ kódok futás idejű hibáit detektáló Codechecker eszköz optimalizálását tűztük ki célul. Ezt egészítették ki, többek között, Erlangos projektek, például az OTP Banknál ledolgozott szakmai gyakorlat, ahol az azonnali fizetési funkciók támogatásában tudtunk segíteni. Ugyanekkor alapítottunk barátaimmal egy céget, ahol többek között lefejlesztettük az egri mozinak a jegyfoglaló rendszerét.

2019 őszén, az MSc-s éveim kezdetén aztán nagy fordulatot vett életem, ugyanis csatlakoztam az EIT Digital nemzetközi kétdiplomás képzéséhez, amely keretein belül lehetőségem nyílt az adatelemző, Data

Scientist tudományágban részletesebben elmerülni az ELTE-n és a finnországi Aalto Egyetemen. Ennek köszönhetően sikerült megismerkednem a finn kultúrával, számtalan érdekes tudományterülettel (például megerősítéses tanulás, adatbányászat, valós idejű adatfeldolgozó rendszerek, stb.), de a program struktúrája lévén üzleti témájú (pl. Startup Experience) és önfejlesztő (Good Life Engine) kurzusokkal is, amelyek jelentősen kibővítették látószögemet. Az Aalto Egyetemen nagyon jóba lettem az Algorithmic Methods of Data Mining nevezetű kurzus professzorával, Wilhelmiina Hamalainennel, azóta többször is segítettem a kurzus projektfeladatának összeállításában kutatói asszisztensként. A 2021-es mesterdiplomám (Horváth Dániel vezetésével) is ezzel rezonált, hiszen a forgalmi állapotok makroszkopikus szinten történő becslését tűztem ki témaként.

Mindeközben, még 2018 őszén csatlakoztam egy akkoriban még kis létszámú startup céghez, a Commsigniához, ahol azóta is dolgozom, immár több mint 5 éve, eleinte kutató gyakornokként, majd 2 éve, kutató menedzserként. Ebben a pozícióban tudom kamatoztatni az elmúlt évtized során megszerzett ismereteimet, hiszen feladatom többek között az innovatív technológiák és módszerek felfedezése, kiértékelése és alkalmazása a jármű-kommunikáció (V2X) és 5G peremterületein, amelyek elősegíthetik a gyalogosok védelmét, a forgalom optimalizálását, vagy konfliktusok megelőzését. Kiemelendő a 2023 őszén általam vezetett, példátlan detroiti demonstráció, ahol bemutattuk több, nagy ipari partnerrel a technológia interoperabilitását.

Idén pedig a vezetésemmel a Commsignia belépett a BME HIT által szervezett PARIPA ipar-orientáltságú programjába, amelynek keretein belül két tehetséges BSc-s hallgató mentorálását és szakmai vezetését vállaltam el a 2024-es év során.

Luksa Norbert

EJC: 2015–2021

norbert.luksa@gmail.com

Elsőéves collegistaként hamar felfedeztem a Collegium lehetőségeit. Serényen kivettem a részem a szakmai foglalkozásokon túl a különböző Társasági programokon is. Először rendszergazda, majd a Választmány Informatikai Bizottságának tagja lettem. Már a második félévemtől a Közösségi programok szervezésében – alelnökként – tevékenykedtem. Ezek után nem szükséges a sorok között olvasni, hamar eljutottam az Estikébe is, ahol az első lehetőségtől kezdve beálltam a pult mögé.

Ezeken túl voltam még – a teljesség igénye nélkül – választmányi elnök, műhelytitkár, kuratóriumi tag.

Az alapképzés során a funkcionális programozás felé terelődött az érdeklődésem, BSc-s TDK és szakdolgozatom Haskell programok párhuzamosítását tette lehetővé refaktorálással. Mesterszakon a típuselmélet témakörében kezdtem kutatni. Mindkét témám előadtam számos konferencia mellett az OTDK-n is.

Egyetemi éveim során több kurzust is vezettem, az egyetemen Ada, Imperatív programozás, és Típuselmélet tantárgyakban. A Collegiumban Csörnyei tanár úr nyugdíjba vonulása után engem ért a megtiszteltetés, hogy a λ-kalkulus szépségeibe bevezethessem a műhely elsőéves tagjait. Tartottam továbbá Katkó Dominikkal Open-source programozást, illetve a nem informatikus Eötvös collegistáknak LaTeX és programozás órákat.

Második MSc félévem elején elkezdtem dolgozni a Cloudera-nál. A Cloudera a big data teljes életciklusára kínál megoldásokat, a tárolástól kezdve a továbbításon át a feldolgozásig mindent lefed a portfólió. Első két évemben az Apache Impala nyílt forráskódú C++/Java alapú, masszívan párhuzamosítható adatbázis-kezelő rendszert fejlesztettem. Ezt követően egy új projektre kerültem, ahol ezeket a rendszereket a

felhőbe telepítve biztosítjuk terhelés alapú skálázással. Hamarosan elérem az ötéves évfordulót, jelenleg a csapatom technikai vezetőjeként a fenti projekt egy privát felhős megoldásán dolgozunk.

Két éve feleségül vettem Pankát, akivel nyolc éve a Collegiumban ismerkedtünk meg. Az esküvő előtt Győrbe költöztünk, de olykor-olykor még fel-felbukkanunk a Collegiumban nosztalgiázni és barátokkal találkozni, számunkra mindig *megmarad ez a hely* otthonunknak.

Manninger Mátyás

EJC: 2012–2014

matyas.manninger@gmail.com

2012-ben nyertem felvételt a Collegiumba és 2014-ig voltam bentlakó tag. Ez idő alatt, szobatársam, Balassi Márton ösztönzésének köszönhetően egyszerre végeztem az ELTE programtervező informatikus BSc-jét és a BME alkalmazott közgazdász képzését.

A Collegium minden feltételt biztosított számomra, hogy mozgalmasan és hasznosan teljenek ezek az évek. Erre az egyik legjobb példa, hogy a Collegium igazgatóságának támogatásával sikerült megszervezni egy 2 kredites választható tárgyat, mely a magyar kártyajátékok kultúráját oktatta, különös figyelmet fordítva a tarokk nevű játékra. Mindenkit ösztönöznék, hogy legyenek kreatívak és vállalkozó szelleműek. Ez elengedhetetlen a collegiumi háttér legélvezetesebb és legproduktívabb kihasználásához.

Ebben az időszakban ismerkedtem még meg a funkcionális programozással. Ennek a területnek a kimagaslóan elegáns megoldásai a mai napig is lenyűgöznek. Nem csoda, hogy ez volt az a témakör, amiből a TDK dolgozatom és a szakdolgozatom is született.

A diákmunkám alatt tetszett meg az akkor divatos „Big Data” terület, így mester képzésnek az EIT (European Institute of Technology) nemzetközi Data Science programját választottam. Ez remekül ötvözte a vállalkozói tevékenységhez szükséges tudást a mély technikai képzéssel. Első évemet Madridban a UPM-en, a másodikat Stockholmban a KTH-n töltöttem.

Itt találtam az egyetem után állást egy tanácsadó cégnél. Google Cloud technológiákkal foglalkoztunk. Azóta kipróbáltam az egyéni vállalkozást is Data Engineer tanácsadóként. Jelenleg ismét az első cégnél dolgozom az adat csapat vezetőjeként (Head of data pozícióban). Továbbra is Google technológiák segítségével építek „data platformokat”,

illetve az utóbbi évben több generatív mesterséges intelligenciával foglalkozó projektben is részt vettem.

Összességében a Collegium életre szóló barátságokkal, a minőség és precizitás utáni vággyal, és felejthetetlen élményekkel ruházott fel, melyek életem végéig segítenek majd.

Márki-Zay János

EJC: 2022–

Már kiskoromban érdeklődéssel fordultam a matematika és logikai fejtörők felé, amikor 12 éves koromban kiköltöztünk szüleimmel Bambergbe, Németországba. Itt az évek során részt vettem több bajor és német matematikaversenyen, amiken eredményesen szerepeltem, és elnyertem a lehetőséget a *Jugend Trainiert Mathematik* német tehetséggondozó programban való részvételre. Ennek keretében, és versenysikereimnek köszönhetően számos matekszemináriumon vettem részt, ahol tovább bővíthettem tudásomat.

