Bevezetés a programozásba[1][2]

Fóthi Ákos, Horváth Zoltán

ELTE Informatikai Kar, 2005.

[1] A mű digitális megjelenítése az Oktatási Minisztérium támogatásával, a Felsőoktatási Tankönyv- és Szakkönyv-támogatási Pályázat keretében történt.

III. rész

Párhuzamos és elosztott programozás
Horváth Zoltán

Tartalomjegyzék

Előszó

A tankönyv második része az ELTE programtervező hallgatói számára tartott párhuzamos programozás alapjai című tárgy előadásai alapján készült, az előadások anyagát és az érdeklődő hallgatók számára háttéranyagot tartalmaz.

Feltételezzük, hogy az olvasó elsajátította a könyv első részében szereplő tananyagot, rendelkezik megfelelő programozási gyakorlattal, írt legalább néhány egyszerű párhuzamos vagy elosztott programot.

A következőkben egy olyan programozási modellt fogalmazunk meg, amely alkalmas arra, hogy párhuzamos és elosztott programok tervezését támogassa. A fogalmak jelentését relációk segítségével írjuk le. Programozási modellt különböző matematikai eszközök segítségével fogalmazhatunk meg, széles körben használnak például algebrai, illetve logikai modelleket. Folyamatalgebrai modellek alapjait tárgyalja például [Hen 88], temporális logikai eszközöket mutat be [Eme Sri 88, Krö 87].

A bemutatott modell két fontos előzményre épít. Az egyik a nemdeterminisztikus szekvenciális programok reláció alapú modellje [Fót 83], a másik Chandy és Misra párhuzamos programozási módszertana. Mindkettő közös gyökere Dijkstra „programozási diszciplínája” [Dij 76].

A párhuzamos programok tervezéséhez készített modell tehát ugyanazt a megközelítést alkalmazza, ugyanazt a fogalomrendszert és eszközkészletet használja, mint könyv első felében a szekvenciális programok leírásakor bemutatott modell.

Bevezetés

Számos programozási feladat megoldása lehet szekvenciális, elosztott vagy párhuzamos program is. A programozási feladatok egy része azonban olyan jellegű, amelynek megoldásához térben elosztva elhelyezkedő adatok, erőforrások felhasználása szükséges [Lyn 02, Tan Ste 02]. Ilyen feladatokat egyetlen számítógépen futó, szekvenciális program nem oldhat meg.

Párhuzamos feldolgozás esetén megkülönböztethetünk adatintenzív, illetve számításintenzív feladatosztályokat. Adatintenzív feladat például fizikai kísérletekből származó nagyszámú mérési adat kiértékelése, vagy diagnosztikai eljárásokkal előállított adatok elemzése. Általában számításigényes feladatok közé tartoznak a modellezési feladatok, pl. időjárás előrejelzés, szélcsatorna kísérletek szimulációja, stb. Ezekben az esetekben elvileg egyetlen számítógép is elvégezhetné az adatok feldolgozását, de párhuzamos algoritmusok segítségével az eredmény gyorsabban előállítható. Az is gyakori eset, hogy az eredmény kiszámítását adott időkorláton belül kell elvégezni és ehhez egyetlen processzor számítási kapacitása nem elegendő. Több processzor alkalmazásával nemcsak a számításhoz szükséges idő rövidíthető le, de az időkorlátra vonatkozó követelmény is teljesíthető [Ivá 03].

Programozási feladatok specifikációjának és megoldásának alkalmas módszereit keressük párhuzamos és elosztott rendszerek esetén.

Elosztott és párhuzamos programok fejlesztése során a program helyességének bizonyítása azért is fontos lehet, mert teszteléssel nehezebben lehet a hibákat felfedezni mint szekvenciális programok esetén. Elosztott és párhuzamos programok ismételt futtatása gyakran vezet eltérő eredményre. Ennek az az oka, hogy a komponensek kölcsönhatása annak függvényében változik, hogy az adott futtatás során az egymástól független események közül melyik következik be előbb. Ha a program tesztelése érdekében nyomkövetési utasításokat helyezünk el, akkor ezzel megváltoztatjuk az egyes folyamatok között fennálló időzítési viszonyokat, így könnyen előfordulhat, hogy a tesztelés során a hiba nem jelentkezik.

Biztonságkritikus alkalmazások készítése során tehát csak bizonyítottan helyes komponenseket használhatunk fel és az elosztott program összetevőit csak az összetett program tulajdonságait garantáló programkonstrukciók segítségével szerkeszthetjük össze.

Feladatok megfogalmazása, programok kódolása mindig valamilyen nyelvi eszköz segítségével valósul meg. A feladat specifikációja egy jelsorozat, ahogy egy adott programozási nyelven írt program is betűk, szimbólumok, esetleg grafikus elemek sorozata. Ahhoz, hogy válaszolni tudjunk arra a kérdésre, hogy egy specifikáció valójában milyen feladatot ír le, hogy egy program futása során mi történhet, meg kell adnunk ezen jelsorozatok jelentését. Programok jelentésének pontos meghatározására matematikai modelleket használunk.

Párhuzamos és elosztott programok tervezésének egy matematikai modelljére teszünk javaslatot oly módon, hogy kiterjesztjük a nemdeterminisztikus szekvenciális programok relációs alapú szemantikai modelljét [Fót 83] és a programozási feladatok megfogalmazására és megoldására korábban sikeresen alkalmazott módszereket [Dij 76, Fót Hor 91] párhuzamos programokra is.

Célunk, hogy a modell eszközei segítségével a feladat specifikációját helyettesíteni tudjuk olyan feladatok specifikációival, amely feladatok megoldása esetén a rendelkezésre álló matematikai eszközökkel belátható az eredeti feladat megoldásának helyessége [Var 81, Fót Hor 91, Cha Mis 89].

Arra törekszünk, hogy a megoldás előállításával párhuzamosan a megoldás helyességének bizonyítását is előállítsuk. Nem célunk az automatikus programszintézis [Lav 78],[Eme Sri 88]/4.1.3., és nem akarjuk kész algoritmusok helyességét utólag igazolni [Eme Sri 88]/4.2. Ennek elsősorban az az oka, hogy párhuzamos programokat a legtöbb esetben részben vagy kizárólag azért írunk, hogy a megoldás hatékonyabb legyen a szekvenciális architektúrán elérhető megoldásnál. A hatékonyság lényeges szempont lehet természeténél fogva párhuzamos programmal megoldandó feladatok esetén is, pl. valós idejű alkalmazásoknál, folyamatszabályozó vagy on-line tanácsadó rendszerek esetén [Hor 88]. Talán csak egyes szimulációs feladatok vagy prototípus szoftver fejlesztése során másodlagos a megoldás hatékonysága. Hatékony megoldás előállítására az automatikus programszintézis bonyolultabb feladatok esetén általában nem alkalmas. Az [Eme Sri 88]-ban ismertetett eredmények meggyőzően mutatják azt is, hogy a szintetizáló algoritmusok általában nagyon rossz hatékonyságúak, a megoldás előállításához a specifikáció hosszával exponenciálisan arányos időre van szükség. Hasonló állítások igazak a programbizonyításra is. A programbizonyítási eljárás sikertelensége esetén nincs elegendő támpont a program javításához sem.

A UNITY [Cha Mis 89] lineáris idejű temporális logikára támaszkodó operátorait újrafogalmazzuk a leggyengébb előfeltétel és más predikátumtranszformerek segítségével. Megvizsgáljuk a bevezetett specifikációs módszer kifejezőerejét és az általánosítási lehetőségeket.

Az egyes fejezetekről

A 1-3. fejezetben megadjuk a relációs modell alapfogalmainak definícióit. A két legfontosabb bevezetett fogalom a feladat és az absztrakt program fogalma.

Az 4. fejezetben adjuk meg, hogy egy absztrakt program mikor old meg egy feladatot. Kimondunk néhány tételt is, amelyek igazolják, hogy a bevezetett megoldásfogalom megfelel elvárásainknak.

A 5. fejezetben több hasznos tételt bizonyítunk, amelyek segítségével a későbbiek során könnyebben igazolhatjuk feladatok és programok tulajdonságait.

A 6. fejezetben összetett problémák megoldása során alkalmazható eszközöket vezetünk be. A megoldást modulokból állítjuk össze. Az absztrakt programokat egyesítjük és szuperponáljuk, támaszkodunk az ún. nyitott specifikáció fogalmára [Cha Mis 89]. Megvizsgáljuk programok szekvenciális ill. valós párhuzamos kompozíciójának lehetőségét.

A 7. fejezetben megvizsgáljuk a bevezetett modell tulajdonságait, kifejezőerejét.

A modell eszközkészletének ismertetése után programozási tételeket mondunk ki és összetett problémák megoldása során alkalmazható eszközöket vezetünk be.

A 8. fejezetben általánosan megfogalmazott programozási feladatokat oldunk meg. A kapott megoldásokat programozási tételeknek nevezzük, mert széles körben alkalmazhatóak konkrét feladatok megoldása során. Az egyik ilyen alapfeladat asszociatív művelet eredményének párhuzamos kiszámítása. A másik az elemenként feldolgozható, ill. a sorozatokon többszörös függvénykompozícióval definiált függvény értékének kiszámítása. Példát mutatunk csatornaváltozók használatára és adatcsatornás megoldási módszerekre is. Megvizsgáljuk, hogy a kapott megoldások milyen architektúrákon implementálhatók hatékonyan. Olyan megoldásokat dolgozunk ki, amelyek osztott és aszinkron osztott memóriás rendszerekre is könnyen leképezhetőek.

A 9-10. fejezetben megvizsgáljuk, hogy a párhuzamos programok leírására alkotott modellek milyen lényeges tulajdonságokban térnek el egymástól. Megadjuk, hogy az általunk javasolt modell milyen jelenségek leírására alkalmas.

A 11. fejezetben ismertetjük azokat a matematikai eszközöket, amely a bevezetett modell mélyebb matematikai hátterét tisztázzák. Összefoglaljuk a leképezések fixpontjaira vonatkozó eredményeket és bevezetjük az olvasót a temporális logikák világába. A temporális logikákról szóló 11.1 bekezdésben bevezetett fogalmakra és tételekre a II. részben általában csak egyes megjegyzésekben és lábjegyzetekben utaltunk, így a II. rész fejezeteinek megértéséhez ez a bekezdés nem feltétlenül szükséges.

