Csörnyei Zoltán

PI-KALKULUS
A mobil rendszerek elmélete

ELTE Eötvös József Collegium
2017

PI-KALKULUS

Csörnyei Zoltán

PI-KALKULUS

A mobil rendszerek elmélete

Budapest, 2017

A kiadvány az ELTE Eötvös József Collegium támogatásával készült.

A borítón a tulajdonos engedélyével az Archimédés-palimpsestusnak az a lapja szerepel, amelyen a kör látható, utalva Archimédésnek a körrel és a később pi-nek nevezett szám értékével kapcsolatos számításaira. Bár a pi-kalkulus elnevezésében a pi a processz szó első betűjéből származik, könyvünk a pi betűn keresztül kapcsolódik a palimpsestushoz, a kézirat első olvasói és bírálói a Collegium Informatikai Műhelyének tagjai voltak, és a Collegiumban a görög szöveg kiolvasására alakult, Horváth László által vezetett és rendkívüli sikereket elért kutatócsoport szoftver támogatásához a Műhely is hozzájárult.

ISBN 978-615-5371-74-5

Felelős kiadó:
Dr. Horváth László,
az ELTE Eötvös József Collegium igazgatója
Borítóterv: Egedi-Kovács Emese
A nyomdai munkákat a Komáromi Nyomda és Kiadó Kft. végezte
2900 Komárom, Igmándi út 1.
Felelős vezető: Kovács János ügyvezető igazgató

Előszó

Ez a könyv egy *oktatási segédlet*, az Eötvös Loránd Tudományegyetem Eötvös József Collegiumában tartott *Mobil rendszerek elmélete* tantárgy anyagát tartalmazza.

A könyvben leírtak megértéséhez különösebb előismeret nem szükséges, de mivel a leírt rendszer sok jellemzője, felépítése, működése nagyon hasonlít a lambda-kalkulus és a típusos lambda-kalkulus tulajdonságaihoz, a lambda-kalkulus ismerete megkönnyítheti az anyag elsajátítását. A lambda-kalkulust a Collegium elsőéves informatikus hallgatóinak, ezt az anyagot, a mobil rendszerek elméletét a másodéves collegistáknak ajánljuk.

A mobil rendszereket, az 1950 környékén végzett kezdeti kísérletektől eltekintve, az 1990-es évek elején kezdték el alaposabban vizsgálni, ennek oka a párhuzamos és disztributív programozás hardver és szoftver feltételeinek ugrásszerű javulása volt. Az elmélet jelenleg is intenzíven fejlődik, szakfolyóiratokban újabb és újabb eredmények jelennek meg. Könyvünkben az elmélet már állandósult, kidolgozott területeiről igyekszünk egy átfogó képet adni.

Elsősorban az elvekre, módszerekre, a kapcsolatokra koncentrálunk, a tételek bizonyításával nem foglalkozunk, ezek a könyv végén, az Irodalomjegyzékben megadott anyagokban megtalálhatók.

Az Irodalomjegyzékben a témához kapcsolódó *felhasznált publikációk* adatai vannak, amelyek az elmélet további tanulmányozásához is segítséget nyújtanak.

Könyvünkkel azt szeretnénk elérni, hogy az informatikus egyetemi hallgatók és az informatikus szakemberek megismerjék ezt az új kutatási területet, a mobil rendszerek elméletét.

* * *

A könyv lektorálására az Eötvös József Collegium programtervező informatikus szakos collegistáit kértem fel. Külön köszönöm

Borbényi Áron,
Boros Attila,
Fogas Lívia,
Luksa Norbert

collegisták észrevételeit, ezek a könyv megírásához nagy segítséget nyújtottak.

Külön köszönet illeti *Horváth László*t, az ELTE Eötvös József Collegiumának igazgatóját, amiért lehetővé tette ennek a könyvnek a kiadását.

Kérem, hogy észrevételeiket a `csz@inf.elte.hu` címre írják meg.

Budapest, 2017. január 31.

Csörnyei Zoltán

Jelölések

- Mivel a pi-kalkulusnak még nincs kialakult jelölésrendszere, az utóbbi évek publikációiban használt, talán már mindenki által elfogadott egységes jelöléseket használjuk, de a korábbi évek publikációiban használt jelöléseket is megadjuk. A könyvben vannak olyan fogalmak, amelyeknek csak az angol nyelvű kifejezése közismert, ezért a Tárgymutatóban

  $\langle$magyar név$\rangle \equiv \langle$angol név$\rangle$

  alakban leírjuk a gyakran használt magyar kifejezések és rövidítések angol megfelelőjét is.
- Mivel a görög ν betű könnyen összetéveszthető a v betűvel, és mindkét betűt intenzíven használjuk a könyv anyagában, a görög ν betűt félkövér (bold) vastagsággal írjuk: $\boldsymbol{\nu}$.
- Ha szükséges, a kifejezésekben a kommunikáló részkifejezéseket jelöljük, a jelölésre a pontozott aláhúzást használjuk, például $\underline{l(t, f)}$.
- A téma könnyebb megértése és elsajátítása érdekében a könyvben nagyon sok példát adunk, a példák végét a □ jellel jelöljük.
- Felhívjuk a figyelmet arra, hogy a mondat végére a pontot kitesszük akkor is, ha a mondatvégi kifejezés önálló, külön sorban van. Ez nem tévesztendő össze a kifejezés „ . ” karaktereivel.

Tartalomjegyzék

1. FEJEZET

Bevezetés

A pi-kalkulus egy jól definiált szintaktikával és szemantikával rendelkező matematikai számítási modell, ahol a folyamatok között az információ névvel ellátott csatornákon áramlik. Alapvető műveletek a csatornára történő adatküldés, egy csatornáról jövő adat fogadása, és két folyamatnak egy csatornán végrehajtható kommunikációja. A kommunikáció jellegzetessége az, hogy a küldő folyamat meghatározza melyik csatornán küldjön adatot, de arra nincs hatással, hogy ezt az adatot melyik folyamat fogja majd megkapni. A fogadó folyamat olvassa az adatot, és az adat, mint egy aktuális paraméter, beépül a fogadó folyamat törzsébe.

Az első ilyen jellegű rendszert Robert Milner dolgozta ki, amelynek a *CCS (Calculus of Communicating Systems)* nevet adta [28]. Tőle függetlenül egy hasonló, de más elveken alapuló rendszert hozott létre C. A. R. Hoare, amelyet *CSP-nek (Communicating Sequential Processes)* neveznek [20, 21]. Munkájukért mindketten *Turing Award* díjat kaptak.

A CCS olyan párhuzamosan futó folyamatok leírására szolgál, amelyek kommunikációs csatornákkal rendelkeznek. A folyamatok ezekre a csatornákra adatokat küldenek és a csatornákon jövő adatokat olvassák, így ha két folyamat között van egy csatorna, akkor ezen a csatornán a két folyamat között kommunikáció történhet. A CCS rendszer egyik célja ennek az *interakciónak*, *kommunikációnak*, az interaktív működésnek a precíz leírása, azonban itt még az inputtal kapott adat behelyettesítésével nem foglalkoztak.

A másik alapvető problémakör annak a gyakorlatban, az alkalmazásokban felvetődő problémának a vizsgálata, hogy két interaktív rendszer mikor tekinthető azonos működésűnek. Ehhez természetesen definiálni kellett a „működés”, a *működési ekvivalencia* fogalmát.

A *pi-kalkulus (Calculus of Mobile Processes)* a CCS rendszer továbbfejlesztése, első kidolgozása szintén Robert Milner nevéhez fűződik [32].

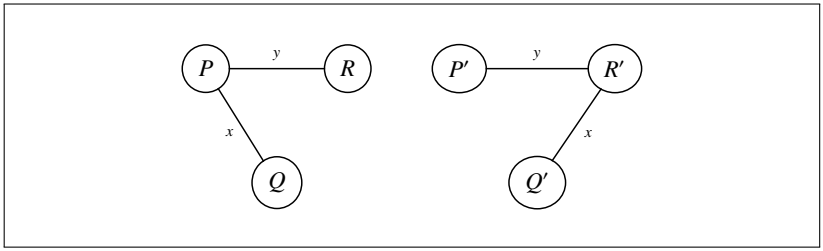
A CCS rendszerben a névvel megnevezett csatornákon adatok továbbítására van lehetőség, a pi-kalkulusban azonban a csatornákon nem csak a megnevezett adatok, hanem nevek, a csatornák nevei, a magasabb rendű rendszerekben pedig folyamatok is továbbíthatók, ami lehetővé teszi a folyamatok topológiájának, azaz a folyamatok közötti kapcsolatoknak a dinamikus megváltoztatását is.

1.0.1. Példa. *(A folyamatok közötti kapcsolatok)*

A P és Q folyamatok között van egy x nevű csatorna, és ezt a nevet a P folyamat a P és R közötti y csatornán átküldi R-nek. Az R folyamat veszi ezt a jelet, és a kommunikáció eredménye az lesz, hogy az x nevű csatorna a Q és R' folyamatok közé kerül át (1.1. ábra).

Ez az absztrakt leírás többféleképpen is értelmezhető. Például,

- Q a felhasználó, aki nyomtatni szeretne, P a nyomtatót kezelő program, és R a nyomtató. A működés eredménye az, hogy a rendszer a felhasználót összekapcsolja a nyomtatóval.
- Q egy ügyfél, aki az igazgatóval szeretne beszélni, P a titkárnő, aki tudja az igazgató telefonszámát, és R az igazgató. Az eredmény most az lesz, hogy a titkárnő átadja az ügyfélnek a telefonszámot, aki így közvetlen kapcsolatot tud teremteni az igazgatóval. □



1.1. ábra. A konfiguráció változása

A CCS másik alapvető továbbfejlesztése az, hogy míg a CCS rendszer csak a folyamatok közötti interakció tényével foglalkozik, a pi-kalkulus elemzi és értelmezi a kommunikációnak a processzekre történő hatását is, ami a csatornán vett adatnak a folyamatba történő beépülését jelenti.

Ezért van az, hogy az 1.0.1. példában az R folyamat az y csatornán kapott

x névvel fog a Q folyamathoz kapcsolódni, és így R'-vé válik. Közben természetesen a P is megváltozik, miután végrehajtotta az y csatornán az output műveletet, P' lesz.

A példában szereplő rendszer precíz leírását és a kommunikáció pontos lezajlását majd a 2.4.2. példában adjuk meg.

A következő fejezetben a pi-kalkulussal foglalkozunk, áttekintjük a pi-kalkulus szintaktikáját, szemantikáját és érdekességképpen megadjuk egyes matematikai és informatikai fogalom és művelet folyamatkifejezéssel történő leírását. Ezután különböző biszimulációk tulajdonságait elemezzük.

Foglalkozunk még a kalkulus algebrai elméletével és speciális pi-kalkulussokkal, például a típusos pi-kalkulussal is.

2. FEJEZET

A pi-kalkulus

2.1. Folyamatkalkulusok

A szekvenciális programok szemantikájának megadása egyszerű, egy függvénnyel leírható a végrehajtandó input-output transzformáció. A konkurens és interaktív programok nemdeterminisztikus viselkedése azonban elbonyolítja ezt a módszert, az elágazásokban egy rossz irány választása könnyen megnehezítheti, sőt akár le is állíthatja az adott irányban haladó vizsgálatot. A disztributív vezérlés miatt az ilyen esetben használatos klasszikus visszalépéses elemzés hatékonyan már nem alkalmazható.

Konkurens rendszerek szemantikájának formális megadására és tulajdonságaik bizonyítására a *folyamatalgebrák* tűnnek a legalkalmasabbaknak. A folyamatalgebra folyamatok matematikai modellje, ahol folyamatok kommunikálnak folyamatokkal egy közös környezetben. A folyamatalgebra a rendszer leírására pontos eszközöket és módszereket ad.

A folyamatalgebra egyik alapeleme a *szintaxisa*, amely meghatározza az elemi kifejezésekből és operátorokból álló folyamatkifejezések jólformáltságát. A szintaxis szabályok halmaza, a szabályok felhasználásával dönthető el a kifejezések szintaktikus helyessége.

Egy szintaktikusan helyes folyamatkifejezés *szemantikájának* megadására többféle módszer alakult ki. A három leggyakrabban alkalmazott módszer a következő.

- A *műveleti szemantika* a rendszer működését *címkézett átmeneti rendszerrel* írja le. A rendszer állapotai a folyamatalgebra kifejezései, azaz maguk a folyamatok, az átmenetek folyamatok közötti relációk, és az átmeneteket a megadott műveletekkel címkézzük meg. A műveleti szemantika viszonylag közel van a folyamatok végrehajtásának „absztrakt

gépi" szintjéhez, és az implementáció matematikai leírásának tekinthető. A címkézett átmeneti rendszerekkel részletesen majd a 2.2. szakaszban foglalkozunk.

- A *denotációs szemantika* a folyamatkifejezések jelentését leképezi absztrakt matematikai modellekre (denotációkra), amelyek kifejezik, hogy a folyamat milyen műveletet hajt végre. Például a denotációs szemantika a folyamatkifejezéseket parciális rekurzív függvényekkel is reprezentálhatja. A denotációs szemantika nem foglalkozik az implementációval, a folyamatkifejezést olyan absztrakciós szinten írja le, amelyik a kifejezés lényeges jelentését adja meg.
- Az *algebrai szemantika*, vagy más néven az *axiomatikus szemantika* a matematikai logika eszköztárát használja, axiómákat és szabályokat ad egy folyamatkifejezés jelentésének megadására. A pi-kalkulus algebrai elméletét majd a 3. fejezetben elemezzük.

2.2. Címkézett átmeneti rendszerek

A mobil folyamatok szemléletes és precíz leírására több lehetőség kínálkozik, mi a címkézett átmeneti rendszernek nevezett módszert fogjuk használni erre a célra.

2.2.1. Definíció. Címkézett átmeneti rendszer:

> *Legyen S a folyamatok halmaza, L az akciók (címkék) halmaza, és legyen $T \subseteq S \times L \times S$ az átmeneti relációk halmaza. Ekkor az (S, L, T) hármast címkézett átmeneti rendszernek nevezzük.*

A *címkézett átmeneti rendszer* szokásos rövidítése LTS, a *labelled transition system* szavak kezdőbetűiből.

Az S és L megszámlálhatóan végtelen halmaz is lehet.

A továbbiakban a folyamatokat $P, Q, \ldots$ betűkkel, a címkéket $a, b, \ldots, x, y, z, \tau$ betűkkel jelöljük, és a (P, a, Q) átmeneti reláció jele

$$P \xrightarrow{a} Q\,.$$

Az elnevezésben a "címkézett" szó onnan származik, hogy a $P \longrightarrow Q$ átmenetet az a akcióval "címkéztük meg". A τ címke a folyamatok belső, kívülről nem megfigyelhető kommunikációjának a jele.

Ha $(P, a, Q) \in T$, akkor azt mondjuk, hogy a P folyamatot az a akció végrehajtása után a Q folyamat írja le, azaz a P folyamat a Q folyamattá válik.

Ha $a_1, a_2, \ldots, a_n \in L$ és $\alpha = a_1 a_2 \ldots a_n$ $(n \geq 1)$, akkor a

$$P_0 \xrightarrow{a_1} P_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} P_n$$

átmenet rövid jelölése $P_0 \xrightarrow{\alpha} P_n$. Az akciósorozatokat az $\alpha, \beta, \ldots$ betűkkel jelöljük. Megengedjük a nulla hosszúságú akciósorozatot is, ennek a jele $\xrightarrow{\varepsilon}$, és a hozzátartozó átmenet:

Megjegyezzük, hogy az (S, L, T) címkézett átmeneti rendszer hasonló egy olyan véges automatához, amelyben a folyamatok S halmazának elemei az automata állapotai, az L címkehalmaz elemei az automata szimbólumai és T az automata állapotátmeneteit adja meg. De ellentétben a véges automatákkal, a címkézett átmeneti rendszernek

- nincs kitüntetett kezdő- és végállapota,
- a folyamatok és az akciók halmaza nem feltétlenül véges,
- az akciók által elvégezhető műveletek választéka erősen korlátozott,
- egy folyamatra végrehajtható akciók vagy akciósorozatok száma nem feltétlenül véges,
- vannak kívülről nem megfigyelhető eseményei.

Tehát míg minden automata leírható LTS-sel, vannak olyan címkézett átmeneti rendszerek, amelyek nem feleltethetők meg automatáknak.

Természetesen egy LTS-nek is lehet akár több végállapota is. Legyen $Followers(P, \alpha) = \{Q \in S | P \xrightarrow{\alpha} Q\}$. Ha $Followers(P, \alpha) = \emptyset$, akkor P a rendszer egy *végállapota*.

Az LTS lehet determinisztikus vagy nemdeterminisztikus. Az átmeneti rendszert *determinisztikusnak* nevezzük, ha minden P folyamatára és minden α-ra a $P \xrightarrow{\alpha} Q$ átmenet egyértelmű, azaz $|Followers(P, \alpha)| \leq 1$. Ellenkező esetben az LTS *nemdeterminisztikus*.

A pi-kalkulusban azonban nem az automata működésének terminálása és az elfogadott vagy felismert szimbólumsorozatok érdekesek számunkra, hanem a rendszer működési tulajdonságai, a folyamatoknak az akciók által meghatározott dinamikája.

2.3. Szintaxis

A pi-kalkulusban az egyik alapvető fogalom a *név*, ami lényegében a kommunikációs csatornákat, kapukat és a csatornákon küldött neveket és adatokat azonosítja. A nevek halmazát $\mathcal{N}$-nel jelöljük. A pi-kalkulusban, ellentétben a programnyelvekben megszokott tulajdonsággal, nincs különbség „változók” és „adatok” között, mivel a kommunikációs csatornákon nem csak adatok, hanem például a csatornák nevei is átküldhetőek.

A neveket az ábécé kisbetűivel, leggyakrabban az $x, y, \ldots$ betűkkel jelöljük, és a szövegben a szóismétlések elkerülésére a „név” helyett a „csatorna”, vagy „kommunikációs csatorna” elnevezést is használjuk.

A pi-kalkulus másik alapvető fogalma a *folyamat*, a folyamatokat a $P, Q, \ldots$ betűkkel jelöljük. A mobil rendszert és a rendszer működését a folyamatokkal, a folyamatok közötti kapcsolatokkal írjuk le.

A folyamatok *akciókat* hajtanak végre, az akciókat a folyamatok elé írt *prefixek* jelölik. Ha a P folyamat prefixe π, akkor ezt $\pi\,.\,P$-vel jelöljük, ami azt jelenti, hogy π akció a P folyamat működése előtt hajtódik végre.

2.3.1. Definíció. Prefixek:

A pi-kalkulusban négy prefixet különböztetünk meg:

$$\pi ::= \overline{x}y \mid x(y) \mid \tau \mid [x = y]\pi \ .$$

Megjegyezzük, hogy a későbbiekben majd további prefixeket is be fogunk vezetni.

A prefixek jelentése a következő:

- $\overline{x}y\,.\,P$ *output prefix*
  A folyamat az x néven keresztül elküldi az y nevet, a P ettől nem változik meg, és az output végrehajtása után a P végrehajtása következik. Az $\overline{x}$ egy kimeneti kapunak tekinthető, és y a kapun kiküldött adat. Az x-et az output prefix *alanyának*, y-t a prefix *tárgyának* nevezzük.
- $x(z)\,.\,P$ *input prefix*
  A folyamat az x néven keresztül fogad egy tetszőleges w nevet, majd a P-ben levő z nevek w-re helyettesítődnek, azaz P a $P[z := w]$ folyamattá válik. Ezután a helyettesítéssel megváltozott folyamat hajtódik végre. (A helyettesítés művelet pontos leírásával majd a 2.3.2. pontban foglalkozunk.)

  A z név egy formális paraméter, azokat a P-beli helyeket jelöli, ahová

majd az inputtal kapott w értéket kell betölteni. Az x egy bemeneti kapunak is tekinthető. Az x-et az input prefix *alanyának*, z-t a prefix *tárgyának* nevezzük.

- $\tau . P$ *nem megfigyelhető művelet*
  Egy belső, azaz a P folyamaton kívülről nem megfigyelhető művelet hajtódik végre, a P nem változik, és a P végrehajtása következik. Megjegyezzük, hogy a τ művelet nem csak a folyamat prefixéből, hanem két folyamat kommunikációjából is származhat.
- $[x = y]\pi . P$ *azonosság prefix*
  Ha az x név és az y név azonos, akkor a végrehajtás a $\pi . P$ folyamattal folytatódik, ha a két név nem azonos, akkor a $\pi . P$ nem hajtódik végre, azaz a végrehajtás az azonosság vizsgálata után azonnal befejeződik. Ha a kifejezésben nincs π, akkor a kifejezés alakja $[x = y]P$.

A funkcionális paradigmából jól ismert a kifejezések *lusta* és *mohó* kiértékelése. Ehhez hasonló stratégia a pi-kalkulusban az, hogy egy input prefix esetében a $P[z := w]$ helyettesítés mikor hajtódik végre, és itt a helyettesítés időpontjától függően „késői” és „korai” végrehajtásról beszélünk. A korai input prefixet a 2.6.3. pontban adjuk majd meg, a prefixeknek megfelelő *késői szemantikát* a 2.4.2., a *korai szemantikát* a 2.6.3. pontban fogjuk elemezni.

Talán feltűnő, hogy a pi-kalkulusban van azonosság prefix, de nincs lehetőség $[x \neq y]\pi$ alakú „nem-azonosság” prefix használatára. Ennek okát majd a későbbiekben, a 2.3.2. pontban mutatjuk meg.

2.3.2. Definíció. Folyamatok:

A pi-kalkulusban a folyamatokat a következő kifejezésekkel adjuk meg:

$$P ::= \mathbf{0} \mid \pi . P \mid P + P \mid P \mid P \mid (\nu x) P \mid !P \ .$$

A definícióban levő folyamatkifejezések a következőket jelentik:

- $\mathbf{0}$ *nem működő folyamat*
  A „vastag nulla” folyamat inaktív, a végrehajtás leállt, befejezte a működését.
- $\pi . P$ *prefixes folyamat*
  A π akcióval leírt művelet elvégzése után a végrehajtás a P-vel folytatódik.
- $P_1 + P_2$ *összegkifejezés, vagy-művelet*
  A két folyamat közül csak az egyik kerül végrehajtásra, azaz ha a

P_1 végrehajtódik, akkor a P_2 nem hajtódik végre, leáll és befejezi működését, ha a P_2 hajtódik végre, akkor a P_1 fejezi be a működését.

- $P_1 \mid P_2$ *kompozíció, párhuzamos végrehajtás*
  A P_1 és P_2 egymástól függetlenül végrehajtódik, és ha egy közös névre az egyik folyamat egy input, a másik folyamat egy output műveletet hajt végre, lehetőség van a közös néven keresztül a két folyamat közötti kommunikációra.
- $(\boldsymbol{\nu} x)\, P$ *korlátozás*
  Az x név hatásköre a P folyamatra van korlátozva. Az x névhez más folyamat nem férhet hozzá, az x név csak a P-n belülről érhető el és csak a P-n belüli folyamatok használhatják.

  Egy korlátozott név hatásköre azonban – szemben más rendszerekkel – egy folyamat végrehajtásakor megváltozhat. Ezt a fontos tulajdonságot majd a későbbiekben megvizsgáljuk (lásd 2.3.4. pont), de már most megemlítjük, hogy ebből a tulajdonságból a pi-kalkulusnak sok előnye származik (2.4.6. példa).

  Mint a könyv elején a *Jelölések* pontban már írtuk, a korlátozást jelölő görög ν betűt ebben a könyvben félkövér (bold) vastagsággal írjuk.

  Felhívjuk a figyelmet arra, hogy a $(\boldsymbol{\nu} x)\, P$ korlátozásra más szakirodalomban gyakran a $\boldsymbol{\nu} x\,.\,P$, $\boldsymbol{\nu} x\, P$, $(x)\, P$, $\mathsf{new}\, x\, P$ jelölést is használják.
- $!P$ *ismétlés, replikáció*
  Ez a kifejezés végtelen sok P folyamat párhuzamos végrehajtását jelöli, azaz $!P = P \mid !P$. Ez a művelet használható nemvéges folyamatok működésének leírására.

A folyamatokra vonatkozó műveleteket, beleértve a prefixeket is, közös néven *operátoroknak* nevezzük.

Ha szükséges, a folyamatkifejezés egyértelműsége érdekében a kifejezésbe zárójelpárok is írhatók, de a sok zárójel elkerülésére a következő precedenciaszabályt alkalmazzuk:

prefix, korlátozás > *kompozíció* > *összeg* .

A folyamatkifejezések utolsó **0** elemét elhagyhatjuk, és ha az output vagy input prefix esetén a küldött vagy fogadott információ nem lényeges, elegendő csak az output vagy input csatorna nevét megadni. Ha egy folyamatra több név korlátozása is vonatkozik, akkor ezek a nevek egyetlen $\boldsymbol{\nu}$ jel után is felsorolhatók.

2.3.3. Példa. *(Példák folyamatkifejezésekre)*

$x(z) \,.\, \overline{y}z \,.\, P$,

$x(z) \,.\, \overline{z}y \,.\, P$,

$x(z) \,.\, [z = y]\overline{z}w \,.\, P$,

$x(z) \,.\, y(w) \,.\, [z = w]\overline{v}u \,.\, P$,

$x(z) \,.\, \overline{z}y \,.\, P + \overline{w}v \,.\, Q$,

$(x(z) \,.\, \overline{z}y \,.\, P_1 + \overline{w}v \,.\, P_2) \mid \overline{x}u \,.\, P_3$,

$(\boldsymbol{\nu} x)(x(z) \,.\, \overline{z}y \,.\, P_1 + \overline{w}v \,.\, P_2) \mid \overline{x}u \,.\, P_3)$,

$!(x(z) \,.\, \overline{y}z \,.\, P)$. □

2.3.4. Példa. *(Rövid jelölések)*

A **0** elhagyható:

$x(z) \,.\, \mathbf{0} \equiv x(z)$,

$\overline{x}y \,.\, \mathbf{0} \equiv \overline{x}y$.

$(\boldsymbol{\nu} x_1)(\boldsymbol{\nu} x_2)\, P \equiv (\boldsymbol{\nu} x_1 x_2)\, P$.

A küldött vagy fogadott információ nem lényeges:

$\overline{x}y \,.\, \mathbf{0} \mid z(w) \,.\, \mathbf{0} \equiv \overline{x} \mid z$,

$\overline{x} \mid y + x \mid \overline{y}$. □

Az output és az input művelet nem egy adott névhez rögzített tulajdonság, egy kifejezésben ugyanazt a nevet használhatjuk az egyik helyen output művelettel, egy másik helyen input művelettel, annak megfelelően, hogy az adott nevű csatornán milyen irányban halad az információ.

2.3.5. Példa. *(Output és input egy néven)*

Az $\overline{x}y \mid x(z)$ kompozíció első folyamatában az x néven kiküldjük az y nevet, a második folyamatban pedig ugyanezen a néven olvasunk. □

2.3.1. Szabad és kötött nevek

A pi-kalkulusban a folyamatkifejezésekben levő nevek használatának korlátozását *kötésnek* nevezzük. Ez azt jelenti, hogy a kötött nevet csak az adott folyamatban használhatjuk, azaz a név a folyamat „saját” neve, a név a folyamaton kívülről nem érhető el. Ezt a nevet a *kötés nevének*, a folyamatot

a *kötés törzsének* vagy a *kötés hatáskörének* nevezzük.

A pi-kalkulusban két lehetőség van a folyamatkifejezésben levő nevek használatának korlátozására. Az egyik ilyen művelet nyilvánvalóan a korlátozás művelet, hiszen a $(\boldsymbol{\nu} y)\, P$ kifejezésben az y név hatásköre a P-re van korlátozva. A másik egy implicit kötés, ez az $x(y)$ input prefix, ami az $x(y)\,.\,P$ kifejezés esetén az y nevet köti meg a P folyamatban.

2.3.6. Definíció. Szabad és kötött nevek:

> *Az y név kötött P-ben a $(\boldsymbol{\nu} y)\, P$ és az $x(y)\,.\,P$ kifejezések esetén. Egy név egy folyamatban szabad, ha nem kötött.*

Tehát az $x(y)$ prefix csak az y nevet köti, az x-t nem, és az $\overline{x}y$ prefix a benne szereplő egyik nevet sem. Nyilvánvaló, hogy a τ és a $[x = y]$ prefixek sem kötnek neveket.

A P folyamat neveit $n(P)$-vel, a kötött neveinek halmazát $bn(P)$-vel, a szabad neveinek halmazát $fn(P)$-vel jelöljük. A kifejezésekre vonatkozó kötéseket a 2.1. táblázatban foglaltuk össze.

	bn	fn
$\overline{x}y\,.\,P$	$bn(P)$	$\{x, y\} \cup fn(P)$
$x(y)\,.\,P$	$\{y\} \cup bn(P)$	$\{x\} \cup fn(P)$
$\tau\,.\,P$	$bn(P)$	$fn(P)$
$[x = y]\pi\,.\,P$	$bn(\pi\,.\,P)$	$fn(\pi\,.\,P)$
$\mathbf{0}$	$\emptyset$	$\emptyset$
$P_1 + P_2$	$bn(P_1) \cup bn(P_2)$	$fn(P_1) \cup fn(P_2)$
$P_1 \mid P_2$	$bn(P_1) \cup bn(P_2)$	$fn(P_1) \cup fn(P_2)$
$(\boldsymbol{\nu} x)\, P$	$\{x\} \cup bn(P)$	$fn(P) \setminus \{x\}$
$!P$	$bn(P)$	$fn(P)$

2.1. táblázat. Szabad és kötött nevek

2.3.7. Példa. *(Szabad és kötött nevek)*

$P \equiv (\overline{z}y + \overline{w}v) \mid \overline{x}u$,

$bn(P) = \emptyset$,

$fn(P) = \{z, y, w, v, x, u\}$.

$Q \equiv (\nu x)((x(z) . \overline{z}y + \overline{w}v) \mid (\nu u)\overline{x}u)$,

$bn(Q) = \{x, z, u\}$,

$fn(Q) = \{y, w, v\}$. □

2.3.2. Helyettesítés

A folyamatkifejezésekben a nevek szabadon választhatók meg, és előfordulhat, hogy egy kifejezésben az egyik nevet egy másik névre akarjuk kicserélni. Ezt a műveletet *helyettesítésnek* nevezzük. A helyettesítés mindig csak nevet névre helyettesít, kifejezésnek vagy névnek kifejezéssel való helyettesítésére nincs lehetőség.

A pi-kalkulusban a helyettesítés szokásos jele a σ, és $P\sigma$ jelzi, hogy a helyettesítés a P folyamatra vonatkozik. Ha a σ helyettesítés csak egy névre vonatkozik, például a P-ben az x nevet y-ra helyettesítjük, akkor ezt a $P[x := y]$ jelöléssel is leírhatjuk.

Megjegyezzük, hogy a helyettesítésre gyakran a σP-t, egy név helyettesítésére a $P\{y/x\}$ vagy az $\{y/x\}P$ jelölést is használják.

Ha $\sigma = [x_1 := y_1], [x_2 := y_2], \ldots, [x_n := y_n]$ (n > 0), akkor $P\sigma$ lényegében a $P([x_1 := y_1], [x_2 := y_2], \ldots, [x_n := y_n])$ szimultán helyettesítések elvégzésének rövid jelölése, ezért a továbbiakban általában csak egy név helyettesítésével foglakozunk.

A helyettesítés egy szintaktikus átalakítás, ezzel indokolhatjuk a helyettesítésre előírt általános tulajdonságokat, amelyeket a következő definícióban adunk meg.

2.3.8. Definíció. Helyettesítés:

> *A $P[x := y]$ helyettesítés esetén csak a P kifejezésben levő szabad x nevek helyettesíthetők az y névre úgy, hogy az eddig szabad nevek szabadok, a kötött nevek kötöttek maradjanak.*

A helyettesítés egyszerű névcserét jelent, problémát, figyelmet csak a nevek kötése okoz. Mivel kötést csak az input és a korlátozás prefixek idéznek elő, az ezekre vonatkozó helyettesítéseket külön megadjuk, a kötések neveinek helyettesítésével majd a következő szakaszban foglalkozunk.

Először vizsgáljuk meg a kötések törzseinek helyettesítéseit,

$$u(x)\,.\,(P[y := z]) \equiv \begin{cases} u(x)\,.\,P, & \text{ha } x \equiv y\,, \\ u(x)\,.\,P, & \text{ha } x \equiv z\,, \\ u(x)\,.\,P', & \text{egyébként, ahol } P' \equiv P[y := z]\,. \end{cases}$$

A definíció első alternatívája szerint tehát a kötés x neve a P-ben nem helyettesíthető egy másik névre, hiszen ezzel éppen a P-ben levő kötéseket szüntetnénk meg. A második alternatíva azt mondja ki, hogy a P egy y neve nem cserélhető le a kötés x nevére, hiszen a névcsere után a P-ben eddig szabad y nevek kötötté válnának. Ha tehát sem az y, sem a z nem azonos x-szel, akkor a helyettesítés végrehajtható, azaz a P szabad y nevei helyettesíthetők z-vel.

Teljesen hasonló tulajdonság és hasonló indoklás érvényes a korlátozás prefix esetén is.

$$(\boldsymbol{\nu} x)\,(P[y := z]) \equiv \begin{cases} (\boldsymbol{\nu} x)\,P, & \text{ha } x \equiv y\,, \\ (\boldsymbol{\nu} x)\,P, & \text{ha } x \equiv z\,, \\ (\boldsymbol{\nu} x)\,P', & \text{egyébként, ahol } P' \equiv P[y := z]\,. \end{cases}$$

Összefoglalva, a helyettesítés a következő definícióval adható meg.

2.3.9. Definíció. Helyettesítés:

A helyettesítést a pi-kalkulus prefixeire és kifejezéseire a következőképpen adjuk meg:

- $\mathbf{0}\sigma \equiv \mathbf{0}$,
- $(\pi\,.\,P)\sigma \equiv \pi\sigma\,.\,P\sigma$,
- $(P_1 + P_2)\sigma \equiv P_1\sigma + P_2\sigma$,
- $(P_1 \mid P_2)\sigma \equiv P_1\sigma \mid P_2\sigma$,
- $((\boldsymbol{\nu} x)\,P)\sigma \equiv (\boldsymbol{\nu} x)\,P\sigma$,
- $(!P)\sigma \equiv !(P\sigma)$.

Már ismerve a helyettesítés műveletét, nézzük meg, hogy mi történne az $[x \neq y]$ prefix bevezetése esetén. Legyen $[x \neq y]\,.\,P$, tehát ha $x \neq y$, akkor a P-t, $x = y$ esetén a $\mathbf{0}$ folyamatot kapjuk eredményül. Ha erre az $[x \neq y]\,.\,P$ folyamatra végrehajtunk egy $[y := x]$ helyettesítést, akkor a prefix $x \neq x$ lesz, és így a folyamat biztosan a $\mathbf{0}$-vá válik. Tehát a helyettesítés, azaz egy szintaktikus átalakítás megváltoztathatja a kifejezés jelentését, és ez nem engedhető meg.

2.3.10. Példa. *(Példák a helyettesítésre)*

$$
\begin{aligned}
(\overline{x}y\,.\,P)[x := y] &\equiv \overline{y}y\,.\,P[x := y]\,,\\
(x(z)\,.\,P)[x := y] &\equiv y(z)\,.\,P[x := y]\,,\\
(x(z)\,.\,P)[z := y] &\equiv x(z)\,.\,P\,,\\
((\boldsymbol{\nu}x)\,P)[x := y] &\equiv ((\boldsymbol{\nu}x)\,P)\,,\\
((\boldsymbol{\nu}x)\,P)[z := x] &\equiv (\boldsymbol{\nu}x)\,P[z := x] \;\equiv\; (\boldsymbol{\nu}x)\,P\,. \qquad \square
\end{aligned}
$$

2.3.3. Az α-konverzió

Ha a kötés nevét szeretnénk helyettesíteni egy másik névvel, a névcserét a kötés törzsében is végre kell hajtani, de következetesen, azaz ügyelve arra, hogy a névcsere csak akkor hajtható végre, ha az eddigi szabad nevek nem válnak kötötté.

Ezt a műveletet α-konverziónak nevezzük, a konverziót az $\leftrightarrow_\alpha$ jellel is jelölhetjük. Az α-konverzió szintaktikus átalakítás, az α-konverzióval egymásba alakítható folyamatok azonosaknak, megegyezőeknek tekinthetőek, ezért ha az α-konverziót nem hangsúlyozzuk, a kifejezések közé a $\equiv$ jelet is írhatjuk.

2.3.11. Definíció. Az α-konverzió:

Ha P-ben nincs z szabad név, azaz $z \notin fn(P)$, akkor

- $x(y)\,.\,P \;\leftrightarrow_\alpha\; x(z)\,.\,P[y := z]$,
- $(\boldsymbol{\nu}y)\,P \;\leftrightarrow_\alpha\; (\boldsymbol{\nu}z)\,P[y := z]$.

Az α-konverzió feltétele az, hogy az input vagy a korlátozás új kötő neve a kifejezés törzsében ne szerepeljen szabadon. Mivel egy folyamatkifejezésben a nevek szabadon választhatók meg, ez a feltétel egy egyszerű vizsgálattal könnyen teljesíthető.

Az α-konverzió $z \notin fn(P)$ feltételére $z \notin fn(P) \subseteq n(P)$, így az a gyengébb, de a gyakorlatban könnyebben vizsgálható feltétel is megfelelő, hogy a korlátozás új kötő neve egyáltalán ne szerepeljen a kifejezés törzsében, és mivel ebben az esetben a helyettesítés és a korlátozás egymástól függetlenek, a helyettesítés ezzel a feltétellel is biztonságosan elvégezhető.

2.3.12. Példa. *(α-konverziók)*

$x(y)\,.\,\overline{y}z \;\leftrightarrow_\alpha\; x(u)\,.\,\overline{u}z$,

$(\boldsymbol{\nu}y)\overline{y}z \;\leftrightarrow_\alpha\; (\boldsymbol{\nu}u)\overline{u}z$,

$(\nu x)(\overline{x}z\,.\,z(y)\,.\,\overline{y}w) \;\leftrightarrow_\alpha\; (\nu y)(\overline{y}z\,.\,z(y)\,.\,\overline{y}w)$,

$(\nu y)(\overline{y}z\,.\,z(y)\,.\,\overline{y}w) \;\leftrightarrow_\alpha\; (\nu u)(\overline{u}z\,.\,z(y)\,.\,\overline{y}w)$. □

2.3.4. Szerkezeti kongruencia

Mint már utaltunk rá, a pi-kalkulus egyik feladata az azonos, megegyező műveleteket végző folyamatok, azaz az azonos szemantikájú folyamatok felismerése. A pi-kalkulusban ezeket a folyamatokat *kongruens* folyamatoknak nevezzük.

Először értelmezzük a *kontextus* fogalmát a folyamatokra.

2.3.13. Definíció. Degenerált és nem degenerált* 0*:

> *Egy P folyamatkifejezésben a* **0***-t degeneráltnak nevezzük, ha a P kifejezés $Q_1 + Q_2$ alakú és $Q_1 \equiv \mathbf{0}$ vagy $Q_2 \equiv \mathbf{0}$. Egyébként a* **0** *nem degenerált.*

2.3.14. Definíció. Kontextus:

> *Egy folyamatkifejezés kontextusát úgy kapjuk meg, hogy a degenerált* **0** *kifejezéseit elhagyjuk, a nem degenerált* **0** *kifejezéseit a* [ ] *„lyukakra” cseréljük ki.*

2.3.15. Példa. *(Kontextusok)*

- A $\mathbf{0} + \overline{x}y\,.\,\mathbf{0}$ kifejezés kontextusa $\overline{x}y\,.\,[\;]$, mivel az első **0** degenerált, de a második nem.
- Az $\overline{x}y \mid z(w)$ kontextusának meghatározásához egészítsük ki a kifejezést a **0** kifejezésekkel:

  $\overline{x}y \mid z(w) \equiv \overline{x}y\,.\,\mathbf{0} \mid z(w)\,.\,\mathbf{0}$,

  mivel egyik **0** sem degenerált, a kontextus:

  $\overline{x}y\,.\,[\;] \mid z(w)\,.\,[\;]$.
- A $(\nu x)(\mathbf{0} \mid x(y)\,.\,\mathbf{0})$ kontextusa $(\nu x)([\;] \mid x(y)\,.\,[\;])$. □

Ha C egy kontextus és P egy folyamat, akkor a C-beli lyukak helyére beírhatjuk a P-t, és az így kapott kifejezést $C[P]$-vel jelöljük. Ez a helyettesítés „szöveghelyettesítés”, a nevek kötéseit nem veszi figyelembe, a P szabad nevei $C[P]$-ben kötötté válhatnak.

2.3.16. Példa. *(Nevek a kontextusban)*

Ha az előző példa utolsó kontextusát C_P-vel jelöljük, akkor

$C_P[\overline{x}z] \equiv (\boldsymbol{\nu} x)(\overline{x}z \mid x(y) \,.\, \overline{x}z)$,

$C_P[y(x)] \equiv (\boldsymbol{\nu} x)(y(x) \mid x(y) \,.\, y(x))$. □

Ezek után definiálhatjuk a kongruencia fogalmát.

2.3.17. Definíció. Kongruencia:

Legyen $\mathcal{R}$ a folyamatokon értelmezett ekvivalencia reláció. Az $\mathcal{R}$-t kongruenciának nevezzük, ha $(P, Q) \in \mathcal{R}$ esetén minden C kontextusra $(C[P], C[Q]) \in \mathcal{R}$.

A kongruencia fontos szerepet játszik két folyamat működési ekvivalenciájának vizsgálatában. Először a szerkezeti kongruenciával foglalkozunk, az általános értelemben vett kongruenciákat majd a biszimulációval kapcsolatban tárgyaljuk.

A *szerkezeti kongruencia*, mint a neve is jelzi, a folyamatok szerkezeti felépítéséből állapítja meg a folyamatok szemantikájának azonosságát. A szerkezeti kongruencia a folyamatok kifejezéseinek felépítéséből adódó, „azonnal látható”, „nyilvánvaló” szemantikai azonossági tulajdonságokat írja le.

A szerkezeti kongruencia jele a $\equiv$. A szerkezeti kongruenciát szabályokkal adjuk meg.

2.3.18. Példa. *(α-konverzió)*

Az $x(u) \,.\, \overline{z}u$ és $x(v) \,.\, \overline{z}v$ folyamatok között csupán az u és v „kommunikációs” csatornák nevében van különbség, mindkét folyamat ugyanazt a műveletet végzi és ugyanazt az eredményt adja, az x csatornán vett jelet kiküldi a z csatornára. □

A példából is látható, hogy az α-konverziónak nevezett átnevezés is a szerkezeti kongruencia egyik fajtája:

$$P \equiv Q, \quad ha\ P \leftrightarrow_\alpha Q \,.$$

A szerkezeti kongruencia további szabályai hasonló intuitív meggondolással állapíthatóak meg.

A folyamatok a *párhuzamos végrehajtásra* és a *vagy*-műveletre kommutatív, asszociatív és egységelemes halmazt (azaz Abel-monoidot[1]) alkotnak,

[1] Az S halmaz monoid, ha elemeire értelmezve van egy egységelemes és asszociatív $S \times S \rightarrow$

ahol az egységelem a **0**.

$$\begin{aligned} P \mid Q &\equiv Q \mid P \\ P \mid (Q \mid R) &\equiv (P \mid Q) \mid R \\ P \mid \mathbf{0} &\equiv P \\ P + Q &\equiv Q + P \\ P + (Q + R) &\equiv (P + Q) + R \\ P + \mathbf{0} &\equiv P \end{aligned}$$

Nyilvánvalóan helyes a következő szabály is:

$$[x = x]\,\pi\,.\,P \equiv \pi\,.\,P\ ,$$

és az ismétléses kifejezésekre:

$$!P \equiv P \mid !P\ .$$

Egy P folyamatra előírt korlátozások felsorolási sorrendje lényegtelen:

$$(\nu x)(\nu y)\,P \equiv (\nu y)(\nu x)\,P\ .$$

A *korlátozás* műveletekre vonatkozó következő szabályokat *hatáskör kiterjesztés*, vagy a szabályt a másik irányban alkalmazva, *hatáskör szűkítés* szabályoknak is nevezhetjük:

$$\begin{aligned} (\nu x)\,P &\equiv P,\ \textit{ha } x \notin \textit{fn}(P) \\ (\nu x)\,P \mid Q &\equiv (\nu x)\,(P \mid Q),\ \textit{ha } x \notin \textit{fn}(Q) \\ (\nu x)\,P + Q &\equiv (\nu x)\,(P + Q),\ \textit{ha } x \notin \textit{fn}(Q) \\ (\nu x)\,\mathbf{0} &\equiv \mathbf{0} \end{aligned}$$

A hatáskör kiterjesztés szabálya fontos szerepet játszik majd két párhuzamosan futó folyamat kommunikációjában akkor, ha a korlátozás csak az egyik folyamatra vonatkozik és a másik folyamatra a szabály alkalmazásának feltétele teljesül. Ekkor a korlátozás kiterjeszthető mindkét folyamatra, és a kommunikáció megvalósulhat (2.4.6. példa).

Felhívjuk a figyelmet arra, hogy nincs például $(\nu x)\,(P \mid Q) \equiv (\nu x)\,P \mid (\nu x)\,Q$ szabály, hiszen a jobboldalon levő két νx kötésben az α-konverzió miatt az x nevek nem feltétlenül jelölik ugyanazt a nevet, míg a

S bináris művelet. A kommutatív monoidot Abel-monoidnak nevezzük.

baloldalon természetesen ugyanarról az x-ről van szó. Ugyanakkor a $(\nu x)\,(P + Q) \equiv (\nu x)\,P + (\nu x)\,Q$ szabály bevezethető, mivel P és Q közül biztosan csak az egyik fog végrehajtódni.

Nem adtunk meg prefixekre és prefixes folyamatokra vonatkozó hatáskör kiterjesztés szabályokat sem, de az ilyen átalakításokat majd a későbbiekben tárgyaljuk.

A szerkezeti kongruencia szabályait a 2.2. táblázatban foglaljuk össze.

$$
\begin{array}{rcl}
P & \equiv & Q, \quad ha\ P \leftrightarrow_\alpha Q \\
P \mid Q & \equiv & Q \mid P \\
P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R \\
P \mid \mathbf{0} & \equiv & P \\
P + Q & \equiv & Q + P \\
P + (Q + R) & \equiv & (P + Q) + R \\
P + \mathbf{0} & \equiv & P \\
[x = x]\,\pi\,.\,P & \equiv & \pi\,.\,P \\
!P & \equiv & P \mid !P \\
(\nu x)(\nu y)\,P & \equiv & (\nu y)(\nu x)\,P \\
(\nu x)\,P & \equiv & P,\ ha\ x \notin fn(P) \\
(\nu x)\,P \mid Q & \equiv & (\nu x)\,(P \mid Q),\ ha\ x \notin fn(Q) \\
(\nu x)\,P + Q & \equiv & (\nu x)\,(P + Q),\ ha\ x \notin fn(Q) \\
(\nu x)\,\mathbf{0} & \equiv & \mathbf{0}
\end{array}
$$

2.2. táblázat. A szerkezeti kongruencia szabályai

Látható, hogy a folyamatkifejezések a szerkezeti kongruencia szabályaival átalakíthatóak azonos műveletet végző, de a leírásukban különböző kifejezésekre. Ezért most definiáljuk a *folyamatkifejezések standard formáját*.

2.3.19. Definíció. Folyamatkifejezés standard formája:

Azt mondjuk, hogy a folyamatkifejezés standard formában van, ha

$$(\nu x_1 x_2 \ldots x_i)\,(P_1 \mid P_2 \mid \ldots \mid P_n \mid !Q_1 \mid \ldots \mid !Q_m)$$

alakú, ahol P_j $(1 \leq j \leq n)$ nem üres összegkifejezések és Q_j $(1 \leq j \leq n)$ standard formában vannak. Ha $i = 0$, akkor a kifejezésben nincs korlátozás, és ha $n = m = 0$, akkor a kifejezés törzse a $\mathbf{0}$.

A standard formában tehát korlátozás csak a kifejezés bal oldalán lehet, és a standard forma törzsében az összegkifejezések párhuzamos műveletekkel vannak összekapcsolva.

Már ismerve a szerkezeti kongruencia szabályait, kimondhatjuk a következő tételt.

2.3.20. Tétel. (Standard forma)

Minden folyamatkifejezés szerkezeti kongruens egy standard formájú folyamatkifejezéssel.

Tehát minden folyamatkifejezés standard formára hozható, de nyilvánvaló, hogy egy kifejezés standard formája (például az összeg és párhuzamos műveletek kommutativitása miatt) nem egyértelmű.

2.4. Műveleti szemantika

A pi-kalkulus lehetőséget ad a folyamatok működésének a leírására. Mint a 2.2. szakaszban láttuk, a leírásra a címkézett átmeneti rendszerek egy szemléletes és könnyen kezelhető eszközt jelentenek. Sokkal absztraktabb módszer az, amikor egy folyamat működését egy hasonló módon működő, egy vele ekvivalens folyamat működésével adjuk meg. Az ekvivalencia meghatározása természetesen egy „külső megfigyelő” vizsgálatán alapul, úgy hogy a folyamatok működésének elemzésekor a folyamatok kívülről megfigyelhető eseményeit hasonlítjuk össze.

Ennek a vizsgálatnak két módszere van:

- A folyamatok műveleti tulajdonságait vizsgáljuk a *műveleti szemantika* megadásával, és ez vezet majd el a biszimulációhoz, azaz a kölcsönös szimulációhoz.

- A folyamatok között egyenlőségeket adunk meg axiómákkal, szabályokkal, és ezek használatával, algebrai módszerekkel határozzuk meg két folyamat ekvivalenciáját.

A műveleti szemantikával ebben a fejezetben, a szimulációval és biszimulációval a 2.6. fejezetben foglalkozunk, és az *algebrai elméletet* egy későbbi fejezetben írjuk majd le.

2.4.1. A szemantikát leíró szabályok

A pi-kalkulus műveleti szemantikáját a *szabályokkal* adjuk meg, a szabályok alakja

$$\frac{I_1, \ \ldots, \ I_n}{I}, \text{[Név]}$$

ahol $I_1, \ \ldots, \ I_n$ a *feltételek* és I a feltételekből származtatott *következmény*. Egy szabály azt mondja ki, hogy ha mindegyik feltétel teljesül, akkor a következmény is helyes. Ha a feltételek halmaza üres, akkor a szabályt *axiómának* nevezzük. A Név a szabály elnevezésére utal.

A szabályokat *levezetések* készítésére használjuk, úgy, hogy a szabályokból egy *levezetési fát* építünk. A szabályokat egymáshoz kapcsoljuk, egy S1 szabály következménye az S2 szabály feltételéhez kapcsolható, ha a bennük levő állítások azonosak.

Azt mondjuk, hogy egy állítás *adott feltételek* mellett *érvényes*, ha az állítás egy levezetési fa gyökérpontjában levő következmény, és az adott feltételek állításai a fa leveleiben szerepelnek.

A szabályokban levő feltételek és a következmény leírására a *címkézett átmeneti rendszerek*

$$P \xrightarrow{\alpha} Q$$

jelölését használjuk, amely szerint a P folyamat az α művelet hatására a Q folyamattá alakul át.

2.4.2. A műveleti szemantika szabályai

A műveleti szemantika szabályait a 2.3. táblázatban adjuk meg.

A műveleti szemantika szabályai viszonylag egyszerűek, könnyen érthetőek, magyarázatra talán elsőnek a Com szabály szorul, amit prefixes folyamatkifejezésekkel az

$$\overline{x}y \,.\, P \mid x(z) \,.\, Q \xrightarrow{\tau} P \mid Q[z := y]$$

$$\frac{P' \equiv P, \quad P \xrightarrow{\alpha} Q, \quad Q \equiv Q'}{P' \xrightarrow{\alpha} Q'} \quad [\textsc{Struct}]$$

$$\frac{}{\overline{x}y\,.\,P \xrightarrow{\overline{x}y} P} \quad [\textsc{Output}]$$

$$\frac{}{x(y)\,.\,P \xrightarrow{x(y)} P} \quad [\textsc{Input}]$$

$$\frac{}{\tau\,.\,P \xrightarrow{\tau} P} \quad [\textsc{Tau}]$$

$$\frac{\alpha\,.\,P \xrightarrow{\alpha} P}{[x = x]\alpha\,.\,P \xrightarrow{\alpha} P} \quad [\textsc{Match}]$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \quad [\textsc{Sum}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad bn(\alpha) \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad [\textsc{Par}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad x \notin n(\alpha)}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'} \quad [\textsc{Res}]$$

$$\frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \quad [\textsc{Rep}]$$

$$\frac{P \xrightarrow{\overline{x}y} P', \quad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'[z := y]} \quad [\textsc{Com}]$$

2.3. táblázat. A műveleti szemantika szabályai

alakban írhatunk fel. A szabály a P és Q folyamatok közötti *kommunikációt* adja meg. A P folyamat az x néven kiküldi az y nevet, a Q pedig az x néven veszi ezt az információt. Látható, hogy kommunikáció csak akkor valósul meg, ha a folyamatokban a kommunikáló nevek azonosak. A P folyamat az információ elküldése után változatlan, de a Q folyamat az x néven

olvasott y információt a szabad z neveibe helyezi. A z tehát a Q egy formális paramétere, és a kommunikáció eredményeképpen a Q-beli z formális paraméterek a kapott y-nal, mint aktuális paraméterrel helyettesítődnek.

A műveleti szemantika szabályai azonban nagyon speciálisak, gyakran előfordulhat, hogy a kifejezésekre a szabályok nem alkalmazhatóak, például a szabályokban levő feltételek, vagy a kifejezésekben levő folyamatok sorrendje miatt. A Par és Com szabály művelete csak akkor végezhető el, ha a szerkezeti kongruencia átalakításait alkalmazva a kifejezést a kívánt alakra hozzuk. Ezt biztosítja a Struct szabály, amely azt mondja ki, hogy a szerkezeti kongruencia szabályaival átalakított kifejezések a szemantika szempontjából azonosak.

2.4.1. Példa. *(Kommunikáció csak a kongruens átalakítás után)*

$(\overline{x}y\,.\,P \mid \mathbf{0}) \mid x(z)\,.\,Q \not\longrightarrow$,

de a kifejezést átalakítva

$(\overline{x}y\,.\,P \mid \mathbf{0}) \mid x(z)\,.\,Q \equiv$
$\overline{x}y\,.\,P \mid x(z)\,.\,Q \xrightarrow{\tau}$
$P \mid Q[z := y]$. □

2.4.2. Példa. *(A folyamatok közötti kapcsolatok)*

A Bevezetőben az 1.0.1. példában levő P, Q és R folyamatok és a kommunikációjuk a következő kifejezéssel adható meg:

$\overline{y}x\,.\,P' \mid y(z)\,.\,x(z)\,.\,z(u)\,.\,R' \mid \overline{x}w\,.\,Q$.

Először az y néven, majd az x néven kommunikáció fog történni:

$\underline{\overline{y}x}\,.\,P' \mid \underline{y(z)\,.\,x(z)\,.\,z(u)\,.\,R'} \mid \overline{x}w\,.\,Q \xrightarrow{\tau}$
$P' \mid (x(z)\,.\,z(u)\,.\,R')[z := x]) \mid \overline{x}w\,.\,Q \equiv$
$P' \mid \underline{x(z)\,.\,z(u)\,.\,R'} \mid \underline{\overline{x}w}\,.\,Q$. □

2.4.3. Példa. *(Felügyelő program)*

Az R folyamat végrehajtását egy Q felügyelő program indítja el. Legyen

$P \equiv \overline{x}z \mid z\,.\,R$,

és

$Q \equiv x(y)\,.\,\overline{y}$.

A P az x néven kiadott z névvel jelzi, hogy az R folyamat végrehajtásra kész. A Q felügyelő program, ha úgy dönt, hogy az R indulhat, veszi az x néven a jelet, és ezt a nevet, azaz a z-t kiküldi a z néven. A P ezt olvassa, és elindítja az R folyamatot.

$$
\begin{aligned}
&P \mid Q \equiv \\
&(\overline{x}z \mid z\,.\,R) \mid x(y)\,.\,\overline{y} \equiv \\
&(\overline{x}z \mid x(y)\,.\,\overline{y}) \mid z\,.\,R \xrightarrow{\tau} \\
&\overline{y}[y := z] \mid z\,.\,R \equiv \\
&\overline{z} \mid z\,.\,R \xrightarrow{\tau} \\
&R\,.
\end{aligned}
$$

□

2.4.4. Példa. *(Felesleges szabályok)*

A szerkezeti kongruencia szabályait figyelembe véve a szemantikát leíró táblázat is rövid lett, mivel például az összeg művelet kommutativitása miatt elegendő a Sum szabályban a $P + Q \xrightarrow{\alpha} P'$ következtetést megadni. Nem kell foglalkozni a $Q + P \xrightarrow{\alpha} P'$ következtetésű szabállyal, mivel ez a Struct és Sum szabályokból levezethető:

$$
\dfrac{\dfrac{\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}\ [\textsc{Sum}]}{Q + P \equiv P + Q, \quad P + Q \xrightarrow{\alpha} P', \quad P' \equiv P'}}{Q + P \xrightarrow{\alpha} P'}\ [\textsc{Struct}]
$$

Ehhez hasonlóan szintén nem kell a szabályok között megadni a

$$Q \mid P \xrightarrow{\alpha} \ldots, \quad Q \mid P \xrightarrow{\tau} \ldots$$

következtetésű Par és Com jellegű szabályokat sem. □

A korlátozások alkalmazására a szerkezeti kongruencia szabályai között sok lehetőséget láttunk. Felhívjuk a figyelmet arra, hogy a Res szabály is a korlátozás bevezetéséről szól. A szabály szerint, ha $x \notin \alpha$, ahol $P \xrightarrow{\alpha} P'$, akkor a P korlátozható x-szel, és ez a korlátozás az α átmenet után P'-re is érvényes lesz. Ez nyilvánvaló, hiszen a korlátozás x neve a α átmenetben egyáltalán nem szerepel.

2.4.5. Példa. *(A* Par *szabály)*

Most vizsgáljuk meg a Par szabály $bn(\alpha) \cap fn(Q) = \emptyset$ feltételének szükségességét. Legyen $\alpha = x(y)$, $P = x(y) \,.\, y$ és $Q = y$. Látható, hogy a feltétel nem teljesül, hiszen $bn(x(y)) \cap fn(y) = y$. Eltekintve a feltétel teljesülésétől, a Par szabály következtetését alkalmazva

$$\dfrac{\dfrac{}{x(y) \,.\, y \xrightarrow{x(y)} y}\ [\textsc{Prefix}]}{(x(y) \,.\, y) \mid y \xrightarrow{x(y)} y \mid y}\ []$$

A szabállyal kapott eredmény valóban nem helyes, mert például egy $\overline{x}z \,.\, u$ folyamattal a Com szabályt alkalmazva

$$\dfrac{\overline{x}z \,.\, u \xrightarrow{\overline{x}z} u, \quad ((x(y) \,.\, y) \mid y) \xrightarrow{x(y)} y \mid y}{((y \mid y)[y := z] \mid u)}\ [\textsc{Com}]$$

$z \mid z \mid u$ eredményt kapjuk, de a helyes eredmény nyilvánvalóan a $(z \mid y) \mid u$ lenne. A problémát a párhuzamos végrehajtásból és a kommunikációból származó névprobléma, a Par szabály feltételeinek be nem tartása okozta. □

2.4.6. Példa. *(Kommunikáció kötött és szabad nevekkel)*

Mint már korábban láttuk,

$\overline{x}y \,.\, P \mid x(z) \,.\, Q \xrightarrow{\tau} P \mid Q[z := y\,]$.

A teljes kifejezésre adott νx korlátozás nem okoz problémát:

$(\nu x)\,(\overline{x}y \,.\, P \mid x(z) \,.\, Q) \xrightarrow{\tau} (\nu x)\,(P \mid Q[z := y\,])$.