Középiskola végére az informatika is egyre jobban érdekelt, és főleg honvágyból hazatértem Magyarországra, ahol megkezdtem tanulmányaimat az ELTE Informatikai Karának programtervező informatikus szakán. A matematika iránti szeretetem is megmaradt, ezért természetes volt, hogy a matematikai összefüggésekkel mélyebben foglalkozó modellező szakirányt válasszam. A Collegiumban és az egyetemi Neumann-körben olyan társaságok részévé váltam, amik motiválnak, hogy a tanulmányaim során és szakmailag is minél kiemelkedőbb eredményeket érjek el.

Szabadidőmben szívesen foglalkozom szabadulószobák tervezésével és készítésével, ahol az informatikai tudásomat is kamatoztatom. Először a kollégium által szervezett *Tehetséggondozó Táborban* rendeztünk be egy szabadulószobát a résztvevők szórakoztatása végett, majd annak sikerén fellelkesülve hasonló programok szakkollégisták számára is készültek. Hogy még színesebb legyen a feladatok palettája, ezekhez online rejtvényeket is programoztam.

Ezen kívül versenyszerűen bridzsezek, és az elmúlt években lehetőségem volt képviselni Magyarországot az ifjúsági válogatott részeként egy-egy U21-es Európa- és világbajnokságon. Jelenleg az U26-os csapattal készülök a 2024 nyarára szervezett *Ifjúsági Európa Bajnokságra*. Ez a hobbi többek között a koncentráció, stresszkezelő és együttműködő képességeimet is fejleszti.

Nádor István

EJC: 2010–2015

istvan.andras.nador@gmail.com

2015-ben végeztem az ELTE Informatikai Kar Modellalkotó informatikus, és egyben az egyetem kapcsolatán keresztül a VU Amsterdam Alkalmazott mesterséges intelligencia szakirányán. Diplomamunkámat a VU Amsterdam Idegtudományi Intézetében töltött szakmai gyakorlatom alatt, egy neurális modellen való munkámból írtam.

Az egyetem alatt, az Eötvös Collégiumban, az Informatikai Műhelyben töltött idő az egyik legmeghatározóbb, legkedvesebb időszaka az életemnek. Az Informatikai Műhely segítő, inspiráló közege és szakmai szellemisége megalapozta szakmai szemléletemet.

Egyetem után a BME-sek által alapított Tresorit startupban gyarapítottam a kevés ELTE-s programozó táborát. A Tresorit végponttól végpontig titkosított, felhőalapú fájlmegosztó szolgáltatásának fejlesztésében vettem részt, amit roppantul élveztem, mert egy hatékony, több platformon is működő, biztonságos alkalmazást fejlesztettünk, amihez érteni kellett a kriptográfiai primitívek sajátosságait, és valamilyen mélységig a mögöttük rejlő matematikát is.

A Tresoritnál töltött három év után hirtelen felindulásból Berlinbe költöztem, és egy, az Uberhez hasonló szolgáltatást nyújtó startupnál dolgoztam két évet. Innen egy váratlan ajánlattól vezérelve átköltöztem Münchenbe, és a Google játék streamelő szolgáltatásán, a Stadianál kezdtem el dolgozni. A Stadia szintén egy nagyon izgalmas, szerteágazó projekt volt. Számtalan technológiát érintett: modern hang és videó kódolásokat és tömörítéseket, videókártya drivereket, szerverparkokat, alacsony késleltetésű hálózatokat, beágyazott rendszereket, a legkülönfélébb architektúrán alapuló klienseket, webböngészőket és bugos routereket.

A Stadia sajnálatos leállítása után az Android Autóhoz kerültem, ahol most leginkább az operációs rendszerek belső élete és a Rust programozási nyelv foglalkoztat. Feleségemmel visszaköltöztünk Berlinbe, annak egy kanálisokkal körbefutott természetvédelmi övezetébe. A helyi hódokkal szeretnénk nagyon találkozni, de eddig csak az azokra megtévesztésig hasonló dél-amerikai nutriákhoz volt szerencsénk.

Novák Ádám

EJC: 2001–2007

novadam1@gmail.com

Az Informatikai Műhely „alapító” tagjainak egyikeként szép emlékeket őrzök a Collegiumban és a Műhelyben töltött éveimről. Különösen jóleső érzés visszaidézni, amikor a fiatal Műhelyben a műhelyórák szinte családi légkörben zajlottak, és a Műhely minden egyes tagjának saját beosztás jutott. Nemigen fogom feledni, ahogyan Csörnyei Zoltán tanár úr, első műhelyvezetőnk igazi átéléssel igyekezett megismertetni bennünket a λ-kalkulus csodáival és a fordítóprogramok tervezésének fortélyaival. Szintén örökre emlékembe vésődtek a remek kirándulások képei a kis létszámú műhely barátságos és összetartó tagjaival.

Érdeklődéssel követem azóta is a Collegium és a Műhely fejlődését, szívből gratulálok az évről évre bővülő kiváló eredményekhez, és egykori műhelyalapító-társam, Lócsi Levente műhelyvezetői munkásságához. Örömmel fogadtam el Levente felkérését e rövid névjegy megírására, amelyben eddigi életutamat alább vázolom. Ezúton is kívánok további nagyszerű sikereket a Műhelynek!

Tanulmányaimat az ELTE programtervező matematikus szakán 2001-ben kezdtem, amelyet 2003-tól kezdve biológus képzéssel párosítottam. Az Eötvös Collegium bentlakó tagjaként (2001–2007) először a Matematika–Fizika Műhelyben tevékenykedtem, majd 2004-ben részt vettem az Informatikai Műhely megalapításában. 2001 és 2003 között a Collegium Diákbizottságának is aktív tagja voltam, az Eötvös Konferencia szervezője, és az Estike csaposa. Továbbá 2003 és 2007 között az Eötvös–Bolyai Pingpongversenyek főszervezőjeként és a Collegium futócsapatának oszlopos tagjaként a collegiumi sportélethez is igyekeztem hozzájárulni. Diplomáimat 2007-ben (programtervező matematikus) és 2010-ben szereztem (biológus) – e két tudományterület ötvözését a következő években az iparban és kutatásban is hasznosítottam.

2008 és 2009 között a DSS Consulting munkatársaként és vezető szoftverfejlesztőjeként dolgoztam. Gyermekek sportválasztásában segítséget nyújtó, antropometriai méréseken és gépi tanuláson alapuló modell és kiértékelő szoftver megalkotását irányítottam.

2009 és 2013 között az Oxfordi Egyetem Statisztika Tanszékén helyezkedtem el Jotun Hein csoportjában, ahol a kutatási területem a bioinformatika, azon belül elsősorban genomika, statisztikus szekvenciaillesztési algoritmusok és a génannotáció volt. E területeken számos nemzetközi folyóiratban megjelent publikációm született. 2009–2012 között a Statisztika Tanszék MSc/PhD Bioinformatika és Statisztika kurzusainak oktatójaként, és az oxfordi „Summer School in Computational Biology" nevű nyári iskolájának oktatójaként vettem részt az egyetemi életben. A részben Oxfordban végzett kutatásaim eredményeire támaszkodva az ELTE Informatikai Karán 2015-ben szereztem PhD fokozatot.

2013-ban Svájc francia nyelvterületén, Lausanne közelében a SOPHiA GENETICS nevű startup cégnél helyezkedtem el bioinformatikusként, ahol ma szoftverfejlesztő csapatok munkáját vezetem. Az általunk fejlesztett SOPHiA DDM nevű felhőalapú SaaS platform precíziós genetikai vizsgálatokhoz és személyre szabott terápiák tervezéséhez ad nélkülözhetetlen segítséget kórházak számára onkológia és ritka genetikai betegségek területén. Emellett a platform gyógyszergyártó cégeknek is nyújt értékes szolgáltatásokat az új gyógyszerhatóanyagok klinikai teszteléséhez szükséges, jól definiált genetikai profillal rendelkező betegcsoportok azonosításában, és a kezelések hatékonyságának felmérésében. A SOPHiA GENETICS ma több mint 750 kórházzal és egészségügyi intézménnyel dolgozik együtt, és 2021-es nyilvános részvénykibocsátása óta részvényeit az amerikai NASDAQ tőzsdén jegyzik.