A 12. fejezetben összefoglaljuk és értékeljük a leírt módszereket.

A függelékben megvizsgáljuk, hogy az absztrakt programokat hogyan kódoljuk C-PVM segítségével.

A könnyebb tájékozódást kereszthivatkozások, tárgymutató és jelölések jegyzéke, a legfontosabb fogalmak definíciói, stb. segíti. Az érdeklődő olvasó gyakorló feladatok megoldásásával ellenőrizheti tudását.

Egyes megjegyzéseket lábjegyzetben helyeztünk el. A lábjegyzetekben általában más modellek rokon fogalmaira utalunk röviden. Ezek a megjegyzések elsősorban azoknak az olvasóknak szólnak, akik ezekben a modellekben járatosak.

Gyakran használt jelölések

$::=$ - definiáló egyenlőség
$A ::= \underset{i \in I}{\times} A_i$ - állapottér
$a = (a_1, \dots, a_n) \in A$ - állapot
$R \subseteq \underset{i \in I}{\times} A_i$ - reláció
$R_n(A)$ az $\underset{i \in [1..n]}{\times} A$ direktszorzat feletti relációk halmaza
$R \subseteq A \times B$ - bináris reláció
$R(a)$ - az R reláció képhalmaza
$\mathcal{D}_R$ - az R reláció értelmezési tartománya
$\alpha = (\alpha_1, \dots, \alpha_n)$ - véges sorozat
$\alpha = (\alpha_1, \dots \dots)$ - végtelen sorozat
$|\alpha|$ - az α sorozat hossza
A^* : A elemeiből képzett véges sorozatok halmaza
A^∞ : A elemeiből képzett végtelen sorozatok halmaza
$A^{**} ::= A^* \cup A^\infty$
$\mathcal{P}(A)$ - az A hatványhalmaza
$R : A \longrightarrow B$ - parciális függvény A-ról B-re
$R : A \longmapsto B$ - függvény A-ról B-re
$pr_{A_1} : A \longmapsto A_1$ - az A térről az A_1 altérre vetítés
$\mathcal{L} ::= \{igaz, hamis\}$ - logikai értékek halmaza
Igaz : $A \longmapsto \mathcal{L}$ - az azonosan igaz,
Hamis : $A \longmapsto \mathcal{L}$ - az azonosan hamis logikai függvény.
$\lceil f \rceil$ - logikai függvény (állítás) igazsághalmaza
$P \Rightarrow Q ::= \lceil P \rceil \subseteq \lceil Q \rceil$
$\lceil P \wedge Q \rceil ::= \lceil P \rceil \cap \lceil Q \rceil$
$\lceil P \vee Q \rceil ::= \lceil P \rceil \cup \lceil Q \rceil$
$\lceil \neg Q \rceil ::= A \setminus \lceil Q \rceil$
$\lceil P \rightarrow Q \rceil ::= \lceil \neg P \vee Q \rceil$
$\Longrightarrow, \Longleftrightarrow, \Longleftarrow$ - „ha, akkor”, „akkor és csak akkor”, ill. „akkor, ha”
$R^{(-1)}$ - inverz reláció

R^{tdl} - reláció tranzitív diszjunktív lezártja
$[R]$ az $R =$ Igaz állítás rövidítése, ahol $R \subseteq A \times \mathcal{L}$
$v_i : A \longmapsto A_i$ - változó
$\mathcal{N}$ - pozitív egészek halmaza
$\mathcal{N}_0$ - nemnegatív egészek halmaza
ω - a természetes számok halmazának rendszáma
$\eta X : G(X)$ - G legnagyobb fixpontja X szerint
$\mu Y : F(Y)$ - F legkisebb fixpontja Y szerint
$P \rhd Q$ - P stabil feltéve, hogy nem Q
$P \mapsto Q$ - P biztosítja Q-t
$P \hookrightarrow Q$ - P-ből elkerülhetetlen Q
FP $\Rightarrow R$ - R teljesül fixpontban
$Q \in$ INIT - Q igaz kezdetben
inv$P$ - P invariáns
$Q \hookrightarrow$ FP, $Q \in$ TERM - Q-ból indítva a program biztosan fixpontba jut
$VR(P)$ - azok a változók, amelyektől a P (logikai) reláció (függvény) függ
F - feladat
B - paramétertér
s - utasítás
I - a változók és az állapottérkomponensek indexeinek halmaza
J - a program utasításainak indexhalmaza
$p(s)$ - az s utasítás hatásrelációja
$SKIP$ - üres utasítás
$VL(s)$ - az s utasítás baloldalán álló változók
$VR(s)$ - az s utasítás jobboldalán álló változók
$V(s){::=}VR(s) \cup VL(s)$
S - absztrakt program
$E(S)(a)$ - S program a-ból elérhető állapotainak halmaza
$VL(S)$ - az S programban baloldalon álló változók
$VR(S)$ - az S programban jobboldalon álló változók
$V(S){::=}VR(S) \cup VL(S)$
$fixpont_S$, φ_S - az S absztrakt program fixpontjait jellemző állítás
P', F', S' - altéren definiált logikai fgv., feladat, program kiterjesztése
$s_1 \| s_2$ - feltételes értékadások szuperpozíciója
$S_1 \cup S_2$ - programok uniója
$F_1 \sqcup F_2$ - feladatok egyesítése
$S_1 ; S_2$ - programok szekvenciája
BT - elágazó idejű temporális logika

LT - lineáris idejű temporális logika

1. fejezet

A relációs modell alapfogalmai

> Programozási modell*nek nevezzük azt a matematikai modellt, amely megadja feladatok és programok szemantikai jelentését, konstrukciós műveleteket definiál feladatok és programok felett, valamint megadja, hogy egy program mikor old meg egy feladatot. Relációs modellről beszélünk, ha a szemantikai tartományok elemei relációk.*

Gondolkodásunk során a programozási feladat [Fót 83] fogalmából indulunk ki. Programozási feladatot mindig egy állapottéren [Dij 76] fogalmazunk meg. A feladat megfogalmazásához tehát meg kell alkotnunk a feladat matematikai modelljét[1], absztrakcióra van szükség. A feladat megfogalmazásához vezető utat most nem vizsgáljuk.[2]

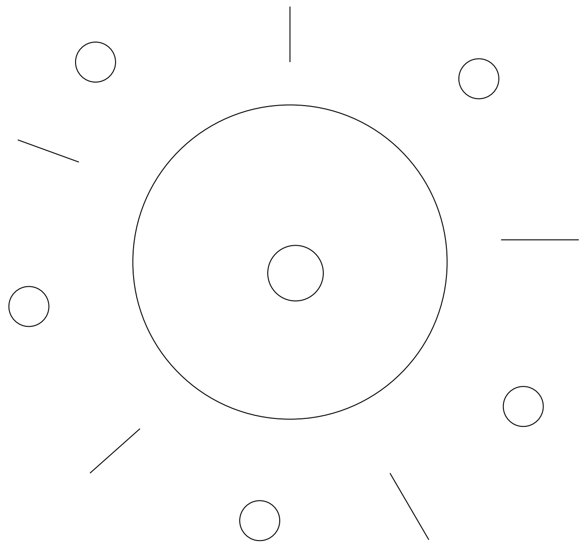
1.1. Feladat: étkező filozófusok

Első példánk E. W. Dijkstrától származik, aki az operációs rendszerek tárgy tanítása során a folyamatok közötti kölcsönös kizáráson alapuló erőforrásmegosztás elvét szemléltette vele. Ez a példa öt, egymással együttműködő párhuzamos folyamatból álló rendszert modellez. A történet szerint egy asztal körül öt filozófus ül, akik egy nagy közös tálból vehetnek maguknak makarónit. Ahhoz, hogy a makaróni a tányérba kerüljön két villára van szükség. A tálat többen is használhatják

[1] A modell szót általános értelemben használjuk. Ha a matematikai logikában, a modellelméletben használt modellfogalomra gondolunk, akkor erre külön hivatkozunk. Egy feladat megfogalmazása nem köthető pl. egy rögzített temporális logikai struktúrához, mert ez esetben a feladat már nem fogalmazható meg függetlenül az időstruktúrát definiáló programtól. A feladatot nem azonosítjuk az őt leíró formulák halmazával, mint szintaktikus egységekkel sem, ugyanis egy interpretálatlan formulahalmaz minden interpretációban más és más szemantikai jelentést hordoz. A feladatot mindig egy rögzített állapottér felett, azon értelmezett relációk segítségével adjuk meg.

[2] Nem vizsgáljuk azt, hogy egy feladat formális alakja valóban azt a feladatot írja-e le, amit valamilyen természetes nyelven megfogalmaztak. A választott programozási modell keretein túlmutat ennek a kérdésnek a vizsgálata.

egyszerre, de az étkezéshez szükséges villák midegyikére külön-külön teljesülnie kell, hogy egyszerre csak egy filozófus használatában lehetnek. Minden filozófus számára két villa érhető el, ezek azonban olyan villák amelyeket szomszédai is szeretnének használni. Ha egy filozófus felemeli az asztalról bal- és jobboldali villáját és eszik, akkor egyik szomszédja sem rendelkezhet az étkezéshez szükséges két villával. Egyidejűleg csak egymással nem szomszédos filozófusok étkezhetnek .



1.1. ábra. Az étkező filozófusok

Jelöljük az i. filozófust $f(i)$-vel, az egyes állapotokat pedig a kezdőbetűjükkel: gondolkodik:g, villákat tart a kezében:v, eszik:e, otthon van:o. A következőkben példákat mutatunk a filozófusok viselkedésére vonatkozó specifikációs követelmények megfogalmazására.