Ha a korlátozás csak az egyik, például az $\overline{x}y \,.\, P$ folyamatra vonatkozik, akkor ha $x \notin fn(Q)$, először a szerkezeti kongruencia hatáskör kiterjesztési szabályát kell alkalmazni:

$(\nu x)\,(\overline{x}y \,.\, P) \mid x(z) \,.\, Q \equiv$

$(\nu x)\,(\overline{x}y \,.\, P \mid x(z) \,.\, Q) \xrightarrow{\tau}$

$(\nu x)\,(P \mid Q[z := y\,]) \equiv$

$(\nu x)\,P \mid Q[z := y\,]$. □

2.4.3. A kötött output prefix

Mint korábban már láttuk, a $(\boldsymbol{\nu} x)\, P$ korlátozás az x név hatáskörét a P folyamatra korlátozza. Például a $(\boldsymbol{\nu} x)\, \overline{x}y\,.\,P$ kifejezésben, ahol a korlátozás az output *alanyára* vonatkozik, az x név az $\overline{x}y\,.\,P$ folyamat saját belső neve, és ezért ez a kifejezés nem kommunikálhat egyetlen $x(z)\,.\,Q$ alakú kifejezéssel sem. Az x névre kikerülő y információt csak a P használhatja, így az $\overline{x}y$ prefixnek túl sok szerepe nincs. Lényegében a $(\boldsymbol{\nu} x)\, \overline{x}y\,.\,P$ kifejezés szemantikusan a $\mathbf{0}$ kifejezésnek felel meg.

Most nézzük meg azt az esetet, amikor a korlátozás az output *tárgyára* vonatkozik. A $(\boldsymbol{\nu} y)\overline{x}y\,.\,P$ folyamat nem az x-et, hanem a kiküldött y információt korlátozza a P-re, és az y már részt vehet például egy $x(z)\,.\,Q$ folyamattal történő kommunikációban, feltéve természetesen, hogy a Q-ban nincs szabad y név. Ekkor ugyanis a szerkezeti kongruencia hatáskör kiterjesztését végző műveletének alkalmazásával az y korlátozása átvihető az $x(z)\,.\,Q$-ra is,

$$(\boldsymbol{\nu} y)\overline{x}y\,.\,P \mid x(z)\,.\,Q \;\equiv\; (\boldsymbol{\nu} y)\,(\overline{x}y\,.\,P \mid x(z)\,.\,Q)\,,$$

és így a kommunikációnak már nem lesz semmilyen akadálya.

Látható tehát, hogy a $(\boldsymbol{\nu} x)\, \overline{x}y$ és a $(\boldsymbol{\nu} y)\, \overline{x}y$ funkciójukban lényegesen különböznek, az utóbbi külön nevet is kapott.

2.4.7. Definíció. Kötött output prefix:

> *A $(\boldsymbol{\nu} y)\overline{x}y$ prefixet kötött output prefixnek nevezzük, és röviden az $\overline{x}(y)$ jelekkel jelöljük.*

Az eddig megismert output és input prefixek kötött és szabad neveit a 2.4. táblázatban adjuk meg.

Az $\overline{x}(y)$ prefixet régebbi publikációkban gyakran $\overline{x}\boldsymbol{\nu} y$-nal is jelölték.

Ha a kötött output prefixszel szemben hangsúlyozni akarjuk az $\overline{x}y$ output prefix jellemző tulajdonságát, akkor ezt a prefixet *szabad output prefixnek* nevezzük.

Az $\overline{x}(y)$ és az $\overline{x}y$ prefixek közötti különbséget a következő példában mutatjuk meg.

2.4.8. Példa. *(Kötött és szabad output prefixek)*

Legyen $P = \overline{x}(z)\,.\,\mathbf{0}$, és $Q = x(y)\,.\,(y + z)$. Ekkor

$P \mid Q =$

$\overline{x}(z)\,.\,\mathbf{0} \mid x(y)\,.\,(y + z) \equiv$

	bn	fn
$\overline{x}y\,.\,P$	$bn(P)$	$\{x, y\} \cup fn(P)$
$\overline{x}(y)\,.\,P$	$\{y\} \cup bn(P)$	$\{x\} \cup fn(P)$
$x(y)\,.\,P$	$\{y\} \cup bn(P)$	$\{x\} \cup fn(P)$

2.4. táblázat. Az output és input prefixek szabad és kötött nevei

$(\nu z)\overline{x}z\,.\,\mathbf{0} \mid x(y)\,.\,(y+z) \;\leftrightarrow_\alpha$
$(\nu w)\overline{x}w\,.\,\mathbf{0} \mid x(y)\,.\,(y+z) \equiv$
$(\nu w)(\overline{x}w\,.\,\mathbf{0} \mid x(y)\,.\,(y+z)) \xrightarrow{\tau}$
$(\nu w)(\mathbf{0} \mid w+z) \equiv$
$(\nu w)(w+z)$,

és ugyanez a szabad output prefixszel:

$\overline{x}z\,.\,\mathbf{0} \mid x(y)\,.\,(y+z) \xrightarrow{\tau}$
$\mathbf{0} \mid z+z \equiv$
$z+z$,

ami az előző eredménytől lényegesen különbözik. □

A szemantikának a 2.3. táblázatban leírt szabályait nézve látható, hogy még egy szabályt sem adtunk meg a kötött outputra. Először fejezzük ki a kötött output jelentését egy szabállyal:

$$\frac{\overline{x}y\,.\,P \xrightarrow{\overline{x}y} P, \quad x \neq y}{\overline{x}(y)\,.\,P \xrightarrow{\overline{x}(y)} P} \quad [\textsc{Open}]$$

2.5. táblázat. A kötött output szabálya

A kötött outputra vonatkozó kommunikációt levezethetjük a Com, Res és Struct szabályok használatával. Nézzük meg, hogy mi lesz az

$$\overline{x}(y)\,.\,P \mid x(z)\,.\,Q$$

eredménye.

$$\dfrac{\dfrac{\overline{x}y\,.\,P \xrightarrow{\overline{x}y} P, \quad x(z)\,.\,Q \xrightarrow{x(z)} Q}{\overline{x}y\,.\,P \mid x(z)\,.\,Q \xrightarrow{\tau} P \mid Q[z := y]}\ [\textsc{Com}]}{(\boldsymbol{\nu}y)(\overline{x}y\,.\,P \mid x(z)\,.\,Q) \xrightarrow{\tau} (\boldsymbol{\nu}y)(P \mid Q[z := y])}\ [\textsc{Res}]$$

Tegyük fel, hogy $y \notin fn(Q)$, így a szerkezeti kongruencia hatáskör szűkítés szabályát alkalmazhatjuk:

$$(\boldsymbol{\nu}y)(\overline{x}y\,.\,P \mid x(z)\,.\,Q) \equiv (\boldsymbol{\nu}y)(\overline{x}y\,.\,P) \mid x(z)\,.\,Q\ ,$$

és a kongruencia kommutativitása miatt

$$\dfrac{\begin{array}{c}(\boldsymbol{\nu}y)(\overline{x}y\,.\,P) \mid x(z)\,.\,Q \equiv (\boldsymbol{\nu}y)(\overline{x}y\,.\,P \mid x(z)\,.\,Q)\ ,\\ (\boldsymbol{\nu}y)(\overline{x}y\,.\,P) \mid x(z)\,.\,Q \xrightarrow{\tau} (\boldsymbol{\nu}y)(P \mid Q[z := y])\end{array}}{(\boldsymbol{\nu}y)(\overline{x}y\,.\,P) \mid x(z)\,.\,Q \xrightarrow{\tau} (\boldsymbol{\nu}y)(P \mid Q[z := y])}\ [\textsc{Struct}]$$

A következményre a kötött input jelölését használva

$$\overline{x}(y)\,.\,P \mid x(z)\,.\,Q \xrightarrow{\tau} (\boldsymbol{\nu}y)(P \mid Q[z := y])\ .$$

Ebből a levezetésből egy új szabályt konstruálhatunk:

$$\dfrac{\overline{x}(y)\,.\,P \xrightarrow{\overline{x}(y)} P, \quad x(z)\,.\,Q \xrightarrow{x(z)} Q}{\overline{x}(y)\,.\,P \mid x(z)\,.\,Q \xrightarrow{\tau} (\boldsymbol{\nu}y)(P \mid Q[z := y])}$$

Az y a folyamatok saját neve, ezért nézzük meg, hogy mi történik az $x(z)\,.\,Q \xrightarrow{x(z)} Q$ átmenettel konkrét $x(y)$ input esetén. Az $x(z)\,.\,Q$-ra egy α-konverziót alkalmazva:

$$x(z)\,.\,Q \equiv x(y)\,.\,Q[z := y]\ ,$$

és ebből a Prefix szabállyal

$$x(y)\,.\,Q[z := y] \xrightarrow{x(y)} Q[z := y]\ ,$$

így ezekből a Struct szabály alkalmazásával

$$\dfrac{x(z)\,.\,Q \equiv x(y)\,.\,Q[z := y], \quad x(y)\,.\,Q[z := y] \xrightarrow{x(y)} Q[z := y]}{x(z)\,.\,Q \xrightarrow{x(y)} Q[z := y]}$$

$$\frac{\overline{x}(y) \,.\, P \xrightarrow{\overline{x}(y)} P, \quad x(z) \,.\, Q \xrightarrow{x(y)} Q[z := y]}{\overline{x}(y) \,.\, P \mid x(z) \,.\, Q \xrightarrow{\tau} (\nu y)(P \mid Q[z := y])} \quad [\textsc{Close}]$$

2.6. táblázat. A Close szabály

és ezzel már kimondhatunk egy új szabályt, amit Close-nak nevezünk (2.6. táblázat).

Látható, hogy a feltétel output műveletében az y-ra adott korlátozás a kommunikáció után már a teljes folyamatkifejezésre vonatkozik.

A műveleti szemantika eddig megismert szabályait a 2.7. táblázatban foglaltuk össze.

2.5. Konstansok és függvények

Ebben a szakaszban megmutatjuk, hogy a pi-kalkulus folyamatkifejezéseivel hogyan lehet leírni konstansokat, ezeken értelmezett műveleteket, függvényeket. Ez a témakör nagyon hasonlít a λ-kalkulusnak ahhoz az érdekes alkalmazási területéhez, amelyben meg lehet mutatni, hogy szinte mindent le lehet írni λ-kifejezésekkel ([8, 9]). Itt is erről lesz szó, látni fogjuk, hogy nem csak a λ-kifejezésekhez, hanem adott folyamatkifejezésekhez is hozzá lehet rendelni közismert matematikai, informatikai fogalmakat. Most is beszélhetünk a tréfás „a típus csak illúzió” kifejezésről, és majd látni fogjuk ennek az állításnak a hátterét.

A leíráshoz a poliadikus pi-kalkulus kifejezéseit fogjuk használni, ezért először ezzel a témakörrel foglalkozunk, és csak utána adjuk meg néhány konstans és függvény folyamatkifejezését. A természetes számokon értelmezett összeadás függvényhez rekurzióra lesz szükség, ezért ebben a fejezetben mutatjuk meg, hogy a pi-kalkulusban hogyan lehet a rekurziót rekurzió nélkül, a replikáció műveletével leírni.

2.5.1. Poliadikus pi-kalkulus

A (monadikus) pi-kalkulusban a prefixek csak egy névre vonatkoznak, a folyamatok mindig csak egy-egy néven keresztül kommunikálnak, azaz a folyamatoknak mindig legfeljebb egy paraméterük van. A többparaméteres kifejezések azonban más területeken megszokottak és természetesek, ezért

$$\frac{P' \equiv P, \quad P \xrightarrow{\alpha} Q, \quad Q \equiv Q'}{P' \xrightarrow{\alpha} Q'} \quad [\textsc{Struct}]$$

$$\frac{}{\overline{x}y\,.\,P \xrightarrow{\overline{x}y} P} \quad [\textsc{Output}] \qquad \frac{\overline{x}y\,.\,P \xrightarrow{\overline{x}y} P, \quad x \neq y}{\overline{x}(y)\,.\,P \xrightarrow{\overline{x}(y)} P} \quad [\textsc{Open}]$$

$$\frac{}{x(y)\,.\,P \xrightarrow{x(y)} P} \quad [\textsc{Input}] \qquad \frac{}{\tau\,.\,P \xrightarrow{\tau} P} \quad [\textsc{Tau}]$$

$$\frac{\alpha\,.\,P \xrightarrow{\alpha} P}{[x = x]\alpha\,.\,P \xrightarrow{\alpha} P} \quad [\textsc{Match}]$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \quad [\textsc{Sum}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad bn(\alpha) \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad [\textsc{Par}]$$

$$\frac{P \xrightarrow{\overline{x}y} P', \quad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'[z := y]} \quad [\textsc{Com}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad x \notin n(\alpha)}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'} \quad [\textsc{Res}]$$

$$\frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \quad [\textsc{Rep}]$$

$$\frac{\overline{x}(y)\,.\,P \xrightarrow{\overline{x}(y)} P, \quad x(z)\,.\,Q \xrightarrow{x(y)} Q[z := y]}{\overline{x}(y)\,.\,P \mid x(z)\,.\,Q \xrightarrow{\tau} (\nu y)(P \mid Q[z := y])} \quad [\textsc{Close}]$$

2.7. táblázat. A műveleti szemantika szabályai

most a pi-kalkulust ezzel a jellemzővel bővítjük, és ezt a kalkulust *poliadikus pi-kalkulusnak* fogjuk nevezni.

2.5.1. Definíció. A poliadikus pi-kalkulus prefixei:

A poliadikus pi-kalkulus kifejezései csak a folyamatok output és input prefixeiben különböznek a 2.3.1. és 2.4.7. definíciókban megadottaktól:

$$\pi ::= \overline{x}\tilde{y} \mid x(\tilde{y}) \mid \overline{x}(\tilde{y}) \mid x\tilde{y} \mid \ldots ,$$

ahol a $\tilde{}$ *jel arra utal, hogy a paraméterek száma egynél nagyobb is lehet.*

Nézzük meg részletesen a prefixeket. A poliadikus kötött output és a kötött input prefixeket az

$$\begin{aligned} \overline{x}(\tilde{y}) &\equiv \overline{x}(y_1, y_2, \ldots, y_n) \equiv \overline{x}(y_1) \,.\, \overline{x}(y_2) \,.\, \ldots \,.\, \overline{x}(y_n) \,, \\ x(\tilde{y}) &\equiv x(y_1, y_2, \ldots, y_n) \equiv x(y_1) \,.\, x(y_2) \,.\, \ldots \,.\, x(y_n) \end{aligned}$$

alakra bonthatjuk ki.

Mivel a ( ) zárójelpár az input és output prefixekben mindig a kötést jelöli, a szabad output és szabad input prefixek leírásában a $\langle\ \rangle$ zárójeleket használjuk. Ezek a zárójelek itt nem paramétert jelölnek, csupán a szabad output és szabad input prefixek tárgyainak megadására szolgálnak:

$$\begin{aligned} \overline{x}\tilde{y} &\equiv \overline{x}\,\langle y_1, y_2, \ldots, y_n\rangle \equiv \overline{x}y_1 \,.\, \overline{x}y_2 \,.\, \ldots \,.\, \overline{x}y_n \,, \\ x\tilde{y} &\equiv x\,\langle y_1, y_2, \ldots, y_n\rangle \equiv xy_1 \,.\, xy_2 \,.\, \ldots \,.\, xy_n \,. \end{aligned}$$

A poliadikus kifejezések kommunikációja azonban nem triviális. Azt várjuk, hogy

$$\overline{x}\tilde{y} \,.\, P \mid x(\tilde{z}) \,.\, Q \xrightarrow{\tau} P \mid Q[\tilde{z} := \tilde{y}]$$

legyen, de azonnal látszik, hogy ha $|\tilde{w}|$ jelöli a $\tilde{w}$ neveinek darabszámát, akkor a kommunikációhoz az $|\tilde{y}| = |\tilde{z}|$ feltételt meg kell követelnünk.

A probléma azonban ennél súlyosabb, ezt a következő példában mutatjuk meg.

2.5.2. Példa. *(Poliadikus kommunikáció – hibásan)*

Legyen

$$\overline{x}\,\langle v_1, v_2\rangle \,.\, P \mid \overline{x}\,\langle w_1, w_2\rangle \,.\, Q \mid x(z_1, z_2) \,.\, R \,.$$

Ha ezeket a rövid jelöléseket a fentiek alapján átírjuk az eredeti kifejezésekre, akkor az

$$\overline{x}v_1 \,.\, \overline{x}v_2 \,.\, P \mid \overline{x}w_1 \,.\, \overline{x}w_2 \,.\, Q \mid x(z_1) \,.\, y(z_2) \,.\, R$$

kifejezést kapjuk, és az elvégezhető redukálásokkal például a következő ered-

ményt is kaphatjuk:

$$\overline{x}v_1 \,.\, \overline{x}v_2 \,.\, P \mid \overline{x}w_1 \,.\, \overline{x}w_2 \,.\, Q \mid x(z_1) \,.\, y(z_2) \,.\, R \quad \xrightarrow{\tau}$$
$$\overline{x}v_2 \,.\, P \mid \overline{x}w_1 \,.\, \overline{x}w_2 \,.\, Q \mid x(z_2) \,.\, R[z_1 := v_1] \quad \xrightarrow{\tau}$$
$$\overline{x}v_2 \,.\, P \mid \overline{x}w_2 \,.\, Q \mid R[z_1 := v_1, z_2 := w_1] \,,$$

vagy egy másik lehetséges redukálás az

$$\overline{x}v_1 \,.\, \overline{x}v_2 \,.\, P \mid \overline{x}w_1 \,.\, \overline{x}w_2 \,.\, Q \mid x(z_1) \,.\, y(z_2) \,.\, R \quad \xrightarrow{\tau}$$
$$\overline{x}v_1 \,.\, \overline{x}v_2 \,.\, P \mid \overline{x}w_2 \,.\, Q \mid x(z_2) \,.\, R[z_1 := w_1] \quad \xrightarrow{\tau}$$
$$\overline{x}v_2 \,.\, P \mid \overline{x}w_2 \,.\, Q \mid R[z_1 := w_1, z_2 := v_1]$$

eredményt adja, és nyilván még további redukálási lehetőségek lehetnek. □

A példából az látszik, hogy az eredményben a különböző paraméterek összetartozó nevei könnyen összekeveredhetnek.

A paraméterek megkülönböztetésének problémáját úgy tudjuk megoldani, hogy bevezetünk egy új nevet, és korlátozzuk ennek a névnek a használatát az output műveletre. Az output nevén kiküldjük ezt az új nevet és az output paramétereit ezen az új néven küldjük ki:

$$\overline{x}\,\langle y_1, y_2, \ldots, y_n\rangle \,.\, P \;\approx\; (\boldsymbol{\nu} p)\,\overline{x}p \,.\, \overline{p}y_1 \,.\, \overline{p}y_2 \,.\; \ldots \;.\, \overline{p}y_n \,.\, P \,.$$

A p név az x néven előírt input-output művelet kommunikációs csatornájának tekinthető. A poliadikus input műveletet úgy alakítjuk át, hogy először olvassuk ennek a kommunikációs csatornának a nevét egy új q névbe, majd ezen a néven vesszük az output művelet jeleit:

$$x(y_1, y_2, \ldots, y_n) \,.\, Q \;\approx\; x(q) \,.\, q(y_1) \,.\, q(y_2) \,.\; \ldots \;.\, q(y_n) \,.\, Q \,.$$

Látható, hogy a két folyamat kompozíciójánál először az $\overline{x}p$ és $x(q)$ kommunikációja fog megtörténni, ezzel „felépül” a két folyamat közötti kapcsolatot tartó csatorna neve, és a további kommunikációk már ezen a néven fognak megtörténni. A $\boldsymbol{\nu} p$ korlátozás biztosítja, hogy másik polimorfikus output ebbe a kommunikációba nem fog beleavatkozni.

2.5.3. Példa. *(Poliadikus kommunikáció – helyesen)*

Nézzük az előző példában szereplő folyamatkifejezést, és a fentiek szerint írjuk át az egyes kifejezéseket. Legyenek p és q új nevek, melyekre $p, q \notin fn(P) \cup fn(Q) \cup fn(R)$. Ekkor

$$\overline{x}\,\langle v_1, v_2\rangle \,.\, P \mid \overline{x}\,\langle w_1, w_2\rangle \,.\, Q \mid y(z_1, z_2) \,.\, R \;\approx$$

$$(\boldsymbol{\nu} p)\,(\overline{x}p\,.\,\overline{p}v_1\,.\,\overline{p}v_2\,.\,P) \mid (\boldsymbol{\nu} p)\,(\overline{x}p\,.\,\overline{p}w_1\,.\,\overline{p}w_2\,.\,Q) \mid x(q)\,.\,q(z_1)\,.\,q(z_2)\,.\,R\,.$$

Feltételünk alapján a p korlátozása kiterjeszthető az R-t tartalmazó folyamatra, és így feltehetjük, hogy a Q és R első prefixeinek kommunikációja hajtódik végre első lépésként. Ennek eredménye:

$$(\boldsymbol{\nu} p)\,(\overline{x}p\,.\,\overline{p}v_1\,.\,\overline{p}v_2\,.\,P) \mid (\boldsymbol{\nu} p)\,(\overline{p}w_1\,.\,\overline{p}w_2\,.\,Q \mid (\,q(z_1)\,.\,q(z_2)\,.\,R)[q := p]\,) \equiv$$
$$(\boldsymbol{\nu} p)\,(\overline{x}p\,.\,\overline{p}v_1\,.\,\overline{p}v_2\,.\,P) \mid (\boldsymbol{\nu} p)\,(\,\underline{\overline{p}w_1}\,.\,\overline{p}w_2\,.\,Q \mid \underline{p(z_1)\,.\,p(z_2)\,.\,R}\,) \xrightarrow{\tau}$$
$$(\boldsymbol{\nu} p)\,(\overline{x}p\,.\,\overline{p}v_1\,.\,\overline{p}v_2\,.\,P) \mid (\boldsymbol{\nu} p)\,(\,\underline{\overline{p}w_2}\,.\,Q \mid \underline{p(z_2)\,.\,R[z_1 := w_1]}\,) \xrightarrow{\tau}$$
$$(\boldsymbol{\nu} p)\,(\overline{x}p\,.\,\overline{p}v_1\,.\,\overline{p}v_2\,.\,P) \mid Q \mid R[z_1 := w_1, z_2 := w_2]\,.$$

Látható, hogy most már a poliadikus paraméterekben lévő nevek nem keveredtek össze. □

A továbbiakban mindenhol, ahol poliadikus, azaz többparaméteres folyamatkifejezések kommunikációjáról lesz szó, az input és az output prefixek fenti átalakítását már nem részletezzük és nem jelöljük, csupán a τ átmenetre koncentrálunk. Az új szabály tehát a következő:

$$\frac{P \xrightarrow{\overline{x}\langle\tilde{y}\rangle} P', \quad Q \xrightarrow{x(\tilde{z})} Q', \quad |\tilde{y}| = |\tilde{z}|}{P \mid Q \xrightarrow{\tau} P' \mid Q'[\tilde{z} := \tilde{y}]} \quad [\textsc{Poly-Com}]$$

2.8. táblázat. A poliadikus kommunikáció szabálya

2.5.2. Folyamatkifejezés absztrakciója

A folyamatoknak lehetnek paramétereik, egy folyamat paramétereinek száma, azaz *aritása* egy nemnegatív konstans szám. A folyamatokat absztrakcióval adjuk meg.

2.5.4. Definíció. Folyamatabsztrakció:

Egy $n \geq 0$ aritású P folyamat absztrakciója az

$$(x_1, x_2, \ldots, x_n)\,.\,P$$

alakú kifejezés, ahol az x_i $(1 \leq i \leq n)$ nevek az absztrakció változói, azaz formális paraméterei, ezek páronként különbözőek és $x_i \notin fn(P)$.

A folyamatkifejezéseket és a folyamatabsztrakciókat együttesen gyakran *ágenseknek* is nevezik.

2.5.5. Definíció. Definiáló egyenlőség:

Ha a fenti absztrakciónak az F nevet adjuk, akkor ezt az

$$F \stackrel{\text{def}}{=\!=} (x_1, x_2, \ldots, x_n) \,.\, P$$

„definiáló” egyenlőséggel jelöljük.

Az absztrakció formális paraméterei kötést is jelentenek, a paraméterek kötöttek P-ben és hatáskörük P. A P kifejezést az absztrakció törzsének nevezzük.

2.5.6. Definíció. Absztrakció példányosítása:

Az $F \stackrel{\text{def}}{=\!=} (x_1, x_2, \ldots, x_n) \,.\, P$ *absztrakcióból egy konkrét folyamatot úgy kapunk meg, hogy P-ben az* x_i *formális paramétereket aktuális, konkrét kifejezésekkel helyettesítjük. Ha az aktuális paraméterek* P_i $(1 \leq i \leq n)$, *akkor az F egy ilyen példánya*

$$F\,\langle P_1, P_2, \ldots, P_n\rangle \equiv P[x_1 := P_1, x_2 := P_2, \ldots, x_n := P_n]\,.$$

Ezt a műveletet pszeudo-applikációnak, vagy röviden példányosításnak nevezzük.

A definícióban leírt azonosságot a *szerkezeti kongruencia* egy új szabályának is tekinthetjük.

Az $F\,\langle P_1, P_2, \ldots, P_n\rangle$ kifejezésben a P_i aktuális paraméterek különkülön is megadhatók, például az $F\,\langle P_1, P_2, P_3\rangle$ kifejezés $F\,\langle P_1\rangle\,\langle P_2\rangle\,\langle P_3\rangle$, $F\,\langle P_1, P_2\rangle\,\langle P_3\rangle$, vagy akár $F\,\langle P_1\rangle\,\langle P_2, P_3\rangle$ alakban is felírható.

2.5.7. Példa. *(Egy absztrakció és két példánya)*

Ha egy P nevű folyamat törzse P' és két paramétere az x és y név, akkor ezt a

$$P \stackrel{\text{def}}{=\!=} (x, y) \,.\, P'$$

absztrakcióval jelöljük. A

$$P\,\langle Q, R\rangle \equiv P'[x := Q, y := R]\,,$$
$$P\,\langle \overline{x}, l\rangle \equiv P'[x := \overline{x}, y := l]$$

a P egy-egy példánya. □

Felhívjuk a figyelmet arra, hogy a folyamatabsztrakció definíciójában a P előtti nevek a folyamat formális paraméterei, és nem tévesztendőek össze a $\boldsymbol{\nu}$ jelű korlátozással vagy egy prefix művelet kötött paramétereivel.

Egy folyamat aktuális paraméterértéket kommunikációval is kaphat, ezt mutatjuk meg a következő példában.

2.5.8. Példa. *(Aktuális paraméter kommunikációval)*

Legyen például

$$P \stackrel{\text{def}}{=\!=} (x) \,.\, x(o,z) \,.\, \overline{o} \,.\, \overline{o} \,.\, \overline{z} \,,$$

ez egyébként a 2 számjegy folyamatkifejezése (lásd 2.5.5. pont). Ekkor

$$P\langle v\rangle \equiv v(o,z) \,.\, \overline{o} \,.\, \overline{o} \,.\, \overline{z} \;.$$

Ha az $u(v) \,.\, P\langle v\rangle$ folyamatot párhuzamosan futtatjuk egy $\overline{u}r$ folyamattal, akkor

$$\begin{aligned}
&\overline{u}r \mid u(v) \,.\, P\langle v\rangle \equiv \\
&\underline{\overline{u}r} \mid \underline{u(v) \,.\, v(o,z) \,.\, \overline{o} \,.\, \overline{o} \,.\, \overline{z}} \xrightarrow{\tau} \\
&r(o,z) \,.\, \overline{o} \,.\, \overline{o} \,.\, \overline{z} \equiv \\
&P\langle r\rangle \,,
\end{aligned}$$

azaz a 2 szám átkerült az r névre. □

2.5.3. Logikai konstansok

Reprezentálják a logikai konstansokat a következő poliadikus folyamatkifejezés absztrakciók:

$$\begin{aligned}
\mathsf{True} &\stackrel{\text{def}}{=\!=} (l) \,.\, l(t,f) \,.\, \overline{t} \,, \\
\mathsf{False} &\stackrel{\text{def}}{=\!=} (l) \,.\, l(t,f) \,.\, \overline{f} \,.
\end{aligned}$$

A kifejezésekben szereplő l név a kifejezések formális paramétere, ez lesz a kommunikációs csatorna, a kifejezések a logikai műveletekkel majd ezen keresztül kapcsolódnak össze.

Látható, hogy a True konstans az l néven vett első, a False konstans az l néven vett második néven küld ki egy közömbös, nem jelölt információt. A logikai konstansokat tehát úgy tudjuk „kiolvasni”, hogy az l néven outputként megadjuk azt, hogy a *true* vagy *false* érték jelzését melyik néven várjuk, és

azt a folyamatot a logikai konstans folyamatkifejezésével párhuzamosan futtatjuk.

2.5.9. Példa. *(A* True *logikai konstans megjelenítése)*

A kommunikációs csatorna legyen az u. A *true* érték jelzése jöjjön a v, a *false* jelzése jöjjön a w néven.

$(\nu u)\,(\,\mathsf{True}\,\langle u\rangle \mid \overline{u}\,\langle v, w\rangle\,) \equiv$

$(\nu u)\,(\,u(t, f)\,.\,\overline{t} \mid \overline{u}\,\langle v, w\rangle\,) \xrightarrow{\tau}$

$(\nu u)\,\overline{v} \equiv$

$\overline{v}\,.$ □

Ezekhez a logikai konstansokhoz adjunk meg egy jól működő If-Then-Else kifejezést. Először adjuk meg a P vagy Q „elágazás” kifejezését. Legyen

$$\mathsf{Cond} \stackrel{\text{def}}{=\!=} (p, q, l)\,.\,(\nu t, f)\,(\,\overline{l}\,\langle t, f\rangle\,.\,(t\,.\,p + f\,.\,q)\,) \qquad l, t, f \notin fn(p, q)\,,$$

így

$$\mathsf{Cond}\,\langle P, Q\rangle \equiv (l)\,.\,(\nu t, f)\,(\,\overline{l}\,\langle t, f\rangle\,.\,(t\,.\,P + f\,.\,Q)\,)\,.$$

Ha a True és False konstansok jelölésére a Bool kifejezést használjuk, akkor a

$$(\nu l)\,(\,\mathsf{Bool}\,\langle l\rangle \mid \mathsf{Cond}\,\langle P, Q\rangle\ \langle l\rangle\,)$$

kifejezés elemei az l néven kommunikálnak és ez a kifejezés $\mathsf{True}\,\langle l\rangle$ esetén a P, $\mathsf{False}\,\langle l\rangle$ esetén a Q kifejezést adja eredményül. Ennek felhasználásával

$$\mathsf{If\text{-}Then\text{-}Else} \stackrel{\text{def}}{=\!=} (f, p, q, l)\,.\,(\nu l)\,(\,f\,\langle l\rangle \mid \mathsf{Cond}\,\langle p, q\rangle\ \langle l\rangle\,)\,,$$

vagy részletesen kiírva a Cond kifejezést is:

$$\mathsf{If\text{-}Then\text{-}Else} \stackrel{\text{def}}{=\!=} (f, p, q, l)\,.\,(\nu l)\,(\,f\,\langle l\rangle \mid (\nu t, f)\,(\,\overline{l}\,\langle t, f\rangle\,.\,(t\,.\,p + f\,.\,q)\,)\,)\,,$$

ahol az f formális paraméter aktuális értéke True vagy False, p és q paraméterekbe kerülnek a kifejezés *then* és *else* ágának aktuális kifejezései, l pedig a kommunikációs csatorna.

2.5.10. Példa. *(Az* If-Then-Else True P' P'' *kifejezés)*

Tegyük fel, hogy $l, t, f \notin fn(P', P'')$, ekkor

$\mathsf{If\text{-}Then\text{-}Else}\,\langle \mathsf{True}, P', P''\rangle\ \langle l\rangle \equiv$

$(\nu l)\,(\,\mathsf{True}\,\langle l\rangle \mid \mathsf{Cond}\,\langle P', P''\rangle\ \langle l\rangle\,) \equiv$
$(\nu l)\,(\,l(t,f)\,.\,\bar{t} \mid (\nu t, f)\,(\,\bar{l}\,\langle t, f\rangle\,.\,(t\,.\,P' + f\,.\,P'')\,) \equiv$
$(\nu l, t, f)\,(\,\underline{l(t,f)\,.\,\bar{t}} \mid \underline{\bar{l}\,\langle t, f\rangle}\,.\,(t\,.\,P' + f\,.\,P'')\,) \xrightarrow{\tau}$
$(\nu l, t, f)\,(\,\underline{\bar{t}} \mid (\underline{t\,.\,P'} + f\,.\,P'')\,) \xrightarrow{\tau}$
$(\nu l, t, f)\,(\,P'\,) \equiv$
P' . □

2.5.11. Példa. *(Az* If-Then-Else False Q' Q'' *kifejezés)*

Most is tegyük fel, hogy $l, t, f \notin fn(Q', Q'')$, ekkor

$\mathsf{If\text{-}Then\text{-}Else}\,\langle \mathsf{False}, Q', Q''\rangle\ \langle l\rangle \equiv$
$(\nu l)\,(\,\mathsf{False}\,\langle l\rangle \mid \mathsf{Cond}\,\langle Q', Q''\rangle\ \langle l\rangle\,) \equiv$
$(\nu l)\,(\,l(t,f)\,.\,\overline{f} \mid (\nu t, f)\,(\,\bar{l}\,\langle t, f\rangle\,.\,(t\,.\,Q' + f\,.\,Q'')\,) \equiv$
$(\nu l, t, f)\,(\,\underline{l(t,f)\,.\,\overline{f}} \mid \underline{\bar{l}\,\langle t, f\rangle}\,.\,(t\,.\,Q' + f\,.\,Q'')\,) \xrightarrow{\tau}$
$(\nu l, t, f)\,(\,\underline{\overline{f}} \mid (t\,.\,Q' + \underline{f\,.\,Q''})\,) \xrightarrow{\tau}$
$(\nu l, t, f)\,(\,Q''\,) \equiv$
Q'' . □

Mivel már ismerjük az If-Then-Else folyamatkifejezést, a konstansokon értelmezett logikai függvények meghatározására használhatjuk a programozásból jól ismert *optimalizált kiértékelés* módszerét:

$$
\begin{aligned}
\mathit{and}\ F\ P &\equiv \mathit{if\text{-}then\text{-}else}\ F\ P\ \mathit{false}\ ,\\
\mathit{or}\ \ F\ Q &\equiv \mathit{if\text{-}then\text{-}else}\ F\ \mathit{true}\ Q\ ,\\
\mathit{not}\ F &\equiv \mathit{if\text{-}then\text{-}else}\ F\ \mathit{false}\ \mathit{true}\ .
\end{aligned}
$$

A logikai függvények argumentumai adott néven vett logikai értékek. Tegyük fel, hogy F-t, az első argumentumot az l néven kapjuk, *and* esetében a P-vel, *or* esetében a Q-val jelölt második argumentumot pedig a j néven. A logikai függvények eredménye kerüljön a k névre.

A logikai függvények *if-then-else* átírásából látszik: lehetséges, hogy az *and* esetében az *if-then-else* második argumentumát, azaz a j néven kapott P-t, *or* esetében az *if-then-else* harmadik argumentumát, vagyis a j néven kapott Q-t kell a k néven eredményként kiadni. Minden más esetben outputként a k névre a megfelelő $\mathsf{Bool}\,\langle k\rangle$ kifejezés kerül.

Készítsünk egy olyan másoló kifejezést, amelyik a p logikai értéket a j

névről átmásolja a k névre, a folyamatkifejezés neve legyen ChCh:

$$\mathsf{ChCh} \stackrel{\text{def}}{=\!=} (p, j, k)\,.\,(p\,\langle j\rangle \mid \overline{j}\,\langle t, f\rangle\,.\,(t\,.\,\mathsf{True}\ \langle k\rangle + f\,.\,\mathsf{False}\,\langle k\rangle))$$

Ennek felhasználásával a logikai függvények leírására az

$$\mathsf{If\text{-}Then\text{-}Else'} \stackrel{\text{def}}{=\!=} (f, p, l, j, k)\,.$$
$$(\boldsymbol{\nu} l)\,(\,f\,\langle l\rangle \mid (\boldsymbol{\nu} t, f)\,(\,\overline{l}\,\langle t, f\rangle\,.\,(t\,.\,\mathsf{ChCh}\,\langle p, j, k\rangle + f\,.\,\mathsf{ChCh}\,\langle p, j, k\rangle\,)\,)\,)$$

kifejezést használhatjuk, amiben majd a megfelelő helyeken a ChCh kifejezéseket lecseréljük az optimalizált kiértékelésben megadott logikai értékekre. Az optimalizált kiértékelés eljárásait használva a logikai függvények a következők:

$$\mathsf{And} \stackrel{\text{def}}{=\!=} (f, p, l, j, k)\,.$$
$$(\boldsymbol{\nu} l)\,(\,f\,\langle l\rangle \mid (\boldsymbol{\nu} t, f)\,(\,\overline{l}\,\langle t, f\rangle\,.\,(t\,.\,\mathsf{ChCh}\,\langle p, j, k\rangle + f\,.\,\mathsf{False}\,\langle k\rangle\,)\,)\,)$$

$$\mathsf{Or} \stackrel{\text{def}}{=\!=} (f, q, l, j, k)\,.$$
$$(\boldsymbol{\nu} l)\,(\,f\,\langle l\rangle \mid (\boldsymbol{\nu} t, f)\,(\,\overline{l}\,\langle t, f\rangle\,.\,(t\,.\,\mathsf{True}\,\langle k\rangle + f\,.\,\mathsf{ChCh}\,\langle q, j, k\rangle\,)\,)\,)$$

$$\mathsf{Not} \stackrel{\text{def}}{=\!=} (f, l, k)\,.$$
$$(\boldsymbol{\nu} l)\,(\,f\,\langle l\rangle \mid (\boldsymbol{\nu} t, f)\,(\,\overline{l}\,\langle t, f\rangle\,.\,(t\,.\,\mathsf{False}\,\langle k\rangle + f\,.\,\mathsf{True}\,\langle k\rangle\,)\,)\,)$$

2.5.12. Példa. *(Az And False True kifejezés)*

A fent megadott logikai függvényeknek megfelelően, legyen a False és True folyamatkifejezése az l és j néven, az eredményt pedig a k néven várjuk.

$\mathsf{And}\,\langle \mathsf{False}, \mathsf{True}, l, j, k\rangle \equiv$
$(\boldsymbol{\nu} l)\,(\,\mathsf{False}\,\langle l\rangle \mid (\boldsymbol{\nu} t, f)\,(\,\overline{l}\,\langle t, f\rangle\,.\,(t\,.\,\mathsf{ChCh}\,\langle p, j, k\rangle + f\,.\,\mathsf{False}\,\langle k\rangle\,)\,)\,) \equiv$
$(\boldsymbol{\nu} l, t, f)\,(\,(\,\underline{l(t, f)\,.\,\overline{f}}\ \mid (\,\underline{\overline{l}\,\langle t, f\rangle}\,.\,(t\,.\,\mathsf{ChCh}\,\langle p, j, k\rangle + f\,.\,\mathsf{False}\,\langle k\rangle\,)\,)\,) \xrightarrow{\tau}$
$(\boldsymbol{\nu} l, t, f)\,(\,\underline{\overline{f}}\ \mid (\,t\,.\,\mathsf{ChCh}\,\langle p, j, k\rangle + \underline{f\,.\,\mathsf{False}\,\langle k\rangle}\,)\,) \xrightarrow{\tau}$
$(\boldsymbol{\nu} l, t, f)\,(\mathsf{False}\,\langle k\rangle\,) \equiv$
$\mathsf{False}\,\langle k\rangle\,.$ □

2.5.13. Példa. *(Az Or False True kifejezés)*

$\mathsf{Or}\,\langle \mathsf{False}, \mathsf{True}, l, j, k\rangle \equiv$

$(\boldsymbol{\nu} l)\,(\,\mathsf{False}\,\langle l\rangle \mid (\boldsymbol{\nu} t, f)\,(\,\bar{l}\,\langle t, f\rangle\,.\,(t\,.\,\mathsf{True}\,\langle k\rangle + f\,.\,\mathsf{ChCh}\,\langle \mathsf{True}, j, k\rangle\,)\,)\,) \equiv$

$(\boldsymbol{\nu} l, t, f)\,(\,\underline{l(t, f)\,.\,\bar{f}} \mid (\,\underline{\bar{l}\,\langle t, f\rangle}\,.\,(t\,.\,\mathsf{True}\,\langle k\rangle + f\,.\,\mathsf{ChCh}\,\langle \mathsf{True}, j, k\rangle\,)\,)\,) \xrightarrow{\tau}$

$(\boldsymbol{\nu} l, t, f)\,(\,\underline{\bar{f}} \mid (t\,.\,\mathsf{True}\,\langle k\rangle + \underline{f\,.\,\mathsf{ChCh}\,\langle \mathsf{True}, j, k\rangle}\,)\,) \xrightarrow{\tau}$

$(\boldsymbol{\nu} l, t, f)\,\mathsf{ChCh}\,\langle \mathsf{True}, j, k\rangle \equiv$

$(\boldsymbol{\nu} l, t, f)\,(\mathsf{True}\,\langle j\rangle \mid \bar{j}\,\langle t, f\rangle\,.\,(t\,.\,\mathsf{True}\,\langle k\rangle + f\,.\,\mathsf{False}\,\langle k\rangle)\,) \equiv$

$(\boldsymbol{\nu} l, t, f)\,(\,\underline{j(t, f)\,.\,\bar{t}} \mid \underline{\bar{j}\,\langle t, f\rangle}\,.\,(t\,.\,\mathsf{True}\,\langle k\rangle + f\,.\,\mathsf{False}\,\langle k\rangle)\,) \xrightarrow{\tau}$

$(\boldsymbol{\nu} l, t, f)\,(\,\underline{\bar{t}} \mid \underline{(t\,.\,\mathsf{True}\,\langle k\rangle + f\,.\,\mathsf{False}\,\langle k\rangle)}\,) \xrightarrow{\tau}$

$(\boldsymbol{\nu} l, t, f)\,\mathsf{True}\,\langle k\rangle \equiv$

$\mathsf{True}\,\langle k\rangle\,.$ □

2.5.14. Példa. *(A* Not True *kifejezés)*

$\mathsf{Not}\,\langle \mathsf{True}, l, k\rangle \equiv$

$(\boldsymbol{\nu} l)\,(\,\mathsf{True}\,\langle l\rangle \mid (\boldsymbol{\nu} t, f)\,(\,\bar{l}\,\langle t, f\rangle\,.\,(t\,.\,\mathsf{False}\,\langle k\rangle + f\,.\,\mathsf{True}\,\langle k\rangle\,)\,)\,) \equiv$

$(\boldsymbol{\nu} l, t, f)\,(\,\underline{l(t, f)}\,.\,\bar{t} \mid (\,\underline{\bar{l}\,\langle t, f\rangle}\,.\,(t\,.\,\mathsf{False}\,\langle k\rangle + f\,.\,\mathsf{True}\,\langle k\rangle\,)\,)\,) \xrightarrow{\tau}$

$(\boldsymbol{\nu} l, t, f)\,(\,\underline{\bar{t}} \mid (\,\underline{t\,.\,\mathsf{False}\,\langle k\rangle} + f\,.\,\mathsf{True}\,\langle k\rangle\,)\,) \xrightarrow{\tau}$

$(\boldsymbol{\nu} l, t, f)\,\mathsf{False}\,\langle k\rangle \equiv$

$\mathsf{False}\,\langle k\rangle\,.$ □

A logikai konstansok és az *if-then-else* kifejezésben szereplő elvet felhasználva nagyon könnyen megadható a többirányú elágazás folyamatkifejezése is.

2.5.15. Példa. *(Többirányú elágazás)*

A példában egy ötirányú elágazást mutatunk be. Az i-edik elágazás kiválasztását az

$$E_i \stackrel{\text{def}}{=\!=} l(a_1, a_2, a_3, a_4, a_5)\,.\,\overline{a_i}$$

folyamattal végezhetjük el, amely az l néven kommunikál az elágazások

$$\bar{l}\,\langle a_1, a_2, a_3, a_4, a_5\rangle\,.\,(a_1\,.\,P_1 + a_2\,.\,P_2 + a_3\,.\,P_3 + a_4\,.\,P_4 + a_5\,.\,P_5)$$

kifejezésével. Nyilvánvaló, hogy ha például az E_2 folyamatot ezzel a kifejezéssel párhuzamosan végrehajtjuk, akkor a P_2 eredményt kapjuk meg. □

2.5.4. Megszűnő és ismétlődő folyamatkifejezések

Az ebben a fejezetben tárgyalt folyamatkifejezések mindegyike olyan volt, hogy ha használtuk a kifejezést, akkor a kifejezés megváltozott. Input esetén a formális paraméterek felvették aktuális értéküket, output esetén a használt prefix még ki is törlődött a kifejezésből. A folyamatok működése a legtöbb esetben az „esemény nem figyelhető meg” állapotot leíró **0** jellel fejeződött be.

Ez azt jelenti, hogy egy kifejezés eredeti alakjában csak egyszer volt használható, ami nem teszi lehetővé a kifejezéssel leírt működés folyamatos, többszöri végrehajtását. A folyamatot az eredeti állapotában mindig újra kell indítani.

Ez a jelenség különösen problémás a konstansok esetében, hiszen ha például egyszer használtunk egy True értéket, akkor a használat után ezt a kifejezést már nem tudjuk többször használni.

2.5.16. Példa. *(A* Not *kétszeri alkalmazása)*

Azt várjuk, hogy a

$$\mathsf{True}\,\langle l\rangle \mid \mathsf{Not}\,\langle l, k_1\rangle \mid \mathsf{Not}\,\langle l, k_2\rangle \longrightarrow^{+} \mathsf{False}\,\langle k_1\rangle \mid \mathsf{False}\,\langle k_2\rangle$$

legyen, hiszen mindkét Not művelet az l néven veszi az input értéket. Ez azonban nem lesz így. Mint azt a 2.5.14. példában láttuk, az első két folyamat kompozíciójának eredménye (a most megadott nevekkel) $\mathsf{False}\,\langle k_1\rangle$, és azonnal látható, hogy a True kifejezés eltűnt, többet ebben a kifejezésben nem használható. Ráadásul a kompozíció False eredménye a k_1 névre került, és a második Not még csak ezzel sem tud kommunikálni, mivel az l néven várja az inputot. □

Ez a probléma a P ismétlésének vagy replikációjának nevezett $!P$ alakú kifejezéssel oldható meg, hiszen mint a 2.3.2. definícióban láttuk,

$$!P = P \mid !P\,.$$

Ha a kifejezésünkben P helyett a $!P$-t használjuk, akkor ha végrehajtáskor a P eltűnik, a $!P$ kifejezés még megmarad, amiből újabb P folyamatot vagy akár több P folyamatot lehet leválasztani.

Az ilyen „örök, állandó” folyamatokat *ismétlődő* folyamatoknak nevezzük, az egyszer végrehajtható folyamatok a *megszűnő* jelzőt kapják.

Egy P megszűnő folyamat kifejezésének ismétlődő változatát $^{*}P$-vel fogjuk jelölni.

A logikai konstansok ismétlődő folyamatkifejezései a következők:

$$
\begin{aligned}
{}^{*}\mathsf{True} &\stackrel{\text{def}}{=\!=} (l)\,.\,!\,l(t,f)\,.\,\bar{t}\,,\\
{}^{*}\mathsf{False} &\stackrel{\text{def}}{=\!=} (l)\,.\,!\,l(t,f)\,.\,\bar{f}\,.
\end{aligned}
$$

2.5.17. Példa. *(A* Not *kétszeri alkalmazása a* *True *konstansra)*

$$
\begin{aligned}
&{}^{*}\mathsf{True}\,\langle l\rangle \mid \mathsf{Not}\,\langle l,k_1\rangle \mid \mathsf{Not}\,\langle l,k_2\rangle \equiv\\
&!\,l(t,f)\,.\,\bar{t} \mid \mathsf{Not}\,\langle l,k_1\rangle \mid \mathsf{Not}\,\langle l,k_2\rangle \equiv\\
&l(t,f)\,.\,\bar{t} \mid !\,l(t,f)\,.\,\bar{t} \mid \mathsf{Not}\,\langle l,k_1\rangle \mid \mathsf{Not}\,\langle l,k_2\rangle \equiv\\
&\underline{l(t,f)\,.\,\bar{t} \mid \mathsf{Not}\,\langle l,k_1\rangle} \mid !\,l(t,f)\,.\,\bar{t} \mid \mathsf{Not}\,\langle l,k_2\rangle\,.
\end{aligned}
$$

Az első két kifejezésre felhasználva a 2.5.14. példa eredményét:

$$
\begin{aligned}
&\mathsf{False}\,\langle k_1\rangle \mid !\,l(t,f)\,.\,\bar{t} \mid \mathsf{Not}\,\langle l,k_2\rangle \equiv\\
&\mathsf{False}\,\langle k_1\rangle \mid l(t,f)\,.\,\bar{t} \mid !\,l(t,f)\,.\,\bar{t} \mid \mathsf{Not}\,\langle l,k_2\rangle \equiv\\
&\mathsf{False}\,\langle k_1\rangle \mid \underline{l(t,f)\,.\,\bar{t} \mid \mathsf{Not}\,\langle l,k_2\rangle} \mid !\,l(t,f)\,.\,\bar{t}\,,
\end{aligned}
$$

és ismét a 2.5.14. példára hivatkozva az eredmény:

$$\mathsf{False}\,\langle k_1\rangle \mid \mathsf{False}\,\langle k_2\rangle \mid !\,l(t,f)\,.\,\bar{t}\,.$$

Tehát megkaptuk a várt eredményt. A kompozíció harmadik tagja a *True $\langle l\rangle$ kifejezés, tehát a *True konstans is megmaradt az l néven. □

2.5.5. Természetes számok

Ebben a szakaszban először a természetes számokra adunk meg folyamatkifejezéseket. Az n természetes szám *ismétlődő* folyamatkifejezését jelölje $[\![n]\!]$. A számok kifejezéseire meg fogunk adni olyan Succ és Zero kifejezéseket, amelyekre

$$
\begin{aligned}
\mathsf{Succ}\,[\![k]\!] &\longrightarrow^{+} [\![k+1]\!]\,,\\
\mathsf{Zero}\,[\![k]\!] &\longrightarrow^{+} \begin{cases} \mathsf{True}, & \textit{ha } [\![k]\!] \equiv [\![0]\!]\\ \mathsf{False} & \textit{egyébként,}\end{cases}
\end{aligned}
$$

így a számok *számjegyrendszert* vagy röviden *számrendszert* fognak alkotni.

2.5.18. Definíció. A természetes számok:

Legyen az n természetes szám folyamatkifejezése

$$[\![n]\!] \stackrel{\text{def}}{=\!=} (x)\,.\,!x(o,z)\,.\,(\overline{o}.)^n\,.\,\overline{z}\,,$$

ahol $(\overline{o}.)^n$ *az* $\underbrace{\overline{o}\,.\ \ldots\ .\overline{o}}_{n}$ *rövid jelölése.*

2.5.19. Példa. *(A természetes számok)*

$$\begin{aligned}
[\![0]\!] &\stackrel{\text{def}}{=\!=} (x)\,.\,!x(o,z)\,.\,\overline{z}\\
[\![1]\!] &\stackrel{\text{def}}{=\!=} (x)\,.\,!x(o,z)\,.\,\overline{o}\,.\,\overline{z}\\
[\![2]\!] &\stackrel{\text{def}}{=\!=} (x)\,.\,!x(o,z)\,.\,\overline{o}\,.\,\overline{o}\,.\,\overline{z}\\
[\![3]\!] &\stackrel{\text{def}}{=\!=} (x)\,.\,!x(o,z)\,.\,\overline{o}\,.\,\overline{o}\,.\,\overline{o}\,.\,\overline{z}\\
\ldots
\end{aligned}$$

□

Látható, hogy a természetes számok kifejezéseiben a $\overline{z}$-re azért van szükség, hogy a *nulla* természetes számot is meg tudjuk adni, különben a nullára nem lenne megfigyelhető esemény.

A számokat úgy tudjuk „olvasni”, hogy az x kommunikációs csatornán megadunk két nevet, és ha a szám értéke k, akkor az első néven k darab output jelenik meg, a második néven egy darab output. A második néven jelentkező output a szám értékét reprezentáló jelsorozat befejezéseként, záró jeleként értelmezhető.

2.5.20. Példa. *(A nulla és a 3 szám folyamatkifejezése)*

Ha a számok kommunikációs csatornája az u, szám értékét a v, végjelét a w néven várjuk, akkor

$$\begin{aligned}
&(\nu u)\,(\,[\![0]\!]\,\langle u\rangle \mid \overline{u}\,\langle v,w\rangle\,) \equiv\\
&(\nu u)\,(\,\underline{!u(o,z)\,.\,\overline{z}} \mid \underline{\overline{u}\,\langle v,w\rangle}\,) \xrightarrow{\tau}\\
&(\nu u)\,(\,!u(o,z)\,.\,\overline{z} \mid \overline{w}\,) \longrightarrow^{+} \qquad *** \quad \overline{w}\\
&(\nu u)\,!u(o,z)\,.\,\overline{z}\,,
\end{aligned}$$

és

$$\begin{aligned}
&(\nu u)\,(\,[\![3]\!]\,\langle u\rangle \mid \overline{u}\,\langle v,w\rangle\,) \equiv\\
&(\nu u)\,(\,\underline{!u(o,z)\,.\,\overline{o}\,.\,\overline{o}\,.\,\overline{o}\,.\,\overline{z}} \mid \underline{\overline{u}\,\langle v,w\rangle}\,) \xrightarrow{\tau}
\end{aligned}$$

$(\nu u)\,(\,!u(o,z)\,.\,\overline{o}\,.\,\overline{o}\,.\,\overline{o}\,.\,\overline{z} \mid \overline{v}\,.\,\overline{v}\,.\,\overline{v}\,.\,\overline{w}\,) \longrightarrow^{+}$ *** $\overline{v}\,.\,\overline{v}\,.\,\overline{v}\,.\,\overline{w}$

$(\nu u)\,!u(o,z)\,.\,\overline{o}\,.\,\overline{o}\,.\,\overline{o}\,.\,\overline{z}\,.$ □

A példából is látható, hogy a számok folyamatkifejezései a műveletek elvégzése után megmaradnak.

A *succ n* kifejezés az $n+1$ számot állítja elő. A *succ* függvény folyamatkifejezése a következő:

$$^{*}\mathsf{Succ} \stackrel{\text{def}}{=\!=} (x,y)\,.\,!y(o,z)\,.\,\overline{o}\,.\,\overline{x}\,\langle o,z\rangle\,.$$

A Succ kifejezésnek két formális paramétere van, az x névre a *succ n* bemenő paraméterét, azaz az n szám folyamatkifejezését kell adni, és az y néven kell megadni, hogy milyen neveken várjuk az eredmény számértékét és végjelét.

2.5.21. Példa. *(A* Succ $[\![2]\!]$ *kiértékelése)*

Határozzuk meg a *succ* 2 folyamatkifejezését. Az eredmény számértéke és végjele kerüljön a p és q névre, és ezt a Succ kifejezésnek a v néven adjuk meg. A növelendő szám kommunikációs csatornája legyen u.

$(\nu u,v)\,(\,\mathsf{Succ}\,\langle u,v\rangle \mid [\![2]\!]\,\langle u\rangle \mid \overline{v}\,\langle p,q\rangle\,) \equiv$

$(\nu u,v)\,(\,\underline{!v(o,z).\overline{o}\,.\,\overline{u}\,\langle o,z\rangle} \mid !u(o,z)\,.\,(\overline{o})^2\,.\,\overline{z} \mid \underline{\overline{v}\,\langle p,q\rangle}\,) \xrightarrow{\tau}$

$(\nu u,v)\,(\,!v(o,z).\overline{o}\,.\,\overline{u}\,\langle o,z\rangle \mid \overline{p}\,.\,\overline{u}\,\langle p,q\rangle \mid !u(o,z)\,.\,(\overline{o})^2\,.\,\overline{z}\,) \xrightarrow{\overline{p}}$ *** $\overline{p}$

$(\nu u,v)\,(\,!v(o,z).\overline{o}\,.\,\overline{u}\,\langle o,z\rangle \mid \underline{\overline{u}\,\langle p,q\rangle} \mid \underline{!u(o,z)\,.\,(\overline{o})^2\,.\,\overline{z}}\,) \xrightarrow{\tau}$

$(\nu u,v)\,(\,!v(o,z).\overline{o}\,.\,\overline{u}\,\langle o,z\rangle \mid (\overline{p})^2\,.\,\overline{q} \mid !u(o,z)\,.\,(\overline{o})^2\,.\,\overline{z}\,) \longrightarrow^{+}$ *** $\overline{p}\,.\,\overline{p}\,.\,\overline{q}$

$(\nu u,v)\,(\,!v(o,z).\overline{o}\,.\,\overline{u}\,\langle o,z\rangle \mid !u(o,z)\,.\,(\overline{o})^2\,.\,\overline{z}\,)\,.$

Látható, hogy a p névre három output került. □

A Zero folyamatkifejezés megadása viszonylag egyszerű, hiszen ha a számjegy kiértékelésekor a definiálásában szereplő o néven nincs jel és megjön a z néven a végjel, akkor a szám biztosan nulla. Mivel a $\overline{z}$ az utolsó művelet, ha először egy $\overline{o}$ jelet kapunk, akkor a szám biztosan nem nulla.

Így a Zero függvény folyamatkifejezése a következő:

$$^{*}\mathsf{Zero} \stackrel{\text{def}}{=\!=} (x,y)\,.\,!\overline{x}\,\langle o,z\rangle\,.\,(o\,.\,\mathsf{False}\,\langle y\rangle + z\,.\,\mathsf{True}\,\langle y\rangle)\,.$$

A folyamatkifejezésnek két formális paramétere van, az első a számjegy kifejezésével a kommunikációs csatorna, és a második néven megadott

csatornán jelenik meg a függvény értéke.

A könyv további részében a rövidebb levezetések érdekében – ha nem feltétlenül szükséges – a $!P$ *helyett csak P-t írunk, azaz az ismétlődő alakjuk helyett a megszűnő alakjukat használjuk*, annak a tudatában, hogy ha ezt nem tennénk, a $!P$ alakú kifejezések a levezetések végére megmaradnának.

2.5.22. Példa. *(A* Zero $[\![0]\!]$ *és a* Zero $[\![3]\!]$ *kifejezés)*

Tegyük fel, hogy a kiértékelések kommunikációs csatornája az u és az eredményt a w néven várjuk.

$$(\nu u, w)\,(\,\mathsf{Zero}\,\langle u, w\rangle \mid [\![0]\!]\,\langle u\rangle\,) \equiv$$
$$(\nu u, w)\,(\,\underline{\overline{u}\,\langle o, z\rangle\,.\,(o\,.\,\mathsf{False}\,\langle w\rangle + z\,.\,\mathsf{True}\,\langle w\rangle)} \mid \underline{u(o, z)\,.\,\overline{z}}\,) \xrightarrow{\tau}$$
$$(\nu u, w)\,(\,\underline{(o\,.\,\mathsf{False}\,\langle w\rangle + z\,.\,\mathsf{True}\,\langle w\rangle)} \mid \underline{\overline{z}}\,) \xrightarrow{\tau}$$
$$(\nu u, w)\,\mathsf{True}\,\langle w\rangle\,,$$

és

$$(\nu u, w)\,(\,\mathsf{Zero}\,\langle u, w\rangle \mid [\![3]\!]\,\langle u\rangle\,) \equiv$$
$$(\nu u, w)\,(\,\underline{\overline{u}\,\langle o, z\rangle\,.\,(o\,.\,\mathsf{False}\,\langle w\rangle + z\,.\,\mathsf{True}\,\langle w\rangle)} \mid \underline{u(o, z)\,.\,(\overline{o})^3\,.\,\overline{z}}\,) \xrightarrow{\tau}$$
$$(\nu u, w)\,(\,\underline{(o\,.\,\mathsf{False}\,\langle w\rangle} + z\,.\,\mathsf{True}\,\langle w\rangle) \mid \underline{\overline{o}}\,.\,(\overline{o})^2\,.\,\overline{z}\,) \xrightarrow{\tau}$$
$$(\nu u, w)\,(\,\mathsf{False}\,\langle w\rangle \mid (\overline{o})^2\,.\,\overline{z}\,) \longrightarrow^{+}$$
$$(\nu u, w)\,\mathsf{False}\,\langle w\rangle\,.$$

□

A pi-kalkulusban – természetesen – megadható a két természetes szám összeadását végző Add folyamat kifejezése is. Tegyük fel, hogy az $n + m = w$ műveletet kell elvégeznünk, tehát az Add folyamatnak három paramétere lesz. Az n és m szám legyen az i és j néven, a w eredményt tegyük a k névre, ahol $\overline{r}$ reprezentálja az eredmény értékét és $\overline{s}$ legyen a számot lezáró output.