Végül néhány szó a családomról is: chilei–olasz származású feleségemet Oxfordban ismertem meg, és 3 közös fiunk született – Dániel (2014), Márk (2016) és Alex (2019) –, akiket igyekszünk korán megismertetni az informatika és bioinformatika szépségeivel!

Ölvedi Tibor

EJC: 2008–2012

teebeey@gmail.com

2008-ban kezdtem az ELTE Informatikai Karán Programtervező informatikus szakon, és ezzel egy időben az Eötvös Collegium bentlakó tagjaként az Informatikai Műhely munkájában is.

Collegiumi éveim alatt korán, már elsőévesként bekapcsolódtam a rendszergazdák munkájának támogatásába, másodévesként már rendszergazdaként tevékenykedtem tovább. Ez idő alatt több projektet is megvalósítottunk. Az alagsori BNC hálózatot lecseréltük Ethernet alapúra, ezzel javítva a hálózat sebességét és megbízhatóságát, emellett az Estike is rákapcsolódott a hálózatra, ennek köszönhetően pedig kiszélesedett a zenei választék. Az Érintésvédelmi Osztály vizsgálata alapján a pizzásdoboz-technológiás rack szekrények elbontására köteleztek minket, ami miatt teljesen megújult a lakószintek hálózata is, a folyosókra kerültek ekkor 24 portos switch-ek, idáig vezetett a gigabites gerinchálózat. Szobánként beépítésre került egy-egy ötportos switch, ezzel szobánként 4 vezetékes végpontot lehetett csatlakoztatni egységesen.

A szerverteremben is történtek változások, a társalgó létrehozásával a szerverterem leköltözött a TMK műhely hátsó szobájába, ami a gerinchálózat átalakítását is szükségessé tette. Béla (fájl- és nyomtatómegosztás) mellett beüzemeltük Bélánét (levelezés, weboldalak), illetve a tűzfal/proxy szervert is új hardverrel váltottuk ki pályázati forrásból. Szoftverek területén bevezetésre került az EIR (Egységes Információs Rendszer), ahol a collegisták a nyomtatószámlájukat tudták kezelni, internetet vásárolni, eseményeket meghirdetni, csevegni, vagy éppen a vizuális szobabeosztást megtekinteni. Megújult a levelezés, és Exchange alapokra került, a Diákbizottság kérésére egy kérdőív-szolgáltatás került bevezetésre, illetve a Collegium új hivatalos weboldalt is kapott egy külső beszállítótól.

Collegiumi éveim alatt két témát mutattam be Eötvös Konferenciák alkalmával. Foglalkoztam a vezeték nélküli hálózatok fejlődésével és az ezek használata során felmerülő biztonsági kockázatokkal, illetve a különböző védelmi, titkosítási funkciókkal. Egy másik alkalommal a monitoring rendszerek gyógyító képességeit vizsgáltam meg, ahol a monitorozó rendszer tisztában van a topológiával, függőségekkel, és ez alapján a hibásan működő rendszeren végre tud hajtani bizonyos hibaelhárítási (takarítás, újraindítás, skálázás stb.) feladatokat önállóan, ezzel redukálva a 24/7-es szolgáltatások üzemeltetési költségeit.

ELTE-s tanulmányaim zárásaként egy céges együttműködés keretében implementáltam egy webes jelszónyilvántartó/megosztó rendszert, aminek az újdonsága az aszimmetrikus kulcsú titkosítás (ideális esetben smartcard alapon), azaz a jelszavak titkosítva voltak tárolva, de csak a kliensnél levő privát kulcs segítségével visszafejthetők. A mögöttes PKI emellett lehetővé tette csoportos jelszavak kezelését (ami minden csoporttag számára önállóan volt titkosítva) és a jelszóküldést (titkosítás a címzett nyilvános kulcsa által).

Ezek után elvégeztem egy kétnyelvű mérnök informatikus mesterképzést is, ahol a robotika és az orvosi informatika volt a két meghatározó tudományterület. Diplomamunkám keretében egy OSLC (Open Services for Lifecycle Collaboration) illesztést implementáltam SVN (Subversion) alapú verziókövetéshez, ahol commit üzenetek segítségével lehet egy OSLC-kompatibilis Bugzilla rendszerben a hibákat kezelni (megjegyzés, lezárás stb.).

2011-ben kezdtem meg az első főállású munkámat egy német cég magyar leányvállalatánál, ahol nagy nemzetközi cégek számára biztosítottunk informatikai szolgáltatásokat. Middleware (alkalmazásszerverek, webszerverek, aszimmetrikus üzenetküldő rendszerek stb.) szerverrendszerek tervezésével, felépítésével, fejlesztésével és üzemeltetésével kezdtem foglalkozni, majd gyorsan ezek automatizálása került a fókuszomba (Szkriptek különböző nyelveken, Ansible, Puppet, Terraform, Cloud Init stb.). 2015-ben megbízást kaptam egy middleware termék Docker-container alapú implementációjára, majd rögtön ezt követte az ezt futtató platform fejlesztése. Azóta fő profilommá a container technológia (Kubernetes, CRI-O, Containerd, OpenShift, Rancher, ArgoCD, Tekton, CloudFoundry stb.) vált.

Jelenleg Svájcban élek és a svájci államigazgatás containerfelhőjének vezető mérnöke vagyok. Így olyan fontos alkalmazásokkal

foglalkozom, mint a pandémiás kontakt-követés, oltási igazolványok, oltási időpontok foglalása, úti okmányok ellenőrzése, vámos teherforgalom nyilvántartása, adónyilvántartó rendszerek, elektronikus útdíjak, az elektronikus személyazonosság, és sok más fontos államigazgatási rendszer. Az aktuális munkámból egy előadással készülök a 2024-es Red Hat Summitra, a témám (amennyiben beválogatnak) OpenShift alapú nagyvállalati biztonságos CI/CD/GitOps megvalósítása.

Mindig is érdekelt a szoftver mellett a hardveres oldal, így hobbi szinten foglalkozom saját használatú eszközök tervezésével és építésével a kapcsolási rajztól a kész, összeforrasztott áramkörön át a rajta futó vezérlőprogramig, illetve a vezérlőprogrammal esetlegesen kommunikáló szerveroldali programig. Fejlesztettem saját lakásautomatizációs rendszert, amin keresztül világításokat, redőnyöket, termosztátokat tudok vezérelni, illetve saját hifi rendszeren is dolgozom. Emellett hobbiszinten elkezdtem foglalkozni 360 fokos fotográfiával és videográfiával.

Parragi Zsolt

EJC: 2006–2010

parragizs@gmail.com

2006-ban kezdtem tanulmányaimat az ELTE Informatika Karán, az akkor még csak második alkalommal indított, BSc-MSc osztású programtervező informatikus szakán, illetve az Informatika Műhely tagjaként.

Tanulmányaim mellett igyekeztem több extra feladatot is vállalni: 2007 és 2009 között a Collegium rendszergazdája voltam, valamint részt vettem különböző egyetemi projektekben, mint a HypereiDoc nevű, kooperatív szerkesztést segíteni szándékozó keretrendszer, vagy a C/C++ projektek statikus elemzését segítő CodeChecker.

A programozási versenyeket is igen kedveltem, ezeken általában Cséri Tamással és Sztupák Szilárd Zsolttal alkottunk csapatot. Részt vettünk híresebb nemzetközi versenyeken is, mint az ACM vagy a Challenge24, de indultunk rengeteg hazai cég által szervezett kisebben is, amiken rendszeresen értünk el dobogós helyet.

Mindezek mellett dolgozni is kezdtem a Nemzeti AudioVizuális Archívumban szoftverfejlesztőként, a sok egyéb következtében pedig tanulmányaim kevésbé maradtak fókuszban, így azokat a tipikusnál lassabban fejeztem be: MSc diplomámat 2015-ben szereztem meg, PhD tanulmányaimat pedig bár elkezdtem, és az abszolutóriumot is megszereztem, teljesen azóta se fejeztem be.