Az „amíg nem” (jelölés: $\triangleright$) típusú kikötésekkel egyes állapotátmeneteket tilthatunk. Megkövetelhetjük, hogy minden filozófus gondolkodik, amíg villát nem tart a kezében vagy haza nem ért: $f(i).g \triangleright f(i).v \vee f(i).o$. Ez azt jelenti, hogy a gondolkodás állapotát nem követheti közvetlenül az étkezés állapota. Kikötjük azt is, hogy a villák kézben tartását csak az evés állapota követheti: $f(i).v \triangleright f(i).e$, a villák megszerzése után nem szabad sem hazamenni, sem gondolkodni. Szigorúbb kikötést tehetünk oly módon, hogy a megengedett állapotváltozások bekövetkezését elő is írjuk: $f(i).e \mapsto f(i).g$. Az étkezést előbb utóbb fel kell váltania a gondolkodásnak. Lazább kapcsolatot is megkövetelhetünk állapotok között:

$f(i).g \hookrightarrow f(i).o$, a gondolkodás állapotát előbb utóbb elkerülhetetlenül követi, hogy a filozófus otthon van. Invariáns tiltja, hogy szomszédok egyszerre étkezzenek: $(f(i).e \rightarrow (\neg f(i+1).e \wedge \neg f(i-1).e)) \in$ inv. Fixpont kikötéseket alkalmazunk arra, hogy előírást tegyünk arra az esetre, ha a rendszer nyugalmi állapotba jut: FP $\Rightarrow f(i).h$. A kikötés szerint, ha további állapotváltozás nincs, akkor minden filozófus otthon van. Megkövetelhetjük, hogy egy állapotból $\forall i : f(i).g \in$ TERM elkerülhetetlen legyen a nyugalmi állapot elérése, a terminálás.

1.2. Absztrakt program: rendezés

Következő példánk az absztrakt program fogalmát mutatja be. Az absztrakt program utasításait nem rendezzük szekvenciális folyamatokká, egyetlen utasításhalmaz formájában adjuk meg. Utasításhalmazok tulajdonságai könnyebben igazolhatóak, mint szekvenciális folyamatok halmazaként megadott párhuzamos programok helyessége. Az utasításhalmaz elemeit felváltva hajtjuk végre (ld. 3. fejezet), a program működése egy több ciklusmaggal rendelkező ciklushoz hasonlít. Ha egynél több processzor áll rendelkezésre, akkor egyszerre vagy időben átfedve több utasítást is végrehajthatunk [3]. Az utasításhalmazt egy inicializáló értékadás előzi meg.

A buborékrendezés algoritmusa szerint két szomszédos megcserélünk, ha rossz sorrendben vannak. Az absztrakt program minden szomszédos elempárhoz tartalmaz egy olyan utasítást, amelyik szükség esetén a két elemet megcseréli. Ha az elemek sorrendje helyes, akkor további állapotváltozás már nem következik be, a program terminál.

$$S = (SKIP,$$
$$\{ \underset{i\in[1..n-1]}{\Box} a(i), a(i+1) := a(i+1), a(i), \text{ ha } a(i) > a(i+1)\})$$

1.1. absztrakt program. Buborékrendezés.

Az absztrakt program egy lehetséges implementációja, ha minden utasításhoz egy önálló folyamat tartozik és a folyamatokhoz egy-egy saját processzort rendelünk hozzá. Egyidejűleg csak egy folyamat számára engedélyezhetjük a vektor elemeihez való hozzáférést. Jelöljük `<lock a(i) and a(i+1)>`-vel azt a műveletet, amellyel egy folyamat az i. és az i+1. elem használatát igényli (a kritikus szakasz kezdetét), illetve `<unlock a(i) and a(i+1)>`-vel azt a művele-

[3] Ebben az esetben csak olyan értékadások egyidejű vagy időben átfedő végrehajtása megengedett, amelyek nem tartalmaznak közös változót.

tet, amellyel lemond az elemek kizárólagos használatáról (kritikus szakasz vége) . Ebben az esetben az *i*. processzoron futó program pszeudokódja a következő lehet:

```
loop
 < lock a(i) and a(i+1) >
 x := a(i);
 y := a(i+1);
 if x > y then
    a(i+1):=x;
    a(i):= y;
 end if;
 < unlock a(i) and a(i+1) >
end loop;
```

$n-1$ folyamat.

Bemutatjuk azt is, hogy ugyenezen absztrakt programot hogyan valósíthatjuk meg egyetlen processzoron futó szekvenciális program formájában:

```
loop
  for i=1 to n-1 do
    x := a(i);
    y := a(i+1);
    if x > y then
      a(i+1):=x;
      a(i):= y;
    end if;
  end for
end loop
```

1.3. A relációs modell alapfogalmai

A feladat és az absztrakt program pontos jelentését úgy adhatjuk meg, hogy a feladat specifikációjában szereplő kikötésekhez, ill. az absztrakt programot leíró utasításhalmazhoz relációt rendelünk hozzá. A feladatot, ill. a programot leíró relációkat az állapottér, az állapottér hatványhalmaza, és ezek direktszorzatai felett értelmezzünk. Az alapfogalmak definíciói megegyeznek a könyv első részében adottakkal.

Az állapottér véges sok legfeljebb megszámlálhatóan végtelen típusértékhalmaz direktszorzata [Fót 83].

1.1. Definíció (Állapottér). *$I \subset \mathcal{N}$. $\forall i_j \in I : A_{i_j}$ megszámlálható halmaz. Az $A ::= \times_{i_j \in I} A_{i_j}$ halmazt* állapottérnek, *az A_{i_j} halmazokat* típusértékhalmazoknak *nevezzük.*

1.2. Definíció (Állapot). *Az állapottér elemeit, az $a = (a_{i_1}, \ldots, a_{i_n}) \in A$ pontokat* állapot*oknak nevezzük.*

A feladat matematikai megfogalmazásához szükségünk lesz a reláció és a sorozat fogalmára. Megismételjük a reláció, bináris reláció és a reláció értelmezési tartománya definícióját (11. fejezet).

1.3. Definíció (Reláció). *$I \subset \mathcal{N}$. Az $R \subseteq \times_{i_j \in I} A_{i_j}$ halmazt* reláció*nak*[4] *nevezzük.*

Az $R \subseteq A \times B$-t *bináris reláció*nak nevezzük.

1.4. Definíció (Reláció értéke). *$R \subseteq A \times B$. $R(a) ::= \{ b \,|\, (a,b) \in R \}$ halmaz az R reláció* értéke[5] az *a* pontban.

1.5. Definíció (Reláció értelmezési tartománya). *$R \subseteq A \times B$. Az R* reláció értelmezési tartománya*:*
$\mathcal{D}_R ::= \{ a \in A \,|\, \exists b \in B : (a,b) \in R \}$. [6]

Jelölések:
véges sorozat: $\alpha = (\alpha_1, \ldots, \alpha_n)$
végtelen sorozat: $\alpha = (\alpha_1, \ldots \ldots)$
A^* : A elemeiből képzett véges sorozatok halmaza
A^∞ : A elemeiből képzett végtelen sorozatok halmaza
$A^{**} ::= A^* \cup A^\infty$

A feladat matematikai megfelelője feltételek együttese. A specifikációs feltételek megfogalmazásához logikai függvényeket használunk. Logikai függvényeket igazsághalmazukkal, egy alaphalmaz adott részhalmazával jellemezhetjük. A specifikációs relációk leírásához szükségünk lesz a hatványhalmaz fogalmára. Minden egyes specifikációs feltétel az állapottér hatványhalmaza felett értelmezett reláció.

[4] Példa: $A ::= \{1,2,5\}$, $B ::= \{2,3\}$. $R \subseteq A \times B$. $R ::= \{(1,3),(1,2),(5,2)\}$.

[5] A példa R relációjának képe az 1 pontban: $R(1) = \{2,3\}$.

[6] A példa R relációjának értelmezési tartománya: $\mathcal{D}_R = \{1,5\}$.

1.6. Definíció (Hatványhalmaz). *Az A halmaz részhalmazainak halmazát az A* hatványhalmaz*ának nevezzük és $\mathcal{P}(A)$-val jelöljük.*

A feladat matematikai megfelelője feltételek együttese. Minden egyes feltétel az állapottér hatványhalmaza felett értelmezett reláció. Az állapottér részhalmazait logikai relációkkal jellemezzük. Megadjuk a logikai reláció, logikai függvény és igazsághalmazuk definícióját.

1.7. Definíció (Parciális függvény). *Az $R \subseteq A \times B$ reláció* determinisztikus reláció*, (vagy* parciális függvény*), ha $\forall a \in A \ : \ |R(a)| \leq 1$.*
Parciális függvények esetén az $R : A \longrightarrow B$ jelölést alkalmazzuk.

1.8. Definíció (Függvény). *Az $R \subseteq A \times B$ reláció* függvény*, ha* *$\forall a \in A \ : \ |R(a)| = 1$.*
Függvények esetén az $R : A \longmapsto B$ jelölést használjuk.

1.9. Definíció (Logikai függvény, logikai reláció). *$f \subseteq A \times \mathcal{L}$ reláció* logikai reláció *, ahol $\mathcal{L}$::=$\{igaz, hamis\}$ a logikai értékek halmaza.* Logikai függvény*-ről beszélünk, ha a logikai reláció függvény.*

Jelölés:
$\text{Igaz} : A \longmapsto \mathcal{L}$ - az azonosan igaz,
$\text{Hamis} : A \longmapsto \mathcal{L}$ - az azonosan hamis logikai függvény.

1.10. Definíció (Logikai függvény igazsághalmaza). *Az f* logikai függvény (állítás) igazsághalmaza[7]*: $\lceil f \rceil$::=$\{a \in A | f(a) = \{igaz\}\}$.*

1.1. Megjegyzés (Logikai műveletek). *A logikai relációk felett értelmezzük a $\wedge$, $\vee$, $\rightarrow$, $\Rightarrow$, $\neg$ műveleteket, oly módon, hogy azok megfeleljenek a relációk igazsághalmazaira vonatkozó halmazműveleteknek.*
Azaz: $P \Rightarrow Q$::=$\lceil P \rceil \subseteq \lceil Q \rceil$, $\lceil P \wedge Q \rceil$::=$\lceil P \rceil \cap \lceil Q \rceil$, $\lceil P \vee Q \rceil$::=$\lceil P \rceil \cup \lceil Q \rceil$, $\lceil \neg Q \rceil$::=$A \setminus \lceil Q \rceil$, és $\lceil P \rightarrow Q \rceil$::=$\lceil \neg P \vee Q \rceil$.
A $\Longrightarrow, \Longleftrightarrow, \Longleftarrow$ jeleket bizonyítások szövegének rövidítésére használjuk, a „ha, akkor”, „akkor és csak akkor”, ill. „akkor, ha” állítások leírásának rövidítésére.