Az Add művelet elvégzésének elve az lesz, hogy

- ha $n = 0$, akkor a $[\![w]\!]$ eredmény r és s neveire az $[\![m]\!]$ adatait kell átmásolni,
- egyébként először a $[\![w]\!]$ eredmény r csatornájára n darab jelet kell kiadni, majd a jelek kiadását az $[\![m]\!]$ adataival kell folytatni az r és s néven, azaz további m darab outputot kell adni a $[\![w]\!]$ eredmény r nevére és egy outputot az s nevére.

Ehhez először készítsünk egy Copy folyamatot, amelyik az x néven levő számot átmásolja az y névre:

$$\mathsf{Copy} \stackrel{\text{def}}{=\!=} (x, y) \,.\, !y(o, z) \,.\, \overline{x}\,\langle o, z\rangle \,.$$

A másolandó számot az x néven kell megadni, és a másolás eredménye az y néven megadott csatornákon jelenik meg.

2.5.23. Példa. *(A $[\![3]\!]$ másolása)*

$(\nu u, v)\,(\,\mathsf{Copy}\,\langle u, v\rangle \mid [\![3]\!]\,\langle u\rangle \mid \overline{v}\,\langle p, q\rangle\,) \equiv$

$(\nu u, v)\,(\,\underline{v(o, z) \,.\, \overline{u}\,\langle o, z\rangle} \mid u(o, z) \,.\, (\overline{o})^3\overline{z} \mid \underline{\overline{v}\,\langle p, q\rangle}\,) \xrightarrow{\tau}$

$(\nu u, v)\,(\,\underline{\overline{u}\,\langle p, q\rangle} \mid \underline{u(o, z) \,.\, (\overline{o})^3\overline{z}}\,) \xrightarrow{\tau}$

$(\nu u, v)\,(\overline{p})^3\overline{q}\,.$ □

A példa utolsó előtti sorából kiolvasható, hogy ha az u néven adott szám értékét és záró jelét a p, q nevekre akarjuk másolni, akkor elegendő az u névvel adott számmal párhuzamosan egy $\overline{u}\,\langle p, q\rangle$ folyamatot futtatni. Ezt használjuk majd fel az Add kifejezésben az m szám adatainak átmásolására.

Így már megadhatjuk az $\mathsf{Add}\,\langle i, j, k\rangle$ kifejezést:

$$\mathsf{Add} \stackrel{\text{def}}{=\!=} (i, j, k) \,.\, k(r, s) \,.\, \overline{i}\,\langle p, q\rangle \,.\, \mathsf{Add}^+ \,,$$

ahol

$$\mathsf{Add}^+ \equiv p \,.\, \overline{r} \,.\, \mathsf{Add}^+ + q \,.\, \overline{j}\,\langle r, s\rangle \,.$$

Az n és m számok összeadását az

$$\mathsf{Add}\,\langle i, j, k\rangle \mid [\![n]\!]\,\langle i\rangle \mid [\![m]\!]\,\langle j\rangle \mid \overline{k}\,\langle r, s\rangle$$

végzi el, ahol a $\overline{k}\,\langle r, s\rangle$ kifejezéssel adjuk meg, hogy az összeadás eredményét az r és s neveken várjuk. Látható, hogy a k néven történő kommunikáció eredménye az Add^+ kifejezésben kétszer is megjelenik,

- az r névre szükség van, mert az r-re másolja az n számból jövő, az n értékét reprezentáló outputokat,
- másrészt az m másolásának is k lesz outputja, azaz a másolás eredménye az r és s neveken jelenik meg.

Az Add^+ kifejezés egy rekurzív kifejezés, de majd a következő szakaszban látjuk, hogy a rekurzió a pi-kalkulusban nem okoz problémát, a rekurzív folyamatkifejezés átírható rekurziómentes kifejezésre.

2.5.24. Példa. *(Az* $\mathsf{Add}\ 2\ 1$ *kifejezés)*

Az összeadás eredménye kerüljön az a és b nevekre.

$$\mathsf{Add}\,\langle i,j,k\rangle \mid [\![2]\!]\,\langle i\rangle \mid [\![1]\!]\,\langle j\rangle \mid \overline{k}\,\langle a,b\rangle \equiv$$

$$\underline{k(r,s)\,.\,\overline{i}\,\langle p,q\rangle\,.\,\mathsf{Add}^+} \mid [\![2]\!]\,\langle i\rangle \mid [\![1]\!]\,\langle j\rangle \mid \underline{\overline{k}\,\langle a,b\rangle} \xrightarrow{\tau}$$

$$\overline{i}\,\langle p,q\rangle\,.\,\mathsf{Add}^+ \mid [\![2]\!]\,\langle i\rangle \mid [\![1]\!]\,\langle j\rangle$$

és a helyettesítést az Add^+-ban is elvégezve

$$p\,.\,\overline{a}\,.\,\mathsf{Add}^+ + q\,.\,\overline{j}\,\langle a,b\rangle\,.$$

Tovább folytatva a műveleteket, beírva a 2 számjegy $i(o,z)\,.\,\overline{o}\,.\,\overline{o}\,.\,\overline{z}$ kifejezését, majd az Add^+ kifejezését is,

$$\underline{\overline{i}\,\langle p,q\rangle\,.\,\mathsf{Add}^+} \mid \underline{i(o,z)\,.\,\overline{o}\,.\,\overline{o}\,.\,\overline{z}} \mid [\![1]\!]\,\langle j\rangle \xrightarrow{\tau}$$

$$\mathsf{Add}^+ \mid \overline{p}\,.\,\overline{p}\,.\,\overline{q} \mid [\![1]\!]\,\langle j\rangle \equiv$$

$$(\underline{p\,.\,\overline{a}\,.\,\mathsf{Add}^+} + q\,.\,\overline{j}\,\langle a,b\rangle) \mid \underline{\overline{p}}\,.\,\overline{p}\,.\,\overline{q} \mid [\![1]\!]\,\langle j\rangle \xrightarrow{\tau}$$

$$\overline{a}\,.\,\mathsf{Add}^+ \mid \overline{p}\,.\,\overline{q} \mid [\![1]\!]\,\langle j\rangle \xrightarrow{\overline{a}} \qquad *** \ \overline{a}$$

$$\mathsf{Add}^+ \mid \overline{p}\,.\,\overline{q} \mid [\![1]\!]\,\langle j\rangle\,.$$

Látható, hogy az a néven megjelenik az első jel. Beírva az Add^+ kifejezést és tovább folytatva, a 2 számjegy második $\overline{p}$ kifejezésére megkapjuk a második $\overline{a}$-t is. Ezután

$$\mathsf{Add}^+ \mid \overline{q} \mid [\![1]\!]\,\langle j\rangle \equiv$$

$$(p\,.\,\overline{a}\,.\,\mathsf{Add}^+ + \underline{q\,.\,\overline{j}\,\langle a,b\rangle}) \mid \underline{\overline{q}} \mid [\![1]\!]\,\langle j\rangle \xrightarrow{\tau}$$

$$\overline{j}\,\langle a,b\rangle \mid [\![1]\!]\,\langle j\rangle\,.$$

Ettől a ponttól, mint a 2.5.23. példa utáni megjegyzésben már láttuk, megkezdődik a j néven lévő 1 szám átmásolása az a,b-re. Ha ez befejeződik, eredményül három $\overline{a}$ és egy $\overline{b}$ jelet kapunk. □

2.5.6. Rekurzió és replikáció

Az előző szakaszban az összeadás egyik rész-műveletét rekurzív folyamatkifejezéssel adtuk meg. Most megmutatjuk, hogy a rekurzív folyamatkifejezések a replikáció alkalmazásával rekurzió nélkül is leírhatók.

A rekurzív kifejezés legyen

$$A \stackrel{\text{def}}{=\!=} (x) \,.\, Q_A \ ,$$

ahol

$$Q_A \equiv \ldots A\,\langle u\rangle \ldots A\,\langle v\rangle \ldots \ ,$$

azaz Q_A-ban akár többször is előfordulhat különböző paraméterekkel az A meghívása. Ez azt jelenti, hogy A végrehajtásakor egymás után nagyon sokszor, akár végtelen sokszor is meghívhatjuk az A kifejezést.

A $!A$ replikáció is lehetőséget ad az A többszöri meghívására, hiszen $!A \equiv A \mid A \mid \ldots \mid !A$. A két végrehajtás között a különbség azonnal látszik, a rekurzióban az A végrehajtásai „vertikálisan”, a replikációban „horizontálisan” hajtódnak végre. A replikáció végrehajtása azonban nem tartja meg az A-knak a kifejtésben megadott sorrendjét, hiszen az A-k egymástól függetlenül hajtódnak végre.

2.5.25. Példa. *(A replikáció és a rekurzió végrehajtása)*

Legyen $P \equiv !l(x) \,.\, \overline{m}x$, és nézzük a

$P \equiv l(x) \,.\, \overline{m}x \mid l(x) \,.\, \overline{m}x \mid !P$

kifejezést. Kapcsoljuk ehhez a kompozíció műveletével az $\overline{l}u$ és $\overline{l}v$ folyamatokat, és tegyük fel, hogy először az u, majd a v adattal történik meg a kommunikáció. A két kommunikáció után az

$\overline{m}u \mid \overline{m}v \mid P$

folyamatot kapjuk, ahol az m néven az u és v tetszőleges sorrendben jelenhet meg. Most adjuk meg a P által végzett műveletet rekurzióval:

$P' \equiv l(x) \,.\, \overline{m}x \,.\, P'$.

Ha ehhez kapcsoljuk a $\overline{l}u$ és $\overline{l}v$ folyamatokat, látható, hogy az m néven az u és v sorrendje megegyezik a kommunikáció sorrendjével. □

Írjuk át a fenti A rekurzív folyamatot úgy, hogy a kifejezésben replikáció szerepeljen, de a példában is látott probléma már ne forduljon elő. Az átalakított folyamatot jelöljük az $A^{(!)}$ jellel.

2.5.26. Definíció. Rekurzió átírása replikációra:

Az $A \stackrel{\text{def}}{=\!=} (x)\,.\,Q_A$ *átírásának lépései a következők:*

1. *vezessünk be egy új nevet, például az a-t,*
2. Q_A*-ban az* $A\,\langle x\rangle$ *előfordulásokat helyettesítsük* $\overline{a}x$*-szel, a helyettesítések eredménye legyen* $\widehat{Q_A}$ *,*
3. *legyen* $A^{(!)} = (x)\,.\,(\boldsymbol{\nu} a)\,(\overline{a} \mid !\overline{a}x\,.\,\widehat{Q_A})$ *.*

Ha $B \equiv D$ egy olyan folyamat, hogy D-ben szerepelnek a rekurzív A-nak az $A\,\langle x_i\rangle$ $(1 \leq i)$ előfordulásai, akkor a fenti algoritmus 2. pontjában leírt helyettesítéseket hajtsuk végre D-re is. A helyettesítések eredményét jelöljük $\widehat{D}$-vel, ekkor

$$B^{(!)} \equiv \boldsymbol{\nu}(a)\,(\widehat{D} \mid !a(x)\,.\,\widehat{Q_A}) \ .$$

2.5.27. Példa. *(A rekurzió helyes átírása)*

Nézzük a 2.5.25. példában szereplő

$$P' \equiv l(x)\,.\,\overline{m}x\,.\,P'$$

rekurzív kifejezést. A P'-nek nincs formális paramétere, ezért $a(x)$ és $\overline{a}x$ helyett a és $\overline{a}$ írható, így a fenti algoritmussal

$$P'^{(!)} \equiv (\boldsymbol{\nu} a)\,(\overline{a} \mid !a\,.\,l(x)\,.\,\overline{m}x\,.\,\overline{a})\ .$$

Készítsük el ezzel a $P'^{(!)} \mid \overline{l}u \mid \overline{l}v$ kompozíciót:

$$(\boldsymbol{\nu} a)\,(\overline{a} \mid !a\,.\,l(x)\,.\,\overline{m}x\,.\,\overline{a}) \mid \overline{l}u \mid \overline{l}v \equiv$$

$$(\boldsymbol{\nu} a)\,(\underline{\overline{a}} \mid \underline{a\,.\,l(x)\,.\,\overline{m}x\,.\,\overline{a}} \mid !a\,.\,l(x)\,.\,\overline{m}x\,.\,\overline{a} \mid \overline{l}u \mid \overline{l}v) \xrightarrow{\tau}$$

$$(\boldsymbol{\nu} a)\,(\underline{l(x)}\,.\,\overline{m}x\,.\,\overline{a} \mid !a\,.\,l(x)\,.\,\overline{m}x\,.\,\overline{a} \mid \underline{\overline{l}u} \mid \overline{l}v) \xrightarrow{\tau}$$

$$(\boldsymbol{\nu} a)\,(\overline{m}u\,.\,\overline{a} \mid !a\,.\,l(x)\,.\,\overline{m}x\,.\,\overline{a} \mid \overline{l}v) \xrightarrow{\overline{m}u}$$

$$(\boldsymbol{\nu} a)\,(\underline{\overline{a}} \mid \underline{a\,.\,l(x)\,.\,\overline{m}x\,.\,\overline{a}} \mid !a\,.\,l(x)\,.\,\overline{m}x\,.\,\overline{a} \mid \overline{l}v) \xrightarrow{\tau}$$

$$(\boldsymbol{\nu} a)\,(\underline{l(x)\,.\,\overline{m}x\,.\,\overline{a}} \mid !a\,.\,\overline{m}x\,.\,\overline{a} \mid \mid \underline{\overline{l}v}) \xrightarrow{\tau}$$

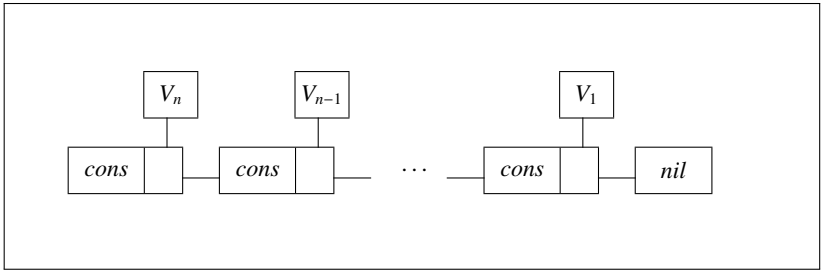
$$(\boldsymbol{\nu} a)\,(\overline{m}v\,.\,\overline{a} \mid !a\,.\,l(x)\,.\,\overline{m}x\,.\,\overline{a}) \xrightarrow{\overline{m}v}$$

$$(\boldsymbol{\nu} a)\,(\overline{a} \mid !a\,.\,l(x)\,.\,\overline{m}x\,.\,\overline{a})\ .$$

A levezetésből is látható, hogy az m néven az u és v sorrendje most is megegyezik a kommunikáció sorrendjével. □

2.5.7. Lista

A *láncolt lista* adatszerkezetnek két konstruktora van, a *nil* és a *cons*. A *nil* a lista végét jelzi, paramétere nincs, ezért a lista adatszerkezet konstansának tekinthető, hiszen önmagában állva is egy listát alkot. A *cons* konstruktor egy új elemet csatol egy már meglévő listához, az új elem a lista fejeleme lesz. A 2.1. ábrán a $[V_n, V_{n-1}, \ldots, V_1, nil]$ lista szerkezete látható.



2.1. ábra. A láncolt lista adatszerkezet

A pi-kalkulusban a konstruktorokat összekapcsoló név legyen a k. Ekkor a lista adatszerkezet konstruktorai a következők:

$$\mathsf{Nil} \stackrel{\text{def}}{=\!=} (k)\,.\,k(n,c)\,.\,\overline{n}\ ,$$

$$\mathsf{Cons} \stackrel{\text{def}}{=\!=} (k,V,L)\,.\,(\boldsymbol{\nu} v,l)\,(k(n,c)\,.\,\overline{c}\,\langle v,l\rangle \ \mid V\langle v\rangle \mid L\,\langle l\rangle)\ ,$$

ahol $V\langle v\rangle$ a v néven levő érték és $L\,\langle l\rangle$ az l néven található lista.

A $[V_n, V_{n-1}, \ldots, V_1, nil]$ lista folyamatkifejezése tehát

$\mathsf{Cons}\,\langle k, V_n, \mathsf{Cons}\,\langle l_n, V_{n-1}, \ldots, \mathsf{Cons}\,\langle l_2, V_1, \mathsf{Nil}\,\langle l_1\rangle\,\rangle \ldots\rangle\rangle$.

2.5.28. Példa. *(A* $[3, 2, nil]$ *lista folyamatkifejezése)*

A lista szerkezete a 2.2. ábrán látható. Mivel $[3, 2, nil] \equiv [3, [2, nil]\,]$, először határozzuk meg a $[2, nil]$ lista kifejezését.

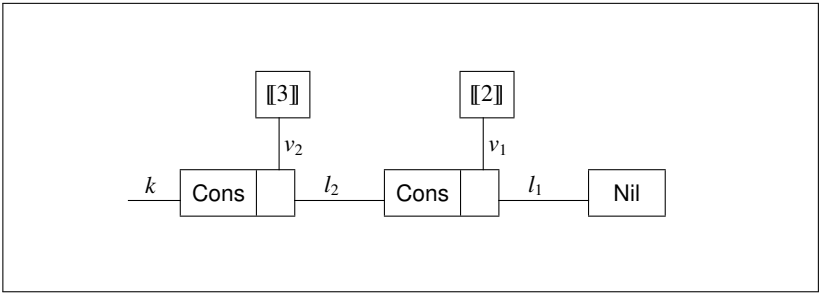
$[[\![2]\!], \mathsf{Nil}] \equiv$

$\mathsf{Cons}\,\langle k\rangle\ \langle [\![2]\!], \mathsf{Nil}\rangle \equiv$

$(\boldsymbol{\nu} v_1, l_1)\,(\,k(n,c)\,.\,\overline{c}\,\langle v_1, l_1\rangle \mid [\![2]\!]\,\langle v_1\rangle \mid \mathsf{Nil}\,\langle l_1\rangle\,)\ .$

Így a $[3, 2, nil]$ lista kifejezése:

$[[\![3]\!],\ [\,[\![2]\!], \mathsf{Nil}\,]\,] \equiv$



2.2. ábra. A $[3, 2, nil]$ lista adatai és csatornanevei

$$\mathsf{Cons}\,\langle k\rangle\ \langle [\![3]\!], [\,[\![2]\!], \mathsf{Nil}\,]\rangle \equiv$$
$$(\boldsymbol{\nu} v_2, l_2)\,(\,k(n,c)\,.\,\overline{c}\,\langle v_2, l_2\rangle \mid [\![3]\!]\,\langle v_2\rangle \mid [\,[\![2]\!], \mathsf{Nil}\,]\,\langle l_2\rangle\,) \equiv$$
$$(\boldsymbol{\nu} v_2, l_2)\,(\,k(n,c)\,.\,\overline{c}\,\langle v_2, l_2\rangle \mid [\![3]\!]\,\langle v_2\rangle \mid$$
$$(\boldsymbol{\nu} v_1, l_1)\,(\,l_2(n,c)\,.\,\overline{c}\,\langle v_1, l_1\rangle \mid [\![2]\!]\,\langle v_1\rangle \mid \mathsf{Nil}\,\langle l_1\rangle\,)\,. \quad \square$$

A lista adatszerkezetre megadunk három függvényt:

$$\mathsf{Head} \stackrel{\text{def}}{=\!=} (r)\,.\,(\boldsymbol{\nu} n, c)\,(\overline{k}\,\langle n, c\rangle\,.\,c(v, l)\,.\,\overline{v}r)\,,$$
$$\mathsf{Tail} \stackrel{\text{def}}{=\!=} (r)\,.\,(\boldsymbol{\nu} n, c)\,(\overline{k}\,\langle n, c\rangle\,.\,c(v, l)\,.\,\overline{l}r)\,,$$
$$\mathsf{IsEmpty} \stackrel{\text{def}}{=\!=} (\boldsymbol{\nu} n, c)\,(\overline{k}\,\langle n, c\rangle\,.\,(n\,.\,\text{„}Yes\text{”} + c\,.\,\text{„}No\text{”}))\,.$$

A Head és Tail kifejezések az r néven adják az eredményt, az IsEmpty egyszerűen a *Yes*/*No* választ adja.

2.5.29. Példa. *(A $[2, nil]$ lista fejeleme)*

Az előző példában szerepelt a $[2, nil]$ lista folyamatkifejezése, ezt alkalmazzuk a Head kifejezésre. A kommunikációs csatorna legyen a k.

$$\mathsf{Head}\,\langle k\rangle \mid [2, nil]\,\langle k\rangle \equiv$$
$$(\boldsymbol{\nu} n, c)\,(\overline{k}\,\langle n, c\rangle\,.\,c(v, l)\,.\,\overline{v}r) \mid$$
$$(\boldsymbol{\nu} v_1, l_1)\,(\,k(n,c)\,.\,\overline{c}\,\langle v_1, l_1\rangle \mid [\![2]\!]\,\langle v_1\rangle \mid \mathsf{Nil}\,\langle l_1\rangle\,) \equiv$$
$$(\boldsymbol{\nu} n, c)\,(\overline{k}\,\langle n, c\rangle\,.\,c(v, l)\,.\,\overline{v}r \mid$$
$$(\boldsymbol{\nu} v_1, l_1)\,(\,k(n,c)\,.\,\overline{c}\,\langle v_1, l_1\rangle \mid [\![2]\!]\,\langle v_1\rangle \mid \mathsf{Nil}\,\langle l_1\rangle\,)\,) \equiv$$
$$(\boldsymbol{\nu} n, c)\,(\underline{\overline{k}\,\langle n, c\rangle}\,.\,c(v, l)\,.\,\overline{v}r) \mid$$
$$(\boldsymbol{\nu} v_1, l_1)\,(\underline{k(n,c)\,.\,\overline{c}\,\langle v_1, l_1\rangle \mid [\![2]\!]\,\langle v_1\rangle \mid \mathsf{Nil}\,\langle l_1\rangle}\,) \xrightarrow{\tau}$$

$$(\boldsymbol{\nu} n, c)\,(c(v,l)\,.\,\overline{v}r) \mid (\boldsymbol{\nu} v_1, l_1)\,(\overline{c}\,\langle v_1, l_1\rangle \mid [\![2]\!]\,\langle v_1\rangle \mid \mathsf{Nil}\,\langle l_1\rangle\,) \equiv$$
$$(\boldsymbol{\nu} n, c, v_1, l_1)\,(\underline{c(v,l)\,.\,\overline{v}r}) \mid (\underline{\overline{c}\,\langle v_1, l_1\rangle} \mid [\![2]\!]\,\langle v_1\rangle \mid \mathsf{Nil}\,\langle l_1\rangle\,) \xrightarrow{\tau}$$
$$(\boldsymbol{\nu} n, c, v_1, l_1)\,(\underline{\overline{v_1}r} \mid \underline{[\![2]\!]\,\langle v_1\rangle} \mid \mathsf{Nil}\,\langle l_1\rangle\,)\,.$$

A 2.5.8. példában láttuk, hogy $\overline{v}r \mid P\,\langle v\rangle \xrightarrow{\tau} P\,\langle r\rangle$, így

$$(\boldsymbol{\nu} n, c, v_1, l_1)\,(\underline{\overline{v_1}r} \mid \mid \underline{[\![2]\!]\,\langle v_1\rangle} \mid \mathsf{Nil}\,\langle l_1\rangle\,) \xrightarrow{\tau}$$
$$(\boldsymbol{\nu} n, c, v_1, l_1)\,([\![2]\!]\,\langle r\rangle \mid \mathsf{Nil}\,\langle l_1\rangle\,)\,.$$

A korlátozásoknak most már nincs szerepük, a *nil* pedig egy saját l_1 néven van, az r néven megkaptuk az eredményt. □

2.5.30. Példa. *(Az* $\mathsf{IsEmpty}\,[2, nil]$ *kifejezés)*

Képezzük az IsEmpty és a $[2, nil]$ lista kifejezésének kompozícióját. A kommunikációs csatorna most is legyen a k.

$$\mathsf{IsEmpty}\,\langle k\rangle \mid [2, nil]\,\langle k\rangle \equiv$$
$$(\boldsymbol{\nu} n, c)\,(\overline{k}\,\langle n, c\rangle\,.\,(n\,.\,„Yes" + c\,.\,„No")) \mid$$
$$(\boldsymbol{\nu} v_1, l_1)\,(\,k(n,c)\,.\,\overline{c}\,\langle v_1, l_1\rangle \mid [\![2]\!]\,\langle v_1\rangle \mid \mathsf{Nil}\,\langle l_1\rangle\,) \equiv$$
$$(\boldsymbol{\nu} n, c)\,(\overline{k}\,\langle n, c\rangle\,.\,(n\,.\,„Yes" + c\,.\,„No") \mid$$
$$(\boldsymbol{\nu} v_1, l_1)\,(\,k(n,c)\,.\,\overline{c}\,\langle v_1, l_1\rangle \mid [\![2]\!]\,\langle v_1\rangle \mid \mathsf{Nil}\,\langle l_1\rangle\,)\,) \equiv$$
$$(\boldsymbol{\nu} n, c)\,(\underline{\overline{k}\,\langle n, c\rangle}\,.\,(n\,.\,„Yes" + c\,.\,„No") \mid$$
$$(\boldsymbol{\nu} v_1, l_1)\,(\,\underline{k(n,c)\,.\,\overline{c}\,\langle v_1, l_1\rangle \mid [\![2]\!]\,\langle v_1\rangle \mid \mathsf{Nil}\,\langle l_1\rangle}\,)\,) \xrightarrow{\tau}$$
$$(\boldsymbol{\nu} n, c)\,((n\,.\,„Yes" + c\,.\,„No") \mid$$
$$(\boldsymbol{\nu} v_1, l_1)\,(\,\overline{c}\,\langle v_1, l_1\rangle \mid [\![2]\!]\,\langle v_1\rangle \mid \mathsf{Nil}\,\langle l_1\rangle\,)\,) \equiv$$
$$(\boldsymbol{\nu} n, c, v_1, l_1)\,((n\,.\,„Yes" + \underline{c\,.\,„No"}) \mid$$
$$(\,\underline{\overline{c}\,\langle v_1, l_1\rangle} \mid [\![2]\!]\,\langle v_1\rangle \mid \mathsf{Nil}\,\langle l_1\rangle\,)\,) \xrightarrow{\tau}$$
$$(\boldsymbol{\nu} n, c, v_1, l_1)\,((„No") \mid ([\![2]\!]\,\langle v_1\rangle \mid \mathsf{Nil}\,\langle l_1\rangle\,)\,)\,.$$

A lista elemei saját néven vannak, így a továbbiakban nem játszanak szerepet, és megjelent a *No* válasz.

A levezetésből az is látszik, hogy az IsEmpty nem is vizsgálja a lista tartalmát, döntését a Nil és Cons listakonstruktorokban levő $k(n,c)\,.\,\overline{n}$ és $k(nc)\,.\,\overline{c}\,\langle v, l\rangle$ műveletek alapján hozza meg. □

2.5.8. Lambda-kalkulus

Pi-kalkulussal a lambda-kalkulus kifejezései is leírhatók, és így már nem olyan meglepő az előző pontok eredménye, azaz hogy a pi-kalkulus is alkalmas matematikai és informatikai fogalmak pontos megadására.

Egy lambda-kifejezés pi-kalkulusbeli folyamatkifejezésére is a $[\![\ ]\!]$ zárójeleket használjuk, az E lambda-kifejezésnek megfeleltetett pi-kalkulus kifejezés legyen $[\![E]\!]$. (Az 2.5.5. pontban így jelöltük a természetes számok folyamatkifejezéseit is, de ez nem okoz problémát, mert most csak az egyszerű (konstans nélküli) lambda-kalkulussal foglalkozunk.)

Először a *név szerinti*, „call-by-name" lambda-kalkulus átalakítási szabályaival foglalkozunk. Emlékeztetőül megadjuk a kalkulus jellemző szabályait. Az E, F, G kifejezések:

$$\lambda x\,.\,E \mid x \mid E\,F \mid x\ ,$$

és a szabályok:

$$(\lambda x\,.\,E)F \to_\beta E[x := F]\,, \qquad \frac{E \to F}{E\,G \to F\,G}\,, \qquad \frac{E \to F}{G\,E \to G\,F}\,.$$

2.5.31. Definíció. A név szerinti lambda-kalkulus kifejezéseinek folyamatkifejezései:

$$\begin{aligned}
[\![x]\!] &\stackrel{\text{def}}{=\!=} (p)\,.\,\overline{x}p\ ,\\
[\![\lambda x\,.\,E]\!] &\stackrel{\text{def}}{=\!=} (p)\,.\,p(x,q)\,.\,[\![E]\!]\,\langle q\rangle\ ,\\
[\![E\,F]\!] &\stackrel{\text{def}}{=\!=} (p)\,.\,(\boldsymbol{\nu} q)\,(\,[\![E]\!]\,\langle q\rangle \mid (\boldsymbol{\nu} r)\,(\,\overline{q}\,\langle r,p\rangle\,.\,r(s)\,.\,[\![F]\!]\,\langle s\rangle\,)\,)\,.
\end{aligned}$$

A képletekben a $\boldsymbol{\nu}$-vel kötött nevek a hatáskörükben szereplő $[\![E]\!]$ kifejezések szabad nevei.

2.5.32. Példa. *(Az y, $\lambda x\,.\,x$ és $\lambda x\,.\,y$ lambda-kifejezések átalakítása)*

Legyen a kommunikációs csatorna p, ekkor a fenti definíció első és második sorában leírtak szerint

$$[\![y]\!]\,\langle p\rangle \equiv \overline{y}p\ ,$$
$$[\![\lambda x\,.\,x]\!]\,\langle p\rangle \equiv p(x,q)\,.\,\overline{x}q\ ,$$
$$[\![\lambda x\,.\,y]\!]\,\langle p\rangle \equiv p(x,q)\,.\,\overline{y}q\ .$$

□

2.5.33. Példa. *(Határozzuk meg a $(\lambda x\,.\,x)y$ folyamatkifejezését)*

A lambda-kalkulusban $(\lambda x\,.\,x)y \longrightarrow_\beta y$, tehát a pi-kalkulusban is az y folyamatkifejezését várjuk eredményként. Az applikáció átalakítása a fenti definíció harmadik sorában van leírva, így

$$
\begin{aligned}
&[\![(\lambda x\,.\,x)y]\!]\,\langle p\rangle \equiv\\
&(\boldsymbol{\nu} q)\,(\,[\![\lambda x\,.\,x]\!]\,\langle q\rangle \mid (\boldsymbol{\nu} r)\,(\,\overline{q}\,\langle r,p\rangle\,.\,r(s)\,.\,[\![y]\!]\,\langle s\rangle\,)\,) \equiv\\
&(\boldsymbol{\nu} q)\,(\,q(x,w)\,.\,\overline{x}w \mid (\boldsymbol{\nu} r)\,(\,\overline{q}\,\langle r,p\rangle\,.\,r(s)\,.\,\overline{y}s\,)\,) \equiv\\
&(\boldsymbol{\nu} q)\,(\boldsymbol{\nu} r)\,(\,\underline{q(x,w)\,.\,\overline{x}w} \;\mid\; \underline{\overline{q}\,\langle r,p\rangle}\,.\,r(s)\,.\,\overline{y}s\,) \xrightarrow{\tau}\\
&(\boldsymbol{\nu} q)\,(\boldsymbol{\nu} r)\,(\,\underline{\overline{r}p} \mid \underline{r(s)\,.\,\overline{y}s} \;\mid\; r(s)\,.\,\overline{y}s) \xrightarrow{\tau}\\
&(\boldsymbol{\nu} q)\,(\boldsymbol{\nu} r)\,(\overline{y}p \mid\; r(s)\,.\,\overline{y}s)\,.
\end{aligned}
$$

A kompozíció második tagjában az r név korlátozva van, az $r(s)$ input nem hajtódhat végre, így $r(s)\,.\,\overline{y}s \equiv \mathbf{0}$. Tehát eredményként az

$$
\begin{aligned}
&\overline{y}p \equiv\\
&[\![y]\!]\,\langle p\rangle
\end{aligned}
$$

kifejezést kaptuk, ami megfelel a lambda-kalkulusban a $(\lambda x\,.\,x)y \longrightarrow_\beta y$ redukció eredményének. □

2.5.34. Példa. *(Határozzuk meg a $(\lambda x\,.\,y)z$ folyamatkifejezését)*

$$
\begin{aligned}
&[\![(\lambda x\,.\,y)z]\!]\,\langle p\rangle \equiv\\
&(\boldsymbol{\nu} q)\,(\,[\![\lambda x\,.\,y]\!]\,\langle q\rangle \mid (\boldsymbol{\nu} r)\,(\,\overline{q}\,\langle r,p\rangle\,.\,r(s)\,.\,[\![z]\!]\,\langle s\rangle\,)\,) \equiv\\
&(\boldsymbol{\nu} q)\,(\,q(x,w)\,.\,\overline{y}w \mid (\boldsymbol{\nu} r)\,(\,\overline{q}\,\langle r,p\rangle\,.\,r(s)\,.\,\overline{z}s\,)\,) \equiv\\
&(\boldsymbol{\nu} q)\,(\boldsymbol{\nu} r)\,(\,\underline{q(x,w)\,.\,\overline{y}w} \;\mid\; \underline{\overline{q}\,\langle r,p\rangle}\,.\,r(s)\,.\,\overline{z}s\,) \xrightarrow{\tau}\\
&(\boldsymbol{\nu} q)\,(\boldsymbol{\nu} r)\,(\,\overline{y}p \mid r(s)\,.\,\overline{y}s\,)\,.
\end{aligned}
$$

A kompozíció második tagjában az r név korlátozva van, az $r(s)$ input nem hajtódhat végre, így $r(s)\,.\,\overline{y}s \equiv \mathbf{0}$. Tehát

$$
\begin{aligned}
&(\boldsymbol{\nu} q)\,(\boldsymbol{\nu} r)\,(\,\overline{y}p \mid r(s)\,.\,\overline{y}s\,) \equiv\\
&\overline{y}p \equiv\\
&[\![y]\!]\,\langle p\rangle\;.
\end{aligned}
$$

Az eredmény y lett, ami megfelel a lambda-kalkulusban a $(\lambda x\,.\,y)z \longrightarrow_\beta y$ redukció eredményének. □

Most nézzük meg az *érték szerinti*, „call-by-value” szabályait. Emlékeztetőül most is megadjuk a kalkulus jellemzőit, V az *értéket* jelöli:

$V ::= \lambda x\,.\,E \mid x\,,$

és a szabályok:

$$(\lambda x\,.\,E)V \to_\beta E[x := V]\,, \qquad \frac{E \to F}{E\,G \to F\,G}\,, \qquad \frac{E \to F}{V\,E \to V\,F}\,.$$

Az E kifejezés átalakítását $[\![E]\!]_v$-vel jelöljük.

2.5.35. Definíció. Az érték szerinti lambda-kalkulus kifejezéseinek folyamatkifejezései:

$$\begin{array}{lcl} [\![x]\!]_v & \stackrel{\text{def}}{=\!=} & (p)\,.\,\overline{p}x\,, \\ [\![\lambda x\,.\,E]\!]_v & \stackrel{\text{def}}{=\!=} & (p)\,.\,(\boldsymbol{\nu} y)\,(\overline{p}y\,.\,(\boldsymbol{\nu} q)\,(y(x,q)\,.\,[\![E]\!]_v\,\langle q\rangle))\,, \\ [\![E\,F]\!]_v & \stackrel{\text{def}}{=\!=} & (p)\,.\,(\boldsymbol{\nu} q)\,([\![E]\!]_v\,\langle q\rangle \mid q(v)\,.\,(\boldsymbol{\nu} r)\,([\![F]\!]_v\,\langle r\rangle \mid r(w)\,.\,\overline{v}\,\langle w,p\rangle))). \end{array}$$

A képletekben a $\boldsymbol{\nu}$-vel kötött nevek a hatáskörükben szereplő $[\![E]\!]_v$ kifejezések szabad nevei.

2.5.36. Példa. *(Az y, $\lambda x\,.\,x$ és $\lambda x\,.\,y$ lambda-kifejezések)*

Legyen a kommunikációs csatorna p, ekkor

$[\![y]\!]_v\,\langle p\rangle \equiv \overline{p}y\,,$
$[\![\lambda x\,.\,x]\!]_v\,\langle p\rangle \equiv (\boldsymbol{\nu} y)\,(\overline{p}y\,.\,(\boldsymbol{\nu} q)\,(y(x,q)\,.\,\overline{q}x\,)\,)\,,$
$[\![\lambda x\,.\,y]\!]_v\,\langle p\rangle \equiv (\boldsymbol{\nu} y)\,(\overline{p}y\,.\,(\boldsymbol{\nu} q)\,(y(x,q)\,.\,\overline{q}y\,)\,)\,.$ □

2.5.37. Példa. *(Határozzuk meg a $(\lambda x\,.\,x)t$ folyamatkifejezését)*

A lambda-kalkulusban $(\lambda x\,.\,x)t \longrightarrow_\beta t$, tehát a pi-kalkulusban a t folyamatkifejezését várjuk eredményként.

$[\![(\lambda x\,.\,x)t]\!]_v\,\langle p\rangle \equiv$
$(\boldsymbol{\nu} q)\,((\boldsymbol{\nu} y)\,(\overline{q}y\,.\,(\boldsymbol{\nu} s)\,(y(x,s)\,.\,\overline{q}x)) \mid q(v)\,.\,(\boldsymbol{\nu} r)\,(\underline{\overline{r}t} \mid \underline{r(w)\,.\,\overline{v}\,\langle w,p\rangle}\,)) \stackrel{\tau}{\longrightarrow}$
$(\boldsymbol{\nu} q,y)\,(\underline{\overline{q}y}\,.\,(\boldsymbol{\nu} s)\,(y(x,s)\,.\,\overline{q}x) \mid \underline{q(v)\,.\,\overline{v}\,\langle t,p\rangle}\,) \stackrel{\tau}{\longrightarrow}$
$(\boldsymbol{\nu} q,y,s)\,(\underline{y(x,s)\,.\,\overline{q}t} \mid \underline{\overline{y}\,\langle t,p\rangle}\,) \stackrel{\tau}{\longrightarrow}$
$(\boldsymbol{\nu} q,y,s)\,\overline{q}t \equiv$
$\overline{q}t\,.$ □

2.6. Biszimuláció

A pi-kalkulus egyik alapvető feladata folyamatok kongruenciájának, azaz egyezőségének a vizsgálata. A vizsgálatra két alapvető módszer alakult ki,

- nyomkövetésen és
- megfigyelésen

alapuló ekvivalencia-vizsgálat. Először a nyomkövetésekkel foglalkozunk.

2.6.1. Nyomkövetésen alapuló ekvivalencia

Egy P folyamat nyomkövetései megadják azokat az akciósorozatokat, amelyek a P-ből kiindulva véges lépésben elvezetnek egy, a P-től nem feltétlenül különböző folyamatba.

2.6.1. Definíció. Nyomkövetés:

Legyen $A = (S, L, T)$ egy címkézett átmeneti rendszer, és legyen $P \in S$ egy folyamat. Ha az $\alpha = a_1 \dots a_n$, $a_i \in L$ $(1 \leq i \leq n)$ sorozatra

$$P \xrightarrow{a_1} P_1 \dots \xrightarrow{a_n} P_n \,,$$

akkor α-t a P egy nyomkövetésének nevezzük.

Egy P folyamatnak több nyomkövetése is lehet, $Trace(P)$ jelölje a P nyomkövetéseinek halmazát.

2.6.2. Definíció. Nyomkövetési ekvivalencia:

Azt mondjuk, hogy a P és Q folyamatok a nyomkövetést tekintve ekvivalensek, ha $Trace(P) = Trace(Q)$.

A nyomkövetési ekvivalenciát a $\simeq$ jellel jelöljük, azaz ha P és Q a nyomkövetést tekintve ekvivalensek, akkor $P \simeq Q$.

Megjegyezzük, hogy ha P-nek és P_n-nek az automaták elméletében az automata egy s_P és s_{P_n} állapotát, az akcióknak az automata ábécéjét feleltetjük meg, akkor a P folyamat α nyomkövetése az automata $s_P \xrightarrow{\alpha} s_{P_n}$ átmenetének felel meg. Ha s_P az automata kezdőállapota és minden nyomkövetésre az $s_{P_{n_i}}$ $(i = 1, 2, \dots)$ állapotok végállapotok, akkor $Trace(P)$ az automata végállapottal felismert nyelvét adja meg.

2.6.3. Példa. *(A tea- és kávéautomata)*

Tegyük fel, hogy van egy olyan, a köznyelvben automatának nevezett

készülékünk, amely egy adott értékű, például 50 Ft-os pénzérme bedobására teát, két pénzérme egymás utáni bedobására kávét ad. Az automata működésére a címkézett átviteli rendszer leírásával kétféle megoldást is tudunk adni (2.3. ábra).

Azonnal látható, hogy az első automata P_1 állapotának nyomkövetése

$$Trace(P_1) = (\ 50Ft\,.\,(\overline{tea} + 50Ft\,.\,\overline{kávé}))^* \ ,$$

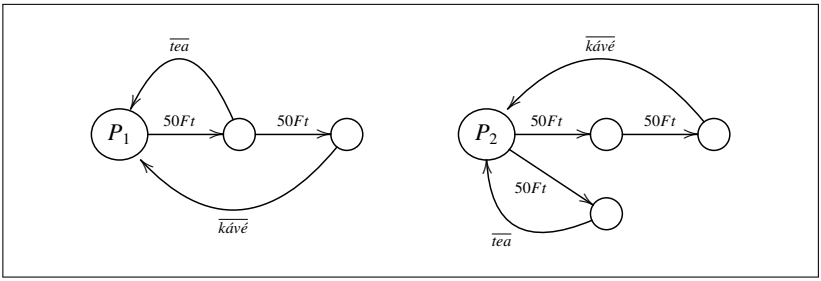
a második automata P_2 állapotának nyomkövetése

$$Trace(P_2) = (\ 50Ft\,.\,\overline{tea} + 50Ft\,.\,50Ft\,.\,\overline{kávé})^* \ ,$$

ahol

$$P^* \quad = \begin{cases} \underbrace{PP\ldots P}_{n}, & \text{ha } n > 0\ , \\ \varepsilon, & \text{ha } n = 0\ . \end{cases}$$

Látható, hogy P_1 és P_2 nyomkövetése azonos, azaz $Trace(P_1) = Trace(P_2)$. □



2.3. ábra. Az automaták

Az automaták és a formális nyelvek elméletében két automatát ekvivalensnek tekintünk, ha ugyanazt a nyelvet fogadják el. Az ekvivalenciának ez a definíciója számunkra nem lesz jó, például azért, mert az automatáknak általában nincs kezdőállapotuk és végállapotuk, és két automata egymástól lényegesen különbözhet még akkor is, ha nyomkövetésük azonos.

2.6.4. Példa. *(A kávéautomaták különbözősége)*

Tekintsük az előző 2.6.3. példában szereplő automatákat. Ha felírjuk a kávét

ivó vásárló folyamatát:

$\overline{50Ft}\,.\,\overline{50Ft}\,.\,kávé$,

akkor felhasználva a $P^* \equiv P\ P^*$ szabályt, az első automatával:

$Trace(P_1) \ \mid\ \overline{50Ft}\,.\,\overline{50Ft}\,.\,kávé \equiv$

$\underline{50Ft}\,.\,(\overline{tea} + 50Ft\,.\,\overline{kávé})\,.\,Trace(P_1) \ \mid\ \underline{\overline{50Ft}}\,.\,\overline{50Ft}\,.\,kávé \xrightarrow{\tau}$

$(\overline{tea} + 50Ft\,.\,\overline{kávé})\,.\,Trace(P_1) \ \mid\ \overline{50Ft}\,.\,kávé \ \equiv$

$(\overline{tea}\,.\,Trace(P_1) + \underline{50Ft}\,.\,\overline{kávé}\,.\ Trace(P_1)) \ \mid\ \underline{\overline{50Ft}}\,.\,kávé \xrightarrow{\tau}$

$\underline{\overline{kávé}}\,.\,Trace(P_1) \ \mid\ \underline{kávé} \xrightarrow{\tau}$

$Trace(P_1)$,

azaz semmilyen probléma nem lép fel. De a második automata esetén akár holtpont is előállhat. A vásárló folyamatát a második automata kifejezésével párhuzamosan futtatva előfordulhat a

$Trace(P_2) \ \mid\ \overline{50Ft}\,.\,\overline{50Ft}\,.\,kávé \equiv$

$(50Ft\,.\,\overline{tea} + 50Ft\,.\,50Ft\,.\,\overline{kávé})\,.\,Trace(P_2) \ \mid\ \overline{50Ft}\,.\,\overline{50Ft}\,.\,kávé \equiv$

$(\underline{50Ft}\,.\,\overline{tea}\,.\,Trace(P_2) + 50Ft\,.\,50Ft\,.\,\overline{kávé}\,.\,Trace(P_2)) \ \mid$

$\underline{\overline{50Ft}}\,.\,\overline{50Ft}\,.\,kávé \xrightarrow{\tau}$

$\overline{tea}\,.\,Trace(P_2) \ \mid\ \overline{50Ft}\,.\,kávé \xrightarrow{\overline{tea}}$

$Trace(P_2) \ \mid\ \overline{50Ft}\,.\,kávé \xrightarrow{\overline{50Ft}}$

$Trace(P_2) \ \mid\ kávé$

levezetés is. A vásárló kávé helyett – teljesen váratlanul – teát kapott, amit nem is fogadott, majd bedobta a második 50 FT-os érmét is. További műveletre itt most már nincs lehetőség, és ez úgy értelmezhető, hogy az automata és a vásárló folyamata holtpontra jutott. □

A nyomkövetési ekvivalencia problémáját vizsgálták úgy is, hogy a nyomkövetésekben csak véges sorozatokat engedtek meg. Ezeket a sorozatokat a szakirodalomban *befejezett nyomkövetéseknek*, az ekvivalenciát pedig *befejezett nyomkövetési ekvivalenciának* nevezték [15].

Vizsgálták még a nyomkövetéseknek azt a speciális esetét is, ahol a τ átmenetek nem számítottak bele a nyomkövetésbe. Az ilyen nyomkövetéseken alapuló ekvivalenciát *gyenge nyomkövetési ekvivalenciának* nevezték el [15]. Azonban ezek a rendszerek sem voltak megfelelőek a folyamatok kongruenciájának vizsgálatára, azaz a folyamatkifejezésekkel leírt automaták megkülönböztetésére.

Az automaták azonosságának vizsgálatában azt szeretnénk, hogy ha két automata, két „gép" *hasonló* állapotban van és az egyik végrehajt valamit, azt a másik is végrehajthassa és mindkettő újra *hasonló* állapotba kerüljön. Folyamatok ekvivalenciájának ezt az elvét, azaz a kölcsönös hasonlóságot vizsgáljuk a következő pontokban.

2.6.2. Biszimuláció és kongruencia

A folyamatok kongruenciájának vizsgálata a biszimuláción alapul. A klasszikus folyamatkalkulusok, mint például a CCS, a következő definíciót használják:

Legyen (S, L, T) egy címkézett átmeneti rendszer, és legyen $\mathcal{R}$ egy bináris reláció az S halmaz felett. Ha $P, P', Q \in S$, $P\,\mathcal{R}\,Q$, és minden α-ra $P \xrightarrow{\alpha} P'$ esetén van olyan $Q' \in S$, melyre $Q \xrightarrow{\alpha} Q'$ és $P'\,\mathcal{R}\,Q'$, akkor az $\mathcal{R}$ relációt az LTS feletti *szimulációnak* nevezzük, és azt mondjuk, hogy a P folyamatot a Q folyamat *szimulálja*. A Q folyamat tehát a működés szempontjából a P-hez *hasonló*. A szimuláció szemléletesen a 2.4. ábrán látható.

$$\begin{tikzcd}
P \arrow[r, dotted, no head] \arrow[d, "\alpha"] & \mathcal{R} \arrow[r, dotted, no head] & Q \arrow[d, "\alpha"] \\
P' \arrow[r, dotted, no head] & \mathcal{R} \arrow[r, dotted, no head] & Q'
\end{tikzcd}$$

2.4. ábra. Szimuláció

Az $\mathcal{R}$ reláció konverziójának jele $\mathcal{R}^{-1}$, azaz ha $P\,\mathcal{R}\,Q$, akkor legyen $Q\,\mathcal{R}^{-1}\,P$. Ha $\mathcal{R}$ és $\mathcal{R}^{-1}$ mindegyike szimuláció, akkor $\mathcal{R}$-t kölcsönös szimulációnak, *biszimulációnak* nevezzük. Ez azt is jelenti, hogy ebben az esetben $\mathcal{R}$ egy *szimmetrikus bináris reláció*.

Megjegyezzük, hogy ha $P\,\mathcal{R}\,Q$ és van olyan $\mathcal{R}'$ reláció, melyre $Q\,\mathcal{R}'\,P$,

biszimulációról csak akkor beszélhetünk, ha $\mathcal{R}^{-1} \equiv \mathcal{R}'$.

A pi-kalkulusban a szimuláció és biszimuláció definíciója nem olyan egyszerű, mint a CCS-ben. Ennek oka az, hogy az $x(y)$ input prefix esetén az y kötött lesz a prefix utáni folyamatban, és például egy $x(y) \,.\, P$ kifejezés esetén a P folyamat egy „y változós függvényként" viselkedik. Ha ezt egy Q szimulálja, akkor a Q-nak szimulálni kell a P-t az y formális paraméter minden aktuális értékére. Ezért a szimuláció definíciójában meg kell különböztetnünk az input és a nem-input prefixeket.

2.6.5. Példa. *(Az input prefix és a szimuláció)*

Legyen

$P \equiv x(y)$,

$Q \equiv x(z) \,.\, (\boldsymbol{\nu} w)\, \overline{w} y$.

Mindkét folyamat, ha az input műveletet végrehajtja, a $\mathbf{0}$ folyamattá válik, hiszen a Q-ban a w egy saját név, és a saját névre az adatküldés már nem figyelhető meg. Tehát a P és Q megfigyelhető eseménye azonos. Azonban látható, hogy a $P \xrightarrow{x(y)}$ átmenetnek nincs a Q-hoz megfelelő átmenete, hiszen a $z := y$ α-konverzió nem hajtható végre, mivel ekkor a $(\boldsymbol{\nu} w)\, \overline{w} y$-ban a szabad y kötötté válna. □

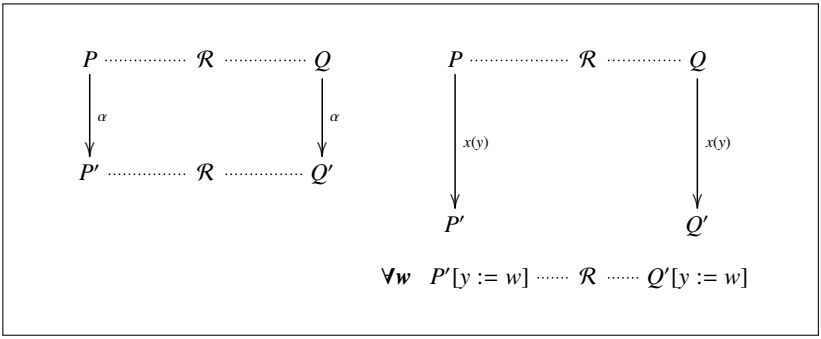
2.6.6. Definíció. Biszimuláció:

> *Legyen $\mathcal{R}$ egy szimmetrikus bináris reláció az S halmaz felett, és legyen $P \mathcal{R} Q$. Az $\mathcal{R}$ biszimuláció,*
>
> - *ha $P \xrightarrow{\alpha} P'$, $\alpha \neq x(y)$ és $bn(\alpha) \notin fn(P, Q)$, akkor $Q \xrightarrow{\alpha} Q'$ esetén $P' \mathcal{R} Q'$,*
> - *ha $P \xrightarrow{x(y)} P'$, $y \notin fn(P, Q)$, akkor $Q \xrightarrow{x(y)} Q'$ esetén minden w-re $P'[y := w] \mathcal{R} Q'[y := w]$.*

A biszimulációt *erős biszimulációnak* is nevezzük azért, mert a címkézett $\xrightarrow{\alpha}$ átmenetekben a τ-val jelzett belső kommunikációkat is figyelembe vesszük.

Ennek a biszimulációnak egy másik jelzője a *késői*, vagy ahogyan a 2.3.1. definíció utáni megjegyzésben utaltunk rá, a „lusta" biszimuláció. Az elnevezés abból adódik, hogy az input prefixből kapott név csak a kommunikációkor helyettesítődik a prefixet követő folyamatba. Felhívjuk a figyelmet arra is, hogy a P és Q folyamatokhoz egy P' és egy Q' folyamatot rendelünk, és

ezekre mondjuk azt, hogy minden w helyettesítésére a helyettesített folyamatoknak $\mathcal{R}$ relációban kell lenniük. A 2.6.6. definícióban szereplő átmenetek a 2.5. ábrán láthatók.



2.5. ábra. Biszimuláció

2.6.7. Definíció. Kölcsönös hasonlóság:

A kölcsönös hasonlóság legyen a biszimulációk uniója, és a kölcsönös hasonlóság jele legyen $\dot{\sim}$ *. Azt mondjuk, hogy P és Q folyamatok kölcsönösen hasonlóak, azaz* $P \dot{\sim} Q$*, ha van olyan* $\mathcal{R}$ *biszimuláció, melyre* $P \mathcal{R} Q$*.*

2.6.8. Tétel. (A kölcsönös hasonlóság tulajdonságai - 1.)

A kölcsönös hasonlóság

- *ekvivalencia reláció, azaz reflexív, szimmetrikus és tranzitív,*
- *önmaga is egy biszimuláció.*

Könnyen belátható, hogy két biszimuláció uniója is biszimuláció. Mivel a kölcsönös hasonlóság biszimulációk uniója, a tétel állításai könnyen igazolhatóak [45].

A kölcsönös hasonlóságot azonban az input prefix nem őrzi meg, azaz ha két folyamat kölcsönösen hasonló, az eléjük írt input prefixszel képzett új folyamatok nem feltétlenül lesznek kölcsönösen hasonlóak.

2.6.9. Példa. *(Biszimuláció és az input prefix)*

Vizsgáljuk az $a \mid \overline{b}$ és az $a\,.\,\overline{b} + \overline{b}\,.\,a$ folyamatokat. Azt találjuk, hogy

$a \mid \overline{b} \;\dot{\sim}\; a\,.\,\overline{b} + \overline{b}\,.\,a\,.$

Ugyanakkor ha a kifejezések elé helyezünk egy $c(a)$ prefixet,

$c(a)\,.\,a \mid \overline{b}, \quad c(a)\,.\,(a\,.\,\overline{b} + \overline{b}\,.\,a)\,,$

a kölcsönös hasonlóság megszűnik, hiszen ha az a néven például a b-t olvassuk, akkor a kifejezések

$b \mid \overline{b}, \quad b\,.\,\overline{b} + \overline{b}\,.\,b$

kifejezésekké válnak, és látható, hogy a bal oldali kifejezésben kommunikáció történik, míg a jobb oldali kifejezésben erre nincs lehetőség.

Nyilvánvaló, hogy a

$b \mid \overline{b} \;\dot{\sim}\; b\,.\,\overline{b} + \overline{b}\,.\,b + \tau$ kapcsolat lenne a helyes. □

2.6.10. Tétel. ***(A kölcsönös hasonlóság tulajdonságai - 2.)***

Ha $P \;\dot{\sim}\; Q$, akkor

- *a P és Q folyamatkifejezések kölcsönös hasonlósága az input prefix kivételével minden operátor alkalmazásakor megmarad,*
- *a σ injektív helyettesítésre $P\sigma \;\dot{\sim}\; Q\sigma$ is fennáll.*

(Megjegyzés: Az $f : A \to B$ leképezés injektív, ha $a, b \in A$ és $f(a) = f(b)$ esetén $a = b$.)

2.6.11. Példa. *(A kölcsönös hasonlóság megmarad)*

A tétel alapján, ha $P \;\dot{\sim}\; Q$, akkor

$(\nu x)\,P \;\dot{\sim}\; (\nu x)\,Q\,,$

és ha még $P' \;\dot{\sim}\; Q'$ is fennáll, akkor

$(\nu x)\,(P \mid P') \;\dot{\sim}\; (\nu x)\,(Q \mid Q')$

is teljesül. □

A kölcsönös hasonlóság tulajdonság a helyettesítés műveletre azonban nem marad meg, ha $P \;\dot{\sim}\; Q$, akkor nincs garancia arra, hogy $P\sigma \;\dot{\sim}\; Q\sigma$ is teljesül. A szakirodalomban az ilyen típusú relációt *alaprelációnak* nevezik.

2.6.12. Példa. *(Biszimuláció és a helyettesítés)*

Nyilvánvaló, hogy az $[x = y]\overline{x}z$ és a $\mathbf{0}$ folyamatkifejezések kölcsönösen hasonlóak. De ha mindkét kifejezésre végrehajtunk egy $[y := x]$ helyettesítést, a kölcsönös hasonlóság megszűnik, hiszen az $[x = x]$ azonosság prefix *igaz* lesz, és a $\overline{x}z$ művelet végrehajtódik. □

Szeretnénk találni és definiálni a kölcsönös hasonlósággal kapcsolatban egy olyan relációt, amely *kongruencia*, azaz amelyik megmarad a folyamatokra alkalmazott minden *operátor* esetén.

A 2.3.17. definícióban már megadtunk egy kongruencia értelmezést a *szerkezeti* kongruenciára a kontextusok felhasználásával. Most a kölcsönös hasonlóságot tekintve definiáljuk a kongruens kifejezéseket.

Ezt a kongruenciát a biszimuláció jelzőivel *erős késői kongruenciának* is nevezhetjük.

2.6.13. Definíció. Kongruencia:

A P és Q folyamat a kölcsönös hasonlóságot tekintve kongruens, ha minden σ helyettesítésre $P\sigma \;\dot{\sim}\; Q\sigma$. A folyamatok kongruenciájának a jele $\sim$, tehát ekkor $P \sim Q$.

A kölcsönös hasonlóság tehát nem kongruencia, erre utal a kölcsönös hasonlóság jelében a $\sim$ szimbólumon a pont jel.

2.6.14. Példa. *(Kongruens kifejezések)*

A 2.6.9. példában láttuk, hogy

$$a \mid \overline{b} \;\dot{\sim}\; a\,.\,\overline{b} + \overline{b}\,.\,a\ .$$

A $[b := a]$ helyettesítés a kölcsönös hasonlóságot megszünteti, mivel

$$a \mid \overline{a} \qquad a\,.\,\overline{a} + \overline{a}\,.\,a\ ,$$
$$\tau \;\not\sim\; a\,.\,\overline{a} + \overline{a}\,.\,a\ ,$$

tehát a két folyamat nem kongruens. Azonban azonnal látszik, hogy

$$a \mid \overline{b} \sim a\,.\,\overline{b} + \overline{b}\,.\,a + [a = b]\tau\ ,$$

vagyis ez a két folyamat kongruens lesz. □

2.6.15. Tétel. (A legnagyobb kongruencia)

A $\sim$ a kölcsönös hasonlóságot tekintve a legnagyobb kongruencia.