Munkám során az első időszakban főleg webfejlesztésre fókuszáltam: először a NAVA-nál, majd később a Famillio biztosítási alkusznál is ez volt a fő feladatom. Azonban idővel itt is a C/C++ fele terelődtem, először az NNG-nél töltöttem pár évet az iGO navigációs szoftver fejlesztésével, majd adatbázisrendszerek fejlesztésére tértem át, már több mint 6 éve a Percona csapatában dolgozom a MySQL, illetve PostgreSQL relációs adatbázisok jobbá tételén, elsődlegesen az (adat)biztonságra fókuszálva: titkosítás, authentikáció, authorizáció, ...

Poór Márk

EJC: 2008–2014

mp2264@cornell.edu

2008 szeptemberében kezdtem programtervező informatikus tanulmányaimat, egyben ugyanettől az időponttól Eötvös collégista pályafutásom is datálódik. Az élet azonban úgy hozta, hogy az akkori „A" szakirány kínálta matek se volt elég nekem és 3 év elteltével az informatika MSc helyett elméleti matematikát kezdtem tanulni a TTK-n.

Az akadémiai közeget viszont azóta se hagytam el, elvégeztem a matematika MSc-t is, viszonylag hamar kiderült, hogy a precízen megfogalmazott problémák, melyek némely esetben nélkülöznek minden való életbeli relevanciát valóban sokkal inkább testhez állnak (ezzel példányosítva a matematika és informatika közötti transzferek rendhagyó, de legalábbis ritkábbik irányát).

Ezután Elekes Márton témavezetésével PhD fokozatot szereztem, a téma valós analízis és halmazelmélet illetve logika határterületei voltak (egyik fő eredmény a disszertációmból, hogy lokálisan kompakt topologikus csoportoknak – ilyenek pl. a valós számok, vagy a véges dimenziós mátrixcsoportok – mindig van nullmértékű, de nem első kategóriájú részcsoportja, azonban ennek fordítottja eldönthetetlen [1, 2]).

Az azóta eltelt időben 3 évet töltöttem Jeruzsálemben a Héber Egyetem és az izraeli akadémia finanszírozásával, ahol Saharon Shelah irányítása alatt dolgoztam. Ez idő alatt kezdtem el érdeklődni kombinatorikai és algebrai alkalmazások iránt [3, 4]. Egyik legutolsó eredményem egy Assaf Rinottal (Bar-Ilan Egyetem, Ramat Gan) közös tétel, mely szerint létezik olyan nem megszámlálható G csoport és egy megfelelően nagy N, hogy G bármely nem megszámlálható H részhalmazára $H^N (= \{h_0 \cdot h_1 \cdot \ldots \cdot h_{N-1} : \ h_0, h_1, \ldots, h_{N-1} \in H\}) = G$ [5]. (A probléma eredetéhez és a kontextushoz érdemes megjegyezni, hogy Stevo Todorcevic alábbi anti-Ramsey típusú tétele a múlt század egyik nagy

áttörése végtelen kombinatorikában, egy sok évtizedes problémát oldott meg: létezik nem megszámlálható $G = (V, E)$ gráf, és az élein egy nem megszámlálható sok színt használó c színezés, hogy bármely nem megszámlálható sok $S \subseteq V$ csúcsot kiválasztva c az S által meghatározott $(S, E \cap (S \times S))$ részgráf élein is *minden egyes* színt felvesz.)

Szerettem volna az előzőektől különböző közegben dolgozni még mielőtt letelepedek valahol, így kerültem Ithacába, a Cornell Egyetem matematika intézetébe a H.C. Wang Assistant Professor ösztöndíjjal (3 éves). Ennyi idő elteltével, elméleti háttérrel kisebb kihívásnak éreztem, hogy az első félévben alkalmazott logikát fogok tanítani többnyire informatikus hallgatóknak. Végül Anil Nerode, akitől a tárgyat örököltem, frissítette fel az ezeréves emlékeimet helyességbizonyításról és λ-kalkulusról.

Ezzel el is értünk napjainkhoz, remélhetően az ezután következő állomás egy állandó állás lesz majd.

Hivatkozások

[1] M. Poór: *Answer to a question of Rosłanowski and Shelah.* `https://arxiv.org/abs/1610.00614`, J. Math. Log. 21. no. 3, 2021.

[2] M. Elekes, M. Poór: *Cardinal invariants of Haar null and Haar meager sets.*, `https://arxiv.org/abs/1908.05776`, P. Roy. Soc. Edinb. A. 151, no. 5, pp. 1568 - 1594. 2021.

[3] M. Poór, S. Shelah: *Universal graphs between a strong limit singular and its power.*, `https://arxiv.org/abs/2201.00741`, submitted.

[4] M. Poór, S. Shelah: *Between uniformization and Whitehead groups.*, `https://arxiv.org/abs/2201.00741`, submitted.

[5] M. Poór, A. Rinot: *A Shelah group in ZFC.*, `https://arxiv.org/abs/2305.11155`, submitted. https://arxiv.org/abs/2305.11155

Sárközi Gergely

EJC: 2023–

Sárközi Gergely vagyok, a Collegium egy bejáró tagja. Egyetemi tanulmányaimat 2020-ban kezdtem meg az ELTE programtervező informatikus BSc képzésén, a világjárvány miatt online oktatási formában. A képzésem harmadik félévében szakirányt kellett választani, ahol én a szoftvertervező specializációt (elterjedtebb nevén a „B szakirányt") választottam, aminek köszönhetően megismerkedtem Fóthi Ákos munkásságával (programozáselmélet, programozási módszertan, bevezetés a programozáshoz).

Régóta foglalkoztattak (illetve továbbra is foglalkoztatnak) az elosztott rendszerek, a számítógépes hálózatok és a konkurens programozás, így 2021-ben, egyéni szoftverfejlesztési vállalkozásomat megszüntetve, az Ericsson Magyarország Kft.-nél kezdtem el dolgozni egy gyakornoki pozícióban. Egy évig maradtam a cégnél, így bőven volt lehetőségem tapasztalatot szerezni a telekommunikáció, hálózati algoritmusok témákban. A felmondásnak köszönhetően szerzett szabadidőmet nem sokáig ízleltem; 2022-től az Informatikai Kar Információs Rendszerek Tanszékén folytatok kutatási tevékenységet a programozható hálózatok, P4 programozási nyelv témákban, Vörös Péter témavezetésével. Ezen túl már több félévben is vállaltam gyakorlatvezetési feladatokat: segítettem mind nappalis, mind estis hallgatóknak megismerkedni a Java programozási nyelvvel a Programozási nyelvek nevű tárgy keretein belül, illetve Eseményvezérelt alkalmazások (EVA) gyakorlatot is tartottam.

A Collegium (illetve az Informatikai Műhely) tagja csak 2023-ban lettem, a programtervező informatikus MSc képzés megkezdésével egy időben. Megpróbálom a tagságom nyújtotta lehetőségeket a lehető legjobban kihasználni, a megszervezett eseményeken részt venni, így bejáró státuszom ellenére viszonylag sok időt töltök a Collegium épületén belül. Jelenleg a Collegium Választmányának gazdasági alelnöki pozícióját töltöm be.

Sümegi Károly

EJC: 2007–2011

2007 az egyik első év volt, hogy már programtervező informatikusnak hívták a szakot, amikor mint bejáró kollégista kezdtem az Eötvös Collegium Informatikai Műhelyében. Erős matematikai hátterem mellett szerettem volna informatikai tudással is felvértezni magam. A kezdetektől érdekelt a diszkrét matematika és az analízis informatikai alkalmazásai, ezért a Modellalkotó (A) szakirány egyértelmű választásnak tűnt. A BSc szakdolgozatomat Zempléni András tanár úrnál, egy valószínűségszámításhoz használható UI formájában alkottam meg Qt-ben. Az MSc szakdolgozatomat és TDK dolgozatomat Fridli Sándor tanár úrnál írtam. Az MSc dolgozat Hough-transzformációval egy képfeldolgozási problémára nyújtott algoritmikus megoldást. A TDK dolgozatot, ami trigonometrikus görbéket elemzett Euler-formulával, csapatban írtuk Kovács Péter collegista műhelytársammal.