1.11. Definíció (Reláció inverz képe). *$R^{(-1)}(H)$::=$\{a \in A \,|\, \exists h \in H \ : \ (a,h) \in R\}$ a $H \subseteq B$ halmaz $R \subseteq A \times B$ relációra vonatkozó* inverz képe*.*

[7]Példa: A::=$\{1,2,5\}$. $R : A \mapsto \mathcal{L}$. R::=$\{(1,igaz),(2,hamis),(5,igaz)\}$. $\lceil R \rceil = \{1,5\}$.

1.12. Definíció (Reláció ősképe). $R^{-1}(H){::=}\{\,a \in A \,|\, R(a) \subseteq H \,\wedge\, R(a) \neq \emptyset\,\}$ *a* $H \subseteq B$ *halmaz* $R \subseteq A \times B$ *relációra vonatkozó* ősképe *[WRMP 95].*

1.13. Definíció (Relációk kompozíciója). *Az* $R_1 \subseteq A \times B$ *és az* $R_2 \subseteq B \times C$ *relációk* kompozíciója*:*
$R_2 \circ R_1{::=}\{\,(a,c) \,|\, \exists b \in B \,:\, (a,b) \in R_1 \,\wedge\, (b,c) \in R_2\,\}.$

1.14. Definíció (Relációk szigorú kompozíciója). *Az* $R_1 \subseteq A \times B$ *és az* $R_2 \subseteq B \times C$ *relációk* szigorú kompozíciója *[Hor 90]:*
$R_2 \odot R_1{::=}\{\,(a,c) \,|\, R_1(a) \subseteq \mathcal{D}_{R_2} \,\wedge\, \exists b \in B \,:\, (a,b) \in R_1 \,\wedge\, (b,c) \in R_2\,\}.$

1.15. Definíció (Reláció igazsághalmaza). $\lceil R \rceil{::=}R^{-1}(\{igaz\})$ *az* $R \subseteq A \times \mathcal{L}$ reláció igazsághalmaza*.*

Haladási tulajdonságok megfogalmazásához szükségünk lesz relációk tranzitív diszjunktív lezártjának fogalmára.

1.16. Definíció (Reláció tranzitív diszjunktív lezártja). $R^{tdl} \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$ *reláció az* $R \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$ *reláció tranzitív diszjunktív lezártja, ha* R^{tdl} *a legkisebb olyan reláció, amelyre:* $R \subseteq R^{tdl}$*, ha* $(a,b) \in R^{tdl}$ *és* $(b,c) \in R^{tdl}$*, akkor* $(a,c) \in R^{tdl}$ *és bármely* W *megszámlálható halmazra:* $(\forall m : m \in W :: (a(m),b) \in R^{tdl})$ $\Longrightarrow ((\bigcup_{m \in W} a(m)),b) \in R^{tdl}$*.*

1.1. Példa (Reláció tranzitív diszjunktív lezártja).
$A = \{1,2,3,4\}, R \subseteq \mathcal{P}(A) \times \mathcal{P}(A),$
$R = \{(\{3\},\{1\}),(\{2\},\{1\}),(\{1\},\{4\})\},$
$R^{tdl} = \{(\{3\},\{1\}),(\{2\},\{1\}),(\{1\},\{4\}),$
$(\{2,3\},\{1\}),(\{2\},\{4\}),(\{3\},\{4\}),$
$(\{2,3\},\{4\}),(\{1,3\},\{4\}),(\{1,2\},\{4\}),(\{1,2,3\},\{4\})\}$

1.17. Definíció ($[R]$). *Legyen* $R \subseteq A \times \mathcal{L}$*.* $[R]$ *annak az állításnak a rövid megfogalmazása, hogy az* R *reláció igazsághalmaza megegyezik* A*-val [Dij Sch 89].*

1.2. Megjegyzés. $\lceil P \rceil \subseteq \lceil Q \rceil \Longleftrightarrow P \Rightarrow Q \Longleftrightarrow [P \rightarrow Q]$.

1.18. Definíció (Változó). *A* $v_{i_j} : A \longmapsto A_{i_j}$ *projekciókat* változó*knak nevezzük.*[8] *A változók megadásakor a projekció értelmezési tartományát, az állapotteret általában elhagyjuk:* $v_{i_j} : A_{i_j}$*.*

[8] A fenti definíció szerint a változó tehát nem szintaktikus fogalom. A továbbiakban amíg a választott programozási modell keretein belül maradunk nem vizsgáljuk formális nyelvek és szemantikai tartományok lehetséges kapcsolatrendszereit. Ha szükséges, a későbbiek során könnyen definiálható formális nyelv és szemantikus leképezés. A szemantikai tartomány elemei az általunk megfogalmazott modellben definiált egyes matematikai objektumok lehetnek.

1.19. Definíció (Vetítés altérre). *Legyen A_1 az A direktszorzat altere. $pr_{A_1} : A \longmapsto A_1$ függvény az A-beli pontokhoz az A_1-beli vetületüket rendeli.*
A pr_{A_1} függvényt általánosítjuk A részhalmazaira, A-beli elemek sorozataira, A felett értelmezett bináris relációkra oly módon, hogy a részhalmazok és sorozatok elemeit ill. a relációk elemeinek komponenseit pontonként vetítjük az altérre [WRMP 95].

Értékadások vizsgálatánál gyakran van arra szükség, hogy meghatározzuk milyen változók kaphatnak új értéket, illetve milyen változók értékétől függ az eredmény. Értékadást hatásrelációja jellemez, így a értékadások vizsgálata a hatásrelációjuk vizsgálatára vezethető vissza.

1.20. Definíció (Reláció független egy változótól). *Legyen $A ::= \times_{i\in[1,n]} A_i$ és $R \subseteq A \times B$. Azt mondjuk, hogy az R reláció* független *az A_i komponenstől és a $v_i : A \longmapsto A_i$ változótól, ha*
$\forall a,b \in \mathcal{D}_R : (\forall k \in ([1,i-1] \cup [i+1,n]) : a_k = b_k) \Rightarrow R(a) = R(b)$

Jelöljük $VR(R)$-rel azon változók halmazát, amelyektől az R reláció függ.

1.21. Definíció (Reláció nem változtatja meg). *Legyen $A ::= \times_{i\in[1,n]} A_i$ és $R \subseteq A \times A$. Azt mondjuk, hogy az R reláció* nem változtatja meg *az A_i komponens és a $v_i : A \longmapsto A_i$ változó értékét, ha $v_i \circ R = v_i$.*

Jelöljük $VL(R)$-rel azon változók halmazát, amelyeket az R reláció megváltoztat. $V(R) ::= VL(R) \cup VR(R)$.

2. fejezet

A feladat fogalmának általánosítása

> *A feladat definíciójának megfogalmazásakor általánosítjuk azt a specifikációs módszert, amely relációk segítségével megfogalmazott elő- és utófeltételeket használ. A most bevezetett feladatfogalom magában foglalja azt az esetet is, amikor egy vagy több nem feltétlenül termináló folyamat, egy* zárt rendszer *együttes viselkedésére teszünk előírásokat. Megjegyezzük, hogy a feladat függetlenül megfogalmazható bármely lehetséges megoldásától, összehasonlítható más feladatokkal, illetve összevethető tetszőleges vele közös állapottéren futó programmal abból a szempontból, hogy az megoldja-e. A feladat megoldása nem feltétlenül csak párhuzamos program lehet.*

A könyv első részében bevezetett feladat fogalmát általánosítjuk, hogy olyan feladatokat is specifikálhassunk, amelyek elő- és utófeltételek segítségével nem írhatóak le. Ilyen feladat például egy operációs rendszer feladata, amelynek folyamatos helyes működésében vagyunk érdekeltek egy utófeltétel teljesülése helyett. Folyamatszabályozó szoftverek, beágyazott rendszerek működése is biztonságossági, haladási feltételekkel jellemezhető utófeltételek megadása helyett.

A feladat matematikai megfelelője *specifikációs reláció*k együttese. A specifikációs relációkat az állapottér hatványhalmaza felett értelmezzük, a relációk elemeit *specifikációs feltétel*eknek nevezzük.

2.1. Specifikációs feltételek

A specifikációs feltételek a programra, mint az állapottér feletti mozgásra fogalmaznak meg kikötéseket. Ezeket a kikötéseket csoportosíthatjuk típusuk szerint. Egy feladat leírásához hét féle feltételt használunk. Az azonos feltételtípushoz tartozó feltételeket egy relációban gyűjtjük össze, így hét specifikációs relációt vezetünk be.

Legyen $P,Q,R,U : A \longmapsto \mathcal{L}$ logikai függvény.
$\rhd, \mapsto, \hookrightarrow \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$ relációk, és $\mathrm{FP}, \mathrm{INIT}, \mathrm{inv}, \mathrm{TERM} \subseteq \mathcal{P}(A)$ halmazok[1].

A relációk megadásakor infix jelölést alkalmazunk, ezért bevezetjük az alábbi *jelöléseket*. Zárójelben megadjuk azt is, hogy hogyan olvassuk azt, ha egy halmaz vagy egy halmazpár eleme az adott relációnak. Az állapottér részhalmazait logikai relációkkal jellemezzük.