A tételben két állítást adunk, az egyik az, hogy a $\sim$ egy kongruencia, a másik pedig az, hogy ez a legnagyobb kölcsönös hasonlóság. Megjegyezzük, hogy a „legnagyobb” jelző halmazelméleti értelemben jelenti a legnagyobbat, azaz azt, hogy a $\sim$ kongruencia vonatkozik a legtöbb olyan P, Q párra, melyre $P \sim Q$, vagyis $P\sigma \mathrel{\dot{\sim}} Q\sigma$. A tétel állításait a következő két példában magyarázzuk meg.

2.6.16. Példa. *(A $\sim$ a kölcsönös hasonlóságot tekintve kongruencia)*

A $\sim$ valóban kongruencia, hiszen az input prefix kivételével a művelettartás már a kölcsönös hasonlóság tulajdonságban benne van (2.6.10. tétel), az input prefixre pedig azt kell belátnunk, hogy $P, Q \in \sim$ esetén $\{x(y).P\,, x(y).Q\} \in \sim$ is fennáll. Az

$$x(y).P \xrightarrow{x(y)} P\,, \quad x(y).Q \xrightarrow{x(y)} Q$$

akciók végrehajtása után a 2.6.6. definíció szerint a

$\forall w \quad P[x := w]\,, Q[x := w]$ kifejezéseket kapjuk. $\{P, Q\} \in \sim$, és a $\sim$ definíciója alapján így minden w-re valóban

$$P[x := w] \sim Q[x := w]\,. \qquad \square$$

2.6.17. Példa. *(A $\sim$ a legnagyobb kongruencia)*

Be kell látnunk, hogy a $\sim$ a kölcsönös hasonlóságot tekintve a legnagyobb kongruencia. Tegyük fel, hogy van a $\mathrel{\dot{\sim}}$ kölcsönös hasonlóságban egy $\sim'$ kongruencia is, ekkor azt kell belátnunk, hogy $P \sim' Q$ esetén a $P\sigma \mathrel{\dot{\sim}} Q\sigma$ is fennáll. Ezt a következő esetre mutatjuk meg.

Legyen $\sigma = [x_1 := y_1, \ldots, x_n := y_n]$. Mivel $P \sim' Q$ és $\sim'$ egy kongruencia, a kongruencia definíciója alapján

$$(\boldsymbol{\nu} z)(z(x_1, \ldots, x_n).P \mid \overline{z}\langle y_1, \ldots y_n\rangle) \sim' (\boldsymbol{\nu} z)(z(x_1, \ldots, x_n).Q \mid \overline{z}\langle y_1, \ldots y_n\rangle)\,,$$

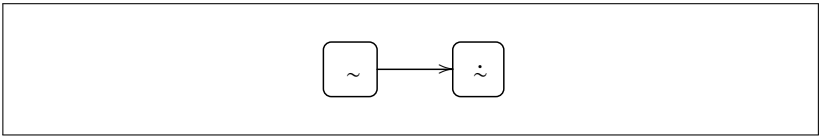
és a kommunikációk végrehajtása után a

$$P\,[x_1 := y_1, \ldots, x_n := y_n] \sim' Q\,[x_1 := y_1, \ldots, x_n := y_n]$$

eredményt kapjuk, azaz $P\sigma \sim' Q\sigma$. Mivel feltettük, hogy $\sim' \subseteq \mathrel{\dot{\sim}}$, a $P\sigma \mathrel{\dot{\sim}} Q\sigma$ is teljesül. $\square$

Jelölje $A \longrightarrow B$ azt a tulajdonságot, hogy az A-ra szigorúbb megkötések vannak megadva, mint B-re, azaz vannak olyan folyamatok, amelyeket az A meg tud különböztetni, de a B nem, vagyis az A finomabb osztályozást tesz lehetővé. Ezzel a jelöléssel a 2.6.15. tétel állítása a 2.6. ábrán látható. Az ábrát

a következő pontokban újabb kongruenciákkal és kölcsönös hasonlóságokkal fogjuk bővíteni.



2.6. ábra. A kongruencia és kölcsönös hasonlóság

2.6.3. Korai szemantika és korai biszimuláció

Mint a 2.3. szakaszban utaltunk rá, az input műveletek végrehajtásának időpontjától függően megkülönböztetünk „késői” és „korai” szemantikát, ami lényegében a funkcionális paradigma *lusta* és *mohó* kiértékelési stratégiájához hasonló.

Eddig mindenhol a *késői szemantikát* használtuk, bár a „késői” jelzőt nem tettük ki. Ezután az egyértelműség érdekében a „késői” és „korai” jelzőt mindenhol kiírjuk.

Ebben a szakaszban először a korai műveleti szemantikával foglalkozunk, majd megadjuk a korai biszimuláció és korai kölcsönös hasonlóság definícióját és a rájuk vonatkozó tulajdonságokat.

A szemantika

A késői és korai rendszerek között lényeges különbség a kommunikáló folyamatokra vonatkozó szabályban van, ezért emlékeztetőül most megismételjük a késői szemantika Com szabályát:

$$\frac{P \xrightarrow{\overline{x}y} P', \quad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'[z := y]} \quad [\text{Com}]$$

2.9. táblázat. A késői szemantika Com szabálya

Mint a 2.3.1. definícióban szerepelt, az $x(z).Q$ kifejezésben, ha majd a folyamat az x néven fogad például egy y nevet, akkor a Q-beli z nevek y-ra helyettesítődnek. A z tehát egy formális paraméternek tekinthető, és y lesz a

z paraméter aktuális értéke. Felhívjuk a figyelmet arra, hogy a helyettesítés itt csak akkor történik meg, ha az $x(z) \,.\, Q$ folyamat az x néven egy másik folyamattal kommunikációt végez.

Definiálható egy másik lehetőség is az $x(z)$ prefix végrehajtására, ezt az Early-Input szabályban adjuk meg, amely a 2.10. táblázatban látható.

$$\frac{}{x(z) \,.\, P \xrightarrow{xw} P[z := w]} \quad [\text{Early-Input}]$$

2.10. táblázat. A korai szemantika Input szabálya

A szabályban egy újabb, már az ötödik fajta prefix jelenik meg, az *xw szabad input*, ami azt jelenti, hogy az x néven jövő w adat azonnal behelyettesítődik a P folyamat z neveibe. Tehát a helyettesítés megtörténik, ha a w az x néven megjelenik, nincs szükség az x néven outputot adó másik folyamattal való τ kommunikációra.

Az xw tehát a szabad inputot jelöli, az $x(y)$ prefixet pedig a továbbiakban megkülönböztetésül *kötött* input prefixnek nevezzük. Emlékeztetőül megismételjük a 2.4. táblázatot, és kibővítjük a szabad input prefix adataival:

	bn	fn
$\overline{x}y \,.\, P$	$bn(P)$	$\{x, y\} \cup fn(P)$
$\overline{x}(y) \,.\, P$	$\{y\} \cup bn(P)$	$\{x\} \cup fn(P)$
$x(y) \,.\, P$	$\{y\} \cup bn(P)$	$\{x\} \cup fn(P)$
$xy \,.\, P$	$bn(P)$	$\{x, y\} \cup fn(P)$

2.11. táblázat. Az output és input prefixek szabad és kötött nevei

$$\frac{P \xrightarrow{\overline{x}y} P', \quad Q \xrightarrow{xw} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \quad [\text{Early-Com}]$$

2.12. táblázat. A korai szemantika Com szabálya

A „korai” elnevezés magyarázata a kommunikációra vonatkozó Early-

Com szabályból látható (2.12. táblázat), nincs helyettesítés a következményben, mert a helyettesítés egy korábbi műveletben, a szabad input feltételben (a $Q \xrightarrow{xw} Q'$-ban) már megtörtént.

A kommunikáció eredményét nézve nincs különbség a korai és késői τ művelet között. Ezt mutatjuk meg a következő példában.

2.6.18. Példa. *(Késői és korai input művelet)*

Nézzük az $\overline{x}y\,.\,P$ és $x(z)\,.\,Q$ folyamatok kommunikációját, először a késői szemantika felhasználásával.

$$\dfrac{\dfrac{}{\overline{x}y\,.\,P \xrightarrow{\overline{x}y} P}\,[\textsc{Prefix}] \qquad \dfrac{}{x(z)\,.\,Q \xrightarrow{x(z)} Q}\,[\textsc{Prefix}]}{\overline{x}y\,.\,P \mid x(z)\,.\,Q \xrightarrow{\tau} P \mid Q[z := y]}\,[\textsc{Com}]$$

A korai szemantikával:

$$\dfrac{\dfrac{}{\overline{x}y\,.\,P \xrightarrow{\overline{x}y} P}\,[\textsc{Prefix}] \qquad \dfrac{}{x(z)\,.\,Q \xrightarrow{xy} Q[z := y]}\,[\textsc{Early-Input}]}{\overline{x}y\,.\,P \mid x(z)\,.\,Q \xrightarrow{\tau} P \mid Q[z := y]}\,[\textsc{Early-Com}]$$

Látható, hogy a két szemantika alkalmazásával kapott eredmény pontosan megegyezik. □

A korai szemantika tehát két szabályban különbözik a késői szemantika szabályaitól. Mivel a típusos pi-kalkulus vizsgálatánál hivatkozni fogunk a korai szemantikára, a 2.13. táblázatban összefoglaljuk a korai szemantika szabályait.

$$\frac{P' \equiv P, \quad P \xrightarrow{\alpha} Q, \quad Q \equiv Q'}{P' \xrightarrow{\alpha} Q'} \quad [\textsc{Struct}]$$

$$\frac{}{\overline{x}y\,.\,P \xrightarrow{\overline{x}y} P} \quad [\textsc{Output}] \qquad \frac{\overline{x}y\,.\,P \xrightarrow{\overline{x}y} P, \quad x \neq y}{\overline{x}(y)\,.\,P \xrightarrow{\overline{x}(y)} P} \quad [\textsc{Open}]$$

$$\frac{}{x(y)\,.\,P \xrightarrow{xw} P[y := w]} \quad [\textsc{Early-Input}] \qquad \frac{}{\tau\,.\,P \xrightarrow{\tau} P} \quad [\textsc{Tau}]$$

$$\frac{\alpha\,.\,P \xrightarrow{\alpha} P}{[x = x]\alpha\,.\,P \xrightarrow{\alpha} P} \quad [\textsc{Match}]$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \quad [\textsc{Sum}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad bn(\alpha) \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad [\textsc{Par}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad x \notin n(\alpha)}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'} \quad [\textsc{Res}]$$

$$\frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \quad [\textsc{Rep}]$$

$$\frac{P \xrightarrow{\overline{x}y} P', \quad Q \xrightarrow{xw} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \quad [\textsc{Early-Com}]$$

$$\frac{\overline{x}(y)\,.\,P \xrightarrow{\overline{x}(y)} P, \quad x(z)\,.\,Q \xrightarrow{xy} Q[z := y]}{\overline{x}(y)\,.\,P \mid x(z)\,.\,Q \xrightarrow{\tau} (\nu y)(P \mid Q[z := y])} \quad [\textsc{Close}]$$

2.13. táblázat. A korai műveleti szemantika szabályai

Biszimuláció

Először most is a biszimulációt definiáljuk, és csak ezután adjuk meg a kölcsönös hasonlóság fogalmát.

2.6.19. Definíció. Korai biszimuláció:

Legyen $\mathcal{R}$ egy szimmetrikus bináris reláció az S halmaz felett, és legyen $P\,\mathcal{R}\,Q$. Az $\mathcal{R}$ korai biszimuláció,

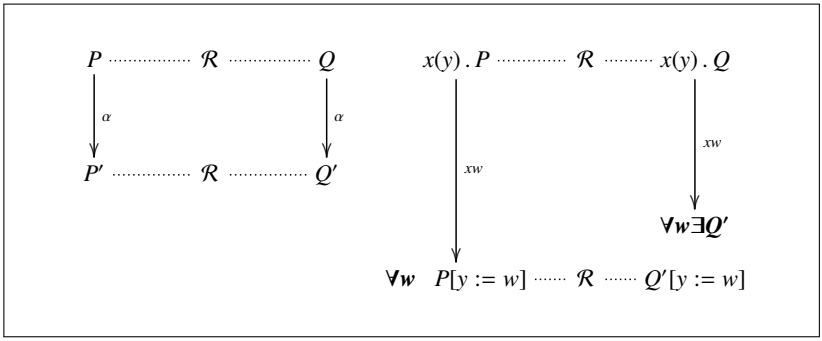
- *ha $P \xrightarrow{\alpha} P'$, $\alpha \neq x(y)$ és $bn(\alpha) \notin fn(P,Q)$, akkor $Q \xrightarrow{\alpha} Q'$ esetén $P'\,\mathcal{R}\,Q'$,*
- *ha $x(y)\,.\,P$ és $x(y)\,.\,Q$-ra $y \notin fn(P,Q)$ és $x(y)\,.\,P \xrightarrow{xw} P[y := w]$, akkor minden w-re van olyan Q', melyre $Q \xrightarrow{xw} Q'$ és $P[y := w]\,\mathcal{R}\,Q'[y := w]$.*

A biszimulációt *erős korai biszimulációnak* is nevezzük. Az „erős” jelző arra utal, hogy a címkézett $\xrightarrow{\alpha}$ átmenetekben a τ-val jelzett belső kommunikációkat is figyelembe veszi.

A biszimuláció másik jelzője a „korai”, a műveletekre a korai szemantika szabályait alkalmazzuk.

Felhívjuk a figyelmet arra is, hogy ellentétben a késői biszimulációval (lásd a 2.6.6. definíció utáni megjegyzést), itt nem a P és Q folyamatokhoz egyértelműen hozzárendelt P' és egy Q' folyamatokat teszteljük a helyettesítéssel, hanem minden w névhez megadunk egy önálló Q' folyamatot, és a P, Q' folyamatokkal végezzük el a helyettesítések tesztelését, azaz azt, hogy a helyettesített folyamatok $\mathcal{R}$ relációban vannak-e.

A definícióban szereplő átmenetek a 2.7. ábrán láthatók.



2.7. ábra. A korai biszimuláció

Meg kell különböztetnünk a korai kölcsönös hasonlóságot és a 2.6.7. definícióban megadott késői kölcsönös hasonlóságot. A továbbiakban a korai kölcsönös hasonlóságot a $\dot{\sim}_e$ jellel, a késői kölcsönös hasonlóságot a

$\dot{\sim}_l$ jellel jelöljük. A jelölésekben az e index az angol *early*, az l index a *late* szó kezdőbetűje.

2.6.20. Definíció. Korai kölcsönös hasonlóság:

A korai kölcsönös hasonlóság legyen a korai biszimulációk uniója. Azt mondjuk, hogy P és Q folyamatok korai kölcsönösen hasonlóak, azaz $P \dot{\sim}_e Q$, ha van olyan $\mathcal{R}$ korai biszimuláció, melyre $P \mathcal{R} Q$.

A korai kölcsönös hasonlóságra is érvényesek a 2.6.8. és 2.6.10. tételekben kimondott állítások, természetesen úgy, hogy a „kölcsönös hasonlóság” kifejezéseket a „korai kölcsönös hasonlóság”-ra cseréljük ki.

A következő példában megmutatjuk a korai és késői kölcsönös hasonlóság közötti kapcsolatot.

2.6.21. Példa. *(Korai és késői kölcsönös hasonlóság)*

Nézzük a következő két folyamatkifejezést,

$P \equiv x(y) \,.\, R + x(y) \,.\, \mathbf{0} + x(y) \,.\, [y = u]R$,

$Q \equiv x(y) \,.\, R + x(y) \,.\, \mathbf{0}$,

és vizsgáljuk a P és Q kölcsönös hasonlóságát. Azonnal észrevehető, hogy a P harmadik tagjának megfelelő kifejezés nincs a Q-ban semmilyen y-ra, tehát a P és Q-ra biztosan nem teljesül a késői kölcsönös hasonlóság.

A korai kölcsönös hasonlóság viszont bizonyítható. Tekintsük a P harmadik tagját. Az Early-Input szabály szerint

$P \equiv x(y) \,.\, [y = u]R \xrightarrow{xw}$

$([y = u]R)\,[y := w] \equiv$

$[w = u]\,(R[y := w])$.

- Ha $w = u$, akkor a P kifejezés eredménye $R[y := w]$. Ha Q első tagjára is végrehajtjuk ezt az input műveletet, akkor

  $Q \equiv x(y) \,.\, R \xrightarrow{xw} R[y := w]$,

  azaz megtaláltuk a Q megfelelő átmenetét.
- Ha $w \neq u$, akkor P eredménye $\mathbf{0}$, és a Q második tagjára:

  $Q \equiv x(y) \,.\, \mathbf{0} \xrightarrow{xw} \mathbf{0}[y := w] \equiv \mathbf{0}$, tehát most is megtaláltuk a Q megfelelő átmenetét.

Megállapíthatjuk, hogy $P \not\dot{\sim}_l Q$, de $P \dot{\sim}_e Q$, azaz a P és Q folyamatokat

a késői kölcsönös hasonlóság megkülönbözteti, de a korai kölcsönös hasonlóság nem tudja. □

A korai kölcsönös hasonlóság is *alapreláció*, azaz a $P\mathcal{R}Q$-ból nem következik az, hogy $P\sigma\mathcal{R}Q\sigma$, ezért a kongruencia fogalmát most is külön definiálnunk kell.

A kongruencia definíciója a 2.6.13. definícióban megadottakhoz hasonló, és az egyértelműség érdekében az ott megadott kongruenciát a továbbiakban *késői kongruenciának* nevezzük és a $\sim_l$ jellel jelöljük.

2.6.22. Definíció. Korai kongruencia:

A P és Q folyamat korai kongruens, ha minden σ helyettesítésre $P\sigma \dot{\sim}_e Q\sigma$. A kongruencia jele $\sim_e$, tehát ekkor $P \sim_e Q$.

A késői kölcsönös hasonlósághoz hasonlóan a korai kölcsönös hasonlóság sem kongruencia, erre utal most is a korai kölcsönös hasonlóság jelében a $\sim_e$ szimbólumon a pont jel.

A korai kongruenciára is kimondhatjuk a 2.6.15. tételnek megfelelő állítást.

2.6.23. Tétel. (A legnagyobb kongruencia)

A $\sim_e$ a korai kölcsönös hasonlóságot tekintve a legnagyobb korai kongruencia.

A tétel alapján a már korábban, a 2.6. szakasz végén bevezetett jelöléssel:

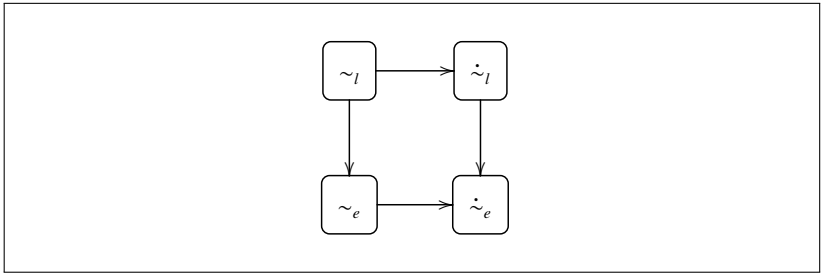
$$\sim_e \quad \longrightarrow \quad \dot{\sim}_e \ .$$

A 2.6.21. példa is arra utal, hogy a korai és késői kölcsönös hasonlóság között a

$$\dot{\sim}_l \quad \longrightarrow \quad \dot{\sim}_e$$

reláció is fennáll, így a a 2.6. ábrán megadott késői kongruenciára és késői kölcsönös hasonlóságra vonatkozó kapcsolatokat kibővíthetjük a korai kongruenciával és a korai kölcsönös hasonlósággal, amit a 2.8. ábrán mutatunk be.

Mivel a késői kongruencia jobban meg tudja különböztetni a folyamatokat, mint a korai, kérdés lehet az, hogy akkor melyik szemantika, melyik kongruencia, a késői vagy a korai a jobb. A válasz az, hogy a döntés „ízlés” kérdése. Azt azonban hangsúlyozni kell, hogy a korai szemantika közelebb van a folyamatok működéséhez, ahol az input adat feldolgozása azonnal



2.8. ábra. A korai és késői kongruencia és kölcsönös hasonlóság

megtörténik, amikor az input néven az adat megjelenik. Ugyanakkor az is látható, hogy a késői szemantika könnyebben kezelhető, hiszen például a késői kölcsönös hasonlóság megállapítására, a 2.6.19. definíció jelöléseit alkalmazva, nincs szükség minden w input adathoz olyan Q' folyamatot keresni, amelyre majd a P folyamattal történő $\mathcal{R}$ reláció fennállását igazolni kell.

2.6.4. Nyilazott biszimuláció

A *nyilazott biszimuláció* azon az elven alapul, hogy egy külső megfigyelő a folyamatok működését vizsgálva

- nem látja a kommunikáció input és output nevét, csupán a kommunikáció tényét jelöli egy τ prefixszel,
- és nem látja a $\boldsymbol{\nu}$ korlátozással jelzett nevekkel történő műveleteket sem.

2.6.24. Példa. *(Megfigyelhető események)*

Nézzük a következő folyamatkifejezést:

$(\boldsymbol{\nu} x)\,(\overline{x}u \mid x(y)\,.\,\overline{y}z + v(w))\ .$

A külső szemlélő a $\overline{x}u$ folyamatból láthatná az x nevet, de a $\boldsymbol{\nu} x$ ezt megtiltja. Az $x(y)\,.\,\overline{y}z + v(w)$ folyamat megfigyelhető nevei x és v, de a korlátozás miatt csak a v lesz látható. A $\overline{y}z$-ben levő $\overline{y}$ név egy formális paraméter, ezért $\overline{y}$-t nem nevezhetjük megfigyelhető névnek.

Ha végrehajtjuk a folyamatkifejezésben levő kommunikációt, akkor az $\overline{u}z$ kifejezést kapjuk, amelyben a megfigyelhető név az $\overline{u}$. □

Most megadjuk a megfigyelhetőség pontos definícióját.

2.6.25. Definíció. Megfigyelhető név:

Tegyük fel, hogy az x név nem esik korlátozás alá, ekkor ha a P folyamat soronkövetkező művelete

- *egy $\overline{x}y$ prefix végrehajtása, akkor az x-t megfigyelhető névnek nevezzük, és ezt a tulajdonságot a $P\downarrow_{\overline{x}}$ jellel jelöljük,*
- *egy $x(y)$ prefix végrehajtása, akkor az x-t megfigyelhető névnek nevezzük, és ezt a tulajdonságot a $P\downarrow_x$ jellel jelöljük.*

A definíció alapján tehát a megfigyelhető név mindig az output vagy input prefix *alanya*. A megfigyelhető nevet a $\downarrow$ nyíllal jelöljük, innen származik a fogalmak elnevezésében a „nyilazott” jelző.

2.6.26. Példa. *(Megfigyelhető nevek)*

A $P \equiv x(y) + \overline{u}v + \tau\,.\,\overline{v}w$ folyamatban $P\downarrow_x$, $P\downarrow_{\overline{u}}$, de $P\not\downarrow_v$. □

2.6.27. Definíció. Nyilazott biszimuláció:

Legyen $\mathcal{R}$ egy szimmetrikus bináris reláció az S halmaz felett, és legyen $P\,\mathcal{R}\,Q$. Az $\mathcal{R}$ relációt nyilazott biszimulációnak nevezzük,

- *ha $P\downarrow_a$, akkor $Q\downarrow_a$,*
- *ha $P \xrightarrow{\tau} P'$, akkor van olyan Q', melyre $Q \xrightarrow{\tau} Q'$ és $P'\,\mathcal{R}\,Q'$,*

ahol a az output vagy input prefixek alanya.

Mint a 2.6.2. és 2.6.3. szakaszokban vizsgált biszimulációknál tettük, a nyilazott biszimulációra is megadjuk a kölcsönös hasonlóság definícióját.

2.6.28. Definíció. Nyilazott kölcsönös hasonlóság:

A nyilazott kölcsönös hasonlóság legyen a nyilazott biszimulációk uniója, és a nyilazott kölcsönös hasonlóság jele legyen $\dot{\sim}_{\downarrow}$. Azt mondjuk, hogy P és Q folyamatok nyilazott kölcsönös hasonlóság relációban vannak, azaz $P \dot{\sim}_{\downarrow} Q$, ha van olyan $\mathcal{R}$ nyilazott biszimuláció, melyre $P\,\mathcal{R}\,Q$.

A nyilazott kölcsönös hasonlóság is ekvivalencia reláció, és erre a kölcsönös hasonlóságra is bizonyíthatóak a 2.6.8. és 2.6.10. tételekben a késői kölcsönös hasonlóságra kimondott állítások.

2.6.29. Példa. *(Nyilazott kölcsönös hasonlóság)*

Vizsgáljuk a

$P \equiv \tau \,.\, \overline{x}y$

és a

$Q \equiv (\boldsymbol{\nu} u)\,(\overline{u}w \mid u(v) \,.\, \overline{x}v)$

folyamatokat, és tegyük fel, hogy $P \mathcal{R} Q$. Nyilvánvaló, hogy $P \downarrow_{\overline{x}}$, és mivel az u kötött a Q-ban, $Q \downarrow_{\overline{x}}$, és más megfigyelhető név nincs sem a P-ben, sem a Q-ban.

$P \xrightarrow{\tau} P' \equiv \overline{x}y$,

és erre $P' \downarrow_{\overline{x}}$. A Q folyamat az u néven végrehajt egy kommunikációt, aminek az eredménye $\overline{x}w$, azaz

$Q \xrightarrow{\tau} Q' \equiv \overline{x}w$,

melyre $Q' \downarrow_{\overline{x}}$.

Tehát ha $P \mathcal{R} Q$, akkor $P' \mathcal{R} Q'$ is fennáll. Hasonlóan megállapítható a tulajdonság az $\mathcal{R}^{-1}$ relációra is, így $P \;\dot{\sim}_{\downarrow}\; Q$. □

2.6.30. Példa. *(Triviális nyilazott kölcsönös hasonlóságok)*

Nyilvánvaló, hogy

$\overline{x}y \;\dot{\sim}_{\downarrow}\; (\boldsymbol{\nu} y)\,\overline{x}y \;\dot{\sim}_{\downarrow}\; \overline{x}z$,

hiszen mindegyik folyamatban a megfigyelhető név $\overline{x}$. A három folyamat tehát nyilazott kölcsönös hasonló, de ugyanakkor azt is látjuk, hogy a három folyamat egymástól lényegesen különbözik. Ezen a tulajdonságon javítunk a nyilazott kongruencia bevezetésével. □

A nyilazott biszimuláció is *alapreláció*, tehát a kongruenciát most is definiálnunk kell.

2.6.31. Definíció. Nyilazott kongruencia:

> *A P és Q folyamat nyilazott kongruens, ha minden C kontextusra $C[P] \;\dot{\sim}_{\downarrow}\; C[Q]$. A kongruencia jele $\sim_{\downarrow}$, tehát ekkor $P \sim_{\downarrow} Q$.*

A késői és korai kölcsönös hasonlóságokról láttuk, hogy nem voltak kongruenciák. Ez elmondható a nyilazott kongruenciáról is.

2.6.32. Példa. *(A 2.6.30. példa kifejezései)*

A 2.6.30. példában szereplő $\overline{x}y$, $(\boldsymbol{\nu} y)\,\overline{x}y$, $\overline{x}z$ folyamatok között fennállt a nyilazott kölcsönös hasonlóság tulajdonság, azaz ezzel a fogalommal nem tudtunk különbséget tenni közöttük. Nézzük a nyilazott kongruenciát:

Legyen például $C \equiv\ [\ ] \mid x(u)\,.\,\overline{u}v$, ekkor

$C[\overline{x}y]\downarrow_{\overline{y}}$,

$C[\overline{x}z]\downarrow_{\overline{z}}$,

és a $((\boldsymbol{\nu} y)\,\overline{x}y) \mid x(u)\,.\,\overline{u}v$ kifejezésben nincs megfigyelhető név, mivel

$((\boldsymbol{\nu} y)\,\overline{x}y) \mid x(u)\,.\,\overline{u}v \equiv$

$(\boldsymbol{\nu} y)\,(\overline{x}y \mid x(u)\,.\,\overline{u}v) \xrightarrow{\tau}$

$(\boldsymbol{\nu} y)\,\overline{y}v$,

azaz a nyilazott kongruencia a három folyamatot már megkülönbözteti. □

A kongruencia meghatározása nem egyszerű feladat, mivel a definícióban a kontextusra egy „minden” kvantor szerepel. Ezen könnyít a következő tétel [45], amely szerint a nyilazott kölcsönös hasonlóság vizsgálatot csak a kontextusok egy részosztályára kell elvégezni.

2.6.33. Tétel. (Kontextus lemma)

$P \sim_{\downarrow} Q$ akkor és csak akkor, ha

$$P\sigma \mid R \ \dot{\sim}_{\downarrow}\ Q\sigma \mid R$$

minden σ-ra és R-re.

2.6.34. Példa. *(Nyilazott kölcsönös hasonlóság és a nyilazott kongruencia)*

A 2.6.29. példában láttuk, hogy

$P \equiv \tau\,.\,\overline{x}y \quad \dot{\sim}_{\downarrow} \quad Q \equiv (\boldsymbol{\nu} u)\,(\overline{u}w \mid u(v)\,.\,\overline{x}v)$.

Legyen a C kontextus $[\ ] \mid x(z)\,.\,\overline{z}s$, ekkor

$C[P] \equiv$

$\tau\,.\,\overline{x}y \mid x(z)\,.\,\overline{z}s \xrightarrow{\tau}$

$\overline{x}y \mid x(z)\,.\,\overline{z}s \xrightarrow{\tau}$

$\overline{y}s$,

és így $C[P]\downarrow_y$. Ugyanakkor

$C[Q] \equiv$

$((\nu u)\,(\overline{u}w \mid u(v)\,.\,\overline{x}v)) \mid x(z)\,.\,\overline{z}s \xrightarrow{\tau}$

$(\nu u)\,\overline{x}w \mid x(z)\,.\,\overline{z}s \equiv$

$(\nu u)\,(\overline{x}w \mid x(z)\,.\,\overline{z}s) \xrightarrow{\tau}$

$(\nu u)\,\overline{w}s \equiv$

$\overline{w}s$,

így $C[Q]\downarrow_w$. Tehát $P \mathrel{\dot{\sim}_\downarrow} Q$, de $P \sim_\downarrow Q$ nem áll fenn. □

A fenti példa is azt mutatja, hogy a nyilazott kongruenciára és a nyilazott kölcsönös hasonlóságra fennáll a

$$\sim_\downarrow \;\longrightarrow\; \dot{\sim}_\downarrow$$

reláció.

A nyilazott és a korai kongruenciákra bizonyítható a következő állítás [45]:

2.6.35. Tétel. *($\sim_\downarrow \;=\; \sim_e$)*

|| $P \sim_\downarrow Q$ akkor és csak akkor, ha $P \sim_e Q$.

A tétel azt mondja ki, hogy a nyilazott kongruencia lényegében azonos a korai kongruenciával, a folyamatok megkülönböztetését tekintve a nyilazott és korai kongruenciák között nincs lényeges különbség, és így továbbra is a késői kongruencia adja a legfinomabb osztályozást.

2.6.36. Példa. *($\sim_\downarrow \;=\; \sim_e$)*

Nézzük a 2.6.32. példában szereplő $\overline{x}y$ és $\overline{x}z$ folyamatokat, amelyekről a példában láttuk, hogy a nyilazott kongruencia szerint különbözőek. Ha ugyancsak ebben a példában szereplő $C \equiv [\,] \mid x(u)\,.\,\overline{u}v$ helyettesítést vesszük, akkor

$C[\overline{x}y] \equiv$

$\overline{x}y \mid x(u)\,.\,\overline{u}v \xrightarrow{\tau}$

$\overline{y}v$,

és

$C[\overline{x}z] \equiv$

$\overline{x}z \mid x(u) \,.\, \overline{u}v \xrightarrow{\tau}$

$\overline{z}v$.

Ezekre pedig a $C[\overline{x}y] \;\dot{\sim}_e\; C[\overline{x}z]$ biztosan nem teljesül, tehát a két folyamatot mind a nyilazott, mind a korai kongruencia megkülönbözteti. □

A 2.6.35. tétel állítása csak a kongruenciák azonosságát mondja ki, a kölcsönös hasonlóságra az azonosság nem teljesül, ezt mutatjuk meg a következő példában.

2.6.37. Példa. *($\dot{\sim}_{\downarrow} \neq \dot{\sim}_e$)*

Legyen

$P \equiv \overline{x}y$,

$Q \equiv \overline{x}y \,.\, \overline{x}z$.

Azonnal látható, hogy $P{\downarrow_{\overline{x}}}$ és $Q{\downarrow_{\overline{x}}}$, tehát P és Q nyilazott kölcsönösen hasonló. Ugyanakkor $P \xrightarrow{\overline{x}y} P'$, ahol $P' \equiv \mathbf{0}$ és $Q \xrightarrow{\overline{x}y} Q'$, ahol $Q' \equiv \overline{x}z$. Tehát ha $P \mathcal{R} Q$, akkor $P' \mathcal{R} Q'$ biztosan nem áll fenn.

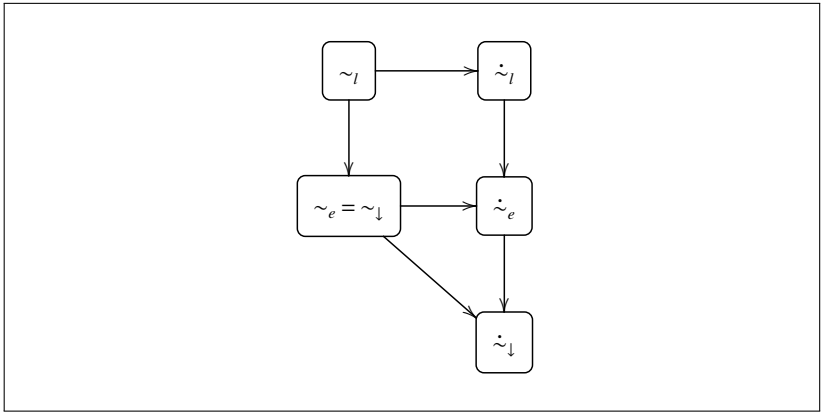
A példából az is látható, hogy a két folyamat között a nyilazott kölcsönös hasonlóság nem tudott különbséget tenni, de a korai kölcsönös hasonlóság a folyamatokat megkülönböztette. □

A 2.8. ábrát a nyilazott kölcsönös hasonlóságra és nyilazott kongruenciára kapott új információkkal bővíthetjük, az eredmény a 2.9. ábrán látható.

2.6.5. Nyitott biszimuláció

Az előző szakaszokban láttuk, hogy a késői, korai és a nyilazott biszimuláció mindegyike *alapreláció* típusú biszimuláción alapul, azaz a kongruencia definíciójában külön elő kellett írni, hogy a helyettesítés megtartja a folyamatok közötti biszimuláció relációt.

A nyitott biszimuláció alapgondolata az volt, hogy a biszimulációt „nyissuk meg”, a helyettesítésre vonatkozó követelményt vigyük be a biszimulációba, és ekkor közvetlenül a biszimulációval tudjuk a kongruenciát definiálni. A gondolat D. Sangiorgitól származik az 1990-es évek közepéről



2.9. ábra. A nyilazott, korai és késői kongruencia és kölcsönös hasonlóság

[44]. Ez az ötlet vezetett el a *nyitott biszimuláció* fogalmához és a legfinomabb kongruenciához, azaz amelyik a legkevesebb folyamatot találja azonosnak.

2.6.38. Definíció. Nyitott biszimuláció:

Legyen $\mathcal{R}$ egy szimmetrikus bináris reláció az S halmaz felett, és legyen $P\mathcal{R}Q$. Az $\mathcal{R}$ relációt nyitott biszimulációnak nevezzük, ha minden σ helyettesítésre

- *$P\sigma \xrightarrow{\alpha} P'$ és $bn(\alpha) \notin fn(P, Q)$ esetén van olyan Q' , melyre $Q\sigma \xrightarrow{\alpha} Q'$ és $P' \mathcal{R} Q'$.*

A definícióban szereplő átmenetek a 2.10. ábrán láthatók.

A korábban vizsgált biszimulációkkal ellentétben a nyitott biszimuláció definíciójában nincs szükség az $x(y)$ input prefix különvételére. Korábban ezt azért tettük, mert a $P \xrightarrow{x(y)} P'$, $Q \xrightarrow{x(y)} Q'$ végrehajtásakor a P'-ben és a Q'-ben az y paraméter egy konkrét névre helyettesítődik, és az y összes lehetséges helyettesítésére vizsgálni kellett a reláció fennállását.

A nyitott biszimuláció definíciójában a „*minden σ helyettesítés*” rekurzív, hiszen ez a kvantált helyettesítés ott van a $P\mathcal{R}Q$ feltételben, és ott van a $P'\mathcal{R}Q'$-ben is. Ez utóbbi elemzéséhez is a helyettesített a $P'\sigma'$ és $Q'\sigma'$ kifejezésekkel kell indulni, hiszen közöttük is ott van az $\mathcal{R}$ reláció, és ezekben már benne van az y formális paraméternek az input műveletből származó

$$
\begin{tikzcd}
 & P \arrow[r, dotted, no head] & \mathcal{R} \arrow[r, dotted, no head] & Q \\
\forall\sigma & P\sigma \arrow[d, "\alpha"] & & Q\sigma \arrow[d, "\alpha"] \\
 & P' \arrow[r, dotted, no head] & \mathcal{R} \arrow[r, dotted, no head] & Q'
\end{tikzcd}
$$

2.10. ábra. A nyitott biszimuláció

helyettesítése is.

Így ebből azonnal látható, hogy a nyitott biszimuláció egyben késői biszimuláció is. Azonban fordított irányban ez az állítás nem teljesül, ezt a következő példában mutatjuk meg.

2.6.39. Példa. *(Nyitott és késői biszimuláció)*

Legyen a P és Q folyamat a következő:

$$P \equiv \tau . \tau + \tau ,$$

$$Q \equiv \tau . [x = y] \tau + \tau . \tau + \tau .$$

Látható, hogy a Q-beli $\tau . \tau$-nak és a τ-nak ott van a megfelelője P-ben, és ha Q-ban $x = y$, akkor a P-beli $\tau . \tau$, $x \neq y$ esetén pedig a τ a megfelelő akciósorozat. Tehát $P \dot{\sim}_l Q$.

Azonban P és Q-ra a nyitott biszimuláció nem áll fenn. Természetesen a kifejezések $\tau . \tau + \tau$ részével nincs probléma, ezért vizsgáljuk a Q folyamat $Q' \equiv \tau . [x = y]\tau$ részkifejezését.

Legyen σ az identitás helyettesítés. Ekkor

$$Q' \xrightarrow{\tau} [x = y]\tau ,$$

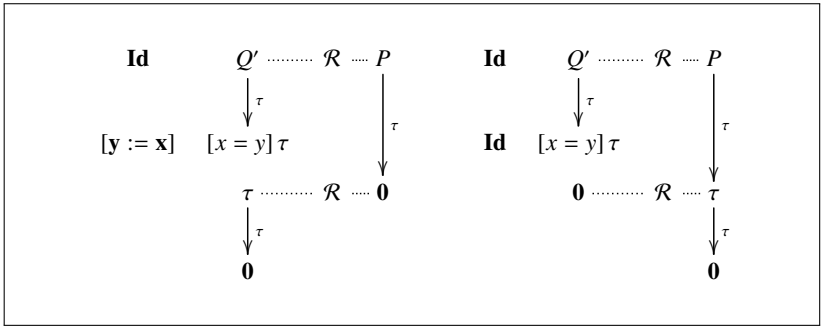
és P-re kettő lehetőség van (2.11. ábra):

$$P \xrightarrow{\tau} \mathbf{0} \quad \text{és} \quad P \xrightarrow{\tau} \tau .$$

Az első esetben hajtsunk végre egy $[y := x]$ helyettesítést, ekkor Q' átmenete τ lesz, amiből τ-val újabb átmenetet képezhetünk. A P-ből kapott $\mathbf{0}$-ból ilyen átmenetre nincs lehetőség.

A második esetben a helyettesítés legyen az identitás, így Q'-ből a $\mathbf{0}$-t

kapjuk, míg P-ből még egy további τ átmenetet képezhetünk, amelynek nincs meg a megfelelő átmenete Q'-ből. □



2.11. ábra. A 2.6.39. példa átmenetei

2.6.40. Definíció. Nyitott kölcsönös hasonlóság:

A nyitott kölcsönös hasonlóság legyen a nyitott biszimulációk uniója, és a nyitott kölcsönös hasonlóság jele legyen $\dot{\sim}_o$ *. Azt mondjuk, hogy P és Q folyamatok nyitott kölcsönös hasonlóság relációban vannak, azaz* $P \dot{\sim}_o Q$ *, ha van olyan $\mathcal{R}$ nyitott biszimuláció, melyre $P \mathcal{R} Q$.*

A jelölésekben az o index az angol *open* szó kezdőbetűje.

A kongruencia definícióját a korábbi biszimulációk esetén helyettesítésekkel vagy környezetekkel adtuk meg, ezek a helyettesítések viszont most a biszimuláció definíciójában már benne vannak. Ezért a nyitott biszimuláció definíciója alapján kimondható a következő állítás:

2.6.41. Tétel. (A nyitott kölcsönös hasonlóság és kongruencia)

A nyitott kölcsönös hasonlóság kongruencia, és ha a nyitott kongruenciát $\sim_o$*-val jelöljük, akkor* $\dot{\sim}_o \equiv \sim_o$ *.*

2.6.42. Példa. *(Nyitott biszimuláció)*

Legyen

$P \equiv \mathbf{0}$,

$Q \equiv [x = y]\tau$.

Mindkét folyamat áll, és a korábbi pontokban tárgyalt kölcsönös hasonlósá-

gokkal nem tudjuk megkülönböztetni ezeket a folyamatokat.

Azonban ha végrehajtjuk mindkettőre az $[x := y]$ helyettesítést, azonnal látjuk, hogy a két folyamat különböző, azaz a nyitott kölcsönös hasonlóság a folyamatokat megkülönbözteti. □

2.6.43. Példa. *(A nyitott biszimuláció és a korlátozás)*

Legyen

$P_1 \equiv \mathbf{0}$,

$Q_1 \equiv (\boldsymbol{\nu} x)\,[x = y]\tau$.

Nincs olyan helyettesítés, amelyik hatással lenne a folyamatokra, amelyik az x-t vagy y-t megváltoztatná, mivel helyettesítések csak a folyamatokban levő szabad nevekre vonatkozhatnak. Ezért $Q_1 \longrightarrow \mathbf{0}$, és a két folyamat nyitott kölcsönös hasonlóság relációban van, $P_1 \;\dot{\sim}_o\; Q_1$.
Hasonló állítás mondható el a

$P_2 \equiv \tau$,

$Q_2 \equiv (\boldsymbol{\nu} x)\,\tau\,.\,[x = y]\tau$

folyamatokról is.

Nézzük meg a kötött output prefixet is. Legyen

$P_3 \equiv \overline{x}(y)$,

$Q_3 \equiv \overline{x}(y)\,.\,[x = y]\tau$.

Az átmenetek után

$P_3 \xrightarrow{\overline{x}(y)} \mathbf{0}$,

$Q_3 \xrightarrow{\overline{x}(y)} [x = y]\tau$,

ezek megegyeznek a 2.6.42. példában szereplő kifejezésekkel, amelyekről láttuk, hogy a nyitott kölcsönös hasonlóság szerint különbözőek. □

Az előző példában szereplő P_3 és Q_3 folyamatok a nyitott kölcsönös hasonlóság szempontjából különbözőek, mert mint láttuk, a korlátozott output végrehajtása feloldotta az y helyettesíthetőségének tilalmát. A P_3 és Q_3 kifejezésekre is a hasonlóságot szeretnénk elérni, hiszen y egy korlátozott név, az $x = y$ névazonosság nem állhat fenn, és ezért az y helyettesíthetőségét meg kellene tiltani.

Úgy tűnik, hogy a nyitott biszimuláció definíciójában szereplő „*minden*

σ helyettesítés" túl sok helyettesítést jelent. (A helyettesítés miatt például a True és False konstansok is kölcsönösen hasonlóak, ha az egyikre egy $[t := f]$ helyettesítést írunk elő. Ez nyilván nem engedhető meg.)

Ezt a problémát úgy oldhatjuk meg, hogy megadjuk azokat a névpárokat, amelyek mindig különböznek egymástól, függetlenül attól, hogy milyen környezetben, milyen műveletben szerepelnek. A 2.6.38. definícióban szereplő „*minden* σ helyettesítés" kitételt pedig úgy módosítjuk, hogy a helyettesítés csak azokra a névpárokra vonatkozik, amelyek nem szerepelnek a megadott névpárok halmazában.

2.6.44. Definíció. Nevek megkülönböztetése:

- *Jelöljük D -vel azt az N névhalmazon értelmezett szimmetrikus és irreflexív bináris relációt, melyre xDy esetén minden σ helyettesítésre $x\sigma \neq y\sigma$.*
- *Legyen $D\sigma = \{(x\sigma, y\sigma) \mid xDy\}$.*
- *A D-t megkülönböztetés relációnak nevezzük.*
- *Azt mondjuk, hogy a σ helyettesítés megfelel a D megkülönböztetésnek, ha xDy esetén $x\sigma \neq y\sigma$.*

A $D\sigma$ halmazt bővíthetjük is, például $D\sigma \cup (y, fn(P\sigma, Q\sigma))^{=}$ jelentse azt a halmazt, amelyet úgy kapunk, hogy a $D\sigma$-t bővítjük azon párok halmazával, amelyeknek

- első eleme y, a második eleme a $P\sigma$ vagy $Q\sigma$ egy szabad neve,
- $H^{=}$ pedig a H szimmetrikus lezárása, azaz $H^{=} = H \cup H^{cs}$, ahol H^{cs} a H-ban levő párok elemeinek megcserélésével kapott halmazt jelenti.

2.6.45. Definíció. Nyitott D-biszimuláció:

Adott D megkülönböztetésre legyen $\mathcal{R}_D$ egy szimmetrikus bináris reláció az S halmaz felett, és legyen $P\mathcal{R}_D Q$. Az $\mathcal{R}_D$ relációt nyitott D-biszimulációnak nevezzük, ha minden D-nek megfelelő σ helyettesítésre $P\sigma \xrightarrow{\alpha} P'$ és $bn(\alpha) \notin fn(P, Q)$ esetén

- *ha $\alpha = \overline{x}(y)$, akkor van olyan Q' , melyre $Q\sigma \xrightarrow{\overline{x}(y)} Q'$ és $P' \mathcal{R}_{D'} Q'$, ahol $D' = D\sigma \cup (y, fn(P\sigma, Q\sigma))$,*
- *ha $\alpha \neq \overline{x}(y)$, akkor van olyan Q' , melyre $Q\sigma \xrightarrow{\alpha} Q'$ és $P' \mathcal{R}_{D\sigma} Q'$.*

$$
\begin{tikzcd}
 & P \arrow[r, dotted, no head] & \mathcal{R} \arrow[r, dotted, no head] & Q \\
\forall\sigma & P\sigma \arrow[d, "\alpha"] & & Q\sigma \arrow[d, "\alpha"] \\
 & P' \arrow[r, dotted, no head] & \mathcal{R} \arrow[r, dotted, no head] & Q'
\end{tikzcd}
$$

2.12. ábra. A nyitott D-biszimuláció

2.6.46. Definíció. Nyitott D-kölcsönös hasonlóság:

A nyitott D-kölcsönös hasonlóság legyen a nyitott D-biszimulációk uniója, és a nyitott D-kölcsönös hasonlóság jele legyen $\dot{\sim}_o^D$ *. Azt mondjuk, hogy P és Q folyamatok nyitott D-kölcsönös hasonlóság relációban vannak, azaz* $P \dot{\sim}_o^D Q$ *, ha van olyan* $\mathcal{R}_D$ *nyitott D-biszimuláció, melyre* $P \mathcal{R}_D Q$ *.*

A nyitott D-kölcsönös hasonlóság nyilvánvalóan *kongruencia*, és a nyitott D-kongruenciát $\sim_o^D$-vel jelöljük.

Megjegyezzük, hogy

$\dot{\sim}_o = \dot{\sim}_o^{\emptyset}$ és $\sim_o = \sim_o^{\emptyset}$.

2.6.47. Példa. *(A 2.6.43. példa nyitott D-biszimulációval)*

A 2.6.43. példában szereplő

$P_3 \equiv \overline{x}(y)$,

$Q_3 \equiv \overline{x}(y) \,.\, [x = y]\tau$

folyamatok a nyitott kölcsönös hasonlóság szempontjából különbözőek voltak. Láttuk, hogy

$P_3 \xrightarrow{\overline{x}(y)} \mathbf{0}$,

$Q_3 \xrightarrow{\overline{x}(y)} [x = y]\tau$,

és a Q_3-ra alkalmazott $[x := y]$ helyettesítés adta azt az eredményt, hogy a két folyamat különböző.

Legyen $D = \{(x, y), (y, x)\}$. Most azt kell belátnunk, hogy

$[x = y]\tau \overset{D}{\underset{o}{\sim}} \mathbf{0}$,

ez pedig teljesül, mert az $[x := y]$ helyettesítést nem hajthatjuk végre, így

$[x = y]\tau \equiv \mathbf{0}$.

A P_3 és Q_3 folyamat között tehát a nyitott D-kölcsönös hasonlóság fennáll. □

Sajnos a nyitott biszimuláció definíciójában levő „*minden* σ helyettesítés" a tulajdonság tesztelését is elég kellemetlenné teszi. A szakirodalomban több módszer is található ennek elkerülésére.

Az egyik módszer (M, α) párokkal dolgozik, ahol M az α-ban szereplő nevekre összegyűjtött olyan információ, ami az α végrehajtásához feltétlenül szükséges. A biszimuláció tesztelése így az α prefix esetén csak azokkal a helyettesítésekkel foglalkozik, amelyek az M-ben vannak összegyűjtve. Például az $[x = y]\alpha \,.\, P$ folyamatban az α prefixhez tartozó információ $(\{[x = y]\}, \alpha)$, és így

$$[x = y]\alpha \,.\, P \overset{(\{[x=y]\},\alpha)}{\longrightarrow} P \,.$$

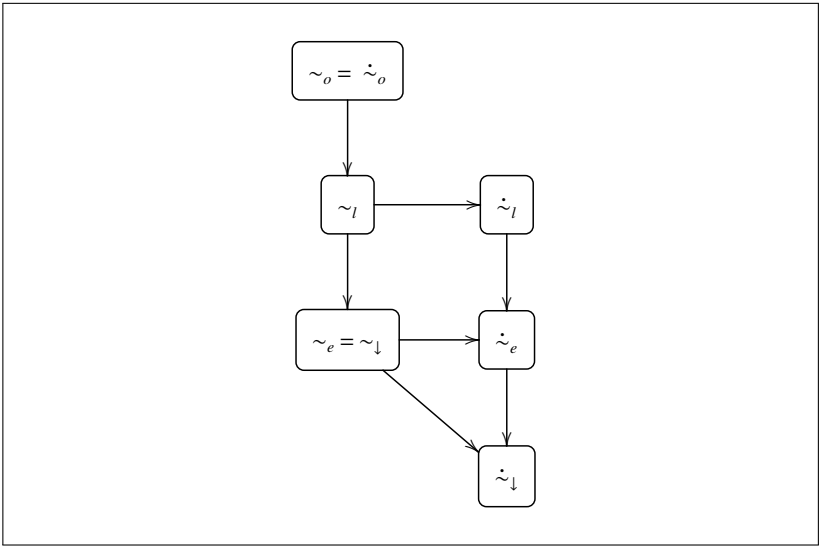
Egy ezen az elven működő nyitott biszimuláció tesztelő program részletes leírása és implementációja a [12]-ben megtalálható.

A nyitott kölcsönös hasonlóságra és nyitott kongruenciára kapott új információkkal tovább bővíthetjük a 2.6.4. pontban megadott 2.9. ábrát, a kapcsolatok a 2.13. ábrán láthatók.

2.6.6. Gyenge biszimuláció

Két folyamat közötti kommunikáció kívülről nem figyelhető meg, a kommunikáció input és output műveletének adatcseréje nem látható, az akciósorozatban a kommunikáció tényére csak a τ szimbólum utal. Ebben a pontban ilyen típusú akciósorozatokkal foglalkozunk, azaz mint egy „külső szemlélő" adjuk meg a folyamatok szimulációjának leírását.

Az eddig vizsgált biszimulációk mindegyike az *erős* jelzőt kapta, mert mint a 2.6.6. definíció után megjegyeztük, az átmenetekben a τ kommunikációkat ugyanolyan súllyal vettük figyelembe, mint a többi „látható" akciót. Ebben a pontban a folyamatok közötti olyan kapcsolatokat vizsgálunk, amelyekben a τ átmenetek nem játszanak szerepet, és ezeket a biszimulációkat, kölcsönös hasonlóságokat, kongruenciákat a *gyenge* jelzővel látjuk el.



2.13. ábra. A nyilazott, korai, késői és nyitott kongruencia és kölcsönös hasonlóság

A biszimulációk vizsgálatakor a következő új jelöléseket fogjuk használni:

$$\begin{array}{ll} \Longrightarrow & \textit{nulla vagy több egymásutáni } \xrightarrow{\tau} \textit{ átmenet}, \\ \overset{\alpha}{\Longrightarrow} & \Longrightarrow\xrightarrow{\alpha}\Longrightarrow\ , \\ \overset{\widehat{\alpha}}{\Longrightarrow} & \begin{cases} \overset{\alpha}{\Longrightarrow}, & \textit{ha } \alpha \neq \tau\ , \\ \Longrightarrow, & \textit{ha } \alpha = \tau\ . \end{cases} \end{array}$$

2.6.48. Definíció. Gyenge késői biszimuláció:

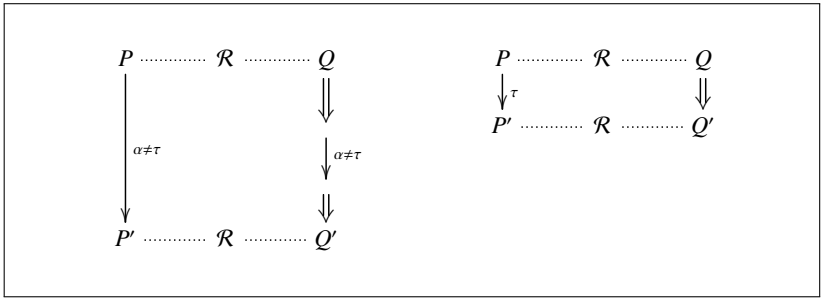
Legyen $\mathcal{R}$ egy szimmetrikus bináris reláció az S halmaz felett, és legyen $P\,\mathcal{R}\,Q$. Az $\mathcal{R}$ gyenge biszimuláció,

- *ha $P \xrightarrow{\alpha} P'$, $\alpha \neq x(y)$ és $bn(\alpha) \notin fn(P,Q)$, akkor $Q \overset{\widehat{\alpha}}{\Longrightarrow} Q'$ esetén $P'\,\mathcal{R}\,Q'$,*
- *ha $P \xrightarrow{x(y)} P'$, $y \notin fn(P,Q)$, akkor $Q \Longrightarrow\xrightarrow{x(y)} Q'$ esetén minden z-re van olyan Q'', melyre $Q'[y := z] \Longrightarrow Q''$ és $P'[y := z]\,\mathcal{R}\,Q''$.*

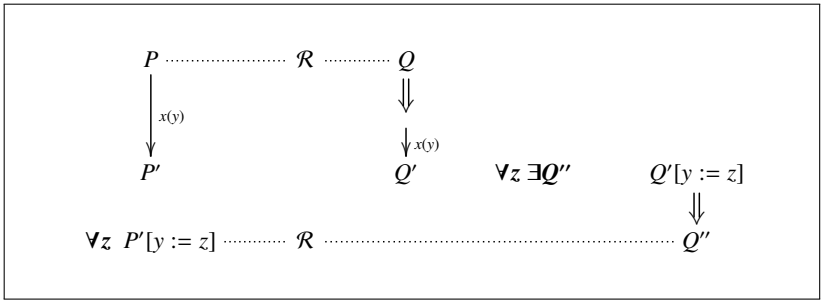
A biszimulációt *gyenge késői biszimulációnak* nevezzük, mert a folya-

matok végrehajtásakor az input prefixből kapott név a kommunikációkor helyettesítődik az input prefixet követő folyamatba, azaz most a *késői szemantika* szabályait alkalmazzuk.

A definícióban szereplő átmenetek a 2.14. és 2.15. ábrán láthatók.



2.14. ábra. Gyenge biszimuláció, $\alpha \neq x(y)$ ($\alpha \neq \tau$ és $\alpha = \tau$)



2.15. ábra. Gyenge biszimuláció, $\alpha = x(y)$

A gyenge késői biszimulációra a kölcsönös hasonlóság és a kongruencia definíciója a megszokott feltételeket írja elő:

2.6.49. Definíció. Gyenge késői kölcsönös hasonlóság:

A gyenge késői kölcsönös hasonlóság legyen a gyenge késői kölcsönös biszimulációk uniója, és a gyenge késői kölcsönös hasonlóság jele legyen $\dot{\approx}_l$ *. Azt mondjuk, hogy P és Q folyamatok gyenge késői kölcsönösen hasonlóak, azaz* $P \dot{\approx}_l Q$*, ha van olyan* $\mathcal{R}$ *gyenge késői biszimuláció, melyre* $P \mathcal{R} Q$ *.*

A késői kölcsönös hasonlóság és a gyenge késői kölcsönös hasonlóság definíciójából megállapíthatjuk a következő tételt:

2.6.50. Tétel. (Az erős és gyenge késői kölcsönös hasonlóság)

Minden erős kölcsönös hasonlóság egyben gyenge kölcsönös hasonlóság is.

Az állítás természetesen fordított irányban nem igaz, hiszen a gyenge kölcsönös hasonlóságra megadott feltételek az erős változatra nem feltétlenül teljesülnek.

2.6.51. Definíció. Gyenge késői kongruencia:

A P és Q folyamat gyenge késői kongruens, ha minden σ helyettesítésre $P\sigma \;\dot{\approx}_l\; Q\sigma$. A kongruencia jele $\approx_l$, tehát ekkor $P \approx_l Q$.

A gyenge késői kölcsönös hasonlóság tehát nem kongruencia, erre utal most is a kölcsönös hasonlóság jelében a $\approx$ szimbólumon a pont jel.

A gyenge késői kölcsönös hasonlóságra is érvényesek az erős késői kölcsönös hasonlóságra a 2.6.8. tételben kimondott tulajdonságok. Például az, hogy a $\dot{\approx}_l$ reláció ekvivalencia reláció.

2.6.52. Példa. *(Gyenge késői kölcsönös hasonlóság)*

Vizsgáljuk a következő kölcsönös hasonlóságot:

$P \;\dot{\approx}_l\; \tau . P$.