2012 óta egy nagy befektetési bank budapesti fejlesztőközpontjában dolgozok. Jelenleg ügyvezető igazgató stratégaként egy kisebb csapatot irányítok. Röviden a stratéga annyit jelent, hogy gazdasági termékekre tervezünk és fejlesztünk kvantitatív modelleket. A stratéga feladatai közé tartozik az aktuális gazdasági folyamatok és regulációk megértése. Ezen problémák megoldása a sztochasztikus differenciálegyenletektől a gépi tanulásig szerteágazó matematikai ötlettárat megkívánhat. A kódolás legtöbbször objektumorientáltan, magas szintű programozási nyelveken történik, de ez alól kivételt jelenthet például magas számításigényű numerikus algoritmusok optimalizációja.

Munkám során szükség van a biztos matematikai alapokra, leginkább a statisztika és az analízis területén. Programozásból napi szinten hasznát veszem Lócsi Levente tanár úr segítségével elsajátított bash alapismereteknek és Csörnyei Zoltán tanár úr funkcionális programozásának is. Ezen felül a collegiumi nyelvórák hozzásegítettek, hogy egyetem mellett felsőfokú nyelvvizsgát tegyek. Az effajta kommunikációs

készség egy nagyvállalatnál kiemelkedő versenyelőnyt jelent. Végül köszönettel tartozom a pezsgő és sokszínű collegista közösségnek, mert az itt szerzett barátságok az embert egy kihívásokkal teli úton elkísérik.

Szijjártó Beáta

EJC: 2007–2011

szijjartobeata@gmail.com

Egyetemi tanulmányaimat követően – egyébként egy collegistatárs ajánlására – a Deutsche Telekom IT Solutions cégcsoportnál helyezkedtem el. Az itt eltöltött lassan 12 év alatt több munkakörben szereztem tapasztalatot az informatikai üzemeltetés világában, elsősorban német nyelvterületen jelenlévő nagyvállalatokkal együttműködve (BMW, BHF-Bank, Huawei, Sopra Financial Technologies).

2012-ben junior adatbázis-adminisztrátorként érkeztem a céghez. A szakterület-választásban nem titkolt szerepe volt az Oracle Junior Program szemináriumsorozatának, amelyet még hallgatóként nagy érdeklődéssel látogattam. Feladataim közé tartozott az Oracle adatbázisok felügyelete, szoftverek telepítése, frissítése, teljesítményük optimalizálása, a hozzájuk kapcsolódó incidensek megoldása. Rögtön a mély vízbe kerültem, hiszen mindezt a BMW vállalat saját fejlesztésű, Linux alapú, automatizált környezetében, gyakran a gyártással közvetlen kapcsolatban álló adatbázisokon gyakorolhattam, ráadásul német munkanyelven.

A BMW projekt lezárulta után 2014-ben tettem egy kis kitérőt a folyamatirányítás és a minőségbiztosítás területére: egy német banki ügyfél (BHF-Bank) change-, problem- és incidensmenedzsment feladataiba kapcsolódtam be. Inspiráló volt megismerni a különböző szakmai csoportok együttműködését a cégen belül (operációs rendszerek, alkalmazások, hálózatok, értékesítés), illetve a gyakorlatban alkalmazni az ITIL vállalatirányítási módszertant (amely addig csak kötelező elméleti tananyag volt). Ebben az időszakban került bevezetésre a HP Service Manager folyamatkezelő rendszer az ágazatban, ennek betanulásában trénerként segítettem a magyar és a német kollégákat.

2016-ban ismét az adatbázisokhoz kapcsolódó feladatok kerültek számomra a fókuszba: tevékenységem a MySQL üzemeltetéssel bővült,

többek között a Huawei és több kisebb német ügyfél révén. A „klasszikus” Unix rendszereken kívül az Open Telekom Cloud felhő alapú architektúráján is dolgoztunk.

Ezt követően 2020-ban vettem át a Sopra Financial Technologies (SFT) ügyfélhez kötődő adatbázis-szolgáltatások koordinációját, majd vezetését. Ebben a szerepben a különböző üzemeltető csoportok munkájának szervezése, összehangolása a feladatom, együttműködve más ágazatokkal, az ügyféllel és az anyavállalat vezetőségével – így a szakmai tudás mellett a korábbi minőségbiztosítási tapasztalataimat is hasznosítani tudom.

Jelenleg az SFT ügyfélhez tartozó adatbázis technológiák (Oracle, MSSQL, DB2LUW) szakmai felelőse, egy agilis DBA csapat (squad) vezetője vagyok.

Visszatekintve az eltelt évekre és a Collegiumra, hálás vagyok a szakmai és baráti közösségért, a sokszínűségért, interdiszciplinaritásért, amelyet – más formában ugyan, de – a munka világában viszek tovább. Örömmel gondolok a Diákbizottságban eltöltött időre, az Eötvös Konferencia és a felvételi szervezésének izgalmaira, illetve a német és angol nyelvórákra is, amelyek komoly lendületet adtak a multinacionális környezetben való érvényesüléshez.

Szilveszter Máté

EJC: 2022–

2019 őszétől az *ELTE Informatika Karának* programtervező informatikus alapképzésén tanultam. Az egyetem első féléve során világossá vált számomra, hogy a matematikához szorosabban köthető tantárgyak iránt nagyobb érdeklődést mutattam, így dőlt el, hogy mely szakirányon szeretném folytatni az alapképzést. 2020 elejétől kezdve a Neumann-kör tagja lettem, majd 2020 őszétől a Modellező szakirányon induló Neumann-csoport keretein belül folytattam tanulmányaimat. 2021 őszétől kezdődően egészen mostanáig a Numerikus Analízis Tanszéken vállaltam feladatokat demonstrátori munkakörben. 2022 tavaszán a Programozáselmélet és Szoftvertechnológiai Tanszék demonstrátora voltam, ahol Objektumelvű programozás tantárgyból vezettem gyakorlatot. 2022-ben az ELTE Informatika Karának programtervező informatikus alapképzésén kitüntetéses oklevelet szereztem.

2022 őszétől az ELTE Informatika Karának programtervező informatikus mesterképzés Modellalkotó szakirányán tanulok. Szakmai mentoraim segítségének köszönhetően lehetőségem nyílt arra, hogy az eddigi mesterszakon töltött féléveim alatt gyakorlatot vezessek Matematikai Alapok és Analízis I. tantárgyakból, és ezt a továbbiakban is szeretném folytatni.

2023 őszén elkezdtem megismerkedni a harmonikus analízissel, azon belül is a Walsh-szerű rendszerekkel. Minden reményem szerint ebből a témakörből egy sikeres diplomamunkát fogok tudni készíteni, és a tervek szerint a későbbiekben is foglalkoznék ezen rendszerek vizsgálatával a doktori képzés keretén belül.

Szirákiné Kovács Gyöngyi

EJC: 2010–2014

Informatikus könyvtáros hallgatóként 2010 és 2014 között voltam az Informatikai Műhely tagja. Abban a megtiszteltetésben volt részem, hogy én voltam a Műhely első bölcsész tagja. A Műhelyben a számítógépes nyelvészetet, ezen belül is a fordítóprogramokat kutattam, és ebben a témában tartottam előadást az Eötvös Konferencián. Később magyar–angol tanári mesterszakot végeztem, és angol irodalomból doktoráltam.

Jelenleg könyvtárosként dolgozom a szolnoki Verseghy Ferenc Könyvtárban, és könyvtárpedagógiára szakosodtam. Azt vizsgálom, hogyan lehet az irodalom tanítását minél élményszerűbbé tenni a diákok számára, és hogyan lehet az órákon feldolgozott irodalmi műveket az ink narratív programozási nyelvet használva interaktív művekké alakítani. Az interaktív irodalmi mű befogadása aktív részvételt kíván a befogadó részéről, így könnyebben fenn tudja tartani a figyelmet, és jobban bevonja az olvasót a mű világába, mint a hagyományos regény vagy novella, ezért érdemes felhasználni az irodalomórán.

Szita István

EJC: 1997–2005

szityu@gmail.com

A Collegiumba 1997-ben kerültem matematikusként. Sőt, még diplomát is matematikusként szereztem 2002-ben, de a témám már számítógéptudományosabb volt. (Megerősítéses tanulás: már akkor látszott, hogy menő téma! Egy standard tanulóalgoritmust alakítottunk át hatékonyabbra.) Aztán a doktorimat már az Informatikai Karon végeztem Lőrincz Andrásnál. Az első két évben még a Collegiumban laktam, aztán utána a XI. kerület távolabbi végéből voltam a műhely tagja.