Jelölések:

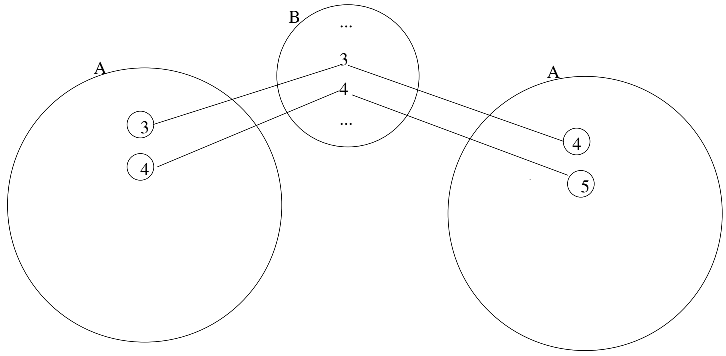
$P \rhd Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \rhd$ (P stabil feltéve, hogy nem Q),
$P \mapsto Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \mapsto$ (P biztosítja Q-t),
$P \hookrightarrow Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \hookrightarrow$ (P-ből elkerülhetetlen Q),
$Q \hookrightarrow \mathrm{FP} ::= \lceil Q \rceil \in \mathrm{TERM}$ (Q-ból a program biztosan fixpontba jut),
$\mathrm{FP} \Rightarrow R ::= \lceil R \rceil \in \mathrm{FP}$ (R teljesül fixpontban),
$\mathrm{inv}P ::= \lceil P \rceil \in \mathrm{inv}$ (P invariáns).
$Q \in \mathrm{INIT} ::= \lceil Q \rceil \in \mathrm{INIT}$ (Q igaz kezdetben),

2.1. Példa (Specifikációs reláció). *Legyen az állapottér $A ::= \mathcal{N}$ egyelemű direkt szorzat. Az egyetlen komponenshez tartozó változót jelöljük i-vel. Legyen $\rhd ::= \{(\lceil i = k \rceil, \lceil i = k+1 \rceil) | k \in \mathcal{N}\}$. Ekkor pl. az $i = 5 \rhd i = 6$ feltétel azt a kikötést fogalmazza meg, hogy a program az $a = (5)$ állapotból csak az $a = (6)$ állapotba juthat. A teljes reláció azt a feltételegyüttest adja meg, amely megköveteli, hogy a program futása során az i változó értéke csak egyesével növekedhet. Megkönnyíti a relációk kezelését, ha egy-egy relációnak csak néhány eleme van. Ezért célszerű a reláció k paraméter szerinti felbontása egyelemű relációkra, erre a célra vezetjük majd be a paramétertér fogalmát.* $\square$

A programra, mint az állapottér feletti mozgásra vonatkozó specifikációs feltételeket négy csoportra osztjuk aszerint, hogy milyen típusú kikötéseket fogalmazunk meg segítségükkel[2]. *A relációcsoportok és az egyes specifikációs relációk elnevezései azt tükrözik, hogy az adott reláció segítségével milyen jellegű feltételeket kívánunk megfogalmazni.* A specifikációs relációk pontos szemantikáját az adja meg, hogy egy program mikor felel meg egy adott relációhoz tartozó feltételnek. Ennek megfogalmazásához szükséges az absztrakt program (3.15. def.)

[1] unáris relációk

[2] A kikötések teljesülését - az invariánsok kivételével - általában nem a teljes állapottér felett vizsgáljuk majd meg, hanem csak az állapottér egy olyan részhalmaza felett, amely tartalmazza az összes elérhető állapotot.



2.1. ábra. $\rhd ::= \{(\lceil i = k \rceil, \lceil i = k+1 \rceil) | k \in \mathcal{N}\}$

és a megoldás (4.1. def.) definíciójának ismerete[3]. Az alábbiakban röviden és informálisan már most megadjuk az egyes feltételek jelentését.

- A $P \rhd Q$ és az inv$P$ alakú feltételeket *biztonságossági feltétel*eknek nevezzük. Ha a program állapotára teljesül a $P \wedge \neg Q$ feltétel, akkor $P \rhd Q$ tiltja, hogy a program Q érintése nélkül közvetlenül egy $\neg P \wedge \neg Q$-beli állapotba jusson. inv$P$ pedig kiköti, hogy a P feltétel igazsághalmazából a program minden elemi lépése a P igazsághalmazába vigyen, valamint, hogy P „kezdetben”[4] is teljesüljön.

- A $P \mapsto Q$, illetve $P \hookrightarrow Q$ *haladási feltétel*ek előírják, hogy ha a program egy P-beli állapotba jut, akkor abból előbb - utóbb Q-ba jusson. $P \mapsto Q$ további megszorítást tesz a haladási irányra. $Q \hookrightarrow$ FP kikötésnek megfelelő program előbb-utóbb *biztosan fixpontba jut* Q-beli állapotából.

- A FP $\Rightarrow R$ *fixpont feltétel*ekkel szükséges feltételeket fogalmazunk meg arra, hogy mi teljesüljön, ha a program fixpontba jut.

- Elégségesnek tekintjük, ha $Q \in$ INIT *kezdeti feltétel*ekkel meghatározott állapotokból indítva helyesen működik a program.

2.1. Megjegyzés. Stabilitási feltétel*nek nevezzük P-t , ha $P \rhd$ Hamis.* P konstans feltétel*, ha $\neg P$ is és P is stabilitási feltétel.*

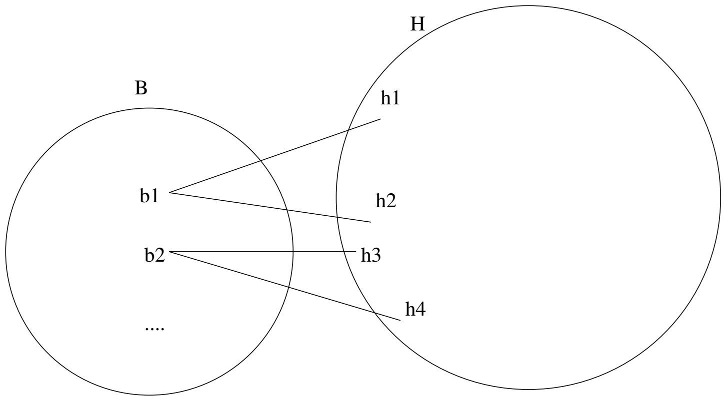
[3]A feladat szemantikáját a specifikációs relációk segítségével meg tudjuk adni oly módon, hogy bármely feladat függetlenül leírható bármely azt megoldó vagy meg nem oldó programtól, azaz a feladatok a programoktól független szemantikával rendelkeznek a modellben.

[4]A kezdeti értékadás végrehajtása után.

A továbbiakban a $\rhd, \mapsto, \hookrightarrow, \mathrm{FP}, \mathrm{INIT}, \mathrm{inv}, \mathrm{TERM}$ relációkat *specifikációs relációk*nak, elemeiket *specifikációs feltétel*eknek, ezen belül az $\rhd$, $\hookrightarrow$, $\mapsto$, inv, TERM relációk elemeit *átmenetfeltétel*eknek, az $\mathrm{INIT}, \mathrm{FP}$ relációk elemeit pedig *peremfeltétel*eknek nevezzük. Az INIT reláció a *környezeti előírás*ok csoportjába tartozik.

2.2. A programozási feladat definíciója

2.1. Definíció (Programozási feladat). *Legyen A egy állapottér, B pedig egy tetszőleges, megszámlálható halmaz. Rendeljünk hozzá a $b \in B$ pontokhoz rendezett reláció heteseket. Minden egyes rendezett hetes kettő, peremfeltételeket megadó, illetve öt, átmenetfeltételeket leíró relációt tartalmaz.*
Az $F \subseteq B \times (\underset{i\in[1..3]}{\times} \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A)) \underset{i\in[1..4]}{\times} \mathcal{P}(\mathcal{P}(A)))$ relációt az A állapottér felett definiált feladat*nak, B-t pedig a feladat* paraméterteré*nek nevezzük.*
A $\underset{i\in[1..3]}{\times} \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A))$ és $\underset{i\in[1..4]}{\times} \mathcal{P}(\mathcal{P}(A))$ direktszorzat $b \in B$-hez rendelt $h \in F(b)$ elemének komponenseit rendre $\rhd_h, \mapsto_h, \hookrightarrow_h$, TERM_h, FP_h, inv_h, $INIT_h$-val jelöljük. Ha $F(b)$ egyelemű, akkor h helyett b-t írunk vagy a h indexet teljesen elhagyjuk, ha ez nem okoz félreértést.



2.2. ábra. $h1 = (\rhd_{h1}, \mapsto_{h1}, \hookrightarrow_{h1}, \mathrm{INIT}_{h1}, \mathrm{FP}_{h1}, \mathrm{inv}_{h1}, \mathrm{TERM}_{h1})$

2.2. Példa (Programozási feladat). *Példaként megadjuk az elemenkénti feldolgozás feladatának specifikációját:*
A specifikációs relációk közül négy üres, három pedig egyelemű. Kikötjük, hogy

bármely fixpontban az y rendezett halmaz-m-es értéke éppen $f(x')$ legyen ((2.3.) feltétel), ahol x' az x változó kezdeti értéke ((2.1.) feltétel). Megköveteljük, hogy a program biztosan elérje valamelyik fixpontját ((2.2.) feltétel). A feladat megfogalmazásában az f függvény argumentuma, mint a specifikációs feltételek paramétere jelenik meg.

$A = X \times Y, x : X, y : Y,\ B = X, x' : X.$

$$(x = x') \in INIT_{x'} \tag{2.1}$$

$$Igaz \hookrightarrow FP_{x'} \tag{2.2}$$

$$FP_{x'} \Rightarrow y = f(x'), \tag{2.3}$$

ahol f elemenként feldolgozható.

Az X, Y típus specifikációja, az elemenként feldolgozható függvény fogalma és a megoldó program megtalálható a 8. fejezetben. □

A paramétertér helyes megválasztásával elérhetjük, hogy a $\rhd_h$, $\hookrightarrow_h$, FP_h, stb. relációk végesek legyenek, vagy éppen csak egyetlen egy halmaz ill. halmazpár legyen az elemük. Ha a B paramétertér végtelen, akkor így összességében végtelen sok relációt adunk meg. Ezek a relációk azonban általában csak a b paraméter értékében különböznek egymástól, így a megoldás definícióját elegendő lesz egyetlen $b \in B$ paraméteres esetre megvizsgálni.

A paramétertér bevezetésével könnyen megfogalmazhatunk olyan feladatokat, amelynek megoldása több alternatív viselkedésminta szerint is lehetséges[5].

2.2. Megjegyzés. *A paramétertér általában maga is az állapottérhez hasonló direktszorzat. Sok esetben van az állapottérnek és a paramétertérnek nem üres, közös altere. A paramétertér projekcióit is változóknak nevezzük, ezeket a változókat megkülönböztetésül ′ jellel egészítjük ki, pl.: v'.*[6]

2.3. Megjegyzés. *A feladat fenti definíciója a [Fót 83, Fót Hor 91]-ben ismertett specifikációs módszer általánosítása. Legyen $\forall b \in B : |F(b)| = 1$ és $Q_b \in \mathrm{TERM}_b$, $\{Q_b\} = INIT_b$ és $\{R_b\} = FP_b$.*

2.4. Megjegyzés. *A specifikációs feltételek szintaktikus alakjától a feladat, mint reláció független. Lényegében ugyanazt a feladatot (2.5. def.) azonban több ekvivalens specifikációs feltételhalmazzal is megfogalmazhatjuk.*

[5] Ha a feladat determinisztikus, akkor megfogalmazhatnánk a feladatot a paramétertér bevezetése nélkül is, mint a $\times_{i\in[1..3]} \mathcal{P}(\mathcal{P}(A)\times\mathcal{P}(A)) \times_{i\in[1..4]} \mathcal{P}(\mathcal{P}(A))$ direktszorzat elemét. Ebben az esetben azonban a feladat egyes komponenseinek számossága kezelhetetlenül nagy lehet, pl. egy-egy átmenetfeltételnek általában végtelen sok halmazpár eleme lenne.