Legyen $\alpha = \varepsilon$, ahol ε a nulla hosszúságú akciósorozat. Az állítás azonnal adódik a 2.6.48. definíció első pontjából. Az $\mathcal{R}$ relációkat a 2.16. ábráról leolvashatjuk. Az ábrán a $\tau . P \overset{\widehat{\varepsilon}}{\Longrightarrow} P$ helyett $\tau . P \overset{\tau}{\longrightarrow} P$ átmenetet látunk, mivel

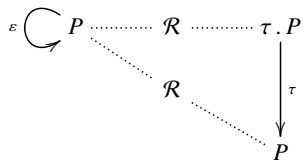
$\overset{\widehat{\varepsilon}}{\Longrightarrow} \quad \equiv$

$\overset{\varepsilon}{\Longrightarrow} \quad \equiv$

$\Longrightarrow\overset{\varepsilon}{\longrightarrow}\Longrightarrow \quad \equiv$

$\Longrightarrow\Longrightarrow \quad \equiv$

$\overset{\tau}{\longrightarrow}$. □



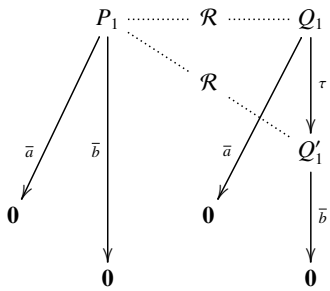
2.16. ábra. $P \;\dot{\approx}_l\; \tau\,.\,P$

2.6.53. Példa. *(Gyenge késői kölcsönös hasonlóságok)*

Hasonlóan könnyen belátható a következő két kölcsönös hasonlóság is:

$$P + \tau\,.\,P \quad \dot{\approx}_l \quad \tau\,.\,P\,,$$
$$\alpha\,.\,(\tau\,.\,P + Q) \quad \dot{\approx}_l \quad \alpha\,.\,P + \alpha\,.\,(\tau\,.\,P + Q)\,. \qquad \square$$

A gyenge késői kölcsönös hasonlóság azonban rendelkezik két kellemetlen tulajdonsággal, a + művelet és a korlátozás a kölcsönös hasonlóságot nem tartja meg. A + műveletre ezt a következő példában mutatjuk meg.



2.17. ábra. (A 2.6.54. példa ábrája)

2.6.54. Példa. *(A + művelet nem kölcsönös hasonlóság tartó)*

Az előző példában láttuk, hogy

$P \;\dot{\approx}_l\; \tau\,.\,P\,.$

Legyen $P \equiv \overline{b}$, és kapcsoljunk a + művelettel például egy $\overline{a}$ folyamatkifejezést mindkét oldalhoz. Legyen

$P_1 \equiv \overline{b} + \overline{a}\,,$

$Q_1 \equiv \tau\,.\,\overline{b} + \overline{a}\,.$

Be kell látnunk, hogy $P_1 \;\dot{\not\approx}_l\; Q_1$, azaz

$\overline{b} + \overline{a} \;\dot{\not\approx}_l\; \tau\,.\,\overline{b} + \overline{a}\,.$

Tegyük fel, hogy találtunk egy megfelelő $\mathcal{R}$ relációt, és $P_1 \,\mathcal{R}\, Q_1$. Legyen $Q_1 \xrightarrow{\tau} Q_1'$, ekkor Q_1'-hez kell keresni egy P_1'-t, melyre $P_1 \Longrightarrow P_1'$ és $P_1' \,\mathcal{R}\, Q_1'$. Ilyen P_1' csak a P_1 lehet, ezért $P_1 \,\mathcal{R}\, Q_1'$. De ez biztosan nem jó, mert $P_1 \xrightarrow{\overline{a}} \mathbf{0}$, és Q_1'-ből nincs semmilyen $\overline{a}$ átmenet (2.17. ábra). □

A gyenge késői biszimuláció definíciójának második része kissé bonyolultnak tűnik, de ha ebben az átmeneteket az erős késői biszimuláció átmeneteihez hasonlóra leegyszerűsítjük, a reláció ekvivalencia tulajdonsága megszűnik, például a tranzitivitás nem teljesül. Ezt mutatjuk meg a következő példában.

2.6.55. Példa. *(Tranzitivitás a módosított definícióval)*

Legyen a 2.6.48. definíció második pontjának módosítása a következő:

- *ha* $P \xrightarrow{x(y)} P'$, $y \notin fn(P, Q)$, *akkor van olyan* Q' , *melyre* $Q \overset{x(y)}{\Longrightarrow} Q'$ *és minden z-re* $P'[y := z] \,\mathcal{R}\, Q'[y := z]$,

a módosított átmenetek a 2.18. ábrán láthatók.

Vizsgáljuk meg a következő három folyamatkifejezést [44].

$P_1 \;\equiv\; x(y)\,.\,(\tau + \tau\,.\,\alpha + \tau\,.\,[y = w]\alpha) \;+\; x(y)\,.\,[y = w]\alpha\,,$

$P_2 \;\equiv\; x(y)\,.\,(\tau + \tau\,.\,\alpha + \tau\,.\,[y = w]\alpha)\,,$

$P_3 \;\equiv\; x(y)\,.\,(\tau + \tau\,.\,\alpha)\,.$

$$
\begin{tikzcd}
P \arrow[r, dotted, no head] \arrow[d, "x(y)"] & \mathcal{R} \arrow[r, dotted, no head] & Q \arrow[d, Rightarrow, "x(y)"] \\
P' & & Q' \\
\forall z\ P'[y := z] \arrow[r, dotted, no head] & \mathcal{R} \arrow[r, dotted, no head] & Q'[y := z]
\end{tikzcd}
$$

2.18. ábra. A módosított gyenge biszimuláció

Az $y = w$ és $y \neq w$ esetek vizsgálatával azonnal látható, hogy

$P_1 \ \dot{\approx}_l \ P_2 \ \dot{\approx}_l \ P_3$, és $P_1 \ \dot{\approx}_l \ P_3$.

Ha a $\dot{\approx}_{l*}$ jellel jelöljük a fenti módosítással kapott relációt, akkor a

$P_1 \ \dot{\approx}_{l*} \ P_2 \ \dot{\approx}_{l*} \ P_3$

kapcsolatok könnyen beláthatók, de erre a relációra már a $P_1 \ \dot{\approx}_{l*} \ P_3$ nem áll fenn, hiszen $P_1 \xrightarrow{x(y)} [y = w]\alpha$ is lehet, ami az y és w-től függően $\mathbf{0}$ vagy α, míg $P_3 \xrightarrow{x(y)} (\tau + \tau \,.\, \alpha)$. Ezek között reláció semmilyen helyettesítésre sem határozható meg. □

A gyenge késői biszimuláció definíciójának második részére egy másik sikertelen kísérlet is közismert, ezt a következő példában mutatjuk meg.

$$
\begin{tikzcd}
P \arrow[r, dotted, no head] \arrow[d, Rightarrow, "\widehat{x(y)}"] & \mathcal{R} \arrow[r, dotted, no head] & Q \arrow[d, Rightarrow, "\widehat{x(y)}"] \\
P' & & Q' \\
\forall z\ P'[y := z] \arrow[r, dotted, no head] & \mathcal{R} \arrow[r, dotted, no head] & Q'[y := z]
\end{tikzcd}
$$

2.19. ábra. A gyenge biszimuláció második módosítása

2.6.56. Példa. *(A 2.6.48. definíció második módosítása)*

Legyen a 2.6.48. definíció második pontjának módosítása a következő:

- *ha* $P \overset{\widehat{x(y)}}{\Longrightarrow} P'$, $y \notin fn(P, Q)$, *akkor van olyan* Q', *melyre* $Q \overset{\widehat{x(y)}}{\Longrightarrow} Q'$ *és minden z-re* $P'[y := z] \mathcal{R} Q'[y := z]$.

A módosított átmenetek a 2.19. ábrán láthatók.

Tekintsük a következő két folyamatot:

$P \equiv x(y) \,.\, [y = v]\,(R' + \tau \,.\, [y = v]\, R'')$,

$Q \equiv x(y) \,.\, [y = v]\,(R' + \tau \,.\, R'')$.

Könnyen belátható, hogy $P \;\dot{\sim}_l\; Q$. Nézzük a következő átmenetet:

$P \overset{\widehat{x(y)}}{\Longrightarrow} [y = v]\, R''$,

amely vagy R'' vagy $\mathbf{0}$. De $Q \overset{\widehat{x(y)}}{\Longrightarrow} R''$, és látható, hogy itt a $\mathbf{0}$ szimulációjára nincs lehetőség, ami ellentmond a 2.6.50. tétel állításának. □

Ahogyan a késői biszimulációhoz megadtuk a gyenge késői biszimuláció definícióját, a korai biszimulációnak is megadhatjuk a gyenge változatát:

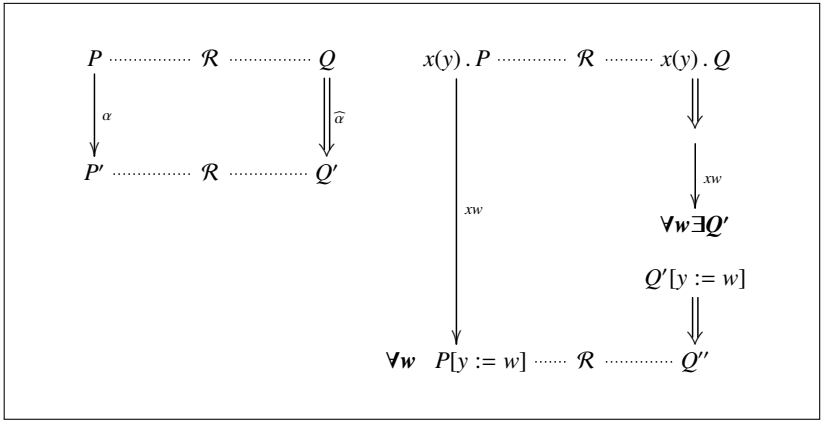
2.6.57. Definíció. Gyenge korai biszimuláció:

Legyen $\mathcal{R}$ *egy szimmetrikus bináris reláció az* S *halmaz felett, és legyen* $P \mathcal{R} Q$. *Az* $\mathcal{R}$ *gyenge korai biszimuláció,*

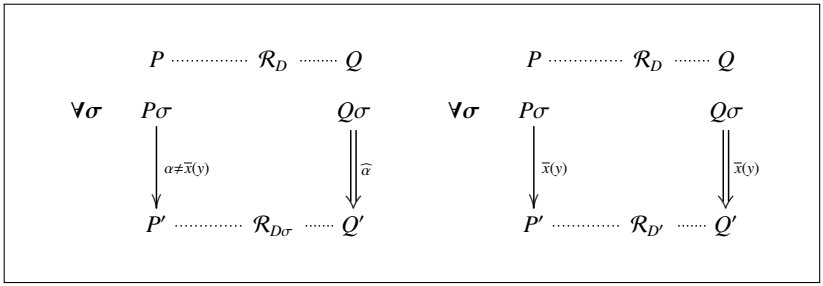
- *ha* $P \xrightarrow{\alpha} P'$, $\alpha \neq x(y)$ *és* $bn(\alpha) \notin fn(P, Q)$, *akkor* $Q \overset{\widehat{\alpha}}{\Longrightarrow} Q'$ *esetén* $P' \mathcal{R} Q'$,
- *ha* $x(y) \,.\, P$ *és* $x(y) \,.\, Q$*-ra* $y \notin fn(P, Q)$ *és* $x(y) \,.\, P \xrightarrow{xw} P[y := w]$, *akkor minden w-re van olyan* Q', *melyre* $Q \Longrightarrow\xrightarrow{xw} Q'$ *és* $Q'[y := w] \Longrightarrow Q''$, *melyre* $P[y := w] \mathcal{R} Q''$.

A gyenge korai biszimuláció átmenetei a 2.20. ábrán láthatók. A kölcsönös hasonlóság és a kongruencia jele $\dot{\approx}_e$ és $\approx_e$, definíciójuk a korábbi ilyen fogalmakhoz hasonló módon adható meg, nem részletezzük.

Megadhatjuk a gyenge nyitott D-biszimulációt is:



2.20. ábra. A gyenge korai biszimuláció



2.21. ábra. A gyenge nyitott D-biszimuláció

2.6.58. Definíció. Gyenge nyitott D-biszimuláció:

Adott D megkülönböztetésre legyen $\mathcal{R}_D$ egy szimmetrikus bináris reláció az S halmaz felett, és legyen $P\,\mathcal{R}_D\,Q$. Az $\mathcal{R}_D$ relációt gyenge nyitott D-biszimulációnak nevezzük, ha minden D-nek megfelelő σ helyettesítésre $P\sigma \xrightarrow{\alpha} P'$ és $bn(\alpha) \notin fn(P,Q)$ esetén

- *ha $\alpha = \overline{x}(y)$, akkor van olyan Q' , melyre $Q\sigma \overset{\overline{x}(y)}{\Longrightarrow} Q'$ és $P'\,\mathcal{R}_{D'}\,Q'$, ahol $D' = D\sigma \cup (y, fn(P\sigma, Q\sigma))^{=}$,*
- *ha $\alpha \neq \overline{x}(y)$, akkor van olyan Q' , melyre $Q\sigma \overset{\widehat{\alpha}}{\Longrightarrow} Q'$ és $P'\,\mathcal{R}_{D\sigma}\,Q'$.*

A gyenge nyitott D-biszimuláció átmeneteit a 2.21. ábrán láthatjuk. A

kölcsönös hasonlóság és a kongruencia jele $\dot{\approx}_o^D$ és $\approx_o^D$, definíciójuk a korábbi ilyen fogalmakhoz hasonló módon írható le, most nem részletezzük.

Befejezésképpen megadunk egy egyszerű állítást a nevek megkülönböztetésének halmazára, amely a folyamatok gyenge nyitott biszimulációjának vizsgálatához hasznosnak bizonyulhat:

2.6.59. Tétel. (A megkülönböztetés)

Ha $P \approx_o^D Q$ és $D \subseteq D'$, akkor $P \approx_o^{D'} Q$.

3. FEJEZET

A pi-kalkulus algebrai elmélete

Ebben a fejezetben a pi-kalkulus algebrai axiomatizálásával és ennek a rendszernek az alkalmazásával foglalkozunk, ami a következőket jelenti:

- megadunk egy axiómahalmazt,
- alkalmazzuk az *egyenlőségi érvelés* módszerét,

és ezzel a pi-kalkulusra vonatkozó „igaz”, azaz az axiómarendszernek megfelelő állításokat határozunk meg.

Az *egyenlőségi érvelés* formulák manipulálására szolgáló olyan módszer, amelynek lényege, hogy egyenlőek bármikor, bármilyen kontextusban helyettesíthetőek egymással. Az *egyenlőség* fogalomba bele kell értenünk a szintaktikai egyenlőséget, amit a $\equiv$ jellel jelölünk, és a szemantikai egyenlőséget, aminek a megszokott jelölése az $=$ jel.

3.1. A kölcsönös hasonlóság axiómarendszere

Első lépésben az erős késői kölcsönös hasonlóságot axiomatizáljuk, majd a 3.2. szakaszban megadjuk az erős késői kongruencia axiómarendszerét is [36, 33]. Ezért először csak azt tesszük fel, hogy az egyenlőségi érvelésben az egyenlőségre a reflexivitás, szimmetria és tranzitivitás szabályok érvényesülnek, de a helyettesítést nem alkalmazhatjuk, mert a helyettesítés szabályának alkalmazásával a 2.6. szakaszban a kongruencia tulajdonságokat definiáltuk.

Most csak a „véges” pi-kalkulussal foglalkozunk, a replikáció műveletét kihagyjuk a rendszerből.

Ebben a fejezetben használni fogjuk a folyamatkifejezések *fejnormálfomáját*, amelyet a következő definícióval adunk meg.

3.1.1. Definíció. A fej-normálforma:

Egy P folyamatkifejezés fej-normálformában van, ha prefixes folyamatkifejezések összege:

$$P \equiv \sum_i \alpha_i \,.\, P_i$$

A kölcsönös hasonlóság axiómarendszere a 3.1. és 3.2. táblázatokban látható. A táblázat első oszlopa az axiómák neveit tartalmazza.

Az axiómarendszert SGE-vel szokás jelölni, a *strong ground equivalence* szavak kezdőbetűiből.

STR	$Ha\ P \equiv Q, akkor\ P = Q$,		
CONGR1	$ha\ P = Q, akkor\ \overline{x}y \,.\, P$	$=$	$\overline{x}y \,.\, Q$,
	$\tau \,.\, P$	$=$	$\tau \,.\, Q$,
	$P + R$	$=$	$Q + R$,
	$P \mid R$	$=$	$Q \mid R$,
	$(\boldsymbol{\nu} x)\, P$	$=$	$(\boldsymbol{\nu} x)\, Q$,
	$[x = y]P$	$=$	$[x = y]Q$,
CONGR2	$Ha\ P[x := y] = Q[x := y],\ \ \forall y \in fn(P, Q) \cup \{y\}$,		
	$akkor\ x(y) \,.\, P$	$=$	$x(y) \,.\, Q$,
SUM0	$P + \mathbf{0} = P$,		
SUM	$P + P = P$,		
MATCH0	$[x = y]P = \mathbf{0}, ha\ x \neq y$,		
MATCH1	$[x = x]P = P$,		
REST0	$(\boldsymbol{\nu} x)\, P = P, ha\ x \notin fn(P)$,		
REST1	$(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)\, P = (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)\, P$,		
REST3	$(\boldsymbol{\nu} x)\,(P + Q) = (\boldsymbol{\nu} x)\, P + (\boldsymbol{\nu} x)\, Q$,		
REST4	$(\boldsymbol{\nu} x)\, \alpha \,.\, P = \alpha \,.\, (\boldsymbol{\nu} x)\, P, ha\ x \notin n(\alpha)$,		
REST5	$(\boldsymbol{\nu} x)\, \alpha \,.\, P = \mathbf{0}, ha\ x\ az\ \alpha\ alanya,$		
DEFEGY	$Ha\ P(x) \stackrel{\text{def}}{=\!=} Q, akkor\ P(y) = Q[x := y]$.		

3.1. táblázat. Az erős késői kölcsönös hasonlóság axiómái (1. rész)

EXP *Ha* $P = \sum_i \alpha_i . P_i$ *és* $Q = \sum_j \alpha_j . Q_j$,
ahol $\forall i, j$*-re* $bn(\alpha_i) \cup fn(Q) = \emptyset$, $bn(\beta_j) \cup fn(P) = \emptyset$
és α_i, β_j *egyike sem kötött output prefix, akkor*

$$P \mid Q = \sum_i \alpha_i . (P_i \mid Q) + \sum_j \beta_j . (P \mid Q_j) + \sum_{\alpha_i \rho \beta_j} \tau . R_{ij} ,$$

ahol $\alpha_i \rho \beta_j$ és R_{ij} *a következők:*
ha $\alpha_i = \overline{x}y$ *és* $\beta_j = x(z)$, *akkor* $R_{ij} = P_i \mid Q_j[z := y]$,
ha $\alpha_i = x(y)$ *és* $\beta_j = \overline{x}z$, *akkor* $R_{ij} = P_i[y := z] \mid Q_j$.

3.2. táblázat. Az erős késői kölcsönös hasonlóság axiómái (2. rész)

A táblázatokban nem szerepelnek az axiómák egyenlőségi relációból származó reflexiv, szimmetrikus és tranzitív következtetései.

A CONGR1 axióma azt mondja ki, hogy az input prefix kivételével minden *operátor* megtartja az egyenlőséget. Az input prefixre a CONGR2 axióma vonatkozik. Sajnos az input prefix nem őrzi meg a folyamatok kölcsönös hasonlóságát, ezért a

Ha $P = Q$ *akkor* $\pi . P = \pi . Q$

szabály nem írható fel a $\pi = x(y)$ prefixre, ezt a szabályt módosítanunk kell. A CONGR2 axióma azt állítja, hogy a kölcsönös hasonlóság következtetéséhez elegendő a kötött x-re vonatkozó helyettesítéseket a folyamatok szabad neveire és az y-ra vizsgálni, minden más helyettesítés az y injektív helyettesítéseiből származik (2.6.10. tétel második állítása). A vizsgálat véges időben elvégezhető, hiszen a helyettesítendő nevek száma véges.

A REST4 axióma szerint a korlátozások „átnyomhatók” a prefixeken vagy a REST5 axióma szerint a folyamattal együtt el is tűnhetnek, kivéve azokat az eseteket, amikor ezek az axiómák nem alkalmazhatók, amikor x az α szabad neve. A REST3 axióma a gyakorlat szempontjából jelentős, mert az összegre vonatkozó korlátozás két kisebb folyamatra, az összeg tagjaira tevődik át.

Látható, hogy a CONGR1 utolsó axiómája redundáns a MATCH0 és MATCH1 axiómák miatt, de az axiómát meghagyjuk, mert később még szükség lesz rá.

Az axiómákat arra is használhatjuk, hogy megmutassuk, minden kifejezés egy közös formára, a *fej-normálformára* hozható.

3.1.2. Tétel. (Fej-normálforma egzisztenciája)

Minden P folyamathoz megadható egy olyan Q folyamat, amelyik fej-normálformában van és a $P = Q$ állítás bizonyítható, azaz

$\mathrm{SGE} \vdash P = Q\,.$

Megjegyezünk még a tétellel kapcsolatban egy fontos állítást: Q-ban a skatulyázott prefixek mélysége nem nagyobb, mint a P-ben.

3.1.3. Tétel. (Az axiómarendszer helyesség tétele)

Ha a fenti axiómarendszerben az = egyenlőségeket az $\dot{\sim}_l$ erős késői kölcsönös hasonlósággal interpretáljuk, akkor az axiómarendszer szabályai érvényesek.

Tehát

$\mathrm{SGE} \vdash P = Q \rightsquigarrow P \;\dot{\sim}_l\; Q\,,$

ahol $\rightsquigarrow$ a következtetés jele. A fordított irányú állítás is igaz:

3.1.4. Tétel. (Az axiómarendszer teljesség tétele)

Minden P és Q folyamatra, ha $P \;\dot{\sim}_l\; Q$, akkor az axiómarendszer szerint $P = Q$.

A tétel állítása formálisan a

$P \;\dot{\sim}_l\; Q \rightsquigarrow \mathrm{SGE} \vdash P = Q$

alakban írható fel.

3.2. A kongruencia axiómarendszere

Próbáljuk meg a kongruenciára alkalmazni a kölcsönös hasonlóságra a 3.1. és 3.2. táblázatokban megadott axiómákat, nézzük az axiómákat a leírás sorrendjében.

A STR, CONGR1,..., SUM axiómákkal semmilyen probléma nem merülhet fel. A CONGR2 axiómát egyszerűsíteni is tudjuk, felírhatjuk

CONGR2' *Ha $P = Q$, akkor $x(y).P = x(y).Q$*

formában, hiszen a kongruencia definíciója ezt lehetővé teszi. A CONGR1 és

CONGR2' axiómák össze is vonhatók az *operátor* fogalmat használva:

CONGR *Ha $P = Q$ és P', Q' jelöli a P-re és Q-ra egy operátor alkalmazásával kapott kifejezést, akkor $P' = Q'$.*

Tovább vizsgálva az axiómákat, látjuk, hogy most a MATCH0 axióma nem alkalmazható, mert például az $[y := x]$ helyettesítésre az axióma nem lesz érvényes.

Az azonosság prefixek könnyű kezelésére bevezetjük az M-mel jelölt *többszörös azonosság* fogalmat, amelyik $x = y$ állítások konjunkciója. Ha az M-et egy $\pi\,.\,P$ folyamat prefixeként használjuk, akkor ezt az $[M]\,\pi\,.\,P$ kifejezéssel jelöljük. Ha $M \equiv m_1 \wedge m_2 \wedge \cdots \wedge m_k$, akkor

$$[M]\,\pi\,.\,P \stackrel{\text{def}}{=\!=} \textit{if } m_1 \textit{ then } (\textit{ if } m_2 \textit{ then } (\ldots (\textit{ if } m_k \textit{ then } P)\ldots))\,.$$

Azt mondjuk, hogy M implikálja az $N \equiv n_1 \wedge n_2 \wedge \cdots \wedge n_k$ többszörös azonosságot, ha minden σ helyettesítésre és minden i-re $m_i\sigma = n_i\sigma$.

$M \Leftrightarrow N$, azaz M ekvivalens N-nel, ha M implikálja N-et és N implikálja M-et.

Ezek után az azonosság prefixek kezelésére bevezethetjük az

MM1 $[M]\,P = [N]\,P, \quad ha\ M \Leftrightarrow N$

axiómát.

Sajnos az utolsó, az EXP axióma nem lesz számunkra jó. Ugyanis a helyettesítés műveletére a párhuzamos folyamatok kölcsönös hasonlósága nem feltétlenül marad meg, azaz a folyamatok nem biztos, hogy kongruensek maradnak, mint azt a 2.6.14. példában láttuk. A módosított axiómát nevezzük EXP'-nek.

A kongruencia axiómarendszere a 3.3. és 3.4. táblázatokban látható. Az axiómarendszert SGE'-vel jelöljük.

Az SGE' axiómarendszer is alkalmas a folyamatkifejezések fejnormálfomája egzisztenciájának bizonyítására, de a fogalmak egy kicsit változnak az előző szakaszban megadottakhoz viszonyítva.

3.2.1. Definíció. Teljes konjukció:

> *Legyen V a nevek egy halmaza. Ekkor az M konjunkciót a V-re nézve teljesnek nevezzük, ha M $x = y$ egyenlőségeket tartalmaz az $x, y \in V$ nevekre.*

Ha $x = y \notin M$, akkor az $x \neq y$ állítás igaz. Az M halmaz tehát egyértelműen deklarálja, hogy a V-ben mely nevek azonosak.

STR	$Ha\ P \equiv Q,\ akkor\ P = Q$,
CONGR	$ha\ P = Q,\ akkor\ \diamond P = \diamond Q,\quad \diamond\ egy\ operátor,$
SUM0	$P + \mathbf{0} = P$,
SUM	$P + P = P$,
MM1	$[M]\,P = [N]\,P,\quad ha\ M \Leftrightarrow N$,
MM2	$[x = y]P = \mathbf{0},\ ha\ x \neq y$,
MM3	$[x = x]P = P$,
MM4	$[M](P_1 + P_2) = [M](P_1) + [M](P_2)$,
MM5	$[M]\alpha\,.\,P = [M](\alpha\,.\,[M]\,P),\ ha\ bn(\alpha) \notin M$,
MM6	$[x = y]\alpha\,.\,P = [x = y](\alpha[y := x])\,.\,P$,
REST0	$(\boldsymbol{\nu} x)[M]P = \mathbf{0},\ ha\ x \neq y$,
REST1	$(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)\,P = (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)\,P$,
REST3	$(\boldsymbol{\nu} x)\,(P + Q) = (\boldsymbol{\nu} x)\,P + (\boldsymbol{\nu} x)\,Q$,
REST4	$(\boldsymbol{\nu} x)\,\alpha\,.\,P = \alpha\,.\,(\boldsymbol{\nu} x)\,P,\ ha\ x \notin n(\alpha)$,
REST5	$(\boldsymbol{\nu} x)\,\alpha\,.\,P = \mathbf{0},\ ha\ x\ az\ \alpha\ alanya,$
DEFEGY	$Ha\ P(x) \overset{\text{def}}{=\!=} Q,\ akkor\ P(y) = Q[x := y]$.

3.3. táblázat. Az erős késői kongruencia axiómái (1. rész)

Azt mondjuk, hogy az M konjunkció *megfelel* egy σ helyettesítésnek, ha M-ben $x = y$ akkor és csak akkor szerepel, ha $x\sigma = y\sigma$.

3.2.2. Definíció. A* V *halmazra vonatkozó fej-normálforma:

Egy P folyamatkifejezés a nevek egy V halmazára nézve fej-normálformában van, ha $[M]\,\alpha\,.\,P$ folyamatkifejezések összege, azaz

$$P \equiv \sum_i [M_i]\,\alpha_i\,.\,P_i\ ,$$

ahol minden i-re $bn(\alpha_i) \notin V$ és M_i a V-re nézve teljes konjunkció.

EXP' *Ha* $P = \sum_i [M_i]\,\alpha_i\,.\,P_i$ *és* $Q = \sum_j [N_j]\beta_j\,.\,Q_j$,
ahol $\forall i, j$*-re* $bn(\alpha_i) \cup fn(Q) = \emptyset$, $bn(\beta_j) \cup fn(P) = \emptyset$
és α_i, β_j *egyike sem kötött output prefix, akkor*

$$P \mid Q = \sum_i [M_i]\,\alpha_i\,.\,(P_i \mid Q) + \sum_j [N_j]\beta_j\,.\,(P \mid Q_j) + \sum_{\alpha_i \rho \beta_j} [M_i \wedge N_j \wedge [x_i = y_j]]\tau\,.\,R_{ij}\ ,$$

ahol $\alpha_i \rho \beta_j, a_i, b_j$ és R_{ij} *a következők:*
ha $\alpha_i = \overline{x_i}y$ *és* $\beta_j = y_j(z)$, *akkor* $R_{ij} = P_i \mid Q_j[z := y]$,
ha $\alpha_i = x_i(y)$ *és* $\beta_j = \overline{y_j}z$, *akkor* $R_{ij} = P_i[y := z] \mid Q_j$.

3.4. táblázat. A kongruencia axiómái (2. rész)

Az axiómákat arra is használhatjuk, hogy megmutassuk, minden kifejezés egy közös formára, a *fej-normálformára* hozható.

3.2.3. Tétel. (Fej-normálforma egzisztenciája)

Minden V névhalmahoz és minden P folyamathoz megadható egy olyan Q folyamat, amelyik a V-re nézve fej-normálformában van és a P = Q állítás bizonyítható, azaz

SGE' $\vdash P = Q$.

Most is teljesül az az állítás, hogy Q-ban a skatulyázott prefixek mélysége nem nagyobb, mint a P-ben.

Ezek után megadjuk az SGE' axiómarendszerre és a kongruenciára vonatkozó két fő állítást.

3.2.4. Tétel. (Az axiómarendszer helyesség tétele)

Ha a kongruenciára megadott axiómarendszerben az = egyenlőségeket az $\sim_l$ *erős késői kongruenciával interpretáljuk, akkor az axiómarendszer szabályai érvényesek.*

Tehát

SGE' $\vdash P = Q \rightsquigarrow P \sim_l Q$.

A fordított irányú állítás is igaz:

3.2.5. Tétel. (Az axiómarendszer teljesség tétele)

Minden P és Q folyamatra, ha $P \sim_l Q$, akkor az axiómarendszer szerint $P = Q$.

A tétel állítása formálisan a

$P \sim_l Q \rightsquigarrow$ SGE' $\vdash P = Q$

alakban írható fel.

4. FEJEZET

Speciális pi-kalkulusok

A elmúlt két és fél évtizedben a pi-kalkulusnak nagyon sok változatát dolgozták ki, mindegyikben a folyamatoknak egy vagy több speciális jellemzőjét helyezve előtérbe. Francesco Zappa Nardelli (Département d'informatique, École normale supérieure, Paris) mondta egy előadásán: "Pi-calculus literature describes *zillions* of slightly different languages, semantics, equivalencies." Ezek között vannak sikeres, használt és jól bevált elméletek, de vannak kevésbé sikeresek is.

A sikeresnek mondható elméletek közül mutatunk be néhányat ebben a fejezetben.

4.1. Az aszinkron pi-kalkulus

A pi-kalkulus kommunikációja szinkronizált, a kommunikáció akkor zajlik le, amikor az output és az input művelet aktív, a két művelet egyidőben és párhuzamosan fut. Ugyanakkor, például a disztributív rendszerekben, az output művelet végrehajtása időben lényegesen korábban végrehajtódhat, jóval megelőzheti az input műveletet. Ezt úgy lehet elképzelni, hogy az output művelet az adatot egy tárolóra helyezi, és az adat addig ott marad, amíg az input művelet onnan ki nem olvassa. Feltesszük, hogy ez a tároló tetszőlegesen nagy kapacitású, és nem őrzi meg az elhelyezett adatok beírási időpontjait, azaz a beírási sorrendet.

Az alapgondolat az, hogy az $\overline{x}y\,.\,P$ folyamatot bontsuk két részre, vegyük le az output műveletet, és az outputot a P-vel futtassuk párhuzamosan (4.1. ábra). Az ábrán a P' arra utal, hogy mire a kommunikáció az x néven megtörténik, a P végrehajtása már sok lépéssel előrehaladhatott.

Ez az aszinkronitás a pi-kalkulusba úgy építhető be, hogy output prefixe

$$\overline{x}y\,.\,P \mid x(z)\,.\,Q \rightarrow \overline{x}y \mid P \mid x(z)\,.\,Q \ldots \rightarrow \ldots \overline{x}y \mid P' \mid x(z)\,.\,Q \xrightarrow{\tau} P' \mid Q[z := y]$$

4.1. ábra. Az $\overline{x}y\,.\,P$ átalakítása

csak a **0** folyamatnak lehet, azaz csak $\overline{x}y\,.\,\mathbf{0} \equiv \overline{x}y$ alakú output műveleteket engedünk meg. Ezt nevezzük *nemfelügyelt outputnak*, mivel az output adatának fogadásáról nincs semmilyen információ.

A pi-kalkulusnak azt az alrendszerét, amelyik ilyen típusú kommunikációt ír le, *aszinkron pi-kalkulusnak* nevezzük, és az $A\pi$ jellel jelöljük [22].

4.1.1. Szintaxis és műveleti szemantika

4.1.1. Definíció. Az Aπ-kalkulus szintaxisa:

Az $A\pi$-kalkulusban a prefixeket és a folyamatokat a következő kifejezésekkel adjuk meg:

$$P ::= \overline{x}y\,.\,\mathbf{0} \mid M \mid P \mid P \mid (\nu x)\,P \mid !P$$
$$M ::= \mathbf{0} \mid x(y)\,.\,P \mid \tau\,.\,P \mid M + M$$

A definícióban nem írtuk ki a kötött output prefixet, mivel ez a 2.4.7. definíció szerint a $(\nu y)\,\overline{x}y$ kifejezés rövid jelölése. Az $A\pi$-kalkulusban nincs $[x = y]$ azonosság prefix, ennek az oka a rendszer könnyebb kezelhetősége.

Az alapvető különbség a pi-kalkulushoz viszonyítva az, hogy nincs *vagy* művelet tetszőleges folyamatok között. A definícióból látható, hogy az összeg kifejezés tagjai csak input vagy τ prefixes folyamatok lehetnek. Ez összhangban van azzal a gondolattal, hogy a kalkulusnak aszinkron kommunikációt kell megvalósítani.

Az $A\pi$-kalkulus szabad és kötött neveinek definíciója, szerkezeti kongruenciájának szabályai a 2.3. szakaszban leírtakkal egyeznek meg. A korai műveleti szemantikát (2.6.3. pont) használjuk, és a 4.1. táblázatban megadjuk az $A\pi$-kalkulus műveleti szemantikájának szabályait.

A táblázatból látható, hogy az Asz-Output szabály pontosan leírja a nemfelügyelt output műveletet, a többi szabály pedig megegyezik a korai szemantika műveleti szabályaival.

$$\frac{P' \equiv P, \quad P \xrightarrow{\alpha} Q, \quad Q \equiv Q'}{P' \xrightarrow{\alpha} Q'} \quad [\textsc{Struct}]$$

$$\frac{}{\overline{x}y \,.\, P \xrightarrow{\overline{x}y} \mathbf{0}} \quad [\textsc{Asz-Output}] \qquad \frac{\overline{x}y \,.\, P \xrightarrow{\overline{x}y} P, \quad x \neq y}{\overline{x}(y) \,.\, P \xrightarrow{\overline{x}(y)} P} \quad [\textsc{Open}]$$

$$\frac{}{x(y) \,.\, P \xrightarrow{xw} P[y := w]} \quad [\textsc{Input}] \qquad \frac{}{\tau \,.\, P \xrightarrow{\tau} P} \quad [\textsc{Tau}]$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \quad [\textsc{Sum}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad bn(\alpha) \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad [\textsc{Par}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad x \notin n(\alpha)}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'} \quad [\textsc{Res}]$$

$$\frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \quad [\textsc{Rep}]$$

$$\frac{P \xrightarrow{\overline{x}y} P', \quad Q \xrightarrow{xw} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \quad [\textsc{Early-Com}]$$

$$\frac{\overline{x}(y) \,.\, P \xrightarrow{\overline{x}(y)} P, \quad x(z) \,.\, Q \xrightarrow{x(y)} Q[z := y]}{\overline{x}(y) \,.\, P \mid x(z) \,.\, Q \xrightarrow{\tau} (\nu y)(P \mid Q[z := y])} \quad [\textsc{Close}]$$

4.1. táblázat. Az $A\pi$-kalkulus műveleti szemantikájának szabályai

A következő példában az $A\pi$-kalkulus folyamataira megadunk néhány egyszerű, de érdekes átalakítási szabályt, amelyek az $A\pi$-kalkulus definíciójából és a műveleti szemantika szabályaiból következnek.

4.1.2. Példa. *(Folyamatok átalakításai)*

- Ha $P \xrightarrow{\overline{x}y} P'$, akkor $P \equiv \overline{x}y \mid P'$,
- ha $P \xrightarrow{\overline{x}(y)} P'$, akkor $P \equiv (\nu y)\,(\overline{x}y \mid P')$,

- ha $P \xrightarrow{\overline{x}y} \xrightarrow{\alpha} P'$ és $P \xrightarrow{\alpha} \xrightarrow{\overline{x}y} P''$, akkor $P' \equiv P''$,
- ha $P \xrightarrow{\overline{x}(y)} \xrightarrow{\alpha} P'$, $P \xrightarrow{\alpha} \xrightarrow{\overline{x}(y)} P''$ és $y \notin n(\alpha)$, akkor $P' \equiv P''$,
- ha $P \xrightarrow{\overline{x}y} \xrightarrow{xw} P'$ és $w \notin fn(P)$, akkor $P \xrightarrow{\tau} P''[w := y]$ és $P' \equiv P''[w := y]$,
- ha $P \xrightarrow{\overline{x}(y)} \xrightarrow{xw} P'$ és $w \notin fn(P)$, akkor $P \xrightarrow{\tau} (\boldsymbol{\nu} y)\, P''[w := y]$ és $P' \equiv (\boldsymbol{\nu} y)\, P''[w := y]$. □

4.1.3. Példa. *(Output műveletek sorozata)*

Ha $y, z \notin fn(P)$, akkor nézzük meg a

$(\boldsymbol{\nu} y, z)\,(\overline{x}y \mid \overline{y}z \mid \overline{z}u \mid P)$

folyamatot. Megmutatjuk, hogy a P működésére nincs semmilyen megkötés, de P az outputokat csak balról-jobbra haladó sorrendben használhatja fel.

Mivel feltettük, hogy $y, z \notin fn(P)$, a P-ben nem lehet $y()$ vagy $z()$ input prefix, így első prefixként sem, ezért az $\overline{y}z$ és $\overline{z}u$ outputok nem kommunikálhatnak. Csak az $\overline{x}y$ jöhet szóba, tehát P legyen $x(u)\,.\,P'$ alakú. Ekkor

$$(\boldsymbol{\nu} y, z)\,(\underline{\overline{x}y} \mid \overline{y}z \mid \overline{z}u \mid \underline{x(u)\,.\,P'}) \xrightarrow{\tau}$$
$$(\boldsymbol{\nu} y, z)\,(\overline{y}z \mid \overline{z}u \mid P'[u := y])\ .$$

Ahhoz, hogy a $\overline{y}z$ következzen, P'-nek $u(v)\,.\,P''$ alakúnak kell lennie, amit az $[u := y]$ helyettesítés $y(v)\,.\,P''$-ra ír át. Tehát

$$(\boldsymbol{\nu} y, z)\,(\overline{y}z \mid \overline{z}u \mid P'[u := y]) \equiv$$
$$(\boldsymbol{\nu} y, z)\,(\underline{\overline{y}z} \mid \overline{z}u \mid \underline{y(v)\,.\,P''})\ ,$$

és ekkor

$$(\boldsymbol{\nu} y, z)\,(\overline{z}u \mid \ .\,P''[v := z])\ .$$

Az előző gondolatmenethez hasonlóan, a $\overline{z}u$ végrehajtásához $P'' = v(w)\,.\,P'''$, ami a helyettesítés után $z(w)\,.\,P'''$ lesz. Így

$$(\boldsymbol{\nu} y, z)\,(\overline{z}u \mid P''[v := z]) \equiv$$
$$(\boldsymbol{\nu} y, z)\,(\overline{z}u \mid z(w)\,.\,P''') \xrightarrow{\tau}$$
$$(\boldsymbol{\nu} y, z)\,P'''[w := u] \equiv$$
$$P^{iv}\ .$$

Tehát az output műveletek valóban balról-jobbra haladva hajtódtak végre, és a

$P \equiv x(u) \,.\, u(v) \,.\, v(w) \,.\, P^{iv}$ folyamat valóban az outputok sorrendjében olvasta a kiküldött adatokat. □

A pi-kalkulus output és input prefixes folyamatai átalakíthatók az $A\pi$-kalkulus kifejezésére, a konverzió jelölésére a $[\![\ ]\!]$ zárójeleket használjuk [5]. (Ez nem tévesztendő össze a természetes számok és a lambda-kifejezések folyamatkifejezéseivel, amelyeket a 2.5.5. és 2.5.8. pontokban használtunk.)

Ha $w, u \notin fn(P, x, y)$, akkor legyen

$$[\![\overline{x}y \,.\, P]\!] \stackrel{\text{def}}{=\!=} (\boldsymbol{\nu} w)\,(\overline{x}w \mid w(u) \,.\, (\overline{u}y \mid [\![P]\!]))\ ,$$

$$[\![x(z) \,.\, P]\!] \stackrel{\text{def}}{=\!=} x(w) \,.\, (\boldsymbol{\nu} u)\,(\overline{w}u \mid u(z) \,.\, [\![P]\!])\ .$$

A műveletek átalakítása a következőképpen történhet:

$$[\![\mathbf{0}]\!] \stackrel{\text{def}}{=\!=} \mathbf{0}$$

$$[\![P \mid Q]\!] \stackrel{\text{def}}{=\!=} [\![P]\!] \mid [\![Q]\!]$$

$$[\![!P]\!] \stackrel{\text{def}}{=\!=} !\,[\![P]\!]$$

$$[\![(\boldsymbol{\nu} x)\,P]\!] \stackrel{\text{def}}{=\!=} (\boldsymbol{\nu} x)\,[\![P]\!]$$

4.1.4. Példa. *(Szinkron kommunikáció az $A\pi$-kalkulusban – 1.)*

Nézzük meg a pi-kalkulusbeli

$$\overline{x}y \,.\, P \mid x(z) \,.\, Q \xrightarrow{\tau} P \mid Q[z := y]$$

kommunikáció végrehajtását az $A\pi$-kalkulusban.

$$[\![\overline{x}y \,.\, P]\!] \mid [\![x(z) \,.\, Q]\!] \stackrel{\text{def}}{=\!=}$$

$$(\boldsymbol{\nu} w)\,(\overline{x}w \mid w(u) \,.\, (\overline{u}y \mid [\![P]\!])) \mid x(w) \,.\, (\boldsymbol{\nu} u)\,(\overline{w}u \mid u(z) \,.\, [\![Q]\!]) \equiv$$

$$(\boldsymbol{\nu} w)\,(\underbrace{\overline{x}w}\ \mid w(u) \,.\, (\overline{u}y \mid [\![P]\!]) \mid \underbrace{x(w) \,.\, (\boldsymbol{\nu} u)\,(\overline{w}u \mid u(z) \,.\, [\![Q]\!])}\ ) \xrightarrow{\tau}$$

$$(\boldsymbol{\nu} w)\,(w(u) \,.\, (\overline{u}y \mid [\![P]\!])\ \mid (\boldsymbol{\nu} u)\,(\overline{w}u \mid u(z) \,.\, [\![Q]\!])\,) \equiv$$

$$(\boldsymbol{\nu} w, u)\,(\underbrace{w(u) \,.\, (\overline{u}y \mid [\![P]\!])}\ \mid \underbrace{\overline{w}u}\ \mid u(z) \,.\, [\![Q]\!]\,) \xrightarrow{\tau}$$

$$(\boldsymbol{\nu} w, u)\,(\overline{u}y \mid [\![P]\!] \mid u(z) \,.\, [\![Q]\!]\,) \xrightarrow{\tau}$$

$$(\boldsymbol{\nu} w, u)\,(\underbrace{\overline{u}y}\ \mid [\![P]\!] \mid \underbrace{u(z) \,.\, [\![Q]\!]}) \xrightarrow{\tau}$$

$$(\boldsymbol{\nu} w, u)\,([\![P]\!] \mid [\![Q]\!][z := y]\,) \equiv$$

$$[\![P]\!] \mid [\![Q]\!][z := y]\ .$$

□

Ismert egy másik, rövidebb és egyszerűbb átalakítás is, polimorfikus input és output műveletekkel:

$$[\![\overline{x}y\,.\,P]\!] \stackrel{\text{def}}{=\!=} (\nu u)\,(\overline{x}\,\langle y,u\rangle \mid u()\,.\,[\![P]\!])$$

$$[\![x(v)\,.\,Q]\!] \stackrel{\text{def}}{=\!=} x(v,w)\,.\,(\overline{w}\,\langle\rangle \mid [\![Q]\!]) \qquad w \notin Q$$

4.1.5. Példa. *(Szinkron kommunikáció az Aπ-kalkulusban – 2.)*

Nézzük meg az előző példa folyamatait a második átalakítási módszerrel.

$$[\![\overline{x}y\,.\,P]\!] \mid [\![x(z)\,.\,Q]\!] \stackrel{\text{def}}{=\!=}$$

$$(\nu u)\,(\overline{x}\,\langle y,u\rangle \mid u()\,.\,[\![P]\!]) \mid x(v,w)\,.\,(\overline{w}\,\langle\rangle \mid [\![Q]\!]) \equiv$$

$$(\nu u)\,(\underline{\overline{x}\,\langle y,u\rangle} \mid u()\,.\,[\![P]\!] \mid \underline{x(v,w)\,.\,(\overline{w}\,\langle\rangle \mid [\![Q]\!])}\,) \stackrel{\tau}{\longrightarrow}$$

$$(\nu u)\,(u()\,.\,[\![P]\!] \mid (\overline{w}\,\langle\rangle \mid [\![Q]\!])[v := y, w := u]\,)) \equiv$$

$$(\nu u)\,(u()\,.\,[\![P]\!] \mid (\overline{u}\,\langle\rangle \mid [\![Q]\!][v := y, w := u]\,)) \stackrel{\tau}{\longrightarrow}$$

$$(\nu u)\,([\![P]\!] \mid [\![Q]\!][v := y, w := u]\,)\,,$$

és mivel feltettük, hogy $w \notin Q$,

$$(\nu u)\,([\![P]\!] \mid [\![Q]\!][v := y, w := u]\,) \equiv$$

$$[\![P]\!] \mid [\![Q]\!][v := y]\,. \qquad \square$$

4.1.2. Biszimuláció

Mivel ebben a kalkulusban a külső szemlélő számára könnyen megfigyelhető esemény csak az output művelet, várható, hogy a kölcsönös hasonlóság szempontjából ez a kalkulus nem lesz nagyon hatékony. Az *Aπ*-kalkulus folyamatainak hasonlóságát a *nyilazott biszimulációval* kapcsolatos fogalmak felhasználásával elemezzük.

Először definiáljuk a megfigyelhető nevet és a nyilazott biszimulációt. Ezek megegyeznek a 2.6.4.. és 2.6.27. definíciókban leírtakkal, attól eltekintve, hogy itt a definíciókban nem szerepel az input prefix.

4.1.6. Definíció. Megfigyelhető név az Aπ-kalkulusban:

Tegyük fel, hogy az x név nem esik korlátozás alá, ekkor ha a P folyamat soronkövetkező művelete

- *egy $\overline{x}y$ vagy $\overline{x}(y)$ prefix végrehajtása, akkor az x-t megfigyelhető névnek nevezzük.*

Ha a P folyamat megfigyelhető neve $\overline{x}$, akkor ezt a tulajdonságot a $P_A{\downarrow}_{\overline{x}}$ jellel jelöljük.

A definíció alapján tehát a megfigyelhető név mindig egy output prefix *alanya*. A $P_A{\downarrow}_{\overline{x}}$ jelölésben az A betű az „aszinkron" jelzőre utal.

4.1.7. Definíció. Nyilazott biszimuláció az Aπ-kalkulusban:

Legyen $\mathcal{R}$ egy szimmetrikus bináris reláció az S halmaz felett, és legyen $P\mathcal{R}Q$. Az $\mathcal{R}$ relációt nyilazott biszimulációnak nevezzük,

- *ha $P_A{\downarrow}_{\overline{x}}$, akkor $Q_A{\downarrow}_{\overline{x}}$,*
- *ha $P \xrightarrow{\tau} P'$, akkor van olyan Q', melyre $Q \xrightarrow{\tau} Q'$ és $P'\mathcal{R}Q'$.*

A nyilazott kölcsönös hasonlóságot a 2.6.28.-ban definiáltuk, ez a definíció változatlanul áthozható az $A\pi$-kalkulusba.

4.1.8. Definíció. Nyilazott kölcsönös hasonlóság:

A nyilazott kölcsönös hasonlóság legyen a nyilazott biszimulációk uniója, és a nyilazott kölcsönös hasonlóság jele legyen $\dot{\sim}_{A\downarrow}$. Azt mondjuk, hogy P és Q folyamatok nyilazott kölcsönös hasonlóság relációban vannak, azaz $P \dot{\sim}_{A\downarrow} Q$, ha van olyan $\mathcal{R}$ nyilazott biszimuláció, melyre $P\mathcal{R}Q$.

A nyilazott kongruenciát a pi-kalkulus környezeteivel adtuk meg, így ennél a fogalomnál már meg kell különböztetnünk a két kalkulust, mivel most az $A\pi$-kalkulus környezeteit kell használnunk. A kongruencia neve legyen *aszinkron nyilazott kongruencia*.

4.1.9. Definíció. Aszinkron nyilazott kongruencia:

A P és Q folyamat aszinkron nyilazott kongruens, ha minden Aπ-kalkulusbeli C kontextusra $C[P] \dot{\sim}_{A\downarrow} C[Q]$. A kongruencia jele legyen $\sim_{A\downarrow}$, tehát ekkor $P{\sim}_{A\downarrow}Q$.

4.1.10. Példa. *(Különbség a nyilazott kongruenciák között)*

Nézzük a

$P \stackrel{\text{def}}{=\!=} !x(y) \,.\, \overline{x}y$

és a

$Q \stackrel{\text{def}}{=\!=} \mathbf{0}$

folyamatokat. Az $A\pi$-kalkulusban egyik folyamatnak sincs megfigyelhető neve, és nincs τ átmenete sem, P és Q aszinkron nyilazott kongruensek. A P úgy értelmezhető, hogy a tárolóból olvas egy nevet és ezt ugyanazon a néven azonnal visszaadja, tehát lényegében, valóban nem csinál semmit. Ugyanakkor a két folyamat nyilvánvalóan nem nyilazott kongruens a pi-kalkulusban. □

A továbbiakban azt vizsgáljuk, hogy az $A\pi$-kalkulusban az aszinkron kongruencia milyen kapcsolatban van más kongruenciákkal. Először az aszinkron biszimulációt, majd az aszinkron kölcsönös hasonlóságot definiáljuk.

4.1.11. Definíció. Aszinkron biszimuláció:

Legyen $\mathcal{R}$ egy szimmetrikus bináris reláció az S halmaz felett, és legyen $P\,\mathcal{R}\,Q$. Feltéve, hogy $y \notin fn(P, Q)$, az $\mathcal{R}$ aszinkron biszimuláció,

- *ha $P \xrightarrow{\alpha} P'$, $\alpha = \overline{x}y, \overline{x}(y)$ vagy τ, akkor $Q \xrightarrow{\alpha} Q'$ esetén $P'\,\mathcal{R}\,Q'$,*
- *ha $P \xrightarrow{xy} P'$, akkor*
  - *ha $Q \xrightarrow{xy} Q'$, akkor $P'\,\mathcal{R}\,Q'$, vagy*
  - *ha $Q \xrightarrow{\tau} Q'$, akkor $P'\,\mathcal{R}\,(Q' \mid \overline{x}y)$ minden y-ra .*

A definíció második fő pontja is az $A\pi$-kalkulusnak azt a tulajdonságát mutatja, hogy csak az output műveleteket tudjuk megfigyelni (4.2. ábra).

Az $A\pi$-kalkulusban az aszinkron kölcsönös hasonlóságot a korábbi rendszerekből már jól ismert definícióval adjuk meg.

4.1.12. Definíció. Aszinkron kölcsönös hasonlóság:

Az aszinkron kölcsönös hasonlóság legyen az aszinkron biszimulációk uniója, és az aszinkron nyilazott kölcsönös hasonlóság jele legyen $\dot{\sim}_a$. Azt mondjuk, hogy P és Q folyamatok aszinkron kölcsönös hasonlóság relációban vannak, azaz $P \;\dot{\sim}_a\; Q$, ha van olyan $\mathcal{R}$ aszinkron biszimuláció, melyre $P\,\mathcal{R}\,Q$.

$$
\begin{tikzcd}
P \arrow[r, dotted, no head] \arrow[d, "xy"] & \mathcal{R} \arrow[r, dotted, no head] & Q \arrow[d, "\tau"] & \\
P' \arrow[r, dotted, no head] & \mathcal{R} \arrow[r, dotted, no head] & Q' \mid \overline{x}y & \forall y
\end{tikzcd}
$$

4.2. ábra. Az aszinkron biszimuláció definíciójának utolsó sora

Az aszinkron kölcsönös hasonlóságnak van egy nagyon kellemes tulajdonsága, bizonyítható, hogy ez a tulajdonság minden helyettesítésre, minden kontextusra, még az input prefixre is, megmarad. Ennek alapján azonnal adódik a következő tétel.

4.1.13. Tétel.

Az Aπ-kalkulusban az aszinkron kölcsönös hasonlóság kongruencia.

Jelöljük az aszinkron kongruenciát $\sim_a$-val, így ennek a tételnek a $\dot{\sim}_a \equiv \sim_a$ nevet is adhatjuk.

Már többször utaltunk rá, a nyilazott kongruencia a megfigyelhető nevekkel foglalkozik, így szerepe a gyakorlati alkalmazásokban jelentős. Éppen ezért az aszinkron rendszerek vizsgálatában különösen nagy szerepet játszik a következő tétel.

4.1.14. Tétel. $(\sim_{A\downarrow} \equiv \sim_a)$

Minden P és Q Aπ-kalkulusbeli folyamatra $P \sim_{A\downarrow} Q$ akkor és csak akkor áll fenn, ha $P \sim_a Q$.

4.2. A lokalizált pi-kalkulus

Az $x(y).P$ folyamat az y néven fogadja a bejövő adatot és ennek a feldolgozása a P-ben történik meg. Erre több lehetőség is van, például

- y egy újabb input alanya lesz, azaz ezen az y néven fog majd olvasni,
- y egy output alanya lesz, vagyis az y néven fog adatot küldeni,
- továbbküldi valamilyen néven az y-t,

- y egy azonosság prefixben szerepel.

Számunkra az első eset az igazán érdekes. Tegyük fel, hogy a pi-kalkulusnak megadtuk egy *elosztott implementációját* (ezzel majd a 4.3.1. pontban fogunk foglalkozni), amelyből most az a fontos számunkra, hogy minden névhez (csatornához) egyértelműen hozzárendelünk egy *helyet*. Például tegyük fel, hogy az x, y, z nevekhez hozzárendeltünk egy l_1 helyet, az u, v, w helyekhez egy másik, l_2 helyet. Az $x(y) \,.\, P$ input prefixes folyamat az x miatt az l_1 helyen van, az $\overline{u}w$ output az implementáció szerint bármelyik helyről jelentkezhet.

Adjunk meg két kifejezést:

$$\overline{x}w \quad \text{és} \quad x(u) \,.\, u(v) \,.\, Q \;.$$

A második folyamat az x név helyén, azaz l_1-en van. Nézzük meg, hogy mi történik:

$$\underline{\overline{x}w} \mid \underline{x(u) \,.\, u(v) \,.\, Q} \xrightarrow{\tau}$$
$$w(v) \,.\, Q[u := w] \;,$$

azaz a Q-t tartalmazó kifejezés helyét senki nem változtatta meg, még mindig az l_1-en van, de a kommunikáció eredményéből azt látjuk, hogy nem itt van a helye, már az l_2-n kellene lennie.

Ezt a problémát kétféleképpen tudjuk megoldani:

- megtiltjuk az $x(u) \,.\, u(v)$ alakú egymás utáni inputok használatát,
- nem engedjük meg az $x(u) \,.\, u(v)$ kifejezések használatát, de az ilyen kifejezést egy új input-konstrukcióval átalakítjuk, és a kalkulusban az átalakított kifejezést használjuk.

Az első változat kalkulusa a *lokalizált pi-kalkulus*, amit $L\pi$-vel jelölünk, a másodikat *lineáris továbbító* pi-kalkulusnak nevezzük, és a kalkulus jele $Lf\pi$, ahol az f az angol továbbító (*forwarder*) szóra utal. Az $Lf\pi$ kalkulussal a következő szakaszban foglalkozunk.

4.2.1. Szintaktika, műveleti szemantika, biszimuláció

A lokalizált pi-kalkulus az aszinkron pi-kalkulus alrendszerének tekinthető. A 4.1. szakaszban tárgyalt fogalmak, definíciók, tételek jelentős része a megfelelő korlátozásokkal a lokalizált pi-kalkulusra is érvényes, ezért ezeket itt nem ismételjük meg, inkább a lokalizált pi-kalkulus specifikus tulajdonságait elemezzük.

4.2.1. Definíció. Az Lπ-kalkulus szintaxisa:

Az Lπ-kalkulusban a prefixeket és a folyamatokat a következő kifejezésekkel adjuk meg:

$$P ::= M \mid P \mid P \mid (\nu x)\, P \mid !P$$
$$M ::= \mathbf{0} \mid \overline{x}y\,.\,P \mid x(y)\,.\,P \mid \tau\,.\,P\ ,$$

ahol $x(y)\,.\,P$ esetén az y név P-ben nem lehet input alanya.

Az $L\pi$-kalkulusban tehát csak az output, input és a τ prefix megengedett.

A definíció utolsó sorában szereplő szöveges feltétel éppen arról a kifejezésről szól, amiről a bevezetőben megmutattuk, hogy milyen problémát okoz. Tehát ebben a a kalkulusban $x(y)\,.\,(\ldots y(v)\ldots)$ szerkezetű folyamatkifejezések nem lehetnek.

Az $L\pi$-kalkulus *műveleti szemantikája* azonos az $A\pi$-kalkulus szemantikájával, a szabályokat nem ismételjük meg. Az aszinkron pi-kalkulusnál a nyilazott biszimulációt elemeztük, itt most a *gyenge nyilazott biszimulációval* és *kongruenciával* kezdjük.

Először definiáljuk a megfigyelhető nevet és a nyilazott biszimulációt.

4.2.2. Definíció. Megfigyelhető név az Lπ-kalkulusban:

Tegyük fel, hogy az x név nem esik korlátozás alá, ekkor ha a P folyamat soronkövetkező művelete

- *egy $\overline{x}y$ vagy $\overline{x}(y)$ prefix végrehajtása, akkor az x-et megfigyelhető névnek nevezzük.*

Ha a P folyamat megfigyelhető neve $\overline{x}$, akkor ezt a tulajdonságot a $P\,{}_L{\downarrow}_{\overline{x}}$ jellel jelöljük. Ha $P \Rightarrow P'$ és $P'\,{}_L{\downarrow}_{\overline{x}}$, akkor a $P\,{}_L{\Downarrow}_{\overline{x}}$ jelölést használjuk.

A definíció alapján tehát a megfigyelhető név mindig egy output prefix *alanya*. A jelölésekben az L betű a „lokalizált” jelzőre utal.