Műhelytagként tartottam néhány órát a Collegiumban (bevezető órákat neuronhálózatokról, gépi tanulásról, egyszer még kriptográfiáról is), de főleg az Estikében építettem a műhelykapcsolatokat. No meg a kirándulásokon.

Közben sikerült elcsípnem néhány ösztöndíjat: a Maastrichti Egyetemre, Pieter Spronckhoz (nagy sikerrel alkalmazott tanulóalgoritmusokat játékokban, Baldur's Gate-tel és Catan telepeseivel foglalkoztunk együtt), aztán Amerikába, a Rutgersre, Michael Littmanhez (ő mindennel foglalkozott, ami megerősítéses tanulás; amíg ott voltam, tanulhatóság-elmélettel foglalkoztunk, azaz, hogy hogy jellemezhetők a könnyen megtanulható feladatok), aztán megint egy picit Maastrichtba. Amikor aztán 2008-ban végre ledoktoriztam (a téma vegyes volt: egyrészt konvergenciabizonyítások és lépésszám-felsőkorlátok néhány algoritmusra, másrészt meg az algoritmusok alkalmazása játékokban, pl. Tetrisben), lehetőséget kaptam, hogy az egyik legjobb megerősítésestanulás-kutatócsoportban legyek posztdok, Rich Suttonnál és Szepesvári Csabánál. (Itt főleg a korábbi elméleti irányokon dolgoztam tovább, konvergenciabizonyításokon, bonyolultságelméleti kérdéseken. Jól jött a matekos képzés.) Apró szépséghiba, hogy ez Edmontonban volt, ahol télen -30 fok van néha (és itt már mindeggyé válik, hogy Celsius vagy Fahrenheit), de ettől még csodálatos hely.

2011-ben elhagytam az akadémiát, a Google zürichi irodájától kaptam egy állásajánlatot, és azóta is ott dolgozom. A feladatom, hogy a Google Shoppingon ne legyenek átverések, meg úgy általában biztonságos legyen ott a vásárlás. Mindenféle gépi tanulást használunk (felügyelt tanítást, klaszterezést – megerősítéses tanulást pont nem), hogy minél jobban felismerjük a csalókat. Ha kicsiny csapatunk jól végzi a munkáját, akkor nem szereplünk a hírekben.

2001-ben megtaláltam életem párját, Rékát (manírosnak hangzik, de az eltelt 23 év ad némi empirikus alátámasztást). Ő szintén Collegista, de nem informatikus (hanem irodalmár), aztán mégis ő tanít az ETH-n. Röpke 10 év együttlét után 2011-ben megházasodtunk, aztán 2017-ben született egy kisfiunk, Brúnó, aki egészen hirtelen 6 éves lett, és természetesen elképzelhetetlenül okos és édes és jóképű.

2011 óta Svájcban lakunk Rékával (és újabban Brúnóval), egy giccsesen gyönyörű faluban Zürich mellett, és élvezzük. Egészen sok Collegista barát sodródott a környékre – egy alkalommal Zürichben tartottunk EC-s fondüzést, és 8 főre kellett asztalt foglalni.

Háziállatunk van. Totónak hívják, és aranyhörcsög származású.

Szokoli Mátyás

EJC: 2018–2021

Jelentős előzetes tapasztalattal kezdtem el az egyetem informatika képzését hála a matematika és informatika szakköröknek és fakultációknak. Egy barátom javaslatára jelentkeztem az Eötvös József Collegiumba is 2018-ban, mert úgy tűnt, hogy sok érdekes dolgot kínál az informatika műhely és a teljes Collegium is. Emlékszem még, ahogy a távolsági buszon Budapestre utazva olvastam műhelyben tanított tantárgyakat, és csak vakartam a fejemet, hogy mi lehet az a *funkcionális* programozás.

Ennek ellenére megszerettem a λ-kalkulus és típusrendszerek tárgyakat, mert egy teljesen új gondolkodásmódot mutattak be. A teljes felsőoktatási képzésemet meghatározta ez a benyomás, felkeltette bennem a fordítóprogramok, típusrendszerek és érdekes programozási nyelvek iránti érdeklődést. Több kis projektet is írtam az alapképzés alatt, pár típusellenőrzőt és egy interpretert.

Miután elvégeztem az alapképzést, jelentkeztem mesterképzésre is, ahol a RefactorErl labor keretében jelenleg az Erlang típusozásával foglalkozom. A BSc-m utolsó két félévében elkezdtem dolgozni az iparban, amit folytattam az MSc képzés mellett is. Amikor egy cég látta, hogy van előzetes Erlang tudásom, gyorsan kaptak az alkalmon, pedig teljesen más technológiára jelentkeztem eredetileg. Itt lehetőségem volt újraírni a fordítóprogramját egy kis belső használatú programozási nyelvnek is, ami egy egyedi feladat volt. Maga a nyelv nem volt bonyolult, de sok át nem gondolt vagy nem hatékony dolog le lett cserélve.

A funkcionális programozás és a fordítóprogramok tárgyak nem csak szemléletet adnak vagy szélesítik a szakember látóterét. Sosem tudhatod, hogy mikor jön jól az ilyen tudás, bármennyire nem hétköznapinak tűnik is.

Tompos Anna

EJC: 2021–

Őszintén szólva a mai napig sem teljesen világos, hogy miért vettek fel ebbe a kollégiumba. Nem én voltam a legokosabb, leggyorsabb, legnagyobb tudású, és valószínűleg a legszebb sem. De van két szuperfegyverem, amikkel mégiscsak beküzdöttem magamat ide: az egyik a kifogyhatatlan lelkesedésem, a másik pedig egy igazán csodálatos illető, aki fentről folyamatosan azon munkálkodik, hogy a fontos dolgok rendben legyenek. Szóval 2021-ben elsőéves matek–infó tanár szakos hallgatóként nem csak a hatalmas Budapest rohanó életébe csöppentem bele, hanem a nagy múltú Eötvös József Collegium varászlatos világába is, amiért soha nem lehetek elég hálás.

Hogy mégis mi mindent adott nekem ez a kollégium, akarom mondani Collegium? Erről a kérdésről valószínűleg napokig tudnék áradozni, de most igyekszem röviden összefoglalni a lényeget. Adott nekem közösséget, kitartást, a határaim feszegetését, lehetőséget a jobbá válásra, hatalmas tűrőképességet, rengeteg abszurd élményt, nem vágyott, de mégis örömmel fogadott hideg zuhanyt, zsúfolt konyhai főzéseket, értékes receptcseréket, és legfőképpen egy helyet, ahova tartozni lehet. Furcsa, hogy amikor beköltöztem, és elkezdtem itteni életemet, az egyáltalán nem volt furcsa. Az első perctől otthon éreztem magamat, és még mindig imádok itt élni a hely és az itt lévő emberek minden kellemetlen, de mégis valahogy szórakoztató különösségével együtt.

A collegiumi Cözéletbe is elég gyorsan belekerültem, a góJatábor után csatlakoztam az EKÉT-hez (Eötvös Kísérleti Énektudományi Tanszék), aminek azóta is kitartó tagja vagyok. Egy felvételin tett ígéretemhez híven a covid alatt elmaradozó Biblia Cört is sikerült újraindítanunk annyira sikeresen, hogy még mindig élünk és virulunk, és vasárnap esténként mindig szakítunk időt Istenre és a hálaadásra. Ilyenkor mindig elképesztő megtapasztalni, hogy mekkora ereje is tud lenni az imának, a beszélgetéseknek és csak az egyszerű együttlétnek.

A Választmányt sem tudtam elkerülni, másodévesként rendíthetetlenül munkálkodtam a Collegium sportéletének fellendítésén, szerintem egészen sikeresen.