[6] Elsőrendű temporális logikai nyelvekben szokás az állapottér változóit lokális változóknak, a paramétertér változóit globális vagy rigid változóknak nevezni.

Az 4.1. definícióban megadjuk, hogy az F feladatnak mikor *megoldás*a egy program, azaz mikor elégíti ki a feladatban előírt feltételeket[7].

2.3. Feladat kiterjesztése

Legyen A_1 az A altere. F az A_1 állapottér és B paramétertér felett definiált feladat. Az F A térre vett kiterjesztése az az F' feladat, amely a kiegészítő altér változóira nem tesz kikötéseket és az A_1 altérre vett vetülete megegyezik F_1-gyel.

Ha P szerepel az F feladat egy specifikációs feltételében, akkor a kiterjesztett feladat megfelelő specifikációs feltételében egy olyan P' logikai függvény szerepel, amelynek A_1-re vett vetülete P és nem függ a kiegészítő altér változóitól.

2.2. Definíció (Logikai függvény kiterjesztése). *Jelöljük P'-vel a P altéren definiált* logikai függvény *teljes térre való* kiterjesztését*. P' igazsághalmaza az a legbővebb halmaz, amelynek vetülete P igazsághalmaza.*

2.3. Definíció (Feladat kiterjesztése). *Legyen A_1 az A altere. F az A_1 állapottér és B paramétertér, F' pedig az A tér és a B paramétertér felett definiált feladat. F'-t az F kiterjesztésének nevezzük, ha $\forall b \in B : pr_{A_1}(F'(b)) = F(b)$ és F' specifikációs feltételeiben előforduló logikai függvények az F specifikációs feltételeiben adott logikai függvények kiterjesztései*[8].

2.4. A feladat finomítása

Célunk, hogy a modell eszközei segítségével a feladat specifikációját helyettesíteni tudjuk olyan feladatok specifikációival, amely feladatok megoldása esetén a rendelkezésre álló matematikai eszközökkel belátható az eredeti feladat megoldásának helyessége [Var 81, Fót Hor 91, Cha Mis 89, Bac Ser 90, Mor 87]. Arra törekszünk, hogy a megoldás előállításával párhuzamosan a megoldás helyességének bizonyítását is előállítsuk.

2.5. Megjegyzés. *A lépésenkénti finomítás szokásos megfogalmazásától eltérően nem a megoldó programot finomítjuk [Bac Ser 90]*[9]*. A feladat finomításának elve*

[7] Azt mondjuk, hogy az S program megoldja az F feladatot, ha $\forall b \in B : \exists h \in F(b)$, hogy az S program *megfelel* a h-ban adott $\mathrm{inv}_h P$, $P \rhd_h U$, $P \mapsto_h U$, $P \hookrightarrow_h U$, $\mathrm{FP}_h \Rightarrow R$, $Q \in \mathrm{TERM}_h$ alakú specifikációs feltételek mindegyikének a $Q \in \mathrm{INIT}_h$ kezdeti feltételek mellett.

[8] A 2.3. def. a szekvenciális modell [Fót 83, Fót 88] kiterjesztési definíciójának általánosítása.

[9] Egy program finomítása egy másiknak, ha minden olyan specifikációnak megfelel, amelyiknek az eredeti program is megfelelt [Bac Ser 90].

különbözik Morris [Mor 87], ill. Lamport [Lam 91] felfogásától is, mert ezekben a modellekben magát a programot is specifikációs eszköznek tekintik és ennek megfelelően finomítják a specifikációt.

A specifikáció finomításának leggyakoribb módja az állapottér bővítése, a régi és új komponensekre további, általában a korábbiaknál szigorúbb feltételek megfogalmazása.

Azt, hogy egy feladat mikor finomítása egy másiknak egy rögzített állapottér felett, a program (3.15. def.) és a megoldás (4.1. def.) definíciójának felhasználásával indirekt úton adjuk meg. Ez a definíciós módszer alkalmas arra, hogy a feladatok lépésenkénti finomítása során az egyes lépéseink helyességét formálisan is igazoljuk[10]. A feladatok felett értelmezett finomítás relációt tehát a feladatok és programok között értelmezett megoldás reláció indukálja.

2.4. Definíció (Feladat finomítása). *Azt mondjuk, hogy az F_1 feladat* finomítás*a az F_2 feladatnak, ha minden olyan S program, ami megoldása az F_1 feladatnak az megoldása az F_2 feladatnak is.*

2.3. Példa (Feladat finomítása). *Az alábbi specifikáció finomítása a (2.1.)-(2.3.) feltételekkel megadottnak.*

$$(x = x') \in INIT_{x'} \tag{2.4}$$

$$Igaz \hookrightarrow FP_{x'} \tag{2.5}$$

$$FP_{x'} \Rightarrow \forall i \in [1..n] : (x_i = \emptyset) \tag{2.6}$$

$$inv_{x'}(\forall j \in [1,m] : (y_j \cup f_j(x_1,...,x_n) = f_j(x'_1,...,x'_n))) \tag{2.7}$$

$$inv_{x'}(\forall j \in [1,m] : (y_j \cap f_j(x_1,...,x_n) = \emptyset)) \tag{2.8}$$

$$inv_{x'}(\forall i,j \in [1,n] : (x'_i \setminus x_i) \cap x_j = \emptyset), \tag{2.9}$$

ahol f elemenként feldolgozható.

Annak bizonyítása, hogy a fenti specifikáció valóban finomítása a (2.1.)-(2.3.) feltételekkel megadottnak megtalálható a 8. fejezetben. □

2.5. Definíció (Ekvivalens feladat). *Azt mondjuk, hogy az F_1 feladat* ekvivalens *az F_2 feladattal, ha az F_1 finomítása az F_2-nek és az F_2 finomítása az F_1-nek.*

[10] A programok felett értelmezett finomítás reláció is a megoldás fogalmához kötött [Bac Ser 90, Mor 87], és a finomítást támogató kalkulus alapja.

2.6. Megjegyzés (Absztrakt feladat). *Nevezzük a most bevezetett ekvivalencia-reláció által létrejött ekvivalenciaosztályokat* absztrakt feladat*nak. A most bevezetett ekvivalenciareláció indukál egy homomorfizmust a feladatokról az absztrakt feladatokra*[11].

Négyféle módon finomítjuk a feladat matematikai modelljét:

- az állapottér komponenseit finomítjuk és mint altereket tekintjük őket,
- az állapottér alterein fogalmazunk meg feladatokat,
- az állapotteret további komponensekkel bővítjük, a hozzájuk tartozó változókra kikötéseket teszünk,
- *állapottér transzformáció*t [Fót 86], vagy más néven koordinátatranszformációt [Dij Sch 89] alkalmazunk.

2.7. Megjegyzés. *Ha el akarjuk dönteni, hogy egy feladat finomítása-e egy másiknak abban az esetben, amikor a két feladat állapottere különbözik, akkor a két feladat állapotterét feleltessük meg egy kiválasztott állapottér egy-egy alterének és adjunk meg egy-egy olyan függvényt, amelyik a kiválasztott altér hatványhalmazára a feladat állapotterének hatványhalmazát leképezi. Ezek a leképezések definiálják a feladatok megfelelőit az új állapottér alterein. A feladatokat ezek után kiterjeszthetjük a közös állapottérre. Legtöbbször a választott állapottér a két feladat egyikének állapottere, a másik feladat állapottere a közös tér egy altere, a leképezés pedig az identitás.*

2.8. Megjegyzés. *Feladatok specifikációjának finomításakor támaszkodunk a* nyitott specifikáció *technikájára. Az állapottér egy alterén definiált részfeladat környezeti feltételeiként olyan biztonságossági-, haladási- és fixpontfeltételeket adunk meg, amelyek az altér felett specifikált komponens és környezete által együttesen alkotott* zárt rendszer*től [Jär 92] elvárt viselkedésre vonatkoznak (6. fejezet) [Col 94, Cha Mis 89]. Az alterekre vonatkozó feltételeket megkülönböztetésül felső indexszel jelöljük, pl.:* $\triangleright^E_h, \mapsto^E_h, \hookrightarrow^E_h$, TERM^E_h, FP^E_h, inv^E_h. *A részfolyamat egyes tulajdonságainak vizsgálatakor felhasználjuk a teljes rendszer (a külső környezet) ismert vagy feltételezett tulajdonságait [Cha Mis 89, Col 94].*

[11] Két absztrakt feladat pontosan akkor különbözik egymástól, ha az egyikhez található olyan megoldás, amelyik a másiknak nem megoldása.

A lépésenkénti finomítás során újabb és újabb részletekkel egészítjük ki a specifikációt, majd az utolsó lépésben előállítjuk a megoldó programot. A program előállítása általában egyszerű, a specifikáció finomítása nehezebb feladat. Minden egyes lépés után igazolnunk kell, hogy az új és részletesebb specifikációt megoldó program megoldja az eredeti feladatot is. A specifikáció finomítása során építjük be a megoldásba mindazt a tudást, amelyet a feladat elemzése során, vagy korábban szereztünk. A finomítás iránya kisebb vagy nagyobb mértékben befolyásolja, hogy milyen architektúrán implementálható hatékonyan a kapott megoldás és melyiken nem. Ezért akárcsak a szekvenciális programok levezetése során, a párhuzamos programok fejlesztésekor is előfordulhat, hogy visszatérünk egy korábban megfogalmazott specifikációhoz és más irányban folytatjuk a specifikáció finomítását.

3. fejezet

Párhuzamos absztrakt program

Az absztrakt párhuzamos program a UNITY-ből ismert programfogalom relációs alapú megfogalmazása. Ahhoz, hogy eldönthessük, hogy egy program megold-e egy feladatot (4. fejezet), össze kell vetnünk a feladatot definiáló relációt a program viselkedését leíró relációval. A programhoz annak viselkedési relációját hozzárendelő leképezést tekinthetjük úgy is, mint egy olyan szemantikai leképezést, amelynek absztrakciós szintje megegyezik az absztrakt feladat szemantikájának absztrakciós szintjével.