4.2.3. Definíció. Gyenge nyilazott biszimuláció az Lπ-kalkulusban:

Legyen $\mathcal{R}$ egy szimmetrikus bináris reláció az S halmaz felett, és legyen $P\,\mathcal{R}\,Q$. Az $\mathcal{R}$ relációt gyenge nyilazott biszimulációnak nevezzük,

- *ha $P\,{}_L{\downarrow}_{\overline{x}}$, akkor $Q\,{}_L{\Downarrow}_{\overline{x}}$,*
- *ha $P \xrightarrow{\tau} P'$, akkor van olyan Q', melyre $Q \Rightarrow Q'$ és $P'\,\mathcal{R}\,Q'$.*

4.2.4. Definíció. Gyenge nyilazott kölcsönös hasonlóság az Lπ-kalkulusban:

A gyenge nyilazott kölcsönös hasonlóság legyen a gyenge nyilazott biszimulációk uniója, és a gyenge nyilazott kölcsönös hasonlóság jele legyen $\dot{\approx}_{L\downarrow}$ *. Azt mondjuk, hogy P és Q folyamatok gyenge nyilazott kölcsönös hasonlóság relációban vannak, azaz* $P \ \dot{\approx}_{L\downarrow} \ Q$*, ha van olyan* $\mathcal{R}$ *gyenge nyilazott biszimuláció, melyre* $P \mathcal{R} Q$*.*

A kongruenciát is a szokásos módon definiáljuk:

4.2.5. Definíció. Gyenge nyilazott kongruencia az Lπ-kalkulusban:

A P és Q folyamat gyenge nyilazott kongruens, ha minden Lπ-kalkulusbeli C kontextusra $C[P] \ \dot{\approx}_{L\downarrow} \ C[Q]$ *. A kongruencia jele legyen* $\approx_{L\downarrow}$*, tehát ekkor* $P \approx_{L\downarrow} Q$ *.*

Két folyamat gyenge nyilazott kongruenciájának meghatározása a gyakorlatban nem egyszerű feladat. Most megmutatjuk, hogy ezt a feladatot hogyan lehet egyszerűbben megoldani. Áttranszformáljuk a vizsgálandó folyamatok kifejezéseit *továbbítóknak* nevezett folyamatkifejezések felhasználásával, és megmutatjuk, hogy ezeknek a kifejezéseknek a kongruenciája az eredeti kifejezések kongruenciáját biztosítja.

A 4.2.1. definíció nem teszi lehetővé az $x(y) \,.\, y(v) \,.\, P$ szerkezetű kifejezések használatát. Emiatt a tiltás miatt kellene tehát egy folyamat, amelyik áthelyez egy P folyamatot az egyik helyről egy másik helyre. A lokalizált pi-kalkulusban erre a célra definiálhatók olyan speciális folyamatok, amelyeket *továbbító* folyamatoknak nevezünk. A szakirodalomban ezt gyakran összekötőnek, kábelnek, linknek is nevezik, de vigyázat, ez nem tévesztendő össze az F_π kalkulusok összekötőtípusával, mint a neve is mutatja, az ott egy típus, a lokalizált pi-kalkulusban ez egy folyamat.

A továbbító folyamat pufferként viselkedik, úgy képzelhető el, hogy az egyik végpontján beolvassa a nevet, és ezt a nevet a másik végpontján adja ki.

Kétfajta továbbító folyamatot definiálunk. Egy triviális megoldás a következő.

4.2.6. Definíció. A statikus továbbító folyamat:

Ha x, y két név, akkor az

$$x \triangleright y \stackrel{\text{def}}{=\!=} !x(u) \,.\, \overline{y}u$$

folyamatot statikus továbbítónak nevezzük.

A folyamat nevében is benne van, hogy statikus, azaz nem kapcsolódik a rendszer dinamikájához. A jó megoldás nem ilyen egyszerű, nem lehet ennyire konkrét. Mint a bevezető példájában is láttuk, a megoldással kapott kifejezésnek lehet, hogy kapcsolódnia kell egy kommunikáció műveletéhez, és itt $\overline{y}u$ egy szabad output, amivel nem biztos, hogy a kívánt eredményt érjük el.

4.2.7. Definíció. A dinamikus továbbító folyamat:

Ha x, y két név, akkor az

$$x \rightarrow y \stackrel{\text{def}}{=\!=} !x(u) \,.\, (\boldsymbol{\nu} z)\,(\overline{y}z \mid z \rightarrow u)$$

folyamatot dinamikus továbbítónak nevezzük.

Az $x \rightarrow y$ folyamat nem felel meg a lokalizált pi-kalkulus szintaktikus szabályainak, mivel a jobb oldalon is szerepel egy $\rightarrow$ operátor, de bizonyítható, hogy a jobb oldalon álló kifejezéshez megadható egy olyan $L\pi$-beli kifejezés, amely korai kölcsönösen hasonló ehhez a kifejezéshez [26].

A dinamikus továbbító folyamat felhasználásával egy folyamatot át tudunk alakítani úgy, hogy a folyamat ne tartalmazzon szabad output prefixet. Ezt az átalakítást a $[\![\ ]\!]$ zárójelekkel jelöljük, és az $L\pi$-kalkulus folyamataira a 4.2. táblázatban adjuk meg.

A fenti szabályokkal átalakított kalkulust nevezzük $[\![L]\!]\pi$-kalkulusnak. Az átalakítás szabályaiból azonnal látható, hogy az átalakítással kapott kifejezések az $L\pi$-kalkulus kifejezései maradnak.

Ez a kalkulus megtartja a gyenge nyilazott kölcsönös hasonlóságot is [4], amit a következő tétellel mondunk ki.

4.2.8. Tétel. (A gyenge nyilazott kölcsönös hasonlóság)

Legyen P és Q az $L\pi$-kalkulus két folyamatkifejezése. A két kifejezésre teljesül a következő állítás:

$$P \approx_{L\downarrow} Q \text{ akkor és csak akkor, ha } [\![P]\!] \approx_{L\downarrow} [\![Q]\!] \,.$$

Az $[\![L]\!]\pi$-kalkulusban a gyenge kongruencia is megmarad, azaz az átalakítás kongruenciatartó.

$$
\begin{array}{rcl}
[\![\mathbf{0}]\!] & \stackrel{\text{def}}{=\!=} & \mathbf{0}\,, \\
[\![\overline{x}y]\!] & \stackrel{\text{def}}{=\!=} & (\boldsymbol{\nu} z)\,(\overline{x}z \mid z \to y)\,, \\
[\![x(y)\,.\,P]\!] & \stackrel{\text{def}}{=\!=} & x(y)\,.\,[\![P]\!]\,, \\
[\![P_1 \mid P_2]\!] & \stackrel{\text{def}}{=\!=} & [\![P_1]\!] \mid [\![P_2]\!]\,, \\
[\![(\boldsymbol{\nu} x)\,P]\!] & \stackrel{\text{def}}{=\!=} & (\boldsymbol{\nu} x)\,[\![P]\!]\,, \\
[\![!x(y)\,.\,P]\!] & \stackrel{\text{def}}{=\!=} & !x(y)\,.\,[\![P]\!]\,.
\end{array}
$$

4.2. táblázat. Az $L\pi \;\Rightarrow\; [\![L]\!]\pi$ átalakítás szabályai

4.2.9. Tétel. (Gyenge kongruencia az $[\![L]\!]\pi$-re)

Legyen P és Q az $L\pi$-kalkulus két folyamatkifejezése. A két kifejezésre teljesül a következő állítás:

$$P \;\approx_{L\downarrow}\; Q \;\; \textit{akkor és csak akkor, ha} \;\; [\![P]\!] \;\approx_{L\downarrow}\; [\![Q]\!]\,.$$

Érdekességképpen megmutatjuk, hogy a helyettesítés művelete hogyan írható le a továbbító folyamattal.

A pi-kalkulusban nyilvánvalóan helyes a helyettesítés

$$P[x := y] \equiv (\boldsymbol{\nu} u)\,(\overline{u}y \mid u(x)\,.\,P)$$

átírása, de az $L\pi$-kalkulusban is meg tudjuk adni a helyettesített folyamatot, nem azonossággal, hanem kongruenciával.

4.2.10. Tétel. (Helyettesítés az $L\pi$-kalkulusban)

Ha $x \neq y$ és $x \notin fn(P)$, akkor

$$P[x := y] \;\approx_{L\downarrow}\; (\boldsymbol{\nu} x)\,(P \mid x \rhd y)\,.$$

4.2.2. A késleltetett input

Az aszinkron kalkulusnál láttuk, hogy egy output művelet a folyamat végrehajtását befejezi. Hasonló gondolat vezetett el a *késleltetett inputos lokalizált pi-kalkulus* bevezetéséhez [26, 25]. A kalkulust $DL\pi$-vel jelöljük, a D betű az angol *delayed* szóból származik.

A kalkulus abban különbözik az $L\pi$ lokalizált kalkulustól, hogy van benne

egy *késleltetett input* prefix, amelynek a jele $a(x)P$, azaz a jelölésben nincs pont a prefix és a folyamat között. A prefix azt jelenti, hogy az input műveletet nem kötelező végrehajtani a P előtt, hiszen nem biztos, hogy az inputra azonnal jön egy olyan output, amivel az input kommunikálni tud. Elindulhat a P folyamat végrehajtása, és a P végrehajtása majd csak akkor áll le, amikor az x néven várt bejövő adatra szüksége lesz.

A *DLπ*-kalkulus prefixei a következők:

$$\pi ::= \overline{x}y \mid x(y) \mid \vartheta\overline{x}y \ ,$$

ahol

$$\vartheta ::= (\boldsymbol{\nu}y) \mid z(y) \mid (\boldsymbol{\nu}z)z(y) \quad z \in \mathcal{N} \ .$$

Tehát a $\vartheta\overline{x}y \,.\, P$ folyamatra a P lehetséges prefixei:

$$(\boldsymbol{\nu}y)\,\overline{x}y, \quad z(y)\overline{x}y, \quad (\boldsymbol{\nu}z)z(y)\overline{x}y \ ,$$

és felhívjuk a figyelmet arra, hogy az utolsó kettő összetett prefix-kifejezésben az első tag egy késleltetett input prefix.

$$\frac{}{x(y)P \xrightarrow{x(y)} P} \ [\text{DL-INPUT}]$$

$$\frac{P \xrightarrow{\alpha} P' \quad x \notin bn(\alpha), y \notin n(\alpha)}{x(y)P \xrightarrow{\alpha} x(y)P'} \ [\text{DL-DEL}]$$

$$\frac{P \xrightarrow{\overline{x}z} P'}{x(y)P \xrightarrow{\tau} (\boldsymbol{\nu}y)P'[y := z]} \ [\text{DL-COMM}]$$

$$\frac{P \xrightarrow{\overline{x}y} P' \quad x \neq y}{z(y)P \xrightarrow{z(y)\overline{x}y} P'} \ [\text{DL-IN-OUT-1}]$$

$$\frac{P \xrightarrow{z(y)\overline{x}y} P' \quad x \neq z}{(\boldsymbol{\nu}z)\,P \xrightarrow{(\boldsymbol{\nu}z)z(y)\overline{x}y} P'} \ [\text{DL-IN-OUT-2}]$$

4.3. táblázat. A *DLπ*-kalkulus műveleti szemantikájának néhány szabálya

4.2.11. Példa. *(Határozzuk meg a felsorolt prefixek szabad és kötött neveit)*

$$
\begin{array}{lclclcl}
fn(\overline{x}y) & = & \{x, y\}\,, & & bn(\overline{x}y) & = & \emptyset\,, \\
fn((\boldsymbol{\nu}y)\,\overline{x}y) & = & \{x\}\,, & & bn((\boldsymbol{\nu}y)\,\overline{x}y) & = & \{y\}\,, \\
fn(z(y)\overline{x}y) & = & \{z, x\}\,, & & bn(z(y)\overline{x}y) & = & \{y\}\,, \\
fn((\boldsymbol{\nu}z)z(y)\overline{x}y) & = & \{x\}\,, & & bn((\boldsymbol{\nu}z)z(y)\overline{x}y) & = & \{z, y\}\,. \qquad \square
\end{array}
$$

A *DLπ*-kalkulus legérdekesebb szabályait a 4.3. táblázatban adtuk meg.

A DL-DEL szabályból látható igazán az input késleltetésének jellege. A P-re egy α átmenetet hajtunk végre, amely teljesül, a P-ből P' lesz. Az $x(y)$ input műveletnek ezen idő alatt nem feltétlenül kell aktivizálódnia, a szabályból látjuk, hogy az input még a P'-nél is ott van, azaz még nem hajtódott végre.

A *DLπ*-kalkulus kifejezései könnyen átalakíthatók az *Lπ* kalkulus kifejezéseire, egy P kifejezés átalakítását a $\{\!|\, P \,|\!\}$ jelöli.

Az átalakítás szabályait a 4.4. táblázatban adjuk meg. A késleltetett inputot a statikus továbbító folyamat felhasználásával alakítottuk át.

$$
\begin{array}{rcl}
\{\!|\, \mathbf{0} \,|\!\} & \stackrel{\text{def}}{=\!=} & \mathbf{0}\,, \\
\{\!|\, \overline{x}y \,|\!\} & \stackrel{\text{def}}{=\!=} & \overline{x}y\,, \\
\{\!|\, x(y)\,.\,P \,|\!\} & \stackrel{\text{def}}{=\!=} & x(y)\,.\,\{\!|\, P \,|\!\}\,, \\
\{\!|\, x(y)P \,|\!\} & \stackrel{\text{def}}{=\!=} & (\boldsymbol{\nu}y)\,(x(z)\,.\,y \rhd z \mid \{\!|\, P \,|\!\})\,. \\
\{\!|\, P_1 \mid P_2 \,|\!\} & \stackrel{\text{def}}{=\!=} & \{\!|\, P_1 \,|\!\} \mid \{\!|\, P_2 \,|\!\}\,, \\
\{\!|\, (\boldsymbol{\nu}x)\, P \,|\!\} & \stackrel{\text{def}}{=\!=} & (\boldsymbol{\nu}x)\,\{\!|\, P \,|\!\}\,, \\
\{\!|\, !x(y)\,.\,P \,|\!\} & \stackrel{\text{def}}{=\!=} & !x(y)\,.\,\{\!|\, P \,|\!\}\,.
\end{array}
$$

4.4. táblázat. A *DLπ* $\Rightarrow$ *Lπ* átalakítás szabályai

A következőkben kimondunk néhány olyan fontos állítást, amelyek két folyamat kongruenciájának vizsgálatánál hasznosak lehetnek.

4.2.12. Tétel. (Gyenge kongruencia a $\{\!\!\{\ \}\!\!\}$ átalakításra)

Legyen P és Q a DLπ-kalkulus két folyamatkifejezése. A két kifejezésre teljesül a következő állítás:

$$P \approx_{L\downarrow} Q \text{ akkor és csak akkor, ha } \{\!\!\{ P \}\!\!\} \approx_{L\downarrow} \{\!\!\{ Q \}\!\!\} .$$

Ha a késleltetett inputot a dinamikus továbbító folyamat felhasználásával alakítjuk át és az átalakítást $\{\!\!\{\ \}\!\!\}_{din}$-nel jelöljük, akkor a 4.4. táblázat $\{\!\!\{ x(y)P \}\!\!\}$ sora így módosul:

$$\{\!\!\{ x(y)P \}\!\!\}_{din} \stackrel{\text{def}}{=\!=} (\nu y)\,(x(z)\,.\,y \to z \mid \{\!\!\{ P \}\!\!\}_{din}) \ ,$$

a táblázat többi sorába az átalakításokhoz csak a *din* indexet kell beírni. Ekkor érvényes a következő állítás is:

4.2.13. Tétel. (Gyenge kongruencia a $\{\!\!\{\ \}\!\!\}_{din}$ átalakításra)

Legyen P a DLπ-kalkulus egy folyamatkifejezése, ekkor

$$[\![P]\!] \approx_{L\downarrow} \{\!\!\{ [\![P]\!] \}\!\!\} .$$

Összefoglalva, a kalkulusok, átalakítások és a kapcsolataik tömör jelöléssel a következők:

$$L\pi \cup a(x)P \equiv DL\pi \ ,$$
$$[\![\,]\!] : L\pi \Rightarrow (L\pi \div \overline{x}y) \equiv [\![L]\!]\pi \ ,$$
$$[\![L]\!]\pi \approx L\pi \ ,$$
$$\{\!\!\{\ \}\!\!\} : DL\pi \Rightarrow L\pi \ ,$$

ahol a ÷ jel az output prefix módosítását, a ≈ jel a „hasonlót” jelenti.

4.2.3. A lokalizált pi-kalkulus és a lambda-kalkulus

A 2.5.8. pontban foglalkoztunk azzal a témával, hogy a *lambda-kalkulus* kifejezései hogyan alakíthatók át a pi-kalkulus kifejezéseire. A név szerinti kalkulus átalakításának szabályai már a 2.5.31. definícióban szerepeltek:

$$[\![x]\!] \stackrel{\text{def}}{=\!=} (p)\,.\,\overline{x}p \ ,$$
$$[\![\lambda x\,.\,E]\!] \stackrel{\text{def}}{=\!=} (p)\,.\,(\nu q)\,(\overline{p}q \mid !q(x,r)\,.\,[\![E]\!]\,\langle r\rangle) \ ,$$
$$[\![E\,F]\!] \stackrel{\text{def}}{=\!=} (p)\,.\,(\nu q)\,(\,[\![E]\!]\,\langle q\rangle \mid (\nu r)\,(\,\overline{q}\,\langle r,p\rangle\,.\,r(s)\,.\,[\![F]\!]\,\langle s\rangle\,)\,)\ .$$

Most itt, ellentétben az előző pontban írottakkal, a $[\![\,]\!]$ zárójelekkel a lambda-

kifejezések átalakítását jelöljük.

Csak az absztrakció átalakításával foglalkozzunk, a változóra és az applikációra az absztrakcióhoz hasonló gondolatmenettel és műveletekkel tudjuk az átalakítást meghatározni.

Azonnal látható, hogy az absztrakció kifejezése $L\pi$-beli kifejezés is lehet, nincs benne $x(u) \,.\, u(v)$ alakú sorozat. A $q(x, r) \equiv q(x) \,.\, q(r)$ pedig azért nem okozhat problémát, mert az utána következő $[\![E]\!]\,\langle r\rangle$ biztosan egy output prefixszel kezdődik.

Éppen ebből a $q(x) \,.\, q(r) \,.\, [\![E]\!]\,\langle r\rangle$ sorozatból látszik az is, hogy az x és r inputok feldolgozása csak az E-ben fog megtörténni, ezért az inputokat átírhatjuk késleltetett inputra is, az absztrakció átalakítása felírható

$$(p) \,.\, (\boldsymbol{\nu} q)\,(\overline{p}q \mid !q(x)q(r)[\![E]\!]\,\langle r\rangle)$$

alakban. Így a kapott kifejezés már egy $DL\pi$-beli kifejezés lett (az inputok között már nincs pont karakter).

A $q(x)q(r)[\![E]\!]\,\langle r\rangle$ kifejezést visszaalakíthatjuk egy $L\pi$-beli kifejezésre, felhasználva az

$$\{\!\{\, x(y)P \,\}\!\} \stackrel{\text{def}}{=\!=} (\boldsymbol{\nu} y)\,(x(z) \,.\, y \rhd z \mid \{\!\{\, P \,\}\!\})$$

szabályt:

$$\begin{aligned}
&\{\!\{\, q(x)q(r)[\![E]\!]\,\langle r\rangle \,\}\!\} = \\
&(\boldsymbol{\nu} x)\,(q(v) \,.\, x \rhd v \mid \{\!\{\, q(r)[\![E]\!]\,\langle r\rangle \,\}\!\}) = \\
&(\boldsymbol{\nu} x)\,(q(v) \,.\, x \rhd v \mid (\boldsymbol{\nu} r)\,(q(w) \,.\, r \rhd w \mid \{\!\{\, [\![E]\!]\,\langle r\rangle \,\}\!\})\,) \equiv \\
&(\boldsymbol{\nu} x, r)\,(q(v, w) \,.\, (x \rhd v \mid r \rhd w) \mid \{\!\{\, [\![E]\!]\,\langle r\rangle \,\}\!\})\,.
\end{aligned}$$

Így a lambda-absztrakcióra a következő szabályt kaptuk:

$$[\![\lambda x \,.\, E]\!] \stackrel{\text{def}}{=\!=} (p) \,.\, (\boldsymbol{\nu} q)\,(\overline{p}q \mid !(\boldsymbol{\nu} x, r)\,(q(v, w) \,.\, (x \rhd v \mid r \rhd w) \mid \{\!\{\, [\![E]\!]\,\langle r\rangle \,\}\!\})\,)\,.$$

4.3. A lineáris továbbító pi-kalkulus

A lineáris továbbító pi-kalkulusban nem vagyunk olyan „szigorúak”, mint a lokalizált kalkulusban, itt is megtiltjuk az $x(u) \,.\, u(v)$ kifejezések használatát, de ezt ellensúlyozva bevezetünk egy új input-konstrukciót, amit *lineáris továbbítónak* nevezünk [16].

A lineáris továbbítót a $\multimap$ jellel jelöljük. Feladata az lesz, hogy egy folyamatot egyik névről egy másik névre helyezzen át.

4.3.1. Definíció. A lineáris továbbító:

A lineáris továbbítót a következő kifejezéssel adjuk meg:

$$[\![x(u)\,.\,u(v)\,.\,Q]\!] \quad \equiv \quad x(u)\,.\,(\boldsymbol{\nu} u')\,(u \multimap u' \mid u'(v)\,.\,[\![Q]\!])\,,$$

ahol a $[\![\ ]\!]$ *zárójelek az átalakítást jelzik, és* $\multimap$ *a lineáris továbbítás jele.*

Az $x \multimap y$ lineáris továbbító esetén x-et a továbbító input nevének, y-t a továbbító output nevének nevezzük.

A 4.2. szakasz bevezetőjében írtuk le a pi-kalkulus egy lehetséges *elosztott implementációját*, ezt az implementációt használjuk ebben a szakaszban is. Az implementáció szerint a 4.3.1. definícióban az azonosság jel két oldalán álló mindkét kifejezés az x-hez tartozó helyen van, így az átalakítással kapott kifejezés is tud kommunikálni egy $\overline{x}w$ kifejezéssel, tehát a kommunikáció lehetősége nem romlott el.

Az új kalkulusban a lokalizált pi-kalkulus prefix- és folyamat-definícióit megtartjuk, de azt a feltételt, hogy „$x(y)\,.\,P$ esetén az y név P-ben nem lehet input alanya”, arra cseréljük le, hogy az ilyen kifejezésekre a fenti 4.3.1. definícióban megadott átalakítást kell alkalmaznunk. Az így kapott kalkulust nevezzük *lineáris továbbító pi-kalkulusnak*, a kalkulust $Lf\pi$-vel jelöljük.

4.3.1. Az $Lf\pi$-gép

Ebben a pontban megadunk egy olyan „gépet”, amelyik az $Lf\pi$-kalkulus műveleteit hajtja végre, megadjuk a kalkulus *elosztott implementációját*. Mint már korábban is említettük, hogy minden névhez (csatornához) egyértelműen hozzárendelünk egy *helyet*.

A helyeket egy-egy téglalappal ábrázoljuk, a téglalap bal felső sarkába írjuk a helyhez tartozó neveket, ezeket a hely címkéjének nevezzük. Hivatkozáskor a téglalapokat balról-jobbra számozzuk. A téglalapba írjuk bele azokat a kifejezéseket, amelyek ehhez a helyhez tartoznak.

A folyamatokat az első tagjuk szerint helyezzük el:

- az $\overline{x}w\,.\,P$ folyamat tetszőleges helyre kerülhet,
- az $x(y)\,.\,P$ input prefixes folyamat az x névhez tartozó helyre kerül,
- a $(\boldsymbol{\nu} x)\,P$ folyamatnak, mivel x a P saját neve, egy új, eddig még nem létező helyet készítünk, ha eddig n hely volt, akkor ez az új hely az l_{n+1} sorszámot kapja,
- az $x \multimap y \mid P$ kifejezést az x névhez tartozó helyre írjuk be.

4.3.2. Példa. *(A kezdeti állapot)*

Legyen a kifejezés

$u(x) \,.\, (\boldsymbol{\nu} v)\,(x \multimap v \mid v(z) \,.\, P) \;\mid\; \overline{u}y \;\mid \overline{y}w$,

és tegyük fel, hogy két hely van, az u és az y név tartozzon ezekhez a helyekhez. Helyezzük az $u(x) \,.\, (\boldsymbol{\nu} v)\,(x \multimap v \mid v(z) \,.\, P)$ kifejezést az u-val jelzett helyre, az outputos prefixű kifejezéseket az y címkéjű helyre. Ekkor a következő elrendezést kapjuk:

u	y
$u(x) \,.\, (\boldsymbol{\nu} v)\,(x \multimap v \mid v(z) \,.\, P)$	$\overline{u}y \mid \overline{y}w$

□

Nézzük meg az $Lf\pi$-gép működését.

Az első helyen a folyamat egy $u(x)$ input művelettel kezdődik. Ekkor a gép megnézi, hogy van-e ezen a helyen az u névre output. Ha talál ilyet, akkor végrehajtja a kommunikációt, ha nincs, akkor olyan helyet keres, ahol egy $\overline{u}z$ output található. Ha nincs ilyen hely, akkor a gép megáll, és egy *error* hibajelzést ad. Ha van ilyen output, akkor először az output prefixű folyamatot átteszi az u névhez tartozó helyre, (ez lesz a $\overset{move_{out}}{\Longrightarrow}$ átmenet), és utána végrehajtódik a megadott kommunikáció, amit a $\overset{comm}{\Longrightarrow}$ átmenettel jelölünk.

4.3.3. Példa. *(A kommunikáció)*

Folytassuk az előző példát:

u	y	
$u(x) \,.\, (\boldsymbol{\nu} v)\,(x \multimap v \mid v(z) \,.\, P)$	$\overline{u}y \mid \overline{y}w$	$\overset{move_{out}}{\Longrightarrow}$

u	y	
$\underline{\overline{u}y} \mid \underline{u(x) \,.\, (\boldsymbol{\nu} v)\,(x \multimap v \mid v(z) \,.\, P)}$	$\overline{y}w$	$\overset{comm}{\Longrightarrow}$

u	y
$(\boldsymbol{\nu} v)\,(y \multimap v \mid v(z) \,.\, P[x := y])$	$\overline{y}w$

□

Most a $\boldsymbol{\nu} v$ korlátozás következik. Mint már korábban a folyamatok elhelyezésénél írtuk, meghatározunk egy új, eddig még nem használt nevet, ez most legyen a v'. A v' nevet ahhoz a helyhez rendeljük, amelyiken a korlátozás volt, és ezt a $u@v'$ kapcsolattal jelöljük. Az új néven futó program

legyen egyelőre a $\mathbf{0}$. Ezenkívül a korlátozás alá esett folyamatban a v korlátozott nevet minden előfordulásánál az új v' névre írjuk át. Ezt az összetett műveletet a $\overset{\boldsymbol{\nu}}{\Longrightarrow}$ jellel jelöljük.

4.3.4. Példa. *(A korlátozás)*

Az előző példában leírt állapot folytatása a következő:

u	y
$(\boldsymbol{\nu} v)\,(y \multimap v \mid v(z)\,.\,P[x := y])$	$\overline{y}w$

$\overset{\boldsymbol{\nu}}{\Longrightarrow}$

u	y	$v'@u$
$y \multimap v' \mid v'(z)\,.\,P[x := y][v := v']$	$\overline{y}w$	$\mathbf{0}$

□

Egy új művelet következik, a $\multimap$, a lineáris továbbító. Itt ebben a példában az y neveket cseréli le v'-re azon a helyen, amelyikhez az y név tartozik. Mivel az első helyhez nem az y, hanem az u név tartozik, a $\overset{\multimap_{\neq}}{\Longrightarrow}$ utasítás hajtódik végre, amelyik a $\multimap$ műveletét átteszi a második helyre.

4.3.5. Példa. *(A lineáris továbbító)*

u	y	$v'@u$
$y \multimap v' \mid v'(z)\,.\,P[x := y][v := v']$	$\overline{y}w$	$\mathbf{0}$

$\overset{\multimap_{\neq}}{\Longrightarrow}$

u	y	$v'@u$
$v'(z)\,.\,P[x := y][v := v']$	$y \multimap v' \mid \overline{y}w$	$\mathbf{0}$

□

A $v'(z)$ input következik, de látjuk, hogy ez rossz helyen van, át kell tenni a folyamatot a v' címkéjű helyre. Ez nem jelent tényleges adatmozgatást, mert a v' címkére egy $v'@u$ kapcsolat van előírva, mint ahogyan azt a korlátozás lépésénél már részleteztük.

4.3.6. Példa. *(A $\overset{move_{in}}{\Longrightarrow}$ művelet)*

Tehát az u névről tegyük át a folyamatot a v' névre

u	y	$v'@u$
$v'(z)\,.\,P[x := y][v := v']$	$y \multimap v' \mid \overline{y}w$	$\mathbf{0}$

$\overset{move_{in}}{\Longrightarrow}$

u	y	$v'@u$
$\mathbf{0}$	$y \multimap v' \mid \overline{y}w$	$v'(z) \,.\, P[x := y][v := v']$

□

A második helyen a lineáris továbbító művelete a következő lépés, a végrehajtással nincs probléma, mert a hely címkéje megegyezik a továbbító input nevével. Az utasítás neve $\overset{\multimap_=}{\Longrightarrow}$.

4.3.7. Példa. *(A lineáris továbbító)*

u	y	$v'@u$	
$\mathbf{0}$	$y \multimap v' \mid \overline{y}w$	$v'(z) \,.\, P[x := y][v := v']$	$\overset{\multimap_=}{\Longrightarrow}$

u	y	$v'@u$
$\mathbf{0}$	$\overline{v'}w$	$v'(z) \,.\, P[x := y][v := v']$

□

Innen már jól ismert műveletek következnek, egy *move*, majd egy *comm* utasítás végrehajtása után nincs már újabb végrehajtható művelet, a folyamat megáll.

4.3.8. Példa. *(Befejező lépések)*

u	y	$v'@u$	
$\mathbf{0}$	$\overline{v'}w$	$v'(z) \,.\, P[x := y][v := v']$	$\overset{move_{out}}{\Longrightarrow}$
$\mathbf{0}$	$\mathbf{0}$	$\overline{v'}w \mid v'(z) \,.\, P[x := y][v := v']$	$\overset{comm}{\Longrightarrow}$
$\mathbf{0}$	$\mathbf{0}$	$P[x := y][v := v'][z := w]$	*stop*

□

Most megadjuk az $Lf\pi$-gép formális leírását.

4.3.9. Definíció. Az Lfπ-gép:

Legyen az Lfπ-gép M, és jelöljük P-vel az Lfπ-kalkulus kifejezéseit.

$M ::= \mathbf{0} \mid [P]_x \mid [P]_{(x)} \mid M, M$,

ahol $[\ ]_x$ *az x névhez tartozó kezelő, és* $[P]_x$ *azt jelenti, hogy a kezelő a P kifejezéssel foglalkozik. Az* (x) *index a korlátozott nevet jelöli.* M, M *két párhuzamosan működő kezelőt jelent.*

Megjegyezzük, hogy az előző példákban a „kezelő” helyett az ott kifejezőbb „hely” szót használtuk.

Ha $\mathcal{L}$ a nevek halmaza és $x, y \in \mathcal{L}$, akkor $x@y$ legyen egy ekvivalencia reláció, ami azt jelenti, hogy az x és y név ugyanahhoz a kezelőhöz van rendelve.

Az *Lfπ*-gép *jólformáltságához* három követelményt adunk meg.

1. *Helyesen rendezett*, azaz minden kifejezés a kifejezéshez hozzárendelt kezelőn fut.
2. *Egyszeresen definiált*, minden névhez csak egy kezelő tartozik.
3. *Teljes*, nincs arra lehetőség, hogy nem létező nevekkel műveleteket végezzen.

4.3.10. Definíció. Az Lfπ-gép működése:

A gép utasításainak neve van az első oszlopban, és az utasítások által végrehajtott műveletek leírása, azaz az átmeneti szabályok találhatók a második oszlopban:

$$
\begin{array}{llll}
comm & [\overline{x}y \mid x(z) \,.\, P \mid Q]_x & \Longrightarrow & [P[z := y] \mid Q]_x \,, \\
\multimap_{=} & [x \multimap y \mid \overline{x}z \mid P]_x & \Longrightarrow & [\overline{y}z \mid P]_x \,, \\
\multimap_{\neq} & [x \multimap y \mid P]_z, [Q]_x & \Longrightarrow & [P]_z, [x \multimap y \mid Q]_x \,, \quad ha \;\; x \neq z \,, \\
\boldsymbol{\nu} & [(\boldsymbol{\nu} x)\, P \mid Q]_u & \Longrightarrow & [P[x := x'] \mid Q]_u, [\mathbf{0}]_{(x')} \,, \quad x' \text{ új}, u@x' \,, \\
move_{out} & [\overline{x}y \mid P]_u, [Q]_x & \Longrightarrow & [P]_u, [\overline{x}y \mid Q]_x \,, \quad ha \;\; x \neq u \,, \\
move_{in} & [x(y) \,.\, P \mid Q]_u, [R]_x & \Longrightarrow & [Q]_u, [x(y) \,.\, P \mid R]_x \,, \quad ha \;\; x \neq u, u@x \,, \\
stop \,, & & & \\
error \,. & & &
\end{array}
$$

Az első utasítás a *comm*, ami a kommunikációt írja le, látható, hogy mind az input, mind az output folyamatnak ugyanazon a kezelőn kell lenni, és az

eredmény is erre a kezelőre kerül.

A $\multimap$ utasításnak két változata is van, az első a $\multimap_{=}$, ez akkor alkalmazható, amikor a lineáris továbbító input neve megegyezik a kezelőhöz rendelt névvel. Hatása az lesz, hogy az $\overline{x}z$ output művelet a x névről átkerül az y névre.

A második változat, a $\multimap_{\neq}$ abban az esetben hajtható végre, ha a lineáris továbbító input neve nem egyezik meg a kezelőhöz rendelt névvel. Ekkor a lineáris továbbító ezen a kezelőn hatástalan, de művelete a vele párhuzamosan futó, az input nevével azonos nevű kezelőre tevődik át.

A $\boldsymbol{\nu}$ jellel jelölt utasítás a korlátozás utasítása, az x név korlátozva van a P folyamatra. A korlátozott x nevet P-ben lecseréli egy új névre, és ezzel megszünteti a P-re előírt korlátozást, és az új névvel elindít egy új kezelőt, egyelőre üresen, azaz folyamat nélkül.

A $move_{out}$ utasításnál látjuk, hogy ha az $\overline{x}y$ output prefix nem jó helyen, az u névhez rendelt kezelőn van, az utasítás ezt az output műveletet áthelyezi az x névhez tartozó kezelőhöz.

A $move_{in}$ utasítás hasonló műveletet hajt végre az $x(y).P$ folyamattal, a folyamat az u kezelőtől átkerül az x-re, de csak akkor,ha az $u@x$ feltétel teljesül. Mivel a két kezelő ugyanazon a helyen van, tényleges adatmozgatás ennél a műveletnél nem lesz.

Ha $ch(M)$ jelenti az M gép kezelőihez rendelt neveket, akkor természetesen a 4.3.10. definícióban megadott műveletek alkalmazásához érvényes a következő szabály is:

$$\frac{M \Longrightarrow M', \quad ch(M') \cap ch(N) = \emptyset}{M, N \Longrightarrow M', N}$$

A 4.3.10. definícióban a gép utasításait egy vagy legfeljebb két kezelőre adtuk meg, ez a szabály ad lehetőséget arra, hogy párhuzamosan tetszőlegesen sok kezelőt futtassunk.

Az $Lf\pi$-gépre is megadhatjuk a *szerkezeti kongruencia* szabályait:

$M, \mathbf{0} \equiv M$,

$M_1, M_2 \equiv M_2, M_1$,

$M_1, (M_2, M_3) \equiv (M_1, M_2), M_3$,

$P \equiv Q \quad \Longrightarrow \quad [P]_x \equiv [Q]_x$ *és* $[P]_{(x)} \equiv [Q]_{(x)}$.

4.3.11. Példa. *(A 4.3.2. példa kifejezésének végrehajtása)*

A $Lf\pi$-gép utasításainak ismeretében adjuk meg a 4.3.2. példában szereplő kifejezés futtatásának egyes lépéseit:

$$[u(x)\,.\,(\boldsymbol{\nu}v)\,(x \multimap v \mid v(z)\,.\,P)]_u, \quad [\overline{u}y \mid \overline{y}w]_y \overset{move_{out}}{\Longrightarrow}$$

$$[\overline{u}y \mid u(x)\,.\,(\boldsymbol{\nu}v)\,(x \multimap v \mid v(z)\,.\,P)]_u, \quad [\overline{y}w]_y \overset{comm}{\Longrightarrow}$$

$$[(\boldsymbol{\nu}v)\,(y \multimap v \mid v(z)\,.\,P[x := y])]_u, \quad [\overline{y}w]_y \overset{\boldsymbol{\nu}}{\Longrightarrow}$$

$$[y \multimap v' \mid v'(z)\,.\,P[x := y][v := v']\,]_u, \quad [\overline{y}w]_y, \quad [\mathbf{0}]_{(v')} \overset{\multimap_{\neq}}{\Longrightarrow} \qquad u@v'$$

$$[v'(z)\,.\,P[x := y][v := v']\,]_u, \quad [y \multimap v' \mid \overline{y}w]_y, \quad [\mathbf{0}]_{(v')} \overset{move_{in}}{\Longrightarrow} \qquad u@v'$$

$$[\mathbf{0}]_u, \quad [y \multimap v' \mid \overline{y}w]_y, \quad [v'(z)\,.\,P[x := y][v := v']\,]_{(v')} \overset{\multimap_{=}}{\Longrightarrow}$$

$$[\mathbf{0}]_u, \quad [\overline{v'}w]_y, \quad [v'(z)\,.\,P[x := y][v := v']\,]_{(v')} \overset{move_{out}}{\Longrightarrow}$$

$$[\mathbf{0}]_u, \quad [\mathbf{0}]_y, \quad [\overline{v'}w \mid v'(z)\,.\,P[x := y][v := v']\,]_{(v')} \overset{comm}{\Longrightarrow}$$

$$[\mathbf{0}]_u, \quad [\mathbf{0}]_y, \quad [P[x := y][v := v'][z := w]\,]_{(v')} \quad \mathit{stop}\,. \qquad \square$$

4.4. A fúzió-kalkulus

A fúzió-kalkulus sok tulajdonságot öröklött a poliadikus pi-kalkulusból, leglényegesebb változás az, hogy

- az input prefix nem köti az input neveit, $x(y, z)\,.\,P$ helyett $x\,yz\,.\,P$ jelzi az inputot,
- a kommunikáció határozza meg az input és output nevek összekapcsolását, társítását, a nevek közötti *ekvivalencia relációra* utal a *fúzió* elnevezés,
- a fúzió meghatározása után csak a külön deklarált kötött nevekre történik meg a szokásos helyettesítés művelet, ráadásul a kötés hatókörében szereplő összes folyamatra, azokra is, amelyek a kommunikációban nem is vettek részt.

Például

$$\overline{x}\langle v, w\rangle . P \mid x\, yz . Q \mid R \mid S \xrightarrow{\{y=v,\ z=w\}} P \mid Q \mid R \mid S \ ,$$

ahol $\{y = v,\ z = w\}$ jelzi a fúziót, és

$$\underline{(\boldsymbol{\nu} y)\,(\overline{x}\langle v, w\rangle . P \mid x\, yz . Q)} \mid R \mid S \xrightarrow{\{z=w\}} P \mid Q[y := v] \mid R \mid S \ ,$$

$$\underline{(\boldsymbol{\nu} z)\,(\,(\boldsymbol{\nu} y)\,(\overline{x}\langle v, w\rangle . P \mid x\, yz . Q) \mid R)} \mid S \xrightarrow{\mathbf{1}}$$
$$P \mid Q[y, z = v, w] \mid R[z := w] \mid S \ ,$$

ahol **1** az identitás reláció.

A fúzió-kalkulust az $f\pi$ jellel jelöljük. A $f\pi$-kalkulus definícióját Joachim Parrow és Björn Victor adta meg 1998-ban [37]. Egy ehhez hasonló kalkulus a χ-*kalkulus*, amelynek kidolgozása Yuxi Fu nevéhez fűződik [14, 13], a fúzió-kalkulus a χ-kalkulus egyszerűsítése és továbbfejlesztése.

Az fúzió-kalkulus nevében a fúzió szót megtartva a továbbiakban a „nevek fúziója” helyett inkább a magyarosabb „nevek társítása” kifejezést fogjuk használni.

4.4.1. Szintaktika és műveleti szemantika

Legyen $\mathcal{N}$ most is a nevek halmaza. Az $\mathcal{N}$-en értelmezett társítás függvény legyen φ, amelyik egy $\{\tilde{x} = \tilde{y}\}$ ekvivalencia relációt határoz meg az $\mathcal{N}$-en, ha $\tilde{x} = x_1, x_2, \ldots, x_n$ és $\tilde{y} = y_1, y_2, \ldots, y_n$, akkor

$$\{\tilde{x} = \tilde{y}\} = \{x_1 = y_1, x_2 = y_2, \ldots, x_n = y_n\} \ .$$

4.4.1. Definíció. Az $f\pi$-kalkulus prefixei:

Az $f\pi$-kalkulusban a következő prefixeket használjuk:

$$\pi ::= \overline{x}\tilde{y} \ \mid \ x\tilde{y} \ \mid \ \varphi \ .$$

Az input és output prefixek *szabad* akciók, ahol x az akció *alanya* és $\tilde{y}$ az akció *tárgya*. Mivel, mint ahogyan a bevezetőben említettük, az input és természetesen az output prefix sem köti a prefix tárgyának nevét, nem kell különbséget tennünk az $\overline{x}$ és x között. Ezeket együttesen a-val fogjuk jelölni, az akció jele $a\tilde{y}$. Ezeknek az akcióknak a neve *kommunikációs akció*, a φ-vel jelölt akciókat *fúzióakcióknak* nevezzük. Ha a fúzióakció eredményét nem részletezzük, akkor az átmeneti szabályokban a fúzióakció eredményének jele a γ lesz.

4.4.2. Definíció. Az $f\pi$-kalkulus folyamatai:

Az $f\pi$-kalkulusban a folyamatokat a következő kifejezésekkel adjuk meg:

$$P ::= \mathbf{0} \mid \pi . P \mid P + P \mid P \mid P \mid (\nu x)\, P .$$

A kifejezések már jól ismertek, csak a korlátozáshoz fűzünk egy megjegyzést. Az $(\nu x)\, P$ korlátozás köti a P-ben levő szabad x neveket, és ezek a nevek nem lesznek láthatóak a társítás számára, azaz a kommunikáció ezeket a neveket nem társítja. A korlátozás tehát a társítás műveletnek a hatáskörét korlátozza.

Az fúzió-kalkulus szabad és kötött neveinek definíciója, *szerkezeti kongruenciájának* szabályai lényegében a 2.3. szakaszban leírtakkal egyeznek meg, de mivel a szintaktika kissé változott, a 4.5. táblázatban megadjuk az érvényes szabályokat.

$$\begin{array}{rcl}
P & \equiv & Q, \quad ha\ P \leftrightarrow_\alpha Q \\
P \mid Q & \equiv & Q \mid P \\
P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R \\
P \mid \mathbf{0} & \equiv & P \\
P + Q & \equiv & Q + P \\
P + (Q + R) & \equiv & (P + Q) + R \\
P + \mathbf{0} & \equiv & P \\
(\nu x)(\nu y)\, P & \equiv & (\nu y)(\nu x)\, P \\
(\nu x)\, P & \equiv & P, \quad ha\ x \notin fn(P) \\
(\nu x)\, P \mid Q & \equiv & (\nu x)\,(P \mid Q), \quad ha\ x \notin fn(Q) \\
(\nu x)\, P + (\nu x)\, Q & \equiv & (\nu x)\,(P + Q) \\
(\nu x)\, \mathbf{0} & \equiv & \mathbf{0}
\end{array}$$

4.5. táblázat. Az $f\pi$-kalkulus szerkezeti kongruencia szabályai

Az $f\pi$-kalkulus műveleti szemantikájának szabályait a 4.6. táblázatban adjuk meg.

A Struct szabály azt mondja ki, hogy a szerkezeti kongruens folyamatok azonosaknak tekinthetők.

Már említettük, hogy az input prefix nem köti az input tárgyának nevét. A kommunikáció végrehajtása így nem helyettesítést jelent, hanem azt, hogy

$$\frac{P' \equiv P, \quad P \xrightarrow{\alpha} Q, \quad Q \equiv Q'}{P' \xrightarrow{\alpha} Q'} \quad [\textsc{Struct}]$$

$$\frac{}{\alpha\,.\,P \xrightarrow{\alpha} P} \quad [\textsc{Pref}]$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \quad [\textsc{Sum}] \qquad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad [\textsc{Par}]$$

$$\frac{x\tilde{y}\,.\,P \xrightarrow{x\tilde{y}} P, \quad \overline{x}\tilde{z}\,.\,Q \xrightarrow{\overline{x}\tilde{z}} Q, \quad |\tilde{y}| = |\tilde{z}|}{x\tilde{y}\,.\,P \mid \overline{x}\tilde{z}\,.\,Q \xrightarrow{\{\tilde{y}=\tilde{z}\}} P \mid Q} \quad [\textsc{Com}]$$

$$\frac{P \xrightarrow{\varphi} P', \quad z\varphi x, \quad z \neq x}{(\nu z)\,P \xrightarrow{\varphi\backslash z} P'[z := x]} \quad [\textsc{Scope}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad x \notin n(\alpha)}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'} \quad [\textsc{Res}]$$

$$\frac{P \xrightarrow{(\tilde{y})a\tilde{x}} P', \quad z \in \tilde{x} - \tilde{y}, \quad a \notin \{z, \overline{z}\}}{(\nu z)\,P \xrightarrow{(z\tilde{y})a\tilde{x}} P'} \quad [\textsc{Open}]$$

4.6. táblázat. Az *fπ*-kalkulus műveleti szemantikájának szabályai

a művelet az input és output prefix tárgyainak nevét társítja, azaz egyenlővé teszi. A Com szabály ezt írja le, és az is látható, hogy a művelet végrehajtásához az input és output paraméterszámának meg kell egyeznie. Egyedül ebben a szabályban szerepel a szabály következményében átmenetként a φ társítás, azaz csak ez a szabály hoz létre egyenlőség relációkat a kalkulus nevei között.

A nevek társításának eredménye *globális*, olyan értelemben, hogy a környezetben futó minden folyamatra érvényes. A bevezetőben említett példát tekintve,

$$\overline{x}\,\langle v, w\rangle\,.\,P \mid x\,yz\,.\,Q \mid R \mid S \xrightarrow{\{y=v,\ z=w\}} P \mid Q \mid R \mid S\ ,$$

az $\{y = v,\ z = w\}$ társítás a környezetben lévő R és S folyamatokat is érinti.

Három különbség is van az fúzió-kalkulus társítása és a pi-kalkulus kommunikációja között. Az *fπ*-kalkulusban a társítás

- nem lokális a társítást eredményező input és output prefixes folyamatokra, mint ahogyan az előzőekben láttuk,
- a $\boldsymbol{\nu} x$ prefixszel meghatározott nevekkel van szabályozva, és
- „szimmetrikus” az input-output műveletekre.

A Scope szabály vonatkozik a társításban szereplő nevek korlátozására. A szabály feltételében az szerepel, hogy P-re meghatároztuk a φ társítást, de a következményben a P-re előírtunk egy $\boldsymbol{\nu} z$ korlátozást. Így a z név a P saját neve lett, és a következményben a z nevet ki kell venni a φ-ből. Ezt jelzi a $\varphi \backslash z$ művelet, amelynek a pontos definícióját a következőképpen lehet megadni:

$$\varphi \backslash z \quad = \quad \varphi \cap (\mathcal{N} - \{z\})^2 \cup \{(z, z)\} \ ,$$

azaz φ-ből ki kell venni az összes olyan társítást, amelyben a z szerepel, és hozzá kell venni a z identitás társítását. A $\{(z, z)\}$ társítás triviális, és nem feltétlenül kell kiírni.

A Scope szabályból az is látszik, hogy nem csak a z-re vonatkozó társítások kerülnek ki a φ-ből, de a P'-re a φ-ben előírt $z = x$ társítás helyett végrehajtódik egy $[z := x]$ helyettesítés.

4.4.3. Példa. *(A $\varphi \backslash z$ művelet)*

$$\begin{aligned} \{x = y\} \backslash y & & = & \ \mathbf{1} \ , \\ \{x = y, y = u\} \backslash y & \equiv \{x = y, y = u, x = u\} \backslash y & = & \ \{x = u\} \ . \end{aligned}$$

□

A „szimmetrikus” tulajdonság azt jelenti, hogy az input és output jelzés a neveken felcserélhető, ettől a kommunikáció eredménye nem változik. Ezt egy példán mutatjuk meg.

4.4.4. Példa. *(Az input és output szimmetriája)*

$$\overline{x}\, vw \,.\, P \mid x\, pq \,.\, Q \xrightarrow{\{v=p, w=q\}} P \mid Q \ ,$$

$$x\, vw \,.\, P \mid \overline{x}\, pq \,.\, Q \xrightarrow{\{v=p, w=q\}} P \mid Q \ .$$

□

4.4.2. Biszimuláció

A korábbi kalkulusokban megkülönböztettünk késői, korai, nyitott biszimuláció relációkat és kölcsönös hasonlóságokat, kongruenciákat. Ezek a fúziókalkulusból hiányoznak, ebben a kalkulusban csak egy biszimuláció reláció definiálható.

Ennek oka az, hogy itt a φ társítás sokkal hatékonyabb kontextust eredményez, mint ami a pi-kalkulusban elérhető volt. A fúzióakciókból származó „környezet”, a φ társítás ugyanis minden folyamatra hat, míg a pi-kalkulusban a kommunikáció eredménye csak az input-prefixes folyamatra hatott.

A biszimuláció definíciójában, mint például a nyitott biszimulációnál, most is helyettesítéseket használunk.

4.4.5. Definíció. Biszimuláció:

Legyen $\mathcal{R}$ egy szimmetrikus bináris reláció az S halmaz felett, és legyen $P\,\mathcal{R}\,Q$. Az $\mathcal{R}$ biszimuláció,

- *ha $P \xrightarrow{\gamma} P'$, $bn(\gamma) \notin fn(Q)$, akkor $Q \xrightarrow{\gamma} Q'$ esetén $P'\sigma_\gamma\,\mathcal{R}\,Q'\sigma_\gamma$, ahol σ_γ egy olyan idempotens helyettesítés, amelyik a γ mindegyik ekvivalenciaosztályának minden tagját az osztály egy tagjára képezi le.*

4.4.6. Példa. *(A σ_γ helyettesítés)*

$$\begin{aligned}
\sigma_{\{x=y\}} &= [y := x]\,, \\
\sigma_{\{x=y\}} &= [x := y]\,, \\
\sigma_{\mathbf{1}} &= \textit{az identitás helyettesítés}\,. \qquad \square
\end{aligned}$$

A biszimulációt *fúzió biszimulációnak* nevezzük. A 4.4.5. definícióban szereplő átmenetek a 4.3. ábrán láthatók.

$$\begin{tikzcd}
P \arrow[r, dotted, no head] \arrow[d, "\gamma"] & \mathcal{R} \arrow[r, dotted, no head] & Q \arrow[d, "\gamma"] \\
P' & & Q' \\
P'\sigma_\gamma \arrow[r, dotted, no head] & \mathcal{R} \arrow[r, dotted, no head] & Q'\sigma_\gamma
\end{tikzcd}$$

4.3. ábra. A fúzió biszimuláció

A fúzió biszimilációval tudunk definiálni egy kölcsönös hasonlóság fogalmat.

4.4.7. Definíció. Fúzió kölcsönös hasonlóság:

A fúzió kölcsönös hasonlóság legyen a fúzió biszimulációk uniója. Azt mondjuk, hogy P és Q folyamatok fúzió kölcsönösen hasonlóak, azaz $P \dot{\sim}_f Q$, ha van olyan $\mathcal{R}$ fúzió biszimuláció, melyre $P \mathcal{R} Q$.

A fúzió kölcsönös hasonlóság is *alapreláció*, azaz a $P \mathcal{R} Q$-ból nem következik az, hogy $P\sigma \mathcal{R} Q\sigma$, ezért a kongruencia fogalmát most is külön definiálnunk kell.

4.4.8. Definíció. Fúzió kongruencia:

A P és Q folyamat fúzió kongruens, ha minden σ helyettesítésre $P\sigma \dot{\sim}_f Q\sigma$. A kongruencia jele $\sim_f$, tehát ekkor $P \sim_f Q$.

Megjegyezzük, hogy a szakirodalomban a fúzió kongruenciát *hiperbiszimulációnak* is nevezik.

4.4.9. Példa. *(Fúzió kongruencia)*

$$\begin{array}{lll} \{x = y\} . x \mid \overline{y} & \nsim_f & \{x = y\} . (x . \overline{y} + \overline{y} . x) \,, \\ \{x = y\} . (x \mid \overline{x}) & \sim_f & \{x = y\} . (x . \overline{y} + \overline{y} . x + \mathbf{1}) \,, \\ \{x = y\} . x . P & \sim_f & \{x = y\} . y . P \,. \end{array}$$

□

4.4.3. A fúzió-kalkulus és a lambda-kalkulus

Ebben a pontban a fúzió-kalkulus és a lambda-kalkulus kapcsolatát mutatjuk be, megadjuk, hogy egy lambda-kifejezés hogyan alakítható át fúzió-kalkulus kifejezésére.

Egy lambda-kifejezés fúzió-kalkulusbeli folyamatkifejezésére is a $[\![\ ]\!]$ zárójeleket használjuk, az E lambda-kifejezésnek megfeleltetett fúzió-kalkulus kifejezés legyen $[\![E]\!]$.

4.4.10. Definíció. A lambda-kifejezések folyamatkifejezései:

$$\begin{array}{lcl} [\![x]\!]u & \stackrel{\text{def}}{=\!=} & \overline{x}u \\ [\![\lambda x . E]\!]u & \stackrel{\text{def}}{=\!=} & (\boldsymbol{\nu} xv)\,(u\,xv \mid [\![E]\!]v) \\ [\![E\,F]\!]u & \stackrel{\text{def}}{=\!=} & (\boldsymbol{\nu} v)\,(\,[\![E]\!]v \mid (\boldsymbol{\nu} x)\,(\,\overline{v}\,xu\,.\,(\boldsymbol{\nu} w)\,(x\,w\,.\,[\![F]\!]w\,)\,)\,) \end{array}$$

4.4.11. Példa. *(Az y, $\lambda x . x$ és $\lambda x . y$ lambda-kifejezések)*

Legyen a kommunikációs csatorna az u, ekkor a fenti definíció első és má-

sodik sorában leírtak szerint

$$[\![y]\!]u \equiv \overline{y}u \ ,$$
$$[\![\lambda x\,.\,x]\!]u \equiv (\boldsymbol{\nu} xv)\,(u\ xv \mid \overline{x}v) \ ,$$
$$[\![\lambda x\,.\,y]\!]u \equiv (\boldsymbol{\nu} xv)\,(u\ xv \mid \overline{y}v) \ . \qquad \square$$

4.4.12. Példa. *(Határozzuk meg a $(\lambda x\,.\,x)y$ folyamatkifejezését)*

A lambda-kalkulusban $(\lambda x\,.\,x)y \longrightarrow_\beta y$, tehát a fúzió-kalkulusban is az y folyamatkifejezését várjuk eredményként.

$$[\![(\lambda x\,.\,x)y]\!]u \equiv$$
$$(\boldsymbol{\nu} v)\,(\ [\![\lambda x\,.\,x]\!]v \mid (\boldsymbol{\nu} x)\,(\ \overline{v}\ xu\,.\,(\boldsymbol{\nu} w)\,(x\ w\,.\,[\![y]\!]w\ )\ )\ ) \equiv$$
$$(\boldsymbol{\nu} v)\,(\ (\boldsymbol{\nu} xp)\,(v\ xp \mid \overline{x}p) \mid (\boldsymbol{\nu} z)\,(\ \overline{v}\ zu\,.\,(\boldsymbol{\nu} w)\,(z\ w\,.\,\overline{y}w\ )\ )\ ) \equiv$$
$$(\boldsymbol{\nu} vxpzw)\,(\ \underline{v\ xp} \mid \overline{x}p \mid \underline{\overline{v}\ zu}\ .\,z\ w\,.\,\overline{y}w\ ) \xrightarrow{\{x=z,\,p=u\}}$$
$$(\boldsymbol{\nu} vzuw)\,(\ \underline{\overline{z}u} \mid \underline{z\ w\,.\,\overline{y}w}\ ) \xrightarrow{\{w=u\}}$$
$$(\boldsymbol{\nu} vzu)\,\overline{y}u \equiv$$
$$\overline{y}u \equiv$$
$$[\![y]\!]u \ . \qquad \square$$

4.4.13. Példa. *(Határozzuk meg a $(\lambda x\,.\,y)q$ folyamatkifejezését)*

$$[\![(\lambda x\,.\,y)q]\!]u \equiv$$
$$(\boldsymbol{\nu} v)\,(\ [\![\lambda x\,.\,y]\!]v \mid (\boldsymbol{\nu} x)\,(\ \overline{v}\ xu\,.\,(\boldsymbol{\nu} w)\,(x\ w\,.\,[\![q]\!]w\ )\ )\ ) \equiv$$
$$(\boldsymbol{\nu} v)\,(\ (\boldsymbol{\nu} xp)\,(v\ xp \mid \overline{y}p) \mid (\boldsymbol{\nu} z)\,(\ \overline{v}\ zu\,.\,(\boldsymbol{\nu} w)\,(z\ w\,.\,\overline{q}w\ )\ )\ ) \equiv$$
$$(\boldsymbol{\nu} vxpzw)\,(\ \underline{v\ xp} \mid \overline{y}p \mid \underline{\overline{v}\ zu}\ .\,z\ w\,.\,\overline{q}w\ ) \xrightarrow{\{x=z,\,p=u\}}$$
$$(\boldsymbol{\nu} vzuw)\,(\ \overline{y}u \mid z\ w\,.\,\overline{q}w\ )\ .$$

A kompozíció második tagjában a z név korlátozva van, a $z\ w$ input nem hajtódhat végre, így $z\ w\,.\,\overline{q}w \equiv \mathbf{0}$. Tehát

$$(\boldsymbol{\nu} vzuw)\,(\ \overline{y}u \mid z\ w\,.\,\overline{q}w) \equiv$$
$$\overline{y}u \equiv$$
$$[\![y]\!]u\ .$$

Az eredmény $[\![y]\!]u$ lett, ami megfelel a lambda-kalkulusban a $(\lambda x\,.\,y)q \longrightarrow_\beta y$ β-redukció eredményének. $\square$

4.5. A belső mobilitás kalkulusa

Az eddig elemzett kalkulusokra két megjegyzést tehetünk,

- feltűnő az input és output közötti aszimmetria, az input tetszőleges nevet fogadhat, az output csak azt küldheti ki, amelyik az operandusában szerepel, de lényegesebb az, hogy az output által küldött név szabad, az input név pedig kötött a műveleteket követő folyamatokban,
- a kötött output fogalmát kissé mesterkélten tudtuk csak bevezetni: $\overline{x}(y) \equiv (\boldsymbol{\nu} y)\overline{x}y$, és a már többször szerepelt Open szabály szerint

  $$\frac{\overline{x}y\,.\,P \xrightarrow{\overline{x}y} P, \quad x \neq y}{\overline{x}(y)\,.\,P \xrightarrow{\overline{x}(y)} P} \quad [\text{Open}]$$

  a kötött output akkor működik, ha P az x néven végre tudja hajtani a szabad y kiküldését.

  Megjegyezzük, hogy ehhez hasonló módon a τ átmenetet is le tudjuk írni:

  $\tau \equiv (\boldsymbol{\nu} x)\,(x\,.\,P \mid \overline{x})$, ha $x \notin fn(P)$.

A kalkulusokban a szabad outputtal végzett

$$\overline{x}y\,.\,P \mid x(z)\,.\,Q \xrightarrow{\tau} P \mid Q[z := y]$$

átmenet *külső mobilitásként* értelmezhető, a Q folyamat az x külső, azaz szabad néven kap egy külső, azaz szabad y értéket, amit a z formális paraméterbe helyettesít. Ugyanakkor a kötött outputos

$$\overline{x}(y)\,.\,P \mid x(y)\,.\,Q \xrightarrow{\tau} (\boldsymbol{\nu} y)\,(P \mid Q)$$

átmenetben mind a P, mind a Q a kötött, tehát belső y-t használja, ezért ezt a műveletet *belső mobilitásnak* nevezzük.