Egyetem előtt azt gondoltam, hogy a kutatás egy igazán bonyolult dolog, és hogy csak az igazán kiemelkedők foglalkozhatnak ilyesmivel. Kiderült, hogy ez nem egészen így van, csupán kíváncsiság és szorgalom kell hozzá. Így hát másodévesen belekerültem egy kutatócsoportba, ahol arra a kérdésre kerestük a választ, hogy hogyan is lehet igazán hatékonyan matematikát tanítani. Én azzal foglalkoztam, hogy a társasjátékokat hogyan lehet úgy bevinni matekórai keretek közé, hogy játszva fejlesszék a gyerekek logikai gondolkodását úgy, hogy az ne menjen a tananyag rovására. Ezzel a témámmal eljuthattam Nagyváradra egy matematika didaktikai konferenciára és a Veszprémben szervezett OTDK-ra. Itt jöttem rá, hogy ami még a kutatásnál is jobb, az a konferenciákon való részvétel. Szuper társaság, felfedezésre váró új területek, érdekes előadások és finom ennivaló. Mi más kell még?

Az egyetem és a Collegium mellett az időm jelentős részét tanítás teszi ki. Először az Informatika Műhely által tartott „Programozási alapismeretek" órán próbálhattam ki magamat pár óra erejéig, ahol Pythonban tanítottuk programozni a lelkes Collegákat. Nem sokkal ezután tanítani kezdtem egy Alphacademy nevű online programozást tanító iskolánál, ahol kéthetente tartok C++ órákat aranyos gyerekeknek, valamint nyaranta táboroztatom őket. 2023 februárjától részben a Collegiumnak köszönhetően lettek egyetemista csoportjaim is, akiknek Excelt és Pythont oktatok. A következő félévben pedig úgy érzem, hogy lépek még egy szintet, mert a „Programozási alapismeretek" tárgy gyakorlatát tarthatom elsőéves tanárisoknak. Ezek az oktatási tevékenységek azok, amiket a legjobban és a legnagyobb erőbedobással szeretek csinálni. Lehet, hogy a tanári pálya nem a legnépszerűbb, legegyszerűbb hivatás, és lehet, hogy a saját (tanár) nagyszüleid próbálnak meg lebeszélni róla, de én mégsem tudnék magamnak szebbet elképzelni. Tudom, hogy nagyon nehéz lesz, de mégis, már alig várom, hogy bemehessek egy középiskolába tanítani, szakköröket tartani, osztálykirándulásokat szervezni, és leginkább csak szeretni a gyerekeket.

Valami ilyesmi lennék én. Sokszor nem tudom követni azt a temérdek dolgot, ami körülöttem történik, de boldog vagyok azzal, amim van és ami vagyok, és izgatottan várom, hogy mit tartogat még az élet.

Tőkés Anna

EJC: 2014–2018

Az én történetem a Collegiummal 2014 januárjában indult. Egy nagyon tartalmas hetet tölthettem el a Tehetségtáborban, amely után eldöntöttem, hogy szeretnék ennek a közösségnek a tagja lenni. Augusztusban felvételt nyertem az Informatikai Műhelybe és szeptemberben Eötvös Collegistaként kezdhettem az egyetemet. Borzasztóan sokat köszönhetek a Collegiumnak. Fantasztikus embereket ismertem meg, rengeteget tanultam Csörnyei és Kozsik tanár urak műhelyóráin, érdekes műhelykirándulásokon vettem részt, Estike csaposként érdekfeszítő beszélgetésekben volt részem. A 2016/17-es tanév második félévét Erasmus ösztöndíjjal Hollandiában töltöttem, amely után a bentlakó státuszomat bejáróra cseréltem, de továbbra is aktív tagja maradtam a Műhelynek.

Az egyetem mellett a Collegium egy olyan biztos tudásalapot adott számomra, amelyet a mai napig hasznosítok munkám során. 2017 nyarán kezdtem el gyakornokoskodni a Morgan Stanleynél mint Securities Lending Strategist. Egyetemi tanulmányaim befejezése után felvettek teljes állásba. 2022-ben kaptam a lehetőséget, hogy New Yorkba költözhessek, és az itteni Trading Deskkel dolgozzak együtt. A fejlesztés mellett én tartom kézben a New York specifikus Securities Lending projekteket, valamint Delta One Structured termékekkel is elkezdtem foglalkozni. 3 nappal ezelőtt, 2024. január 10-én pedig előléptettek Vice Presidentnek.

Mindezt nem tudtam volna elérni Miki, a férjem nélkül, valamint a kiskutyánknak, Nikonak is az érdeme, hogy sikerül megőriznem a mentális egészségemet a karrierem mellett.

Ungvári Gábor

EJC: 2021–2023

gabor.ungvri@gmail.com

Jelenleg az ELTE Informatikai Kar passzív státuszú hallgatója vagyok, a következő záróvizsga-időszak folyamán szeretném megszerezni az alapdiplomám. Szakdolgozati munkám egy szoftver, amely a hozzá tartozó dolgozattal együtt magában foglalja a gépi tanulásról, bonyolultságelméletről, valamint telekommunikációs hálózatokról megszerzett ismereteim. Tanulmányaim a bécsi Universität Wien egyetemen tervezem folytatni, ide felvételt nyertem Data Science mesterszakra.

Az addig hátralevő időt kihasználva további területeken igyekszem fejleszteni magam: harcművészeteket és hangszeres játékot gyakorlok, igyekszem beleásni magam a hangmérnökség módszereibe, továbbá nemrégiben megszereztem a B kategóriás vezetői engedélyt.

A műhely jelenlegi és volt műhelyvezetőinek, tagjainak ezúton szeretném megköszönni, hogy a tagságom alatt fejlődést biztosítottak számomra szakterületem, valamint szabadidős tevékenységeim terén, illetve hogy megannyi élménnyel lehettem gazdagabb.

Varga Balázs

EJC: 2017–2022

Egyetemi tanulmányaim alatt, 2017 és 2022 között voltam az Informatikai Műhely aktív tagja. Kutatási területem és diplomamunkám konkurens Erlang programok elosztottá alakításával volt kapcsolatos.

2020 elején gyakornokként kezdtem dolgozni a Cloudera-nál. Ezzel a lehetőséggel egy műhelyszemináriumon találkoztam, ahol későbbi csapatvezetőm (szintén volt műhelytag) előadása felkeltette azóta is tartó érdeklődésemet a big data és az elosztott rendszerek iránt.

Három és fél évig ebben a csapatban tevékenykedtem (a gyakornokság után teljes állásban), egy Apache Flink alapú szoftvert fejlesztettünk. Ennek használatával adatelemzők és programozók egyszerűen tudnak SQL segítségével valós idejű, streaming adatokat feldolgozó programokat írni és futtatni. A Flink programok elosztott módon futnak, így a működésük skálázható, tehát nagy mennyiségű adat is alacsony késleltetéssel dolgozható fel. Munkám során a cég szoftverének (SQL Stream Builder) fejlesztése mellett a nyílt forráskódú Flinkhez is hozzá tudtam járulni egy-két apróbb fejlesztéssel.

2023 közepe óta más szerepben veszek részt a szakmában, data engineerként dolgozok az Aliz nevű, magyar alapítású cégnél. Itt a Google felhőszolgáltatásait használva különböző ügyfelek adattárházaival kapcsolatos projektjeit valósítjuk meg. A Flinkes érdeklődés megmaradt, de most felhasználói oldalról dolgozom a rendszerrel egy-egy projekt részeként. Emellett persze számos (számomra) új technológiával van lehetőségem megismerkedni.

Az egyetem alatti két külföldön (Linz, München) töltött félév óta Budapesten élek, de az utazás továbbra is fontos része az életemnek. A collegiumi éveimre örömmel tekintek vissza, és azóta is szorosan tartjuk a kapcsolatot több collegista társammal.

Varga Norbert

EJC: 2022–

Másodéves programtervező informatikus hallgató vagyok, illetve a Collegiumban is másodéves. Már gimnáziumi éveim alatt is, az ELTE Radnótiban, különösen érdekelt nemcsak a matematika, de az informatika is. Nem volt hát kérdés, hogy az ELTE Informatikai Karának szakára jelentkezzek. E mellé szerettem volna egy kollégiumban is helyet tudni magamnak, hogy közelebbi ismeretségbe kerülhessek szaktársaimmal. Végül édesapám ajánlotta az Eötvös Collegiumot, amely még jobbnak tűnt, tekintve erős tudásszomjamat.

Már a felvételin és a gólyatáborban is látszódott, milyen különleges, kiemelkedő és összetartó közösség ez a Collegium, illetve maga a műhely is. A túráktól kezdve a főzőcskézéseken keresztül egészen a műhelyórákig mindegyik csupa élmény, emlék, lehetőség.