3.1. Az absztrakt program szerkezete

Az absztrakt program struktúrája nem eredményezheti, hogy olyan szinkronizációs kényszerek épüljenek be a megoldásba, amelyek feleslegesek, valódi párhuzamos architektúrán szükségtelenül lassítják a program futását. Ha a programot szekvenciális folyamatok halmazának tekintenénk, ahogy ezt pl. CSP-ben [Hoa 78] vagy Adában [ALRM 83] megszoktuk, akkor ezzel eleve végrehajtási sorrendet definiálnánk utasítások nagy részhalmazai felett. Ezért a párhuzamos programot feltételes értékadások halmaza segítségével adjuk meg. Az egyes utasítások bármikor végrehajthatóak, állapotváltozás azonban csak akkor következhet be, ha az értékadás feltétele teljesül. A feltételek helyes megválasztásával elérhetjük, hogy az állapotátmenetek a kívánt sorrendben következzenek be. A program tulajdonságok meghatározását is megkönnyíti, ha a programot utasítások halmazaként definiáljuk. Általában megköveteljük, hogy az egyes értékadások végrehajtása pillanatszerű, atomi legyen.

3.1. Példa (Absztrakt program megadása).
S::=(SKIP,{

$(s_1: \ x := x+1,$ *ha* $x \le y \parallel y := y+x)$,
$s_2: \ z := x+y\})$.$\Box$

Ha az 3.1. program utasításainak pillanatszerű végrehajtása biztosított, akkor az egyik lehetséges végrehajtási út mentén a következő állapotokat és állapotátmeneteket figyelhetjük meg: $(2,3,0)-(s_1)->(3,5,0)-(s_2)->(3,5,8)$. Ha az értékadások atomicitása nem biztosított, akkor a következő állapotátmenetsorozat is megfigyelhető: $(2,3,0)-->(3,3,0)-(s_2)->(3,3,6)-->(3,5,6)$, pedig nincs olyan érvényes állapot, amelyben $x+y=6$.

A nemdeterminisztikus végrehajtási sorrend korlátozására szinkronizációs feltételeket használunk (pl. termelő-fogyasztó esetén üres pufferből nem lehet fogyasztani). Az absztrakt program tehát szimultán feltételes értékadások (3.9. def.) [Fót 83, Hor 93] véges halmazával adható meg [Cha Mis 89], ahol az egyes értékadások jobboldalán függvénykompozíciók is szerepelhetnek (pl. kapcsos zárójellel megadott függvény).

3.1.1. A feltételes értékadás fogalma

3.1. Definíció (Utasítás). *Az* $s \subseteq A \times A^{**}$ *relációt* utasítás*nak nevezzük, ha*

- $\mathcal{D}_s = A$,
- $\forall a \in A \ : \ \forall \alpha \in s(a) \ : \ \alpha_1 = a$,
- $(\alpha \in \mathcal{R}_s \wedge \alpha \in A^{\infty}) \Rightarrow (\forall i \in N \ (\alpha_i = \alpha_{i+1} \rightarrow (\forall k (k>0) : \alpha_i = \alpha_{i+k})))$,
- $(\alpha \in \mathcal{R}_s \wedge \alpha \in A^{*}) \Rightarrow (\forall i \ (1 \le i < |\alpha|) : \alpha_i \ne \alpha_{i+1})$.

3.2. Definíció (Hatásreláció). *A* $p(s) \subseteq A \times A$ *reláció az* $s \subseteq A \times A^{**}$ *utasítás* hatásreláció*ja, ha*

- $\mathcal{D}_{p(s)} = \{a \in A | s(a) \subseteq A^*\}$,
- $p(s)(a) = \{b \in A | \exists \alpha \in s(a) \ : \ \tau(\alpha) = b\}$,

ahol $\tau : A^* \rightarrow A$ *függvény az* $\alpha = (\alpha_1, \ldots, \alpha_n) \in A^*$ *véges sorozathoz annak végpontját rendeli.* $\tau(\alpha) ::= \alpha_n$.

3.3. Definíció. .

- *Azt mondjuk, hogy az $v_i : A_i$ változó* konstans *függvény az s utasításban, ha* $\forall a \in A : \forall \alpha \in s(a) : (\forall \alpha_k \in \alpha : \alpha_{ki} = v_i(a))$.
- *Azt mondjuk, hogy az s utasítás végrehajtása* biztosan nem változtatja meg *az $v_i : A_i$ változót, ha $\forall a \in \mathcal{D}_{p(s)} : p(s)(a)_i = v_i(a)$.*

Elemi utasítás az értékadás és az üres utasítás.

3.4. Definíció (Üres utasítás, SKIP). Üres*nek nevezzük, és SKIP-pel jelöljük azt az utasítást, amire $\forall a \in A : SKIP(a) = \{(a)\}$.*

3.5. Definíció (Általános értékadás). *Legyen $A = A_1 \times \ldots \times A_n$, $F = (F_1, \ldots, F_n)$, ahol $F_i \subseteq A \times A_i$. Az s utasítás* általános értékadás *[Fót 83] , ha*
$s = \{(a, red(a,b)) | a, b \in A \wedge a \in \bigcap_{i \in [1,n]} \mathcal{D}_{F_i} \wedge b \in F(a)\} \cup$
$\{(a, (aaa\ldots)) | a \in A \wedge a \notin \bigcap_{i \in [1,n]} \mathcal{D}_{F_i}\}$, *ahol*
az $\alpha \in A^{**}$ redukáltj*ának nevezzük, és $red(\alpha)$-val jelöljük azt a sorozatot, amit úgy kapunk, hogy az α sorozat minden azonos elemekből álló véges részsorozatát a részsorozat egyetlen elemével helyettesítjük.*

3.6. Definíció (Változó az értékadás baloldalán). *Azt mondjuk, hogy a $v_i : A \longmapsto A_i$ változó az* értékadás baloldalán áll*, az értékadás* értéket ad a v_i változónak*, ha az $F_i \subseteq A \times A_i$ reláció nem egyenlő a v_i projekcióval [Fót 86], azaz az értékadás hatásrelációja megváltoztatja a v_i változót (1.21. def.). Az s értékadás baloldalán álló változók halmazát $VL(s)$-sel jelöljük. Az értékadás azoknak a változóknak ad értéket, amelyek a baloldalán állnak*[1].

3.1. Megjegyzés (Szimultán értékadás). *Az értékadás egyszerre több változó értékét is megváltoztathatja, ezért ún.* szimultán *értékadásról van szó.*

3.7. Definíció (Egyszerű értékadás). *Ha legfeljebb egy változó áll az értékadás baloldalán, akkor* egyszerű *értékadásról beszélünk.*

3.8. Definíció (Változó az értékadás jobboldalán). *Azt mondjuk, hogy a $v_i : A \longmapsto A_i$ változó az értékadás* jobboldalán áll*, ha az értékadás hatásrelációja nem független (1.20. def.) az A_i állapottérkomponenstől. Az s értékadás jobboldalán álló változók halmazát $VR(s)$-sel jelöljük.*

---

[1] A 3.6. definíció azokat a változókat nevezi az értékadás baloldalán állónak, amelyek az értékadás végrehajtása során megváltozhatnak, azaz van olyan $a \in A$ állapot, amelyre $a_i = v_i(a) \neq v_i \circ F(a) = F_i(a)$. A definíció tehát független az értékadás szintaktikus alakjától.

3.9. Definíció (Feltételes értékadás). *Legyen* $A = A_1 \times \ldots \times A_n$, $F = (F_1, \ldots, F_n)$, *ahol* $F_i \subseteq A \times A_i$. *Legyen* $\lceil \pi_i \rceil ::= \mathcal{D}_{F_i}$. $F_i|_{Igaz}$ *az* F_i kiterjesztése a Igaz feltételre nézve[2]*:* $F_i|_{Igaz}(a) = F_i(a)$*, ha* $a \in \lceil \pi_i \rceil$ *és* $F_i|_{Igaz}(a) = a_i$*, különben. Az* $F||_{Igaz} = (F_1|_{Igaz}, \ldots, F_n|_{Igaz})$ *relációval megadott általános s értékadást* feltételes értékadás*nak nevezzük, ha* $\forall a \in A : |p(s)(a)| < \omega$.

3.2. Megjegyzés. *A* v_i *változóra vonatkozó egyszerű, determinisztikus, feltételes értékadást a* $(v_i := F_i(v_1,..,v_n)$*, ha* $\pi_i(v_1,...,v_n))$ *alakban adjuk meg. Röviden:* $(v_i := F_i$*, ha* $\pi_i)$*-vel jelöljük. Szimultán, nemdeterminisztikus, feltételes értékadás megadható a* $(v_i :\in F_i(v_1,..,v_n)$*, ha* $\pi_i) \parallel (v_k :\in F_k(v_1,..,v_n)$*, ha* $\pi_k)$ *alakban. Ha lehetséges, akkor sok változó esetén a* $\underset{i\in[1,n]}{\parallel}(\ldots)$ *rövidítést alkalmazzuk.*

Ha a feltételes értékadásban szereplő valamelyik egyszerű értékadáshoz rendelt egyetlen feltétel egy $a \in A$ állapothoz *hamis* értéket rendel, akkor ez a 3.9. definíció szerint annak a rövid megfogalmazása, hogy az értékadás az a pontból indítva nem változtatja meg a baloldalon álló változó értékét. Ezáltal a feltételes értékadások mindenütt értelmezve vannak az állapottér felett.

3.2. Állapotátmenetfák

Az absztrakt programot egy olyan bináris relációként definiáljuk, amelyik egy kezdeti feltételes értékadás hatásrelációja, illetve véges sok feltételes értékadás hatásrelációjának diszjunkt uniója által generált fák ekvivalenciaosztályait rendeli az állapottér egyes pontjaihoz (3.15. def.).