Az input és output szimmetriája a kötött output bevezetésével még nem teljes, mert nincs szabad input művelet. A most vizsgálandó kalkulusban azonban nem a szabad inputot vezetjük be, hanem a szabad outputot töröljük. A szimmetria elérése tulajdonképpen nem feltétlenül szükséges, de megnézzük, hogy egy ilyen „szép” szimmetrikus rendszer milyen előnyöket tud adni a többi kalkulushoz képest.

Mivel ebben az új kalkulusban csak belső (privát) mobilitás lehetséges, ezért a kalkulust *privát kalkulusnak* nevezzük és a $P\pi$ jellel jelöljük. A régebbi szakirodalomban ezt a kalkulust πI-kalkulusnak is nevezték.

4.5.1. Szintaktika és műveleti szemantika

4.5.1. Definíció. A privát kalkulus prefixei:

Az $P\pi$-kalkulusban a következő prefixeket használjuk:

$$\pi ::= \overline{x}(y) \mid x(y) \mid \tau \ .$$

A $P\pi$-kalkulusban tehát csak a kötött output prefix, a kötött input prefix és a τ prefix megengedett.

4.5.2. Definíció. A privát kalkulus folyamatai:

A $P\pi$-kalkulusban a folyamatokat a következő kifejezésekkel adjuk meg:

$$P ::= \mathbf{0} \mid \pi \,.\, P \mid P + P \mid P \,|\, P \mid (\boldsymbol{\nu} x)\, P \,.$$

Az input és output szimmetrikus, az $x(y)$ input azt jelenti, hogy olvasunk egy „friss” nevet az x-en, az $\overline{x}(y)$ pedig azt, hogy kiküldünk egy privát nevet az x-en, tehát a műveletek egymás duálisai.

Jelöljük a π prefix *komplemensét* $\overline{\pi}$-vel, ahol

$$\overline{\overline{x}(y)} = x(y) \ , \quad \overline{x(y)} = \overline{x}y \ , \ \text{és} \ \ \overline{\tau} = \tau \ .$$

A P folyamat $\overline{P}$ *duálisát* úgy kapjuk meg P-ből, hogy a P minden π prefixét a komplemensére, $\overline{\pi}$-re transzformáljuk.

A privát kalkulus műveleti szemantikáját a pi-kalkulus korai szemantikájára építjük. A műveleti szemantika szabályait a 4.7. táblázatban adjuk meg.

Nézzük meg részletesen a Com szabályt. A feltételben az egyik átmenetben π, a másikban a π duálisa szerepel, és ez egy erős megkötés, például az $\overline{x}(y) \mid u(y)$ kifejezésben nem lehet kommunikáció. A $P\pi$-kalkulusban az *α-konverzió* megengedett, és mivel a kifejezésben az u szabad név, a kifejezés α-konverzióval átalakítható $\overline{x}(y) \mid x(y)$ alakra. Az $\overline{x}(y) \mid x(z)$ kifejezésben a $[z := y]$ helyettesítést végző α-konverzió azonban nem végezhető el, mivel a prefixek tárgyai kötöttek. A szabályból az is látható, hogy a prefixek tárgyainak közös x neve kötött lesz a kommunikáció eredményében.

A duális képzése szintaktikai átalakítás, de a következő tétel azt állítja, hogy ez a művelet szemantikus is.

4.5.3. Tétel. (A duális művelet)

Ha $P \xrightarrow{\pi} P'$, akkor $\overline{P} \xrightarrow{\overline{\pi}} \overline{P'}$. Mivel $\overline{\overline{P}} = P$ és $\overline{\overline{\pi}} = \pi$, a tétel állítása fordított irányban is fennáll.

$$\frac{P' \equiv P, \quad P \xrightarrow{\pi} Q, \quad Q \equiv Q'}{P' \xrightarrow{\pi} Q'} \quad [\textsc{Struct}]$$

$$\frac{}{\pi \,.\, P \xrightarrow{\pi} P} \quad [\textsc{Pre}]$$

$$\frac{P \xrightarrow{\pi} P'}{P + Q \xrightarrow{\pi} P'} \quad [\textsc{Sum}]$$

$$\frac{P \xrightarrow{\pi} P', \quad bn(\pi) \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\pi} P' \mid Q} \quad [\textsc{Par}]$$

$$\frac{P \xrightarrow{\pi} P', \quad x \notin n(\pi)}{(\nu x)P \xrightarrow{\pi} (\nu x)P'} \quad [\textsc{Res}]$$

$$\frac{P \xrightarrow{\pi} P', \quad Q \xrightarrow{\overline{\pi}} Q', \quad \{x\} = bn(\pi)}{P \mid Q \xrightarrow{\tau} (\nu x)\,(P' \mid Q')} \quad [\textsc{Com}]$$

4.7. táblázat. A $P\pi$-kalkulus műveleti szemantika szabályai

4.5.2. Biszimuláció

A 2.6.9. példában láttuk, hogy az eredeti pi-kalkulus késői biszimulációját tekintve, vizsgálva az $a \mid \overline{b}$ és az $a \,.\, \overline{b} + \overline{b} \,.\, a$ folyamatokat,

$$a \mid \overline{b} \;\dot{\sim}_l\; a \,.\, \overline{b} + \overline{b} \,.\, a \,.$$

Ha a kifejezések elé helyeztünk például egy $c(a)$ prefixet, akkor a

$$c(a) \,.\, a \mid \overline{b}, \quad c(a) \,.\, (a \,.\, \overline{b} + \overline{b} \,.\, a)$$

kifejezések kölcsönös hasonlósága megszűnt. Nyilvánvalóan a

$$c(a) \,.\, a \mid \overline{b} \;\dot{\sim}_l\; c(a) \,.\, (a \,.\, \overline{b} + \overline{b} \,.\, a + [x = y]\tau)$$

volt a helyes kapcsolat.

Ez egyrészt azt jelentette, hogy a kölcsönös hasonlóság nem volt kongruencia, másrészt azt, hogy egy konkrét rendszerben a biszimuláció elemzésekor minden névpár esetén a pár neveinek azonosságát vizsgálni kell, ami egy bonyolultabb rendszerben nagyon munka- és időigényessé válhat.

Ez a probléma azonban a $P\pi$-kalkulusban nem jelentkezhet, mert itt az input prefix a biszimuláció tulajdonságot megtartja,

$$c(a) \,.\, a \mid \overline{b} \;\; \dot{\sim} \;\; c(a) \,.\, (a \,.\, \overline{b} + \overline{b} \,.\, a) \;.$$

Ha az input után $a \equiv b$, akkor a jobboldali kifejezésben az összeg biztosan megmarad, a bal oldal viszont **0**-ra redukálható. Ez a redukció, pontosabban az $a \equiv b$ azonosság azonban a $P\pi$-kalkulusban nem jöhet létre. Írjuk ki részletesen a baloldali kifejezést:

$$c(a) \,.\, a(z) \mid \overline{b}(z) \;.$$

Kommunikáció akkor jöhetne létre, ha az $a(z)$ kifejezés $b(z)$ lenne, azaz a $c(a)$ inputból az a formális paraméter értéke b lenne. Ez csak a $c(a)$-nak egy $\overline{c}(b)$-vel való kommunikációjával állhatna elő, de a $P\pi$-kalkulusban a $\overline{c}(b) \mid c(a)$ folyamat nem redukálható, tehát nincs olyan, hogy a c néven b-t olvasnánk.

Ezek után a kellemes eredmények után definiáljuk pontosan a biszimulációval kapcsolatos fogalmakat.

4.5.4. Definíció. Biszimuláció:

Legyen $\mathcal{R}$ egy szimmetrikus bináris reláció az S halmaz felett, és legyen $P \,\mathcal{R}\, Q$. Az $\mathcal{R}$ biszimuláció,

- *ha $P \xrightarrow{\alpha} P'$, $\alpha = x(z), \overline{x}(z)$ vagy τ, és $z \notin fn(P, Q)$, akkor $Q \xrightarrow{\alpha} Q'$ esetén $P' \,\mathcal{R}\, Q'$.*

A biszimuláció ábrázolása rendkívül egyszerű (4.4. ábra), mivel a definícióban csak az α-ra van megkötés előírva.

$$\begin{tikzcd}
P \arrow[r, dotted, no head] \arrow[d, "\alpha\in\{x(z),\overline{x}(z),\tau\}"] & \mathcal{R} \arrow[r, dotted, no head] & Q \arrow[d, "\alpha"] \\
P' \arrow[r, dotted, no head] & \mathcal{R} \arrow[r, dotted, no head] & Q'
\end{tikzcd}$$

4.4. ábra. A privát biszimuláció

4.5.5. Definíció. Kölcsönös hasonlóság:

A kölcsönös hasonlóság legyen a biszimulációk uniója, és a kölcsönös hasonlóság jele legyen $\dot{\sim}_p$ *. Azt mondjuk, hogy P és Q folyamatok kölcsönösen hasonlóak, azaz* $P \dot{\sim}_p Q$*, ha van olyan* $\mathcal{R}$ *biszimuláció, melyre* $P \mathcal{R} Q$ *.*

4.5.6. Definíció. Kongruencia:

A P és Q folyamat a kölcsönös hasonlóságot tekintve kongruens, ha minden σ *helyettesítésre* $P\sigma \dot{\sim}_p Q\sigma$ *. A folyamatok kongruenciájának a jele* $\sim_p$*, tehát ekkor* $P \sim_p Q$ *.*

A korábbiakban már utaltunk rá, hogy az $P\pi$-kalkulusban a kölcsönös hasonlóság kongruencia is, ezt mondja ki a következő tétel.

4.5.7. Tétel. (A privát kalkulus kongruenciája)

A $P\pi$*-kalkulusban* $\dot{\sim}_p \equiv \sim_p$ *.*

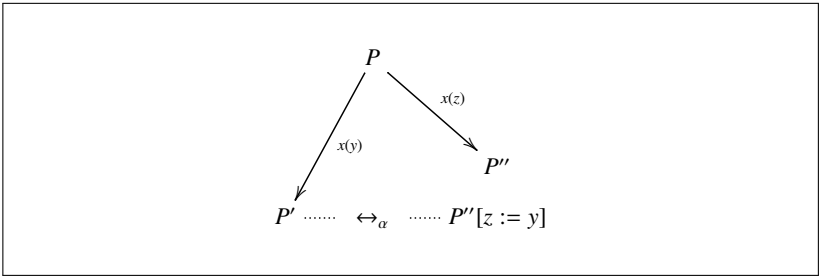
A tétel alapján a privát kalkulusban több nagyon jól használható összefüggést kapunk, ilyenek például a következők:

- Ha $P \leftrightarrow_\alpha Q$, akkor $P \sim_p Q$,
- ha $P \sim_p Q$ és $y \notin fn(P)$, akkor mindex x-re $P[x := y] \sim_p Q[x := y]$,
- ha $P \xrightarrow{x(y)} P'$ és $z \notin fn(P)$, akkor $P \xrightarrow{x(z)} P''$, ahol $P''[z := y] \leftrightarrow_\alpha P'$ (lásd 4.5. ábra).

Ha $P \sim_p Q$, akkor

- $\alpha . P \sim_p \alpha . Q$,
- $P + R \sim_p Q + R$,
- $(\nu x) P \sim_p (\nu x) Q$,
- $P \mid R \sim_p Q \mid R$.

Ebben a kalkulusban is definiálhatjuk a gyenge biszimulációt és a gyenge kölcsönös hasonlóságot is. A biszimuláció definíciója sokkal egyszerűbb, mint a pi-kalkulusban volt:



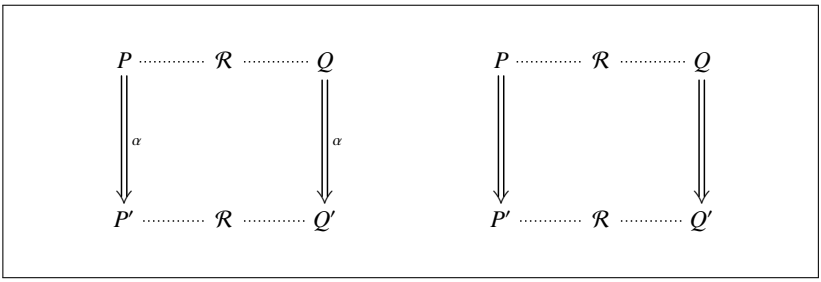
4.5. ábra. Helyettesítés és az α-konverzió

4.5.8. Definíció. Gyenge biszimuláció:

Legyen $\mathcal{R}$ egy szimmetrikus bináris reláció az S halmaz felett, és legyen $P\,\mathcal{R}\,Q$. Az $\mathcal{R}$ gyenge biszimuláció a privát kalkulusban,

- *ha $P \overset{\alpha}{\Longrightarrow} P'$, $\alpha \neq \tau$ és $bn(\alpha) \cap fn(P, Q) = \emptyset$, akkor $Q \overset{\alpha}{\Longrightarrow} Q'$ esetén $P'\,\mathcal{R}\,Q'$,*
- *ha $P \Longrightarrow P'$, akkor $Q \Longrightarrow Q'$ esetén P' R Q' .*

a definícióban szereplő átmenetek a 4.6. ábrán láthatók.



4.6. ábra. A $P\pi$ gyenge biszimulációja

A gyenge biszimulációval definiáljuk a gyenge kölcsönös hasonlóságot és a gyenge kongruenciát, és utána a 4.5.7. tételhez hasonlóan, kimondjuk azt privát kalkulusban bizonyítható állítást, hogy a gyenge kölcsönös hasonlóság azonos a gyenge kongruenciával.

4.5.9. Definíció. Gyenge kölcsönös hasonlóság:

A gyenge kölcsönös hasonlóság legyen a gyenge kölcsönös biszimulációk uniója, és a gyenge kölcsönös hasonlóság jele legyen $\dot{\approx}_p$ *. Azt mondjuk, hogy P és Q folyamatok gyenge kölcsönösen hasonlóak, azaz* $P \dot{\approx}_p Q$*, ha van olyan* $\mathcal{R}$ *gyenge biszimuláció, melyre* $P \mathcal{R} Q$ *.*

4.5.10. Definíció. Gyenge kongruencia:

A P és Q folyamat gyenge kongruens, ha minden σ *helyettesítésre* $P\sigma \dot{\approx}_p Q\sigma$ *. A kongruencia jele* $\approx_p$*, tehát ekkor* $P \approx_p Q$ *.*

Ezek után következhet a privát kalkulus gyenge kongruenciájának nagy jelentőségét kimondó tétel:

4.5.11. Tétel. (A privát kalkulus gyenge kongruenciája)

A $P\pi$*-kalkulusban* $\dot{\approx}_p \equiv \approx_p$ *.*

5. FEJEZET

Típusos pi-kalkulusok

Ebben a fejezetben először [9] alapján áttekintjük a típusos kalkulusokra, a *típusrendszerekre* vonatkozó alapfogalmakat, és csak ezután foglalkozunk a típusos pi-kalkulusokkal. Elemezzük a legegyszerűbb típusos pi-kalkulus tulajdonságait, majd a típusrendszert bővítve eljutunk a *magasabb rendű* típusos pi-kalkulushoz.

5.1. Formális típusrendszerek

Egy formális típusrendszer három komponensből áll, az első a típusrendszer *szintaxisa*, amely a folyamatkifejezések és a típusok szintaktikáját határozza meg.

A formális típusrendszer kifejezései két összetevőből állnak, egy folyamatkifejezésből és a kifejezés típusát leíró típuskifejezésből. Ennek megfelelően a szintaktika is két részből áll, az egyik rész a típusok kifejezéseinek, a másik rész a folyamatkifejezéseknek a szintaktikáját írja le. Mindkét szintaktika *környezetfüggetlen grammatikával* írható le.

5.1.1. Definíció. Jól formált folyamatkifejezés:

> *Egy folyamatkifejezés jól formált, ha szintaktikusan helyes, azaz megfelel a folyamatkifejezések szintaktikájának leírásában megadott szintaktikai szabályoknak.*

5.1.2. Definíció. Jól formált típus:

> *Egy típus jól formált, ha szintaktikusan helyes, azaz megfelel a típusok szintaktikájának leírásában megadott szintaktikai szabályoknak.*

A „jólformáltság” rövid leírására a következőkben a *wf* jelölést fogjuk használni.

Ha egy jól formált kifejezésben csak jól formált típusok szerepelnek, ez még nem garantálja azt, hogy a kifejezés a típus szempontjából helyes. Egy kifejezés típus szerinti helyessége ugyanis nem írható le környezetfüggetlen grammatikával. A típus helyességét csak a

- típuskörnyezet és a
- következtetési szabályok

ismeretében tudjuk eldönteni. A formális típusrendszer másik két komponense, a következtetések *formái* és a következtetési *szabályok* ezt a célt szolgálják.

5.1.3. Definíció. A típuskörnyezet szintaktikája:

$$\begin{array}{lcl} \langle \textit{típuskörnyezet} \rangle & ::= & \emptyset \\ & | & \langle \textit{típuskörnyezet} \rangle\,,\ \langle \textit{név} \rangle : \langle \textit{típus} \rangle \end{array}$$

A típuskörnyezetekre a következő tulajdonságot is megköveteljük:

- *Ha x_i egy T_i típusú név ($1 \leq i \leq n$) és a típuskörnyezet az $x_i : T_i$ párokból felépített*

  $\{x_1 : T_1, x_2 : T_2, \ldots, x_n : T_n\}$

  halmaz, akkor $x_i \neq x_j \ \ (i \neq j)$.

A definícióban szereplő feltétel azt jelenti, hogy a típuskörnyezetben egy név csak egyszer szerepelhet, és így egy névnek legfeljebb egy típusa lehet.

A típuskörnyezet szokásos jelölése Γ . Ha a Γ típuskörnyezet egy párt sem tartalmaz, akkor a típuskörnyezetre az $\emptyset$ („üres halmaz”) jelet használjuk. A Γ-ban szereplő nevek halmazát $dom(\Gamma)$-val jelöljük.

A típuskörnyezetre is definiáljuk a jólformáltságot:

5.1.4. Definíció. Jól formált típuskörnyezet:

Egy típuskörnyezet jól formált, ha szintaktikusan helyes, azaz megfelel az 5.1.3. definícióban megadott szabályoknak.

Egy formális típusrendszerben *következtetéseket* adhatunk meg. Egy következtetés alakja

$$\Gamma \vdash \mathcal{I}\ ,$$

ahol Γ a *típuskörnyezet*, és $\mathcal{I}$ a Γ-ból adódó *állítás*.

Ha a Γ típuskörnyezet jól formált, akkor ezt a

$$\Gamma \vdash wf$$

következtetéssel jelöljük. Az $\emptyset$ üres típuskörnyezet nyilvánvalóan mindig jól formált, azaz $\emptyset \vdash wf$. Ha a Γ típuskörnyezet alapján a T típus jól formált, akkor ezt a tulajdonságot a

$$\Gamma \vdash T$$

jelöli. Számunkra most a

$$\Gamma \vdash P : T$$

alakú *következtetések* a fontosak. A $\Gamma \vdash P : T$ következtetés azt mondja ki, hogy a Γ típuskörnyezetben levő információt figyelembe véve a P kifejezés típusa T.

A formális típusrendszerben egy következtetés lehet *érvényes* vagy *érvénytelen*. Egy következtetés érvényességének bizonyítására a típusrendszer *szabályai* szolgálnak. Egy szabály

$$\frac{\Gamma_1 \vdash \mathcal{I}_1 \quad \ldots \quad \Gamma_n \vdash \mathcal{I}_n}{\Gamma \vdash \mathcal{I}} \ [\text{SZABÁLYNÉV}]$$

alakú, ahol a vízszintes vonal fölött a *feltétel* következtetései, a vonal alatt a feltételekből származtatott *következmény* következtetése van. Egy szabály a feltételben szereplő következtetések érvényességét nem vizsgálja, a szabály azt mondja ki, hogy ha a feltétel mindegyik következtetése érvényes, akkor a következmény következtetése is érvényes.

A szabályoknak nevet is adhatunk, és ha szükséges, a feltételek mellett jobb oldalon még a szabály alkalmazásának feltételeit is megadhatjuk.

Ha egy szabályban a feltételek halmaza üres, akkor a következtetés mindig érvényes. Az ilyen következtetéseket a típusrendszer *axiómáinak* nevezzük.

A szabályokat *típuslevezetések* készítésére használjuk úgy, hogy egy *következtetési fát* vagy más néven *levezetési fát* építünk fel. A szabályokat egymáshoz kapcsoljuk, az egyik szabály következményének következtetése egy másik szabály feltételének egy következtetéséhez kapcsolható, ha a két következtetés megegyezik.

5.1.5. Definíció. Érvényes következtetés:

Azt mondjuk, hogy a $\Gamma \vdash P : T$ következtetés érvényes, ha létezik olyan típuslevezetés, ahol a következtetés a típuslevezetéshez tartozó következtetési fa gyökérpontja.

5.1.6. Definíció. Jól típusozott kifejezés:

Ha egy típusrendszerben a $\Gamma \vdash P : T$ következtetés érvényes, akkor azt mondjuk, hogy a P kifejezés jól típusozott.

Így most már ismerjük a jól típusozott kifejezés fogalmát. A gyakorlatban természetesen az a cél, hogy a vizsgált folyamatok jól típusozott folyamatok legyenek.

5.2. Az $F_{\pi 0}$ típusrendszer

A legegyszerűbb típusos pi-kalkulussal kezdjük, amit $F_{\pi 0}$ típusrendszernek nevezünk. Ez a típusrendszer lényegében a *CCS-VP* „CCS with Value-Passing” rendszeren alapul [7].

5.2.1. A típusrendszer szintaxisa

A pi-kalkulusban az egyik alapvető fogalom a *név*, ami lényegében a kommunikációs csatornákat, kapukat és a csatornákon küldött neveket és adatokat azonosítja. Az $F_{\pi 0}$ típusos kalkulusban a nevek fogalmát az *értékek* fogalma veszi át, és ezen belül beszélhetünk *nevekről* és *alapértékekről*.

Ebben a típusrendszerben az alapértékek és az értékek típusa nem tetszőleges, hanem a típusrendszertől függnek, egy előre definiált *alapérték-halmaznak* és az *alaptípus-halmaznak* az elemei lehetnek. Az alapértékek halmazát B_{val}-lal, az alaptípusok halmazát $T_{B,\pi 0}$-val jelöljük.

A nevek lényegében a típusrendszer változói szerepét töltik be. A nevek közül azokat, amelyeken a kommunikáció történik, *összekötő neveknek*, röviden *összekötőknek*, típusukat *összekötőtípusnak* nevezzük. Az összekötőkön haladó adatok az alapértékek. Egy összekötő típusa statikus és nem tetszőleges, az összekötő típusát meghatározza az összekötőn áthaladó érték típusa.

5.2.1. Definíció. Az $F_{\pi 0}$ típusrendszer típusai:

$$
\begin{array}{lcl}
\langle \textit{típus} \rangle & ::= & \textit{wf} \\
 & | & \langle \textit{értéktípus} \rangle \\
 & | & \langle \textit{összekötőtípus} \rangle \\
\langle \textit{értéktípus} \rangle & ::= & \langle \textit{alaptípus} \rangle \\
\langle \textit{összekötőtípus} \rangle & ::= & \sharp \langle \textit{értéktípus} \rangle
\end{array}
$$

A *wf* típust a folyamatok jóltípusozottságának jelölésére használjuk.

A definícióból látható, hogy az összekötőtípusokat a $\sharp$ jellel különböztetjük meg a többi típustól. Megjegyezzük, hogy néhány publikációban a $\sharp$ helyett a ch karakterpárt használják.

Az összekötőtípusnak nagyobb a precedenciája, mint az értéktípusnak, így például

$$\sharp T_1 \times T_2 \equiv (\sharp T_1) \times T_2 \ .$$

Ha T egy összekötőtípus, akkor $O(T)$-vel jelöljük azoknak az értékeknek a típusát, amelyek egy input vagy output művelettel a T összekötőn haladhatnak át. Például ha $\sharp T$ egy összekötőtípus, akkor $O(\sharp T) = T$.

A típuskörnyezetet az 5.1.3. definícióban írtuk le, ebben a szakaszban használni fogjuk majd a *zárt típuskörnyezet* fogalmat, és ehhez kapcsolódik a *zárt folyamat* kifejezés is.

5.2.2. Definíció. Zárt típuskörnyezet:

A Γ típuskörnyezetet zártnak nevezzük, ha Γ-ban csak olyan $x : T$ párok vannak, ahol T összekötőtípus.

5.2.3. Definíció. Zárt folyamat:

A P folyamatot zártnak nevezzük, ha a Γ típuskörnyezet zárt, és $\Gamma \vdash P$.

A folyamatok kifejezéseiben használt prefixek és a folyamatok definícióját a 2.3.1. és 2.3.2. definíciókban határoztuk meg, de a könnyebb kezelhetőség kedvéért itt is megadjuk:

$$\pi ::= \overline{x}y \mid x(y) \mid \tau \mid [x = y]\pi \ ,$$

$$P ::= \mathbf{0} \mid \pi \,.\, P \mid P + P \mid P \,|\, P \mid (\nu x)\, P \mid !P \ .$$

Az $F_{\pi 0}$ típusrendszer folyamatait a következő definíció adja meg.

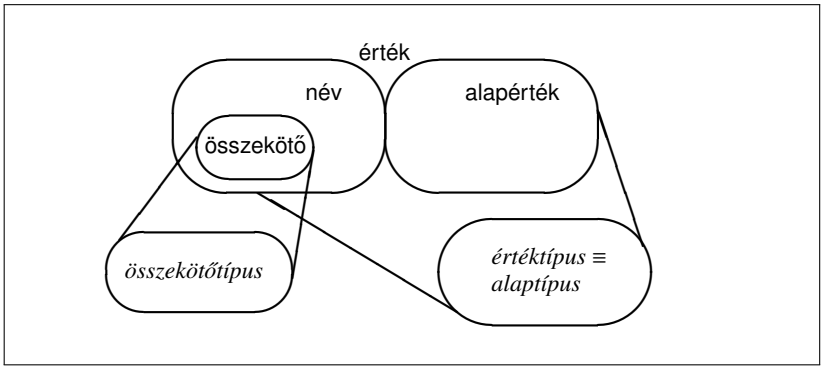
5.2.4. Definíció. ***Az $F_{\pi 0}$ típusrendszer folyamatai:***

$$
\begin{array}{lcl}
\langle \textit{érték} \rangle & ::= & \langle \textit{név} \rangle \\
 & | & \langle \textit{alapérték} \rangle \\
\langle \textit{folyamat} \rangle & ::= & \ldots \\
 & | & (\boldsymbol{\nu} \langle \textit{név} \rangle : \langle \textit{összekötőtípus} \rangle)\, \langle \textit{folyamat} \rangle \\
 & | & \ldots \\
 & | & \texttt{wrong}
\end{array}
$$

A definícióban nem soroltuk fel az összes lehetőséget a folyamatokra, mivel a típusnélküli folyamatokhoz viszonyítva változás csak a $(\boldsymbol{\nu} x)\, P$ kifejezésre van, ahol megjelenik a kötött név típusa. Ez a típus csak összekötőtípus lehet.

Új a folyamatok között a `wrong`, amely a típusosan hibás folyamat runtime hibajelzését fogja majd jelenteni.

Az értékek és a típusok kapcsolatát szemléletesen az 5.1. ábrán láthatjuk.



5.1. ábra. Az $F_{\pi 0}$ típusrendszer értékei és típusai

5.2.5. Példa. *(Alaphalmazok)*

Legyen $F_{\pi 0}$ egy olyan típusrendszer, ahol

$$
\begin{array}{lcl}
B_{val} & = & \{\, 0, 1, \ldots, \mathsf{True}, \mathsf{False}, \mathsf{Unit} \,\}\,, \\
T_{B,\pi 0} & = & \{\, \mathit{Nat}, \mathit{Bool}, \mathit{Unit} \,\}\,.
\end{array}
$$

□

Az értékekre és a típusokra adott erős korlátozásokat majd a következő típusrendszerben fogjuk megszüntetni.

5.2.2. Következtetések és típusszabályok

Most megadjuk az $F_{\pi 0}$ típusrendszer további leírását, a következtetések alakját és a típusrendszer szabályait. Felhívjuk a figyelmet arra, hogy a műveleti szemantika átmeneti szabályait korábban egyszerűen „szabályoknak” neveztük, ezeket most már meg kell különböztetnünk a típusrendszer típusokra vonatkozó szabályaitól. Ezért a továbbiakban a szemantika szabályait *átmeneti szabályoknak*, a típusokkal kapcsolatos szabályokat *típusszabályoknak* nevezzük.

5.2.6. Definíció. Az $F_{\pi 0}$ típusrendszer következtetései:

Γ	$\vdash$	wf	Γ jól formált környezet ,
Γ	$\vdash$	T	Γ-ban a T jól formált típus ,
Γ	$\vdash$	$P : T$	Γ-ban a P kifejezés típusa T .

Az első két következtetés a típuskörnyezet és a típus jólformáltságát mondja ki, a harmadik egy kifejezés típusát adja meg.

Az $F_{\pi 0}$ típusrendszerben a következtetések érvényességét a következő típusszabályok felhasználásával tudjuk bizonyítani. A leírásban v, w értéket, az x nevet jelöl.

5.2.7. Definíció. Az $F_{\pi 0}$ típusrendszer típusszabályai:

A környezetre vonatkozó típusszabályok:

$$\frac{}{\emptyset \vdash wf} \quad [\text{Env-}\emptyset]$$

$$\frac{\Gamma \vdash T \quad x \notin dom(\Gamma)}{\Gamma, x : T \vdash wf} \quad [\text{Env-}x]$$

A típusra vonatkozó típusszabályok:

$$\frac{\Gamma \vdash wf \quad T \in T_{B,\pi 0}}{\Gamma \vdash T} \quad [\text{Type-Base}]$$

$$\frac{\Gamma \vdash wf \quad T \in T_{B,\pi 0}}{\Gamma \vdash \sharp T} \quad [\text{Type-Link}]$$

$$\frac{\Gamma',\ x : T,\ \Gamma'' \vdash wf}{\Gamma',\ x : T,\ \Gamma'' \vdash x : T} \quad [\text{Type-Name-}x]$$

A folyamatra vonatkozó típusszabályok:

$$\frac{}{\Gamma \vdash \mathbf{0} : wf} \quad [\textsc{Type-}\mathbf{0}]$$

$$\frac{\Gamma \vdash v : \sharp T, \quad \Gamma \vdash w : T, \quad \Gamma \vdash P : wf}{\Gamma \vdash \overline{v}w \,.\, P : wf} \quad [\textsc{Type-Out}]$$

$$\frac{\Gamma \vdash v : \sharp T, \quad \Gamma, x : T \vdash P : wf}{\Gamma \vdash v(x) \,.\, P : wf} \quad [\textsc{Type-Inp}]$$

$$\frac{\Gamma \vdash P : wf}{\Gamma \vdash \tau \,.\, P : wf} \quad [\textsc{Type-}\tau]$$

$$\frac{\Gamma \vdash v : \sharp T \quad \Gamma \vdash w : \sharp T \quad \Gamma \vdash P : wf}{\Gamma \vdash [v = w] \,.\, P : wf} \quad [\textsc{Type-Match}]$$

$$\frac{\Gamma \vdash P : wf \quad \Gamma \vdash Q : wf}{\Gamma \vdash P + Q : wf} \quad [\textsc{Type-Sum}]$$

$$\frac{\Gamma \vdash P : wf \quad \Gamma \vdash Q : wf}{\Gamma \vdash P \mid Q : wf} \quad [\textsc{Type-Par}]$$

$$\frac{\Gamma, x : \sharp T \vdash P : wf}{\Gamma \vdash (\boldsymbol{\nu} x : \sharp T)\, P : wf} \quad [\textsc{Type-Res}]$$

$$\frac{\Gamma \vdash P : wf}{\Gamma \vdash !P : wf} \quad [\textsc{Type-Rep}]$$

Néhány megjegyzés a típusszabályokkal kapcsolatban:

Az Env-x típusszabályban x tetszőleges értéket, azaz nevet vagy alapértéket jelöl, és $dom(\Gamma)$ a típuskörnyezetben levő nevek, azaz a párosok első komponenseinek halmaza.

Az első négy típusszabály a definíciók alapján magától értetődik. Az x névre vonatkozó Type-Name-x azt mondja ki, hogy egy név a típusával együtt kiolvasható a típuskörnyezetből, és ettől a művelettől a típuskörnyezet nem változik meg.

A típusszabályok szerint a **0** kifejezéssel semmi probléma nem lehet. Ha mindkét komponens jól típusozott, az összegkifejezés és a kompozíció esetén a velük képzett folyamatok is jól típusozottak. Ugyanez teljesül, csak egy folyamatkifejezéssel, az ismétlés műveletével kapott kifejezésre is.

A korlátozás típusszabályánál x egy név, típusa $\sharp T$, azaz x típusa csak összekötőtípus lehet. Az is látható, hogy ez a név–típus páros a kifejezés típusozásánál kikerül a típuskörnyezetből.

Az output prefixes kifejezésnél a v névnek, a prefix alanyának a típusa egy összekötőtípus, és ez összhangban van w-vel, a prefix tárgyának típusával, hiszen a v típusa a w értéktípusából képzett összekötőtípus. Ugyanez a tulajdonság mondható el az input prefixes kifejezésre is, csak itt a prefix tárgya nem a w, hanem az x lesz.

5.2.8. Példa. *(Típusos nevek)*

Az $x : \sharp T$ azt jelenti, hogy az x egy olyan összekötő neve, amelyen T típusú adatok áramolhatnak.

Legyen, mint az 5.2.5. példában,

$B_{val} = \{0, 1, \ldots, \mathsf{True}, \mathsf{False}, \mathsf{Unit}\}$,

$T_{B,\pi 0} = \{Nat, Bool, Unit\}$,

ekkor $0 : Nat$, $1 : Nat$, $\ldots$, $\mathsf{True} : Bool$, $\mathsf{False} : Bool$, $\mathsf{Unit} : Unit$.

A $\mathsf{True} : Bool$ állítás típusszabályok szerinti levezetése a következő:

$$\dfrac{\dfrac{\dfrac{}{\emptyset \vdash wf}\,[\textsc{Env-}\mathbf{0}] \qquad Bool \in T_{B,\pi 0}}{\emptyset \vdash Bool}\,[\textsc{Type-Base}] \qquad \mathsf{True} \notin dom(\Gamma)}{\mathsf{True} : Bool \vdash wf}\,[\textsc{Env-}x]}{\mathsf{True} : Bool \vdash \mathsf{True} : Bool}\,[\textsc{Type-Name-}x]$$

□

5.2.3. Műveleti szemantika

Az $F_{\pi 0}$ típusrendszer műveleti szemantikáját a pi-kalkulus korai szemantikájára építjük rá, a pi-kalkulus korai műveleti szemantika átmeneti szabályait a 2.13. táblázatokban foglaltuk össze.

Az $F_{\pi 0}$ típusrendszer műveleti szabályait, azaz az átmeneti szabályokat az 5.2. és 5.3. táblázatban adjuk meg.

A táblázatból látható, hogy azokon a helyeken van változás, ahol kötött nevek szerepelnek, mivel a kötött nevek típusát minden esetben meg kell adni.

Az Err-... nevű típusszabályok bevezetésének szükségességét a következő példában mutatjuk meg.

$$\frac{P' \equiv P, \quad P \xrightarrow{\alpha} Q, \quad Q \equiv Q'}{P' \xrightarrow{\alpha} Q'} \quad [\textsc{Struct}]$$

$$\frac{}{\overline{x}y\,.\,P \xrightarrow{\overline{x}y} P} \quad [\textsc{Output}] \qquad \frac{\overline{x}y\,.\,P \xrightarrow{\overline{x}y} P, \quad x \neq y}{(\nu y : T)\overline{x}y\,.\,P \xrightarrow{(\nu y : T)\overline{x}y} P} \quad [\textsc{Open}]$$

$$\frac{}{x(y)\,.\,P \xrightarrow{xw} P[y := w]} \quad [\textsc{Early-Input}] \qquad \frac{}{\tau\,.\,P \xrightarrow{\tau} P} \quad [\textsc{Tau}]$$

$$\frac{\alpha\,.\,P \xrightarrow{\alpha} P}{[x = x]\alpha\,.\,P \xrightarrow{\alpha} P} \quad [\textsc{Match}]$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \quad [\textsc{Sum}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad bn(\alpha) \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad [\textsc{Par}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad x \notin n(\alpha)}{(\nu x : T)P \xrightarrow{\alpha} (\nu x : T)P'} \quad [\textsc{Res}]$$

$$\frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \quad [\textsc{Rep}]$$

$$\frac{P \xrightarrow{\overline{x}y} P', \quad Q \xrightarrow{xw} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \quad [\textsc{Early-Com}]$$

$$\frac{(\nu y : T)\overline{x}y\,.\,P \xrightarrow{\overline{x}(y)} P, \quad x(z)\,.\,Q \xrightarrow{x(y)} Q[z := y]}{(\nu y : T)\overline{x}y\,.\,P \mid x(z)\,.\,Q \xrightarrow{\tau} (\nu y : T)(P \mid Q[z := y])} \quad [\textsc{Close}]$$

5.2. táblázat. Az $F_{\pi 0}$ átmeneti szabályai (1. rész)

5.2.9. Példa. *(Az* Err-*... szabályok)*

Tudjuk, hogy $\mathsf{True} : Bool$, és legyen $x : \sharp Bool$. Ekkor az

$\overline{x}\,\mathsf{True}\,.\,\mathbf{0}$

folyamat típusosan helyes. Nézzük meg, hogy mi történik, ha ezt a folyamatot

$$\frac{v \text{ nem név}}{\overline{v}w\,.\,P \xrightarrow{\tau} \texttt{wrong}} \quad [\textsc{Err-Output}]$$

$$\frac{v \text{ nem név}}{v(x)\,.\,P \xrightarrow{\tau} \texttt{wrong}} \quad [\textsc{Err-Input}]$$

$$\frac{v \text{ vagy } w \text{ nem név}}{[v = w]\,.\,P \xrightarrow{\tau} \texttt{wrong}} \quad [\textsc{Err-Match}]$$

5.3. táblázat. Az $F_{\pi 0}$ átmeneti szabályai (2. rész)

az $x(y)\,.\,\overline{y}z\,.\,\mathbf{0}$ folyamattal párhuzamosan összekapcsoljuk:

$\overline{x}\,\mathsf{True}\,.\,\mathbf{0} \mid x(y)\,.\,\overline{y}z\,.\,\mathbf{0} \xrightarrow{\tau}$

$\overline{\mathsf{True}}\,z\,.\,\mathbf{0} \xrightarrow{\tau}$

$\texttt{wrong}$.

Mivel a futtatás előtt nem végeztünk típusellenőrzést és a kifejezésünk típushibás, run-time hibajelzést kaptunk. A következő pontban az 5.2.17. példában majd látjuk, hogy ha típusellenőrzést végzünk, már akkor, már a futtatás előtt hibajelzést kapunk. □

5.2.4. Az $F_{\pi 0}$ típusrendszer tulajdonságai

Először a típuskörnyezetre mondunk ki három egyszerű állítást.

Egy típuskörnyezet permutációján a benne levő változó–típus párok tetszőleges számú megcserélését értjük.

5.2.10. Tétel. (Típuskörnyezet permutációja)

Legyen a Γ *típuskörnyezet permutációja* Γ' .
Ha $\Gamma \vdash P : T$, *akkor* $\Gamma' \vdash P : T$.

5.2.11. Tétel. (Típuskörnyezet gyengítése)

Ha $\Gamma \vdash P : T$, *akkor* $\Gamma, x : T' \vdash P : T$.

5.2.12. Tétel. (Típuskörnyezet szűkítése)

Ha $\Gamma, x : T' \vdash P : T$ *és* $x \notin fn(P)$, *akkor* $\Gamma \vdash P : T$.

Az 5.1.3. definíció szerint a típuskörnyezet $x_i : T_i$ $(1 \leq i \leq n)$ párok listája. Az első állítás azt mondja ki, hogy a jóltípusozottságot a lista elemeinek sorrendje nem befolyásolja. A következő két lemmában a típuskörnyezetet módosítjuk.

A második állítás szerint a típuskörnyezet bővítése a kifejezések típusát nem változtatja meg. Ha $x \in fn(P)$, akkor $x \in \Gamma$ és a $\{\Gamma, x : T'\}$ típuskörnyezet nem lesz jól formált. Ha $x \notin fn(P)$, akkor mint a harmadik lemmából is látjuk, az x nem játszik szerepet a P típusának meghatározásában.

A harmadik lemma szerint egy kifejezés típusa nem változik, ha a típuskörnyezetből elhagyjuk a kifejezés kötött változóira vonatkozó típusállításokat és azokat a változó–típus párokat, amelyeknek a változói nem szerepelnek P-ben, azaz a típus nem változik, ha a típuskörnyezetben csak a kifejezés szabad változóit és azoknak a típusait hagyjuk meg.

A helyettesítésekre vonatkozik a következő állítás.

5.2.13. Tétel. (A helyettesítés tétele)

Ha $\Gamma \vdash P : T$, $\Gamma \vdash x : T'$, $\Gamma \vdash y : T'$, *akkor* $\Gamma \vdash P[x := y] : T$.

A tétel szerint tehát egy helyettesítés a típus szempontjából csak akkor lesz helyes, ha a helyettesítendő típusa megegyezik a behelyettesített típusával. A tétel azt is megadja, hogy a P kifejezés típusa a helyettesítéssel nem változik meg.

Minden típusrendszer két legfontosabb tétele a tárgyredukció és a típushelyesség tétele.

5.2.14. Tétel. (Tárgyredukció tétele)

Ha Γ *zárt típuskörnyezet,* $\Gamma \vdash P : T$ *és* $P \xrightarrow{\alpha} P'$, *akkor* $\Gamma \vdash P' : T$.

A tétel szerint tehát ha egy jól típusozott P folyamatra átmeneteket hajtunk végre, akkor a kapott folyamatok típusa megegyezik az eredeti folyamat típusával.

A tétel bizonyítása viszonylag egyszerű, ha az egyes α átmeneteket különkülön elemezzük. Például $\alpha = xy$ esetén azt kell belátni, hogy van olyan T' típus, melyre $\Gamma \vdash x : \sharp T'$, és ha $\Gamma \vdash y : T'$ akkor $\Gamma \vdash P' : wf$ [46].

A típushelyesség, vagy más néven a típusbiztonság tétele garantálja azt a Milnertől származó sok helyen idézett állítást, hogy „Well-typed programs cannot *go wrong*” [27].

5.2.15. Tétel. (A típushelyesség tétele)

Ha Γ zárt típuskörnyezet, $\Gamma \vdash P : T$ és $P \rightarrow^+ P'$, akkor P'-ben nincs `wrong` *kifejezés.*

A típushelyesség tétele nem azt mondja ki, hogy ha egy jól típusozott folyamatra átmeneteket hajtunk végre, akkor az egymásután végrehajtható átmenetek száma véges, vagyis a műveletsorozat terminál, hanem csak azt állítja, hogy a végrehajtás során típushiba miatti megállás nem fordulhat elő.

5.2.16. Példa. *(Nem termináló folyamat)*

A következő triviális kifejezés nyilvánvalóan típusosan helyes, átmenetsorozata azonban nem terminál:

$!\,\overline{x}\,\mathsf{True}$,

ahol $x : \sharp Bool$. □

5.2.17. Példa. *(Típusellenőrzés)*

Az 5.2.9. példában láttuk, hogy a

$\overline{x}\,\mathsf{True}\,.\,\mathbf{0} \mid x(y)\,.\,\overline{y}z\,.\,\mathbf{0}$

folyamat a futtatásakor `wrong` eredménnyel leállt. A problémát az $x(y)\,.\,\overline{y}z$ kifejezés okozza. Próbáljuk meghatározni ennek a kifejezésnek a típusát.

Az $\overline{y}z$ prefixből a Type-Output szabály alapján azt látjuk, hogy ha $x(y)\,.\,\overline{y}z\,.\,\mathbf{0} : wf$, akkor $z : T$ esetén $y : \sharp T$. Ezeket az adatokat figyelembe véve az $x(y)$ prefixre a Type-Input szabály szerint $y : T'$ és $x : \sharp T'$. Egy névnek csak egy típusa lehet, ez az állítás igaz y-ra is, ezért $T' = \sharp T$-nek kellene lennie, de ez biztosan nem áll fenn, mivel T' alaptípus és $\sharp T$ összekötőtípus, és a T' és $\sharp T$ típusokat tartalmazó két típushalmaz diszjunkt.

Tehát a kifejezés ebben az $F_{\pi 0}$ típusrendszerben típushibás, és ez okozta az 5.2.9. példában a run-time hiba megjelenését. □

5.3. Az F_π típusrendszer

Az $F_{\pi 0}$ típusrendszer a legegyszerűbb típusos pi-kalkulus, és az 5.2.1. pontban is láttuk, hogy a nevekre, értékekre, típusokra sok megszorítást kellett tennünk. Ebben a szakaszban olyan típusos kalkulust adunk meg, amelyik már nem a CCS rendszeren, hanem a 2. fejezetben leírt pi-kalkuluson alapul.

A típusrendszert F_π típusrendszernek nevezzük.

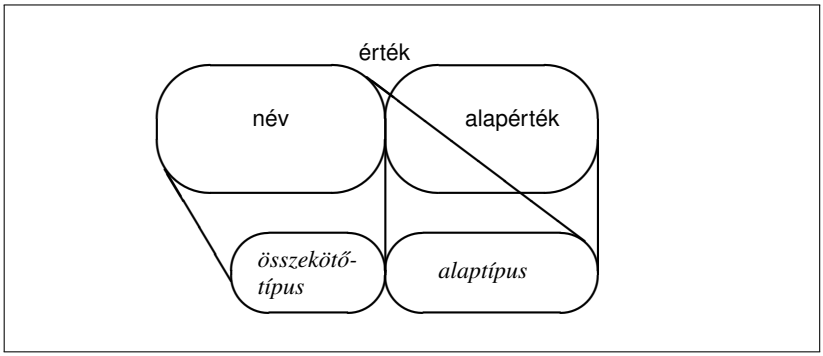
Ebben a típusrendszerben is használjuk az *alapértékek* és *alaptípusok* halmazát, és ezek itt is a típusrendszertől függő, előre definiált halmazok lesznek. Az alapértékek halmazát B_{val}-lal, az alaptípusok halmazát $T_{B,\pi}$-vel jelöljük.

Az $F_{\pi 0}$ típusrendszer szintaxisának alapvető jellemzője az volt, hogy megkülönböztettük a neveket és összekötőket, és különbséget tettünk ezek típusai között. Az F_π típusrendszerben ilyen megkötést nem teszünk. *Nevekről* beszélünk, és bármelyik név lehet *összekötő* is. Egy név típusa alaptípus, és ha a név egy összekötő, akkor a név típusa *összekötőtípus*.

5.3.1. Definíció. Az F_π típusrendszer típusai:

$$
\begin{array}{lcl}
\langle \textit{típus} \rangle & ::= & \textit{wf} \\
 & | & \langle \textit{névtípus} \rangle \\
\langle \textit{névtípus} \rangle & ::= & \langle \textit{alaptípus} \rangle \\
 & | & \langle \textit{összekötőtípus} \rangle \\
\langle \textit{összekötőtípus} \rangle & ::= & \sharp \langle \textit{névtípus} \rangle
\end{array}
$$

A *wf* típust most is a folyamatok jóltípusozottságának jelölésére használjuk.



5.2. ábra. Az F_π típusrendszer értékei és típusai

A definícióból az is látható, hogy lehetőség van $\sharp T, \sharp\sharp T, \sharp\sharp\sharp T, \ldots$ alakú összekötőtípusok definiálására. Ezzel lehetővé válik az F_π típusrendszerben is a pi-kalkulusnak az a tulajdonsága, amit mobilitásnak nevezünk, nevezetesen az, hogy a neveken át nem csak adatokat, hanem neveket is küldhetünk és fogadhatunk.

5.3.2. Példa. *(Többszörös összekötőtípus)*

Nézzük a korábbi példákban már szerepelt

$x(y) \,.\, \overline{y}z \,.\, \mathbf{0}$

kifejezést. Jobbról balra haladva: ha z típusa T, akkor $y : \sharp T$. A kifejezés első prefixében $y : \sharp T$, így $x : \sharp\sharp T$. □

A típusokól eltekintve az F_π típusrendszer minden definíciója és tétele megegyezik az $F_{\pi 0}$ típusrendszer megfelelő definíciójával és tételével, de természetesen a $T_{B,\pi 0}$ jelöléseket mindenhol a $T_{B,\pi}$-re kell kicserélni.

A definíciókat és a tételeket nem ismételjük meg, csupán a két legfontosabb tételt idézzük:

5.3.3. Tétel. ***(Tárgyredukció tétele)***

Ha Γ zárt típuskörnyezet, $\Gamma \vdash P : T$ és $P \xrightarrow{\alpha} P'$, akkor $\Gamma \vdash P' : T$.

5.3.4. Tétel. ***(A típushelyesség tétele)***

Ha Γ zárt típuskörnyezet, $\Gamma \vdash P : T$ és $P \rightarrow^+ P'$, akkor P'-ben nincs `wrong` *kifejezés.*

5.3.5. Példa. *(Típusellenőrzés)*

Az 5.2.9. példa

$\overline{x}\,\mathsf{True} \,.\, \mathbf{0} \mid x(y) \,.\, \overline{y}z \,.\, \mathbf{0}$

kifejezésére az 5.2.17. példában mutattuk meg, hogy az $F_{\pi 0}$ típusrendszerben a kompozíció második tagja típushibás. Az 5.3.2. példában láttuk, hogy ez a második tag az F_π típusrendszerben már típusosan helyes lesz, de sajnos a teljes kifejezés még mindig típushibás. Az első tagban $x : \sharp Bool$, a másodikban $x : \sharp\sharp T$, és mivel ugyanarról az x névről van szó, nincs olyan T, melyre a $\sharp Bool = \sharp\sharp T$ egyenlőség teljesülne. □

5.3.1. Az alaptípus-halmaz bővítése

A F_π típusrendszerben az alaptípusok halmazának megválasztása meghatározza a definiálható típusok halmazát. A példákban eddig a $\{Nat, Bool, Unit\}$ halmazt választottuk, most ezt a halmazt bővítjük. Példaképpen megadjuk a *Pair*, *Union* és a *Record* típusra vonatkozó szabályokat.

A *Pair* típus

Ha a *rendezett pár* első eleme $x_1 : T_1$, a második eleme $x_2 : T_2$, akkor a párt a $\langle x_1, x_2 \rangle$ zárójelezéssel jelöljük. (Ez természetesen nem tévesztendő össze a polimorfikus output tárgyával.) A pár típusát *szorzat típusnak* is nevezzük, és a típus egyszerűen $T_1 \times T_2$-vel jelölhető. A pár elemeit egy P folyamatba a

$$\mathsf{With}\ (x_1, x_2) = v\ \mathsf{Do}\ P$$

kifejezéssel töltjük be.

Az F_π típusrendszer szintaxisát a következőkkel bővítjük:

$$\begin{array}{lcl}
\langle \textit{típus} \rangle & ::= & \ldots \\
 & | & T_1 \times T_2 \\
\langle \textit{érték} \rangle & ::= & \ldots \\
 & | & \langle x_1, x_2 \rangle \\
\langle \textit{folyamat} \rangle & ::= & \ldots \\
 & | & \mathsf{With}\ (x_1, x_2) = v\ \mathsf{Do}\ P\ .
\end{array}$$

A With szerkezetben az x_1 és x_2 nevek kötöttek, és hatáskörük a P folyamat.

A *Pair* típusra vonatkozó típusszabályok:

$$\frac{\Gamma \vdash x_1 : T_1 \quad \Gamma \vdash x_2 : T_2}{\Gamma \vdash \langle x_1, x_2 \rangle : T_1 \times T_2} \quad [\textsc{Type-Pair}]$$

$$\frac{\Gamma \vdash y : T_1 \times T_2 \qquad \Gamma, x_1 : T_1, x_2 : T_2 \vdash P : wf}{\Gamma \vdash \mathsf{With}\ (x_1, x_2) = y\ \mathsf{Do}\ P : wf} \quad [\textsc{Type-With-Pair}]$$

és az átmeneti szabályok:

$$\frac{}{\mathsf{With}\ (x_1, x_2) = \langle y_1, y_2 \rangle\ \mathsf{Do}\ P \xrightarrow{\tau} P[x_1 := y_1, x_2 := y_2]} \quad [\textsc{Pair}]$$

$$\frac{y \neq \langle y_1, y_2 \rangle}{\mathsf{With}\ (x_1, x_2) = y\ \mathsf{Do}\ P \xrightarrow{\tau} \mathtt{wrong}} \quad [\textsc{Err-Pair}]$$

A *Union* típus

A *Pair* típus egy eleme olyan pár, amelyeknek az első komponense T_1, a második komponense T_2 típusú, azaz azt is mondhatjuk, hogy a pár típusa

„T_1 *és* T_2”. Ha egy olyan típust definiálunk, amelynek elemei „T_1 *vagy* T_2” típusúak, akkor a *Union* típust kapjuk meg, és a típust $T_1 \oplus T_2$-vel jelöljük. A $T_1 \oplus T_2$ típus tehát azt jelenti, hogy a típus egy eleme vagy T_1, vagy T_2 típusú. Megjegyezzük, hogy ezt a típust gyakran *diszjunkt összeg típusnak* vagy röviden *összeg típusnak* nevezik, és a típust gyakran T_1+T_2-vel is jelölik.

A $T_1 \oplus T_2$ típus konstruktora az InLeft és az InRight folyamat. Azt, hogy egy *Union* típusú kifejezés aktuális típusa T_1 vagy T_2, az IsLeft és az IsRight *Bool* értékű folyamatokkal lehet eldönteni. Ha a típus T_1, akkor az IsLeft a true értéket, az IsRight a false értéket adja, a T_2 típusra az értékek rendre false és true.

A *Union* típus eleme egy „Case”-nek nevezett „if-then-else” jellegű

Case x Of [IsLeft x Then P_1; IsRight x Then P_2]

utasítás használatára ad lehetőséget, ahol a típus egy x elemére alkalmazott IsLeft x és IsRight x határozza meg, hogy a Case utasítás melyik ága hajtódik végre. Az x kötött, hatásköre a P_1 és P_2 folyamat.

Az F_π típusrendszer szintaxisát a következőkkel bővítjük:

$$
\begin{array}{lcl}
\langle \textit{típus} \rangle & ::= & \ldots \\
 & | & T_1 \oplus T_2 \\
\langle \textit{érték} \rangle & ::= & \ldots \\
 & | & \mathsf{InLeft}\ x \\
 & | & \mathsf{InRight}\ x \\
 & | & \mathsf{IsLeft}\ x \\
 & | & \mathsf{IsRight}\ x \\
\langle \textit{folyamat} \rangle & ::= & \ldots \\
 & | & \mathsf{Case}\ x\ \mathsf{Of}\ [\mathsf{IsLeft}\ x\ \mathsf{Then}\ P_1;\ \mathsf{IsRight}\ x\ \mathsf{Then}\ P_2]
\end{array}
$$

A *Union* típus típusszabályai a következők:

$$\frac{\Gamma \vdash x : T_1 \quad \Gamma \vdash T_2}{\Gamma \vdash \mathsf{InLeft}\ x : T_1 \oplus T_2} \quad [\textsc{Type-InLeft}]$$

$$\frac{\Gamma \vdash T_1 \quad \Gamma \vdash x : T_2}{\Gamma \vdash \mathsf{InRight}\ x : T_1 \oplus T_2} \quad [\textsc{Type-InRight}]$$

$$\frac{\Gamma \vdash x : T_1 \oplus T_2}{\Gamma \vdash \mathsf{IsLeft}\ x : Bool} \quad [\textsc{Type-IsLeft}]$$

$$\frac{\Gamma \vdash x : T_1 \oplus T_2}{\Gamma \vdash \mathsf{IsRight}\ x : Bool} \quad [\textsc{Type-IsRight}]$$

$$\frac{\Gamma \vdash x : T_1 \oplus T_2 \quad \Gamma \vdash P_1 : wf \quad \Gamma \vdash P_2 : wf}{\mathsf{Case}\ x\ \mathsf{Of}\ [\mathsf{IsLeft}\ x\ \mathsf{Then}\ P_1;\ \mathsf{IsRight}\ x\ \mathsf{Then}\ P_2]\ : wf} \quad [\textsc{Type-Case}]$$

A *Union* típus folyamatainak átmeneti szabályai:

$$\frac{}{\mathsf{Case}\ (x = \mathsf{InLeft}\ y)\ \mathsf{Of}\ [\mathsf{IsLeft}\ x\ \mathsf{Then}\ P_1;\ \mathsf{IsRight}\ x\ \mathsf{Then}\ P_2] \xrightarrow{\tau} P_1[x := y]} \quad [\textsc{Case-Left}]$$

$$\frac{}{\mathsf{Case}\ (x = \mathsf{InRight}\ y)\ \mathsf{Of}\ [\mathsf{IsLeft}\ x\ \mathsf{Then}\ P_1;\ \mathsf{IsRight}\ x\ \mathsf{Then}\ P_2] \xrightarrow{\tau} P_2[x := y]} \quad [\textsc{Case-Right}]$$

$$\frac{x \neq \mathsf{InLeft}\ y \textit{ és } x \neq \mathsf{InRight}\ y}{\mathsf{Case}\ x\ \mathsf{Of}\ [\mathsf{IsLeft}\ x\ \mathsf{Then}\ P_1;\ \mathsf{IsRight}\ x\ \mathsf{Then}\ P_2] \xrightarrow{\tau} \mathtt{wrong}} \quad [\textsc{Err-Case}]$$

A *Record* típus

A *Record* típus a *Pair* rendezett pár típus olyan általánosításának tekinthető, ahol a párt n-esre bővítjük, és az n-es minden eleméhez egy-egy címkét rendelünk.

Egy rekord mezőkből áll, a mezőknek neveik vannak, minden mező egy megadott típusú adatot tartalmaz.

Egy n-elemű rekord mezőnevei, azaz a mezők címkéi legyenek l_i-k, az l_i mező adatának a típusa pedig legyen T_i ($1 \leq i \leq n$). Ekkor a rekord típusa legyen

$$\{\!|\, l_1 : T_1, \ldots, l_n : T_n \,|\!\} \,.$$

Ha a $P : \{\!|\, l_1 : T_1, \ldots, l_n : T_n \,|\!\}$ rekord mezőit a $Q_1 : T_1, \ldots, Q_n : T_n$ kifejezésekkel töltjük fel, akkor a rekordot az

$$\{l_1 = Q_1, \ldots, l_n = Q_n\}$$

kifejezéssel írjuk le.

A rekord és a *Record* típus kezeléséhez az F_π típusrendszer szintaxisát a következőkkel bővítjük:

$$
\begin{array}{lcl}
\langle \textit{típus} \rangle & ::= & \ldots \\
 & | & \{\!\!\{ l_1 : T_1, \ldots, l_n : T_n \}\!\!\} \\
\langle \textit{érték} \rangle & ::= & \ldots \\
 & | & \{l_1 = Q_1, \ldots, l_n = Q_n\} \\
\langle \textit{folyamat} \rangle & ::= & \ldots \\
 & | & \mathsf{With}\ \{l_1 = (x_1), \ldots, l_n = (x_n)\} = y\ \mathsf{Do}\ P
\end{array}
$$

A With szerkezetben az x_i $(1 \leq i \leq n)$ nevek kötöttek, és hatáskörük a P folyamat.