Az első félévben rögtön belevetettem magamat egy izgalmas témába, mesterséges intelligencia témakörben végeztem enyhébb „kutatást”, amely inkább a témában való elmélyülésről szólt. Azóta is szeretem ezt a témát, fel is vettem ezzel kapcsolatos tárgyakat.

Most azonban fő kutatási témámnak a típuselméletet tekintem. Már első félévben nagyon megtetszett a funkcionális programozási paradigma, majd érdeklődésemet tovább növelte a témában a második félévben műhelyóraként tartott típuselmélet óra. Ebben az évben már saját kutatást végzek, most épp egész számok reprezentálásával foglalkozok.

Ezeken a témákon kívül különösen foglalkoztat még a grafika, izgatottan várom, hogy a következő félévben belemerülhessek ennek rejtelmeibe egy egyetemi tárgy keretei között.

Tanulmányaim mellett informatikát tanítok is, illetve hobbiként bridzselek.

Varró Máté

EJC: 2022–

Másodéves programtervező informatikusként úgy érzem, az utóbbi másfél év alatt rengeteget fejlődtem, és ez részben (elég nagy részben) a Collegium Informatikai Műhelyének köszönhetem. Nem csupán szakmai tapasztalatról beszélek itt, hanem emberi, humánus értékekről, demokráciáról és interdiszciplinaritásról.

Varró Máté vagyok, jelenleg az ELTE IK tanulója, és az Informatikai Műhely tagja. Kutatásom még nem realizáltam teljes mértékben, de mostani projektem (mely legközelebb áll valamifajta kutatáshoz) egy *álmosság detektálásával* foglalkozó projektben való „részvétel" Kovács Szilárd tanár úr vezetésével.

A kutatás egy gépjárművezető fáradtsága, és a vezetőnél mért EEG jelek közötti korrelációt vizsgálja mesterséges intelligencia segítségével. Ez azért fontos, mivel rengeteg autóúton történő baleset közvetlen okozója a vezető fáradtsága, mely az esetek többségében megelőzhető lenne. Így egy olyan eszköz, ami képes lenne megállapítani egy vezetőről, hogy megfelelő állapotban van-e a vezetéshez, sok életet tudna megmenteni.

„Kutatásom" mellett pár hobbi projekttel is rendelkezem, melyek többsége nem kapcsolódik az informatikához. Azonban az utóbbi időben kipróbáltam a Rust programozási nyelvet, és magával ragadott a furcsa féltökéletessége. Ha le kéne írnom a nyelvet, azt mondanám, minden, ami benne van konzisztens és jól használható, de ami nincs, annak hiánya érződik. A Rust félkész, azonban rengeteg olyan előnnyel rendelkezik, ami miatt úgy döntöttem, elkezdek benne egy új programozási nyelvet írni. Ebből még lehet is valami.

Végh Zoltán

EJC: 2002–2008

vozo@vozo.hu

Most már elárulhatom azt az esetet, amikor vágóstúdiót rendeztem be a műhelyszobában és oda hívtam az ügyfelemet... Az ELTE-n és az Universität Klagenfurton végzett informatikai tanulmányaim mellett (helyett) elvégeztem a Polifilm mozgóképgyártó iskolát és a Magyar Operatőrök Társaságának kameraasszisztensi képzését.

A 2006/2007-es tanévben a Collegium Diákbizottságának elnöke, és az ELTE Hallgatói Önkormányzatának alelnöke voltam. Ugyanennyire fontos emlék, hogy elnyertem a Mr. Eötvös címet is!

A filmes karrieremet a különböző munkahelyeimen és produkcióimon keresztül tudom szemléltetni:

- *Nap-Kelte.* Egy időben ez volt az ország vezető politikai magazinműsora. A szakma alapjait itt sajátítottam el, több miniszterelnökre is én helyeztem fel a mikrofont.
- *Plotpoint reklámügynökség.* Itt volt lehetőségem először reklámot rendezni, amit aztán moziban is vetítettek.
- *LEN (Európai Úszószövetség).* A különböző Európa-bajnokságokon interjúkat készítettünk az úszókkal. Sok utazás.
- *FINA (Nemzetközi Úszószövetség).* A különböző Világbajnokságokon interjúkat és kisfilmeket készítettünk az úszókkal. Még több utazás.

Szabadúszó rendezőként több sorozatot forgattam 2013-tól kezdve (RTL, TV2, Duna):

- *Családi Titkok, Magánnyomozók, Zsaruk.* Ezek ún. scripted reality formátumú sorozatok, de legtöbben trash realityként ismerjük. A végeredménynél sokkal érdekesebb ahogyan a gyártásuk zajlik. Három napos forgatással egy órás műsoridőt produkálunk. A nézettség pedig sokszor veri a nagyságrendekkel több pénzből készült sorozatokét.

- *Barátok Közt.* A valaha volt leghosszabban sugárzott magyar szappanopera. Rettenetesen izgultam az elején. Végül a 23 évnyi műsorfolyam legutolsó jelenetét én rendeztem, sőt! Átírattam! Túl magabiztos fiatal filmeseknek szoktam válaszolni, hogy valóban nem nehéz elkészíteni egy szappanopera-jelenetet. Naponta 28-at elkészíteni annál inkább!

- *Pince.* Jó ha az ember tudja, mikor volt élete legmélyebb pontja.

- *Oltári Csajok.* Az első sorozatom, aminek az elejétől végéig alkotója voltam. A színészek kedvenc rendezőjéből a legutáltabb rendező lettem, mert egyszer csúnyán kiabáltam velük.

- *Családi Kör.* Értékelni kell azokat a szakmai lehetőségeket, amikor az ember dolgába semmilyen szinten nem szól bele senki. Ennek a munkámnak a végére éreztem magamat magabiztos rendezőnek.

- *Bátrak Földje.* Kosztümös történelmi sorozat gyönyörű ruhákkal, lovakkal, hintókkal, kastélyokkal. Viszont ha túl sokat dolgozik az ember, kiégés a vége.

- *Oltári történetek.* A gázszámlát is ki kell fizetni valamiből.

- *Szomszéd Tehene.* Ez még friss, jelenleg is sugározza a TV. Talán annyi, hogy hosszútávon a család fontosabb, mint a karrier.

- *Psycho60.* Ez kakukktojás, mert saját kisfilm. 1960-ban készítette Hitchcock a korszakos jelentőségű Psycho című filmjét. A leghíresebb jelenete a „Zuhanyjelenet”, amikor egy felismerhetetlen gyilkos leszúrja a zuhanyzó Janet Leigh-t. Ezt a jelenetet forgattam újra a film bemutatásának 60. évfordulójára, 60 különböző színésznővel. Minden vágás után újabb színésznő játszotta újra az eredeti szerepet. Néha csak egy könyököt láttunk vagy lábat

vagy ujjakat – mégis minden mozdulatot más-más színésznő játszott el. A legnagyobb színészlegenda Törőcsik Mari utolsó filmes megjelenése volt mielőtt eltávozott közülünk.

Létrehoztam a saját gyártócégemet (2017, VozoFilms), így producerként is készítek filmes megbízásokat. Ahogy a kisfiam egyik kedvenc meséjében hangzik el: „A producer a fancy neve annak, aki az emberekből sztárt csinál!" Ennek szellemében végzem a munkámat.

Van egy fantasztikus feleségem, most várjuk a második fiunkat. A magánélet kiegyensúlyozottsága alapvető fontosságú. Sok-sok hibát kívánok mindenkinek az igazi pár megtalálása felé vezető úton!

Madaras fickó lettem. Kis hullámos papagájokkal kezdtem, de sajnos elhullott Skrillex, Jeremy és Csip is. Majd egy kékhomlokú amazon papagájra vártam egy évet, és a szombathelyi mentőállomáson ki is kelt a tojásból. (A tenyésztője mentőtiszt.) Ráspoly már négy éves és imád énekelni. Kedvence a Don't worry, be happy!

Névmutató

$$E = CCCCCCCCC$$
$$C = \lambda\, icugolem.m(collegium)$$

Fixpontkombinátor az Eötvös Collegiumhoz