A *Record* típusra vonatkozó típusszabályok a következők:

$$
\frac{\Gamma \vdash x_1 : T_1 \quad \ldots \quad \Gamma \vdash x_n : T_n}{\Gamma \vdash \{l_1 = (x_1), \ldots, l_n = (x_n)\} : \{\!\!\{ l_1 : T_1, \ldots, l_n : T_n \}\!\!\}} \quad [\textsc{Type-Rec}]
$$

$$
\frac{\Gamma \vdash y : \{\!\!\{ l_1 : T_1, \ldots, l_n : T_n \}\!\!\} \qquad \Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash P : wf}{\Gamma \vdash \mathsf{With}\ \{l_1 = (x_1), \ldots, l_n = (x_n)\} = y\ \mathsf{Do}\ P : wf}
$$

[Type-With-Rec]

A rekordra vonatkozó két átmeneti szabály:

$$
\frac{}{\begin{array}{c}\mathsf{With}\ \{l_1 = (x_1), \ldots, l_n = (x_n)\} = \{l_1 = y_1, \ldots, l_n = y_n\}\ \mathsf{Do}\ P \\ \xrightarrow{\tau} P[x_1 := y_1, \ldots, x_n := y_n]\end{array}} \quad [\textsc{Record}]
$$

$$
\frac{y \neq \{l_1 = y_1, \ldots, l_n = y_n\}}{\mathsf{With}\ \{l_1 = (x_1), \ldots, l_n = (x_n)\} = y\ \mathsf{Do}\ P \xrightarrow{\tau} \mathtt{wrong}} \quad [\textsc{Err-Record}]
$$

5.4. A lineáris típusrendszer

Az 5.2.4. pontban volt szó a típuskörnyezet három strukturális szabályáról (5.2.10.-5.2.12. tételek). Ha csak a *permutáció* tétel teljesülését követeljük meg, akkor a típusrendszert *lineáris típusrendszernek* nevezzük.

Ha a típusrendszerre nincs *gyengítés* és *szűkítés* szabály, akkor a rendszer típusainak értékei olyan adatszerkezetek lesznek, amelyek *pontosan*

típusrendszer	PERMUTÁCIÓ	GYENGÍTÉS	SZŰKÍTÉS	a változók használata
F_π	✓	✓	✓	nincs megkötés
lineáris	✓	×	×	pontosan egyszer

5.4. táblázat. Típusrendszerek

egyszer érhetők el, azaz úgy tekinthető, hogy az adatnak a használata után azonnal deallokálódnia kell. A *gyengítés* szabály használatának tiltása azt is biztosítja, hogy egy kifejezés típusának meghatározásához *a típuskörnyezet összes adatát* fel kell használni.

Ezekben a rendszerekben

- nincs deallokáció olyan adatszerkezetre, amelynek a felhasználása még nem történt meg,
- nincs adatfelülírás,
- nincs adatduplikáció, minden adatra pontosan egy pointer mutat,
- nem kell lusta stratégia, mert nem fordulhat elő, hogy egy számítást nem hajtunk végre,
- nincs szükség szemétgyűjtésre (garbage collection-re), hiszen a használat után az adat automatikusan deallokálódik.

A memóriakezelés az ilyen típusú adatokra biztonságos lesz, egy adat módosítása biztosan helyesen fog végrehajtódni: a duplikáció tiltása biztosítja azt, hogy egy adatnak pontosan egy példánya van; és egy adat törlésére, deallokációjára vonatkozó korlátozás pedig azt garantálja, hogy nem törlünk olyan adatot, amelyet a későbbiekben még használni szeretnénk [9].

A következőkben a *Lin*π lineáris típusrendszerrel foglalkozunk [48, 17]. Az alaprendszer a 2. fejezetben tárgyalt pi-kalkulus, kissé módosítva, és erre építjük rá a lineáris típusok rendszerét.

5.4.1. Szintaxis és műveleti szemantika

Az egyszerűség és a könnyű kezelhetőség kedvéért ebben a kalkulusban nem foglalkozunk az azonosság prefixekkel, a folyamatok közül az összegkifejezésekkel, de használjuk a True és False konstansokat, valamint az If-Then-Else kifejezést.

5.4.1. Definíció. A Linπ típusrendszer kifejezései:

A Linπ típusrendszer prefixei a következők:

$\pi ::= \overline{x}y \mid x(y) \mid \tau$.

A konstansok:

$v ::= \mathsf{True} \mid \mathsf{False}$,

és a folyamatok:

$P ::= \mathbf{0} \mid \pi . P \mid P \mid P \mid (\nu x) P \mid !P \mid \mathsf{If}\ v\ \mathsf{Then}\ P\ \mathsf{Else}\ P$.

A szerkezeti kongruencia szabályait az 5.5. táblázatban mutatjuk be, lényeges változás a korábbi rendszerekhez viszonyítva nincs.

$$
\begin{array}{rcl}
P \mid Q & \equiv & Q \mid P \\
P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R \\
P \mid \mathbf{0} & \equiv & P \\
!P & \equiv & P \mid !P \\
(\nu x)(\nu y)\, P & \equiv & (\nu y)(\nu x)\, P \\
(\nu x)\, P & \equiv & P,\ ha\ x \notin fn(P) \\
(\nu x)\, P \mid Q & \equiv & (\nu x)\,(P \mid Q),\ ha\ x \notin fn(Q) \\
(\nu x)\, \mathbf{0} & \equiv & \mathbf{0}
\end{array}
$$

5.5. táblázat. A szerkezeti kongruencia szabályai

A műveleti szemantika bővült az If-Then-Else kifejezés szokásos működésének leírásával, és a kommunikáció művelete a késői szemantika szerint működik. A *Linπ* típusrendszerben használt szabályok az 5.6. táblázatban láthatók.

5.4.2. Típusszabályok

Az F_π típusrendszerben egy adatátviteli csatorna (név) típusát a csatornán áthaladó adat határozta meg. Ebben a típusrendszerben tovább finomítjuk ezt az elvet, a csatorna két *végpontjához* rendelünk egy-egy típust, és a csatorna típusát az ezekből képzett párral adjuk meg. Ha S a végpontokhoz rendelhető típusok halmaza, akkor egy *csatorna típusát* egy (S_1, S_2) párral jelöljük, ahol

$$\frac{P' \equiv P, \quad P \xrightarrow{\alpha} Q, \quad Q \equiv Q'}{P' \xrightarrow{\alpha} Q'} \quad [\textsc{Struct}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad bn(\alpha) \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad [\textsc{Par}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad x \notin n(\alpha)}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'} \quad [\textsc{Res}]$$

$$\frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \quad [\textsc{Rep}]$$

$$\frac{P \xrightarrow{\overline{x}y} P', \quad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'[z := y]} \quad [\textsc{Com}]$$

$$\frac{}{\mathsf{If\ True\ Then}\ P\ \mathsf{Else}\ Q \to P} \quad [\textsc{L-If-True}]$$

$$\frac{}{\mathsf{If\ False\ Then}\ P\ \mathsf{Else}\ Q \to Q} \quad [\textsc{L-If-False}]$$

5.6. táblázat. A műveleti szemantika szabályai

S_1 az egyik, S_2 a másik végpont típusa. Az angol szakirodalomban ez a *session typing*, amit talán, utalva a csatornákon történő információ-továbbításra, magyarul „munkamenet” típusozásnak lehet nevezni.

A végpontok típusai *minősített típusok*, két komponensből állnak, az első tag egy lin vagy unlin minősítő, és a második tagban kell megadni, hogy az adott végponton input vagy output történik, milyen az input vagy output adat típusa, és ha az átvitel lezajlott, milyen típussal történik a folytatás.

A csatornavégpont típusának minősítése határozza meg, hogy ezt a pontot hányszor lehet használni. A lin minősítés azt jelenti, hogy pontosan egyszer, az unlin pedig azt, hogy akárhányszor, és ebbe a nulla, azaz az *egyszer sem* is beleértendő.

Ebben a szakaszban az output típus jele a „!”, az input típus jele a „?”, ezek a jelek a típus előtt állnak, így a „!” nem téveszthető össze a folyamatokra vonatkozó replikáció jelével.

Ha a csatorna végpontjának típusa end, akkor ez azt jelenti, hogy ezután a csatornavégponton már semmilyen adatforgalom nem lehet.

A típus lehet *alaptípus* is, ebben a szakaszban alaptípusnak az egyszerűség kedvéért, és mivel már bevezettük a True és False értékeket, a *Bool* típust választjuk.

A típus pontos leírását a következő definícióban adjuk meg.

5.4.2. Definíció. A Linπ típusrendszer típusai:

⟨*típus*⟩	::=	⟨*csatornatípus*⟩
	|	*Bool*
⟨*csatornatípus*⟩	::=	(⟨*végponttípus*⟩ , ⟨*végponttípus*⟩)
⟨*végponttípus*⟩	::=	⟨*minősítő*⟩ ⟨*elő-utó-típus*⟩
	|	*end*
⟨*minősítő*⟩	::=	*lin*
	|	*unlin*
⟨*elő-utó-típus*⟩	::=	? ⟨*típus*⟩ . ⟨*végponttípus*⟩
	|	! ⟨*típus*⟩ . ⟨*végponttípus*⟩

A rövid leírás érdekében a továbbiakban a típust T-vel, a végpont típusát S-sel, a minősítőt q-val és az elő-utó-típust p-vel fogjuk jelölni. Tehát egy csatorna típusa például

$(q?T\,.\,S\ ,\ q!T\,.\,S)$

alakban írható fel. Ezekkel a jelölésekkel a fenti definícióban használt fogalmak röviden így írhatók le:

T ::= $(S\ ,\ S)\ |\ Bool$,

S ::= $q\ p\ |$ end ,

q ::= lin | unlin ,

p ::= $?T\,.\,S\ |\ !T\,.\,S$

5.4.3. Példa. *(Típusok)*

Csatorna végpontok típusai és egy csatornatípus:

lin ? *Bool* . lin ? *Bool* . end ,

T_1 = unlin ? *Bool* . T_1 ,

T_2 = unlin ! *Bool* . T_2 , (unlin ? *Bool* . T_1 , unlin ! *Bool* . T_2) . □

Egy csatorna két végpontja közötti megfeleltetést a *dualitás* fogalom bevezetésével tudjuk leírni.

A definíció nagyon egyszerű, az input duálisa az output, és fordítva, az output duálisa az input, az end duálisa önmaga.

A duálist „felülvonás”-sal jelöljük, ez nem téveszthető össze az előző szakaszokban használt output jelzéssel, mivel lineáris típusrendszerek leírásában, azaz ebben a szakaszban az output jelzésére a „!” jelet használjuk.

5.4.4. Definíció. A duális:

A Linπ típusrendszerben a duálist a következőképpen képezzük:

$$\begin{array}{lcl} \overline{q?T\,.\,S} & = & q!T\,.\,\overline{S}\ , \\ \overline{q!T\,.\,S} & = & q?T\,.\,\overline{S}\ , \\ \overline{\mathsf{end}} & = & \mathsf{end}\,. \end{array}$$

A típuskörnyezetet a szokásos módon definiáljuk:

5.4.5. Definíció. A típuskörnyezet szintaktikája:

$$\begin{array}{lcl} \langle \textit{típuskörnyezet} \rangle & ::= & \emptyset \\ & \mid & \langle \textit{típuskörnyezet} \rangle\,,\ \langle \textit{név} \rangle : \langle \textit{típus} \rangle \end{array}$$

Természetesen most is megköveteljük azt, hogy ha x_i egy T_i típusú név $(1 \leq i \leq n)$ és a típuskörnyezet az $x_i : T_i$ párokból áll, akkor $x_i \neq x_j \ \ (i \neq j)$.

Bevezetünk egy új predikátumot, amit *un*-nel jelölünk, és először a típusokra és a végponttípusokra értelmezünk.

Az *un* predikátum majd arra szolgál, hogy a nemlineáris egységeket könnyen megkülönböztethessük a lineárisaktól.

5.4.6. Definíció. Az un predikátum:

Az un predikátum True *értéket ad a következő típusokra és csatornatípusokra:*

- $un(Bool)$,
- $un(\,(S_1\,,\,S_2)\,)$, *ha* $un(S_1)$ *és* $un(S_2)$,
- $un(\mathsf{end})$,
- $un(\mathsf{unlin}\ p)$.

5.4.7. Definíció. A lineáris típus és végponttípus:

- *Azt mondjuk, hogy a T típus nem korlátozott, ha $un(T) = \mathsf{True}$,*
- *az S végponttípus nem korlátozott, ha $un(S) = \mathsf{True}$.*

A korlátozott típusokat lineáris típusoknak nevezzük.

Az S végponttípus tehát lineáris, ha $\mathsf{lin}\ p$ alakú.

A lineáris típusú adatokat pontosan egyszer lehet és kell használni, ezért a felhasználás helyessége érdekében célszerű az adatokat két részre bontani, úgy, hogy az egyik részbe csak a lineáris adatok kerüljenek. Ezt a műveletet *hasításnak* nevezzük, és a $\circ$ jellel jelöljük.

5.4.8. Definíció. Típus és végponttípus hasítása:

Típusra a hasítás művelete a következőképpen adható meg:

- $Bool = Bool \circ Bool$,
- *ha* $R = R_1 \circ R_2$ *és* $S = S_1 \circ S_2$, *akkor* $(R, S) = (R_1, S_1) \circ (R_2, S_2)$, *ahol* R_1, R_2, S_1, S_2 *végponttípusok,*

és ha S végponttípus:

- $S = S \circ \mathsf{end}$,
- $S = \mathsf{end} \circ S$,
- $\mathsf{unlin}\ p = \mathsf{unlin}\ p \circ \mathsf{unlin}\ p$.

A *Bool* típus tehát a hasítás mindkét tagjába belekerül, a csatornatípusok a csatorna két végpontja szerint hasíthatóak, és a hasítással a nemlineáris végponttípusok is duplikálódnak. Látható, hogy a lineáris típusok nem duplikálódnak és nem is törlődnek.

5.4.9. Példa. *(Típusok hasítása)*

Legyen egy csatornatípus (U_1, U_2), ahol mindkét végtípus nem korlátozott, azaz nemlineáris típus. Ekkor

$U_1 = U_1 \circ U_1$ és $U_2 = U_2 \circ U_2$,

így

$(U_1, U_2) = (U_1, U_2) \circ (U_1, U_2)$.

Mivel $U_1 = U_1 \circ \mathsf{end}$,

$(U_1, U_2) = (U_1, U_2) \circ (U_1, \mathsf{end})$,

sőt mivel $U_1 = \mathsf{end} \circ U_1$,

$(U_1\,,\,U_2) = (\mathsf{end}\,,\,U_2) \circ (U_1\,,\,U_2)$

is fennáll. Ha L_1 és L_2 lineáris végtípusok, akkor

$(L_1\,,\,L_2) = (\mathsf{end}\,,\,\mathsf{end}) \circ (L_1\,,\,L_2)$,

$(L_1\,,\,L_2) = (L_1\,,\,\mathsf{end}) \circ (\mathsf{end}\,,\,L_2)$,

de például

$(L_1\,,\,L_2) \neq (L_1\,,\,\mathsf{end}) \circ (\mathsf{end}\,,\,\mathsf{end})$,

és ha $T \neq (\mathsf{end}\,,\,\mathsf{end})$, akkor $T \neq (\mathsf{end}\,,\,\mathsf{end}) \circ (\mathsf{end}\,,\,\mathsf{end})$.

Az $(\mathsf{end}\,,\,\mathsf{end})$ csatornatípus egyébként a típus teljes megszűnését jelenti. □

Az, hogy egy típuskörnyezetben levő név, azaz csatorna típusa milyen minősítésű, a típuskörnyezetben is jelölve van. Ezért a típuskörnyezetre is definiáljuk a hasítás műveletét.

Ha a Γ típuskörnyezet hasításával a Γ_1 és Γ_2 típuskörnyezetek keletkeznek, akkor ezt a

$\Gamma = \Gamma_1 \circ \Gamma_2$

egyenlettel írjuk le.

5.4.10. Definíció. A típuskörnyezet hasítása:

A hasítás szabályai a következők:

$$\frac{}{\emptyset = \emptyset \circ \emptyset}$$

$$\frac{\Gamma = \Gamma_1 \circ \Gamma_2 \qquad T = T_1 \circ T_2}{\Gamma, x\!:\!T = (\Gamma_1, x\!:\!T_1) \circ (\Gamma_2, x\!:\!T_2)}$$

$$\frac{\Gamma = \Gamma_1 \circ \Gamma_2 \qquad T = T_2 \circ T_1}{\Gamma, x\!:\!T = (\Gamma_1, x\!:\!T_1) \circ (\Gamma_2, x\!:\!T_2)}$$

Szükség lesz arra is, hogy a típuskörnyezetben levő típusinformációt megváltoztassuk, a típust egy új információval bővítsük.

A bővítést a + jellel jelöljük, és erre a műveletre a következő szabályt adhatjuk meg:

$(\Gamma, x\!:\!T_1) + (x\!:\!T_2) = (\Gamma, x\!:\!T_1 \circ T_2)$.

5.4.11. Példa. *(A változó típusának bővítése a típuskörnyezetben)*

Jelöljön most is L egy lineáris, és U egy nemlineáris típust. A típuskörnyezetben levő x változó típusát a bővítés után a végponttípusok hasítására vonatkozó szabályok határozzák meg. Például:

$$(\Gamma, x:(\mathsf{end}\,,\ U)) + x:(L\,,\ U) = (\Gamma, x:(L\,,\ U))\ ,$$
$$(\Gamma, x:(L_1\,,\ \mathsf{end})) + x:(\mathsf{end}\,,\ L_2) = (\Gamma, x:(L_1\,,\ L_2))\ ,$$
$$(\Gamma, x:(\mathsf{end}\,,\ \mathsf{end})) + x:(\mathsf{end}\,,\ \mathsf{end}) = (\Gamma, x:(\mathsf{end}\,,\ \mathsf{end}))\ .$$ □

Ezek után már megadhatjuk a *Linπ*-típusrendszer konstansaira és folyamataira vonatkozó típusszabályokat.

5.4.12. Definíció. A Linπ típusrendszer típusszabályai:

$$\frac{un(\Gamma)}{\Gamma \vdash \mathbf{0} : wf} \quad [\text{LTYPE-}\mathbf{0}]$$

$$\frac{un(\Gamma)}{\Gamma, x:T \vdash x:T} \quad [\text{LTYPE-VAR}]$$

$$\frac{un(\Gamma)}{\Gamma \vdash \mathsf{true} : Bool} \quad [\text{LTYPE-TRUE}]$$

$$\frac{un(\Gamma)}{\Gamma \vdash \mathsf{false} : Bool} \quad [\text{LTYPE-FALSE}]$$

$$\frac{\Gamma_1 \vdash v : Bool \quad \Gamma_2 \vdash P_1 \quad \Gamma_2 \vdash P_2}{\Gamma_1 \circ \Gamma_2 \vdash \mathsf{if}\ v\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2} \quad [\text{LTYPE-IF}]$$

$$\frac{\begin{array}{c}\Gamma_1 \vdash x:(q!T\,.\,S\,,\ S') \quad \Gamma_2 \vdash v:T \quad \Gamma_3 + x:(S\,,\ S') \vdash P \\ ha\ q \equiv \mathsf{unlin},\ akkor\ q!T\,.\,S = S\end{array}}{\Gamma_1 \circ \Gamma_2 \circ \Gamma_3 \vdash \overline{x}v\,.\,P : wf} \quad [\text{LTYPE-OUT-L}]$$

$$\frac{\begin{array}{c}\Gamma_1 \vdash x:(S'\,,\ q!T\,.\,S) \quad \Gamma_2 \vdash v:T \quad \Gamma_3 + x:(S'\,,\ S) \vdash P \\ ha\ q \equiv \mathsf{unlin},\ akkor\ q!T\,.\,S = S\end{array}}{\Gamma_1 \circ \Gamma_2 \circ \Gamma_3 \vdash \overline{x}v\,.\,P : wf} \quad [\text{LTYPE-OUT-R}]$$

$$\frac{\begin{array}{c}\Gamma_1 \vdash x:(q?T\,.\,S\,,\ S') \quad (\Gamma_2 + x:(S\,,\ S')), y:T \vdash P \\ ha\ q \equiv \mathsf{unlin},\ akkor\ q?T\,.\,S = S\end{array}}{\Gamma_1 \circ \Gamma_2 \vdash x(y)\,.\,P : wf} \quad [\text{LTYPE-IN-L}]$$

$$\frac{\Gamma_1 \vdash x : (S' \,,\, q?T \,.\, S) \quad (\Gamma_2 + x : (S' \,,\, S)), y : T \vdash P \qquad ha\ q \equiv \mathsf{unlin},\ akkor\ q?T \,.\, S = S}{\Gamma_1 \circ \Gamma_2 \vdash x(y) \,.\, P : wf} \ [\text{LType-In-R}]$$

$$\frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_2}{\Gamma_1 \circ \Gamma_2 \vdash P_1 \mid P_2 : wf} \ [\text{LType-Par}]$$

$$\frac{\Gamma, x : (S \,,\, \overline{S}) \vdash P}{\Gamma \vdash (\nu x)\, P : wf} \ [\text{LType-Res}]$$

$$\frac{\Gamma \vdash P \qquad un(\Gamma)}{\Gamma \vdash !\, P : wf} \ [\text{LType-Rep}]$$

Az első négy szabályhoz nem kell sok magyarázatot fűzni, csupán annyit, hogy a szabályokban az az információ is benne van, hogy a **0**, azaz a befejezett, álló folyamat, a logikai konstansok és az x név a nemlineáris típuskörnyezetből típusozható, a lineáris csatornák ezek típusára nincsenek hatással.

Az LType-If szabály szerint az If-Then-Else kifejezés feltételének típusozása a Γ_1-ből, a két ág kifejezésének típusozása a Γ_2-ből történik. Így az If típusozásához a $\Gamma_1 \circ \Gamma_2$ típuskörnyezetet kell használni, és ellentétben például a típusos lambda-kalkulussal [9], nem kell a két ág típusának azonosnak lennie.

Az input prefixre is kettő szabály van, a szabályok abban különböznek, hogy az input művelet x alanya a csatorna melyik végpontján van. Az input típuskörnyezete két részből áll, az elsőből az x típusa, a másodikból az inputot követő P folyamat típusa vezethető le. Az x típusa $(q?T \,.\, S \,,\, S')$ alakú, amiből azt tudjuk, hogy az y típusa T, és ezt az információt felhasználjuk a P típusának meghatározásakor. A $(q?T \,.\, S \,,\, S')$ típusból azt is látjuk, hogy az input végrehajtása után az x típusa $(S \,,\, S')$ lesz, azaz az $x(y) \,.\, P$ folyamatban az x típusa $q?T \,.\, S$, de a P-ben az x típusa megváltozik, S lesz. A szabályból azt is látjuk, hogy nemlineáris típusra az x típusa egyszerűen $(S \,,\, S')$.

Az output prefixekre vonatkozó szabályoknál a típuskörnyezetet három részre osztottuk, az elsőből az output alanya, a másodikból az output tárgya, a harmadikból az outputot követő folyamat típusa vezethető le. Hasonlóan az inputhoz, az $\overline{x}v \,.\, P$ alatt az x típusa $(q!T \,.\, S \,,\, S')$, de az output elvégzése után a típusa $(S \,,\, S')$-re változik. Itt is elmondhatjuk azt, hogy ha az x típusa nemlineáris, akkor már az output végrehajtása alatt is $(S \,,\, S')$ a típusa.

Az LType-Par szerint ha a párhuzamos végrehajtás egyik tagja a Γ_1-ből, a másik tagja Γ_2-ből típusozható, akkor a $P_1 \mid P_2$ folyamat típusozásához egy

olyan Γ kell, amelyből hasítással a Γ_1 és Γ_2 állítható elő.

A korlátozásra azt a megkötést kapjuk, hogy a korlátozott x név csatornatípusú, és olyan, hogy a csatorna végponttípusai egymás duálisai. A szabály szerint tehát a korlátozott csatornán az információ átáramolhat.

5.4.3. A *Linπ* típusrendszer tulajdonságai

Ebben a szakaszban azt vizsgáljuk, hogy a *Linπ* típusrendszerben a redukció vajon típustartó-e, érvényes-e Milnernek a helyes típososság és a helyes működés kapcsolatára tett állítása (*lásd* 5.2.15. tétel előtti megjegyzés).

5.4.13. Példa. *(A redukció típustartása)*

Legyen a vizsgálandó folyamat

$P \equiv x(z)\,.\,\mathsf{if}\ z\ \mathsf{then}\ \mathbf{0}\ \mathsf{else}\ \mathbf{0} \mid (\boldsymbol{\nu} y)\,\overline{x}y\ .$

Azonnal látszik, hogy P típusozható a

$\Gamma = \{x : (\mathsf{lin}!(\mathsf{end}\,,\ \mathsf{end})\,.\,\mathsf{end}\ ,\ \mathsf{lin}?\mathit{Bool}\,.\,\mathsf{end})$

típuskörnyezettel. Azonban

$P \xrightarrow{\tau} (\boldsymbol{\nu} y)\,\mathsf{if}\ y\ \mathsf{then}\ \mathbf{0}\ \mathsf{else}\ \mathbf{0}\ ,$

és a redukcióval kapott kifejezés egyáltalán nem típusozható, mivel a korlátozás helyes típusozására az Ltype-Res szabályban az van előírva, hogy y-ra az $y : (S\ ,\ \overline{S})$ tulajdonságnak kell teljesülnie. □

A példából is látszik, hogy a jól típusozottság nem elegendő, ezért bevezetjük a *kiegyensúlyozott típuskörnyezet* fogalmát, ami meg fogja oldani a típustartás problémáját.

5.4.14. Definíció. Kiegyensúlyozott típuskörnyezet:

A Γ típuskörnyezetet kiegyensúlyozottnak nevezzük, ha

$$\Gamma = \begin{cases} \emptyset\ , \\ \Gamma', x : \mathit{Bool}\ , \\ \Gamma', x : (S\ ,\ \mathsf{end})\ , \\ \Gamma', x : (\mathsf{end}\,,\ S)\ , \\ \Gamma', x : (S\ ,\ \overline{S})\ , \end{cases}$$

ahol Γ' egy kiegyensúlyozott környezet.

A kiegyensúlyozottság definíciójában azért kellett külön felsorolni a *Bool* típust, mert ennek a típusnak nincs duálisa.

Most felsorolunk néhány érdekes tételt.

5.4.15. Tétel. (Típuskörnyezet gyengítése)

- *Ha* $\Gamma \vdash v : T$, $x \neq v$ *és* $T' = \mathsf{unlinp}$ *vagy* $T' = (\mathsf{unlinp}_1, \mathsf{unlinp}_2)$, *akkor* $\Gamma, x : T' \vdash v : T$.
- *Ha* $\Gamma, v : S \vdash v : S$, *akkor* $\Gamma, v : (S, \mathsf{unlinp}) \vdash v : S$.
- *Ha* $\Gamma \vdash P$ *és* $T = \mathsf{unlinp}$ *vagy* $T = (\mathsf{unlinp}_1 , \mathsf{unlinp}_2)$, *akkor* $\Gamma, x : T \vdash P$.
- *Ha* $\Gamma, x : S \vdash P$, *akkor* $\Gamma, x : (S , \mathsf{unlinp}) \vdash P$.

5.4.16. Tétel. (Típuskörnyezet szűkítése)

Ha $\Gamma, x : T \vdash P$, $x \notin fn(P)$ *és* $T = \mathsf{unlinp}$ *vagy* $T = (\mathsf{unlinp}_1 , \mathsf{unlinp}_2)$, *akkor* $\Gamma \vdash P$.

5.4.17. Tétel. (A helyettesítés tétele)

Ha $\Gamma_1, x : T \vdash P$ *és* $\Gamma_2 \vdash v : T$, *akkor* $\Gamma_1 \circ \Gamma_2 \vdash P[x := v]$.

5.4.18. Tétel. (Típuskörnyezet vágása)

Legyen $T_1 = (q_1!T'_1 . S_1 , q_2?T'_2 . S_2)$. *Ha* $\Gamma = (\Gamma_1, x : T_1) \circ \Gamma_2$, *akkor* $x : T_1 \in \Gamma$ *és* $x : T_2 \in \Gamma_2$, *ahol* T_2 *az alábbi négy típus egyike:*

- $(\mathsf{end}, \mathsf{end})$,
- T_1, *ha* $q_1 = q_2 = \mathsf{un}$,
- $(q_1!T'_1 . S_1 , \mathsf{end})$, *ha* $q_1 = \mathsf{un}$,
- $(\mathsf{end} , q_2!T'_2 . S_2)$, *ha* $q_2 = \mathsf{un}$.

Egy negatív eredményeket adó tétel:

5.4.19. Tétel. ($\nvdash P$)

Legyen L egy lineáris végponttípus, és legyen $R = \mathsf{end}$ *vagy* $R = \overline{L}$. *Ekkor*

- *ha* $\Gamma, x : (L, S) \vdash P$, *akkor* $\Gamma, x : (\mathsf{end}, R) \nvdash P$,
- *ha* $\Gamma, x : (S, L) \vdash P$, *akkor* $\Gamma, x : (R, \mathsf{end}) \nvdash P$.

Végül két tétel, ezek már csak a kiegyensúlyozott típuskörnyezetekre érvényesek.

5.4.20. Tétel. (A szerkezeti kongruencia megtartása)

Ha a Γ kiegyensúlyozott típuskörnyezetre $\Gamma \vdash P$ és $P \equiv Q$, akkor $\Gamma \vdash Q$ is teljesül.

5.4.21. Tétel. (Tárgyredukció tétele)

Ha a Γ kiegyensúlyozott típuskörnyezetre $\Gamma \vdash P$ és $P \to P'$, akkor a következő két eset egyike fennáll:

- $\Gamma \vdash P'$,
- *van olyan $x \in dom(\Gamma)$, melyre $\Gamma = \Gamma', x : (q?T\,.\,S\ ,\ q!T\,.\,\overline{S})$ és $\Gamma', x : (S\ ,\ \overline{S}) \vdash P'$.*

A felsorolt tételek bizonyítása nagyon nehézkes és hosszú, az érdeklődő olvasó például [17, 48]-ben megtalálja a részletes leírásukat.

Ugyanitt megtalálható az is, hogy a lineáris lambda-kalkulus [9] kifejezései hogyan alakíthatók át a lineáris pi-kalkulus kifejezéseire. Ebből azonnal következik, mint ahogyan várható volt, hogy a lineáris lambda-kalkulus a lineáris pi-kalkulus alrendszerének tekinthető.

6. FEJEZET

A magasabb rendű pi-kalkulus

A pi-kalkulus egyik továbbfejlesztése a *magasabb rendű pi-kalkulus*, amelynek fő jellemzője, hogy a csatornákat reprezentáló neveken nem csak adatokat és neveket, hanem folyamatokat is lehet küldeni és fogadni.

6.0.22. Példa. *(A „végrehajtó" folyamat)*

Tegyük fel, hogy egy folyamat az x néven fogad egy folyamatot, és ezt a folyamatot azonnal végre is hajtja, ezért nevezhetjük ezt a folyamatot végrehajtó folyamatnak. A folyamat egyszerűen $x(X) \,.\, X$ alakban írható le, ahol X egy folyamatváltozó. Ha ezt a folyamatot párhuzamosan futtatjuk az $\overline{x}P \,.\, \mathbf{0}$ folyamattal, akkor megtörténik a kommunikáció:

$$\overline{x}P \,.\, \mathbf{0} \mid x(X) \,.\, X \xrightarrow{\tau} P$$

azaz a P folyamat végrehajtása következik. □

A magasabb rendű pi-kalkulus jele $HO\pi$, az elnevezés a *higher order* szavak kezdőbetűiből származik.

Először a magasabb rendű pi-kalkulus szintaktikáját és műveleti szemantikáját adjuk meg, majd röviden a biszimulációval foglalkozunk, végül megmutatjuk a pi-kalkulus, a magasabb rendű pi-kalkulus, és a lambda-kalkulus közötti kapcsolatot.

6.1. A $HO\pi$-kalkulus szintaxisa és műveleti szemantikája

A $HO\pi$-kalkulus alapjának a 2. fejezetben elemzett pi-kalkulust választjuk, és ezt a rendszert fogjuk bővíteni. A kalkulusunk az egyszerűség kedvéért

monadikus, de nyilvánvalóan nem okozna problémát a poliadikus kalkulusra való áttérés (2.5.1. pont). Ebben a fejezetben, szintén az egyszerűség kedvéért, nem foglalkozunk a $!P$ replikáció folyamattal sem.

Legyen a folyamatok halmaza $\mathcal{P}$, és bevezetünk egy új név-halmazt, a folyamatváltozók halmazát, amelyet $\mathcal{A}$-val jelölünk. A folyamatváltozók halmazának elemeit az $X, Y, \ldots$ betűk jelölik, a név-halmaz elemei továbbra is $x, y, \ldots$. Az egyszerű jelölés érdekében legyen

$$\mathcal{U} = \mathcal{A} \cup \mathcal{N} ,$$

$$\mathcal{K} = \mathcal{P} \cup \mathcal{N} ,$$

tehát $\mathcal{U}$ a változókat, $\mathcal{K}$ a változók által felvehető kifejezések halmazát jelöli. $\mathcal{K}$-t a szakirodalomban néha *értékeknek* is nevezik. Jelöljük U-val és K-val ezeknek a halmazoknak az elemeit, azaz legyen $U \in \mathcal{U}$, $K \in \mathcal{K}$.

6.1.1. Példa. *(Az U és K jelölés)*

A továbbiakban, mint majd a definíciókból látni fogjuk, U egy x, y nevet vagy egy X, Y folyamatváltozót jelöl, míg K egy x, y nevet vagy például az $\overline{x}y \,.\, P$, $x(U) \,.\, Q$ kifejezést, vagy akár a $\mathbf{0}$ kifejezést jelöli. □

6.1.2. Definíció. A HOπ-kalkulus prefixei:

A HOπ-kalkulusban a következő prefixeket használjuk:

$$\pi ::= \overline{x}K \;\mid\; x(U) \;\mid\; \tau \;\mid\; [x = y]\pi$$

Az első két prefix különbözik a pi-kalkulus prefixeitől, de csak annyiban, hogy itt az output és az input nem csak nevekre, hanem folyamatokra is vonatkozhat, azaz a műveletek tárgyai folyamatok is lehetnek.

6.1.3. Definíció. A HOπ-kalkulus folyamatai:

A HOπ-kalkulusban a folyamatokat a következő kifejezésekkel adjuk meg:

$$P ::= \textstyle\sum_{i \in I} \pi_i \,.\, P_i \;\mid\; P \mid P \;\mid\; (\boldsymbol{\nu} U)\, P \;\mid\; D\,K \;\mid\; X\,K$$

A $\sum_{i \in I} \pi_i \,.\, P_i$ kifejezésben az I halmazról feltesszük, hogy véges. Ha $I = \emptyset$, akkor ezt a kifejezést $\mathbf{0}$-val jelöljük.

Látható, hogy ebben a rendszerben is vannak prefixes folyamatok, összegkifejezés, kompozíció és korlátozás. Az input prefix és a korlátozás itt is kötést jelent, az $x(U) \,.\, P$ és a $(\boldsymbol{\nu} U)\, P$ kifejezésben az U kötött a P folyamatban, azaz az U *hatásköre* a P folyamat.

A definícióban a $D\,K$ és az $X\,K$ kifejezésekben a K „üres” is lehet, ekkor a

D név egy folyamatabsztrakció neve, X pedig egy folyamatváltozó, midkettő egy folyamatot reprezentál.

A folyamatabsztrakció pi-kalkulusbeli jelentéséről a 2.5.2. pontban már volt szó, de most részletesen kifejtjük, hogy ez mit is jelent a $HO\pi$-kalkulusban.

Az $(U)\,.\,P$ kifejezéssel a *folyamatabsztrakciót* jelöljük, ahol U az absztrakció változója, azaz formális paramétere. A definiáló egyenlőség

$$D \stackrel{\text{def}}{=\!=} (U)\,.\,P$$

alakú, ahol a folyamatabsztrakciónak a D nevet adjuk. Ekkor az absztrakció *példányosítása*

$$D\,K \equiv ((U)\,.\,P)K\ .$$

Az absztrakció tetszőlegesen magasszintű lehet, tehát megengedett például a

$$D' \stackrel{\text{def}}{=\!=} (U')\,.\,D \equiv (U')\,.\,((U)\,.\,P)$$

kifejezés is. Ennek példányosítása $((U')\,.\,D)K$.

6.1.4. Példa. *(A folyamatváltozó)*

Az X egy folyamatváltozó, és legyen például $X \equiv (U)\,.\,P$, azaz X egy $(U)\,.\,P$ folyamatabsztrakciót reprezentál. Ha erre az absztrakcióra egy K folyamatot applikálunk, akkor

$$X\,K \equiv ((U)\,.\,P)K\ ,$$

azaz a 6.1.3. definícióban szereplő $X\,K$ valóban egy folyamat kifejezése.

Még jobban látszik a folyamatváltozó szerepe a következő levezetésben. A kezdő kifejezés egy kompozíció, ahol a második tag törzse egy $X\,K$ típusú folyamatkifejezés.

$$\overline{x}\,\langle((Y)\,.\,P\rangle\,.\,Q' \ \mid\ x(X)\,.\,X\,Q'' \xrightarrow{\tau}$$
$$Q' \mid ((Y)\,.\,P)\,Q'' \qquad \square$$

A szabad és kötött nevekkel, folyamatváltozókkal, az α-konverzióval nem foglalkozunk, a már ismert definíciók, átalakítások könnyen értelmezhetők a $HO\pi$-kalkulusra. Megadjuk viszont a helyettesítést a prefixekre és a kifejezésekre, mert a műveletek egy része nem triviális:

6.1.5. Definíció. Helyettesítés:

Legyen $\sigma = [X := Y]$, *ekkor*

$$
\begin{array}{lcl}
\mathbf{0}\sigma & \equiv & \mathbf{0} \\
(X\,K)\sigma & \equiv & \begin{cases} \textit{ha } X \equiv (U)\,.\,P, \quad \textit{akkor } P[U := (K\sigma)]\,, \\ Y(K\sigma), \quad \textit{egyébként}, \end{cases} \\
(D\,K)\sigma & \equiv & D(K\sigma) \\
((U)\,.\,P)\sigma & \equiv & (U)\,.\,(P\sigma) \\
(\overline{x}K\,.\,P)\sigma & \equiv & \overline{x}(K\sigma)\,.\,P\sigma \\
(x(U)\,.\,P)\sigma & \equiv & x(U)\,.\,P\sigma \\
(P_1 + P_2)\sigma & \equiv & P_1\sigma + P_2\sigma \\
(P_1 \mid P_2)\sigma & \equiv & P_1\sigma \mid P_2\sigma \\
((\boldsymbol{\nu} x)\,P)\sigma & \equiv & (\boldsymbol{\nu} x)\,P\sigma \\
([x = y]P)\sigma & \equiv & [x = y](P\sigma)
\end{array}
$$

A pi-kalkulus *szerkezeti kongruenciájának* szabályai a 2.2. táblázatban szerepeltek. Ezek érvényesek a $HO\pi$-kalkulusra is, csak egy új szabállyal kell bővíteni ezt a táblázatot:

Ha $D \stackrel{\text{def}}{=\!=} (U)\,.\,P$, *akkor* $D\,K \equiv P[U := K]\,.$

Ez a szabály megadja azt, hogy egy folyamatabsztrakcióra applikálva egy K folyamatot vagy nevet, milyen művelet hajtódik végre.

A *műveleti szemantika* szabályai a 6.2. táblázatban láthatók. A korábbi kötött outputhoz hasonlóan, a folyamatváltozóra vonatkozó $(\boldsymbol{\nu} U)\overline{x}U$ kötött output prefixet az $\overline{x}(U)$ kifejezéssel rövidítjük.

6.2. A *HOπ*-kalkulus és a pi-kalkulus kapcsolata

A magasabb rendű pi-kalkulus nem nevezhető egy egyszerű rendszernek, használata bonyolult. Tegyük fel, hogy egy fizikai folyamatot a $HO\pi$-kalkulus kifejezéseivel tudtunk leírni. Nyilván kellemesebb lenne, ha ezt a pi-kalkulus kifejezéseivel is meg tudnánk tenni, vagy legalább át tudnánk alakítani a $HO\pi$-kalkulus kifejezését a pi-kalkulus kifejezésére. Ezzel a témakörrel foglalkozunk ebben a szakaszban [41, 42].

$$\frac{P' \equiv P, \quad P \xrightarrow{\alpha} Q, \quad Q \equiv Q'}{P' \xrightarrow{\alpha} Q'} \quad [\textsc{Struct}]$$

$$\frac{}{\overline{x}K \,.\, P \xrightarrow{\overline{x}K} P} \quad [\textsc{Output}] \qquad \frac{\overline{x}K \,.\, P \xrightarrow{\overline{x}K} P, \quad x \neq y}{(\nu y)\,\overline{x}K \,.\, P \xrightarrow{(\nu y)\,\overline{x}(K)} P} \quad [\textsc{Open}]$$

$$\frac{}{x(U) \,.\, P \xrightarrow{xK} P[U := K]} \quad [\textsc{Input}] \qquad \frac{}{\tau \,.\, P \xrightarrow{\tau} P} \quad [\textsc{Tau}]$$

$$\frac{\alpha \,.\, P \xrightarrow{\alpha} P}{[x = x]\alpha \,.\, P \xrightarrow{\alpha} P} \quad [\textsc{Match}]$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \quad [\textsc{Sum}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad bn(\alpha) \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad [\textsc{Par}]$$

$$\frac{P \xrightarrow{\overline{x}K} P', \quad Q \xrightarrow{x(U)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'[U := K]} \quad [\textsc{Com}]$$

$$\frac{P \xrightarrow{\alpha} P', \quad x \notin n(\alpha)}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'} \quad [\textsc{Res}]$$

$$\frac{P \xrightarrow{(\nu y)\overline{x}K} P', \quad Q \xrightarrow{xK} Q'}{P \mid Q \xrightarrow{\tau} (\nu y)(P' \mid Q')} \quad [\textsc{Close}]$$

6.2. táblázat. A műveleti szemantika szabályai

Először bevezetünk egy jelölést:

Ha a környezettől függően U egy név vagy folyamatváltozó, akkor legyen

$P\{m := F\} \equiv (\nu m)\,(P \mid !m(U) \,.\, F\ U)$.

Ennek a felhasználásával fogjuk majd megadni az átalakítás szabályait, az átalakítást most is a $[\![\ ]\!]$ zárójelekkel jelöljük.

6.2.1. Definíció. A $HO\pi$-kalkulus kifejezései a pi-kalkulusban:

$$
\begin{array}{lcl}
[\![\mathbf{0}]\!] & \stackrel{\text{def}}{=\!=} & \mathbf{0}\,, \\[2ex]
[\![\alpha\,.\,P]\!] & \stackrel{\text{def}}{=\!=} & \begin{cases} (\overline{x}m\,.\,[\![P]\!])\,\{m := [\![F]\!]\}\,, & \textit{ha } \alpha = \overline{x}F\,, \\ x(y)\,.\,[\![P]\!]\,, & \textit{ha } \alpha = x(Y)\,, \\ \alpha\,.\,[\![P]\!] & \textit{egyébként}, \end{cases} \\[2ex]
[\![[x = y]\,.\,P]\!] & \stackrel{\text{def}}{=\!=} & [x = y]\,.\,[\![P]\!]\,, \\[2ex]
[\![X]\!] & \stackrel{\text{def}}{=\!=} & \begin{cases} [\![(Y)\,.\,X(Y)]\!]\,, & \textit{ha } X \textit{ folyamatabsztrakció}, \\ [\![(y)\,.\,X\,y]\!]\,, & \textit{egyébként}\,, \end{cases} \\[2ex]
[\![P + Q]\!] & \stackrel{\text{def}}{=\!=} & [\![P]\!] + [\![Q]\!]\,, \\[2ex]
[\![Xy]\!] & \stackrel{\text{def}}{=\!=} & \overline{x}y\,.\,\mathbf{0}\,, \\[2ex]
[\![XP]\!] & \stackrel{\text{def}}{=\!=} & (\overline{x}m\,.\,\mathbf{0})\,\{m := [\![P]\!]\}\,, \\[2ex]
[\![(X)\,.\,P]\!] & \stackrel{\text{def}}{=\!=} & (x)\,.\,[\![P]\!]\,, \\[2ex]
[\![(x)\,.\,P]\!] & \stackrel{\text{def}}{=\!=} & (x)\,.\,[\![P]\!]\,, \\[2ex]
[\![P \mid Q]\!] & \stackrel{\text{def}}{=\!=} & [\![P]\!] \mid [\![Q]\!]\,, \\[2ex]
[\![(\nu x)\,P]\!] & \stackrel{\text{def}}{=\!=} & (\nu x)\,[\![P]\!]\,.
\end{array}
$$

6.2.2. Példa. *(Az $\{m := F\}$ jelentése)*

A $HO\pi$-kalkulusban az

$$\overline{x}F\,.\,Q \mid x(Y)\,.\,Yz \xrightarrow{\tau} Q \mid Fz$$

átmenet nyilvánvaló, az F-et azonnal tudjuk használni a z-vel való applikációban. Ha átalakítjuk a két párhuzamosan futó kifejezést a pi-kalkulus kifejezéseire, akkor elhagyva a replikációt a következőket kapjuk:

$$
\begin{array}{l}
[\![\overline{x}F\,.\,Q]\!] = \\
(\nu m)\,(\,(\overline{x}m\,.\,[\![Q]\!])\,\{m := [\![F]\!]\}\,) = \\
(\nu m)\,(\overline{x}m\,.\,[\![Q]\!] \mid m(u)\,.\,[\![F]\!]u)\,,
\end{array}
$$

és

$$
\begin{array}{l}
[\![x(Y)\,.\,Yz]\!] = \\
x(y)\,.\,[\![Yz]\!] =
\end{array}
$$

$x(y) \,.\, \overline{y}z \,.$

Így

$(\boldsymbol{\nu} m)\,(\overline{x}m \,.\, [\![Q]\!] \;\mid\; m(u) \,.\, [\![F]\!]u) \;\mid\; x(y) \,.\, \overline{y}z \equiv$

$(\boldsymbol{\nu} m)\,(\underline{\overline{x}m} \,.\, [\![Q]\!] \;\mid\; m(u) \,.\, [\![F]\!]u \;\mid\; \underline{x(y) \,.\, \overline{y}z}) \xrightarrow{\tau}$

$(\boldsymbol{\nu} m)\,([\![Q]\!] \;\mid\; \underline{m(u) \,.\, [\![F]\!]u} \;\mid\; \underline{\overline{m}z}) \xrightarrow{\tau}$

$(\boldsymbol{\nu} m)\,([\![Q]\!] \mid [\![F]\!]z) \equiv$

$[\![Q]\!] \mid [\![F]\!]z \,.$

Látható. hogy az $\{m := F\}$ kifejezésnek az a szerepe, hogy „elindítsa” a z-t az F-hez, ezért ezt a kifejezést *indítónak* is szokás nevezni. □

Végül megmutatjuk, hogy a lambda-kalkulus is a $HO\pi$-kalkulus alrendszerének tekinthető, a lambda-kalkulus kifejezései átalakíthatók a $HO\pi$-kalkulus kifejezéseivé. A definícióban és az utána következő példában az input és output műveletre poliadikus kifejezéseket használunk, a műveletek értelmezésével a 2.5.1.pontban foglalkoztunk.

6.2.3. Definíció. A név szerinti lambda-kalkulus kifejezései a HOπ-kalkulusban:

$$
\begin{array}{lcl}
[\![x]\!] & \stackrel{\text{def}}{=\!=} & X \\
[\![\lambda x \,.\, E]\!] & \stackrel{\text{def}}{=\!=} & (p) \,.\, p(X, q) \,.\, [\![E]\!]\,\langle q\rangle \\
[\![E\,F]\!] & \stackrel{\text{def}}{=\!=} & (p) \,.\, (\boldsymbol{\nu} q)\,(\;[\![E]\!]\,\langle q\rangle \;\mid\; \overline{q}\,\langle [\![F]\!], p\rangle\;) \,.
\end{array}
$$

6.2.4. Példa. *(A $(\lambda x \,.\, E)F$ kifejezés)*

$[\![(\lambda x \,.\, E)F]\!]\,\langle p\rangle =$

$(\boldsymbol{\nu} q)\,([\![\lambda x \,.\, E]\!]\,\langle q\rangle \mid \overline{q}\,\langle [\![F]\!], p\rangle) =$

$(\boldsymbol{\nu} q)\,(\underline{q(X, v) \,.\, [\![E]\!]\,\langle v\rangle} \;\mid\; \underline{\overline{q}\,\langle [\![F]\!], p\rangle}) \xrightarrow{\tau}$

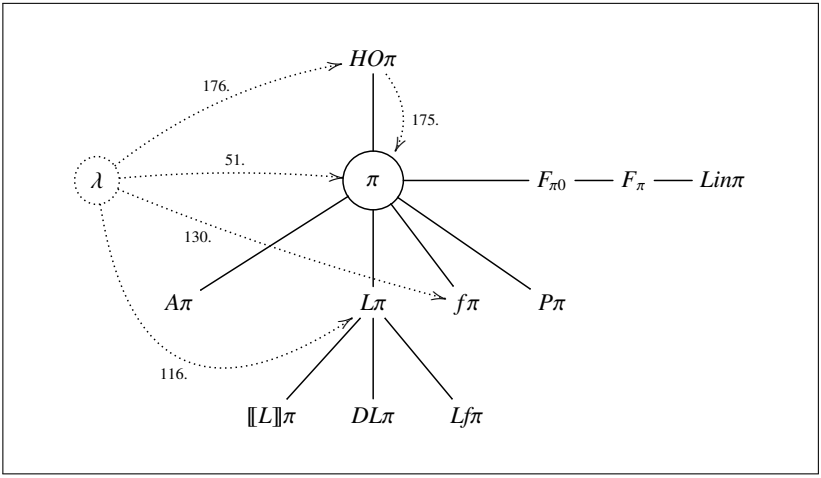
$(\boldsymbol{\nu} q)\,(\;([\![E]\!]\,\langle p\rangle)\,[X := [\![F]\!]]\;) \equiv$

$[\![E]\!]\,\langle p\rangle\,[X := [\![F]\!]] \equiv$

$[\![E]\!]\,[X := [\![F]\!]]\,\langle p\,[X := [\![F]\!]]\rangle \equiv$

$[\![E[x := F]]\!]\,\langle p\rangle \,.$ □

A következő 6.1. ábrán a különböző kalkulusok kapcsolatai és a közöttük végezhető konverziók láthatók, azok, amelyekkel a könyvben foglalkoztunk. A nyilakon levő számok oldalszámot jelentenek.



6.1. ábra. Kalkulusok és konverziók

A $HO\pi$-kalkulusban is lehet definiálni konstansokat, itt is megadhatunk különböző biszimulációkat, kongruenciákat, és vizsgálhatjuk ezek kapcsolatait. A típusos kalkulusokban láttuk, hogy a típusrendszer használata megvédi a felhasználót a folyamatkifejezések feldolgozásakor jelentkező kellemetlen hibajelenségektől, mert a hibák sokkal hamarabb kiderülnek. Ha átalakítjuk a $HO\pi$-kalkulust típusos kalkulusra, akkor erre a rendszerre is bebizonyíthatjuk a tárgyredukció és a típushelyesség tételét.

Ezek a témakörök nagyon érdekesek, de a jelenlegi eredmények, szakirodalmi adatok alapján úgy tűnik, hogy ezen területek vizsgálata már új elveket, új módszereket nem ad, lényegében a korábban ismertetett kalkulusokban alkalmazott eljárások kisebb-nagyobb módosításokkal átvihetők és alkalmazhatók a $HO\pi$-kalkulusra is.

* * *

A pi-kalkulus elmélete természetesen folyamatosan fejlődik, az új irányzatok nem a magasabb absztrakciós szintekre, hanem egy-egy speciális tématerület felé haladnak, ilyen például a *molekuláris biológia*, a *kriptográfia*, vagy az *üzleti folyamatok* vizsgálata, de ezekkel a témakörökkel most itt nem foglalkozunk.

Irodalomjegyzék

[1] Alexandru, Andrei – Ciobanu, Gabriel: Nominal techniques for πi-calculus. *Romanian Journal of Information Science and Technology*, 16. évf. (2013) 4. sz., 261–286. p.

[2] Amadio, Roberto – Castellani, Ilaria – Sangiorgi, Davide: On bisimulations for the asynchronous pi-calculus. *Theoretical Computer Science*, 195. évf. (1998) 2. sz., 291–324. p.

[3] Bonchi, Filippo – Montanari, Ugo: Symbolic semantics revisited. In *Lecture Notes in Computer Science*. 4962. köt. 2008, Springer-Verlag, 395–412. p.

[4] Boreale, Michele: On the expressiveness of internal mobility in name-passing calculi. In *Theoretical Computer Science*. 195/2. köt. 1998, Springer-Verlag, 205–226. p.

[5] Boudol, Gérard: Asynchrony and the π-calculus. INRIA Research Report 1702. Jelentés, 1992, INRIA Sophia-Antipolis.

[6] Chen, Taolue – Han, Tingting – Lu, Jian: On the bisimulation congruence in χ-calculus. In *Lecture Notes in Computer Science*. 3821. köt. 2005, Springer-Verlag, 128–139. p.

[7] Cleaveland, Rance – Yankelevich, Daniel: An operational framework for value-passing processes. In *Proceedings of the 21th ACM Symposium on Principles of Programming Languages* (konferenciaanyag). 1994, 326–338. p.

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

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

[10] Engberg, Uffe – Nielsen, Mogens: A calculus of communicating systems with label-passing. DAIMI PB-208. Jelentés, 1986, Computer Science Department, University of Aarhus.

[11] Fokking, Wan: *Introduction to Process Algebra*. 2007, Springer-Verlag.

[12] Frendrup, Ulrik – Jensen, Jesper Nyholm: Checking for open bisimilarity in the π-calculus. BRICS Report Series. Jelentés, 2001, Computer Science Department, University of Aarhus.

[13] Fu, Yuxi: The χ-calculus. In *Proceedings of the 1997 International Conference on Advances in Parallel and Distributed Computing* (konferenciaanyag). 1997, 74–81. p.

[14] Fu, Yuxi: A proof theoretical approach to communication. In *Lecture Notes in Computer Science*. 1256. köt. 1997, Springer-Verlag, 325–335. p.

[15] Gardner, Philippa: Models of concurrent computation. https://www.doc.ic.ac.uk/˜pg/Concurrency/course.html, 2016.

[16] Gardner, Philippa – Laneve, Cosimo – Wischik, Lucian: Linear forwarders. In *Lecture Notes in Computer Science*. 2761. köt. 2003, Springer-Verlag, 415–430. p.

[17] Giunti, Marco – Vasconcelos, Vasco Thudichum: Linearity, session types and the pi calculus. *Mathematical Structures in Computer Science*, 26. évf. (2014) 2. sz., 206–237. p.

[18] Hennessey, Matthew: *A Distributed Pi-Calculus*. 2007, Cambridge University Press.

[19] Hepburn, Mark – Wright, David: Execution contexts for determining trust in a higher-order π-calculus. TR-01-2003. Jelentés, 2003, School of Computing, University of Tasmania.

[20] Hoare, C. A. R.: The emperor's old clothes. *Communications of the ACM*, 24. évf. (1981) 2. sz., 75–83. p.

[21] Hoare, C. A. R.: *Communicating Sequential Processes*. 1985, Prentice Hall.

[22] Honda, Kohei – Tokoro, Mario: An object calculus for asynchronous communications. In *Lecture Notes in Computer Science*. 512. köt. 1991, Springer-Verlag, 133–147. p.

[23] Katoen, Joost-Pieter: Probalistic models for concurrency. http://www-i2.informatik.rwth-aachen.de/Teaching/Course/PMC, 2005.

[24] Kobayashi, Naoki – Pierce, Benjamin C. – Turner, David N.: Linearity and the pi-calculus. *ACM Transactions on Programming Languages and Systems*, 21. évf. (1999) 5. sz., 914–947. p.

[25] Merro, Massimo: *Locality in the π-calculus and applications to distributed objects*. Phd értekezés (Ecole des Mines de Paris). 2000.

[26] Merro, Massimo – Sangiorgi, Davide: On asynchrony in name-passing calculi. In *Lecture Notes in Computer Science*. 1443. köt. 1998, Springer-Verlag, 856–867. p.

[27] Milner, Robin: A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17. évf. (1978), 348–375. p.

[28] Milner, Robin: Calculus of communicating systems. In *Lecture Notes in Computer Science*. 92. köt. 1980, Springer-Verlag, 1–169. p.

[29] Milner, Robin: *Communication and Concurrency*. 1989, Prentice Hall.

[30] Milner, Robin: The polyadic π-calculus: a tutorial. ECS-LFCS-91-180. Jelentés, 1991, Department of Computer Science, The University of Edinburgh.

[31] Milner, Robin: Functions as processes. *Mathematical Structures in Computer Science*, 2. évf. (1992), 119–141. p.

[32] Milner, Robin: *Communicating and Mobile Systems: the Pi-Calculus*. 1999, Cambridge University Press.

[33] Milner, Robin – Parrow, Joachim – Walker, David: A calculus of mobile processes, Part I–II. *Journal of Information and Computation*, 100. évf. (1992), 1–77. p.

[34] Monroy, Raúl: Introduction to communication and concurrency. http://homepage.cem.itesm.mx/raulm/teaching/cc/notes/, 2016.

[35] Ostheimer, Gerard K. – Davie, Antony J. T.: π-calculus characterisations of some practical λ-calculus reduction strategies. Technical reports CS/93/14. Jelentés, 1993, Department of Mathematical and Computational Sciences, University of St. Andrews.

[36] Parrow, Joachim: An introduction to π-calculus. http://user.it.uu.se/~joachim/intro.ps, 2016.

[37] Parrow, Joachim – Victor, Björn: The fusion calculus: expressiveness and symmetry in mobile processes. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science* (konferenciaanyag). 1998, 176–185. p.

[38] Pierce, Benjamin C.: Foundational calculi for programming languages. http://www.cis.upenn.edu/~bcpierce/papers/crchandbook.ps, 1995.

[39] Quaglia, Paola: The π-calculus: Notes on labelled semantics. BRICS Report Series. Jelentés, 1994, Computer Science Department, University of Aarhus.

[40] Rickmann, Christina – Wagner, Christoph – Nestmann, Uwe – Schmid, Stefan: Topological self-stabilization with name-passing process calculi. In *27th International Conference on Concurrency Theory (CONCUR 2016)* (konferenciaanyag), 59. köt. 2016, 19:1–19:15. p.

[41] Sangiorgi, Davide: Expressing mobility in process algebra: First-order and higher-order paradigms. ECS-LFCS-93-266. Jelentés, 1993, Department of Computer Science, The University of Edinburgh.

[42] Sangiorgi, Davide: From π-calculus to higher-order π-calculus – and back. In *Lecture Notes in Computer Science*. 668. köt. 1993, Springer-Verlag, 151–166. p.

[43] Sangiorgi, Davide: Pi-calculus, internal mobility, and agent-passing calculi. INRIA Research Report 2539. Jelentés, 1995, INRIA Sophia-Antipolis.

[44] Sangiorgi, Davide: A theory of bisimulation for π-calculus. *Acta Informatica*, 33. évf. (1996) 1. sz., 69–97. p.

[45] Sangiorgi, Davide: *Introduction to Bisimulation and Coinduction*. 2012, Cambridge University Press.

[46] Sangiorgi, Davide – Walker, David: *The π-calculus, A Theory of Mobile Processes*. 2001, Cambridge University Press.

[47] Vasco Thudichum Vasconcelos: A note on a typing system for the higher-order π-calculus. http://www.di.fc.ul.pt/~vv/papers/vasconcelos_typing-system-hopi.ps.gz, 1993.

[48] Vasconcelos, Vasco Thudichum: Fundamentals of session types. *Information and Computation*, 217. évf. (2012), 52–70. p.

[49] Victor, Björn: The fusion calculus. https://www.it.uu.se/research/group/mobility/kurs0203/fusion-slides.pdf, 2016.

Tárgymutató

A, Á

B