3.10. Definíció (Címkézett állapotátmenetfa). *A címkézett állapotátmenetfa egy* (r,N,V,L,S) *rendezett ötös, ahol* r *a fa gyökere,* N *a csúcsok halmaza,* $V \subseteq N \times N$ *az élek halmaza,* $L : N \longmapsto A$ *a gráf csúcsaihoz állapotokat rendelő címkefüggvény,* $S : V \longmapsto J$ *az élekhez természetes számokat rendelő címkefüggvény,* $\forall x \in N : (x,r) \notin V$ *és pontosan egy út vezet* r*-ből minden* $x \in N$ *csúcsba.*

3.11. Definíció (Izomorf állapotátmenetfák). *Izomorfnak mondjuk a* $G_1 = (r_1,N_1,V_1,L_1,S_1)$ *és a* $G_2 = (r_2,N_2,V_2,L_2,S_2)$ *fát, ha van olyan* $f : N_1 \longmapsto N_2$ *bijekció, amelyre* $\forall x \in N_1 : f(V_1(x)) = V_2(f(x)) \wedge L_1(x) = L_2(f(x))$ *és* $\forall x,y \in N_1 : (x,y) \in V_1 \Leftrightarrow (f(x),f(y)) \in V_2$ *és* $\forall (x,y) \in V_1 : S_1(x,y) = S_2(f(x),f(y))$.

[2] Általánosabban egy *R reláció* π *feltételre vonatkozó kiterjesztését* az alábbi módon definiálhatjuk:
Legyen B altere A-nak. $pr_B : A \longmapsto B$. A pr_B függvény az A-beli pontokhoz B-beli vetületüket rendeli hozzá [Fót 83]. $R \subseteq A \times B$. $R|_\pi ::= (R \cap (\lceil \pi \rceil \times B)) \cup \{(a, pr_B(a)) | a \in \lceil \pi \rceil \setminus \mathcal{D}_R\}$.

3.1. Tétel. (Az izomorfia reláció ekvivalenciareláció) *Az 3.11. definícióban megfogalmazott izomorfia reláció ekvivalenciareláció az A felett generált fák halmazán.*

Biz.: A reláció reflexív, mert minden fához létezik adott tulajdonságú leképezés, az identitás. A reláció szimmetrikus, mert a bijekció inverze rendelkezik az adott tulajdonságokkal, ha a bijekció rendelkezett vele. Végül a reláció tranzitív, mert két adott tulajdonságú bijekció kompozíciója is rendelkezik a megkövetelt tulajdonságokkal. □

3.12. Definíció (Állapotátmenetfák ekvivalenciaosztályai). *A^{***} jelölje az A felett generált fák ekvivalenciaosztályainak halmazát.*

3.13. Definíció (Generált állapotátmenetfa). *Legyenek az $R_0, R \subseteq A \times A$ relációk mindenütt értelmezve az A állapottér felett, azaz $\mathcal{D}_R = \mathcal{D}_{R_0} = A$. $J \subset \mathcal{N}_0$. Tetszőleges $a \in A$ pontra az (R_0, R) relációpár által az a ponthoz generált fának nevezzük a $GR(a) = (r, N_a, V_a, L_a, S_a)$ irányított fát, ha*

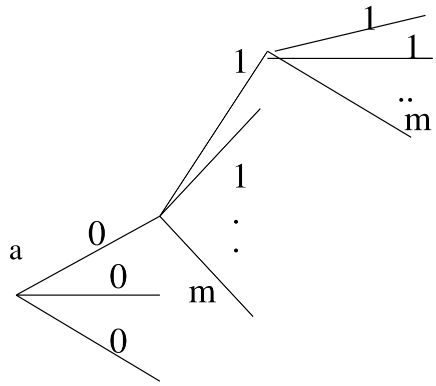
- $L_a(r) = a$,
- $\forall x \in N_a \setminus \{r\} : L_a(V_a(x)) = R(L_a(x))$,
- $L_a(V_a(r)) = R_0(L_a(r)) = R_0(a)$.

Reprezentáljuk az R reláció által az a ponthoz generált gráfok ekvivalenciaosztályait annak egy elemével.

Legyen $S = (s_0, \{s_1, ...s_m\})$ egy s_0 feltételes értékadás és véges sok feltételes értékadás nem üres halmazának rendezett párja. J={1,..,m}, $UP(S)$ a $p(s_j)$ relációk diszjunkt uniója.

3.14. Definíció (Helyesen címkézett állapotátmenetfa). *Az $(p(s_0), UP(S))$ relációpár által generált fa címkézése helyes, ha minden r-ből induló él címkéje* 0 *és minden j-vel címkézett élre amely a-val címkézett csúcsból b-vel címkézett csúcsba mutat teljesül, hogy $(a, b) \in p(s_j)$.*

3.2. Lemma. (Helyes címkézés és ekvivalencia) *Ha egy állapotátmenetfa címkézése helyes, akkor a vele ekvivalens állapotátmenetfák címkézése is helyes.*
Biz.: a 3.11. def. következménye. □



3.1. ábra. Címkézett állapotátmenetfa

3.15. Definíció (Absztrakt program). *Absztrakt programnak nevezzük az $UPG(S) \subseteq A \times A^{***}$ relációt, ha az állapottér pontjaihoz a $(p(s_0), UP(S))$ relációpár által generált azon állapotátmenetfák ekvivalenciaosztályait rendeli, amelyek címkézése helyes. Az $S = (s_0, \{s_1, ...s_m\})$ által definiált $UPG(S)$ programot röviden S-sel jelöljük, és azt mondjuk, hogy $s_j \in S$, ha $j \in J$.*

3.3. Megjegyzés (Műveleti szemantika). *A 3.15. definíció megadja a párhuzamos absztrakt program szemantikai jelentését. Ez a szemantika műveleti jellegű, a programnak megfelelő gráf valójában a műveleti szemantika címkézett állapotátmenet gráfjával azonosítható (9. fejezet). Megjegyezzük, hogy ez a szemantika olyan programok között is különbséget tesz, amelyek egyaránt megoldásai (4.1. def.) ugyanannak a feladatnak, de más-más végrehajtási utakat definiálnak.*

3.16. Definíció (Végrehajtási út). *A $b \in UPG(S)(a)$ ekvivalenciaosztály reprezentánsának bármelyik útját* végrehajtási út*nak nevezzük.*

3.17. Definíció (Elérhető állapotok halmaza). *Az $UPG(S)(a)$ halmaz elemeinek végrehajtási útjain elhelyezkedő csúcsok címkéinek halmazát jelöljük $E(S)(a)$-val. $E(S)(a)$ az a* állapotból elérhető állapot*ok halmaza*[3].

3.18. Definíció (Absztrakt program változói). *Jelöljük $VL(S)$-sel az S program utasításainak baloldalán álló változók (3.6. def.) halmazát, azaz $VL(S) ::= \bigcup_{s \in S} VL(s)$. Jelöljük a jobboldalon álló változók (3.8. def.) halmazát $VR(S)$-sel. $V(S) ::= VL(S) \cup VR(S)$.*

[3] Ezen állapotok az absztrakt program állapotátmenetfájában érhetőek el, de valamely ütemezési kikötés mellett a konkrét program által már nem feltétlenül elérhetőek.

Feltételezzük, ha több processzor hajtja végre a konkrét programot, akkor ez hatékonysági szempontoktól eltekintve hatásában megegyezik azzal, mintha egyetlen processzor válogatott volna valamilyen nemdeterminisztikus sorrendben az utasításhalmaz elemei közül. Megengedjük ugyan, hogy két vagy több processzor időben átfedve hajtsa végre ugyanazon utasítás különböző elemi lépéseit vagy különböző utasításokat, de az így kapott eredménynek meg kell egyeznie valamelyik eredménnyel azok közül, amelyet valamelyik végrehajtási út mentén egyetlen processzor állított volna elő. *Feltételezzük* tehát, hogy az absztrakt programot oly módon implementáljuk, hogy elemi építőköveire, a szimultán feltételes értékadásokra párhuzamos végrehajtás esetén teljesül a *sorbarendezhetőség* [Lam Lyn 90] követelménye. Egy-egy sorrendet egy-egy végrehajtási út ír le.

3.4. Megjegyzés. *Az 3.15. definíció alapján megállapíthatjuk, hogy a modellben szimultán feltételes értékadások* valós aszinkron párhuzamos *végrehajtását nem tudjuk kifejezni*[4]. *A modell programfogalma* összefésüléses *szemantikán (9. fejezet) alapszik. A szinkron párhuzamos végrehajtás leírására a szimultán értékadás alkalmas.*

Az utasítások halmazát sok esetben halmazműveletek segítségével állítjuk majd elő a megoldás logikai struktúrájának megfelelő modulokból. Egy-egy modul leírhatja például egy-egy objektum viselkedését [Cha Mis 89, Sin 91]. A modulok uniójaként vagy szuperpozíciójaként kapott program [Cha Mis 89] utasításait hatékonysági szempontok figyelembevételével képezhetjük le logikai vagy fizikai processzorokra. A modul tehát programtervezési, a folyamat pedig implementációs fogalom.

3.2.1. Utasítások kiterjesztése, szuperpozíciója

Definiáljuk az A állapottér egy altere felett definiált s utasítás s' kiterjesztettjét oly módon, hogy abban a kiegészítő altér változói ne álljanak egyetlen utasítás baloldalán és jobboldalán sem és a kiterjesztett utasítás A_1-re vett vetülete éppen s legyen [Fót 83, Fót 88].

3.19. Definíció (Utasítás kiterjesztése). *Legyen B altere az A állapottérnek, B' a B altér kiegészítő altere az A-ra.*
Az $s \subseteq B \times B^{**}$ utasítás kiterjesztése *A-ra:*
$s' = \{(a, \alpha) \in A \times A^{**} | (pr_B(a), pr_B(\alpha)) \in s \wedge \forall i \in \mathcal{D}_\alpha : pr_{B'}(\alpha_i) = pr_{B'}(a)\}$.

[4] Valós párhuzamosság esetén összetett feladatok megoldását általában nem lehet modulokból előállítani (7. fejezet)

3.3. Lemma. *Legyen az A_1 tér az A állapottér altere. Legyen az A_1 altér felett definiált s utasítás kiterjesztése az s' utasítás. Ekkor: $p(s) = pr_{A_1}(p(s'))$.*

Biz.: Az utasítás kiterjesztésének definíciója szerint $pr_{A_1}(s') = s$, így $p(pr_{A_1}(s')) = p(s)$. Felhasználva, hogy egy utasítás hatásrelációjának (3.2. def.) kiszámítása és a vetítés kommutatív, a lemma állításához jutunk. □

Gyakran alkalmazzuk egyes utasítások definíciójánál azt a módszert, hogy egy meglévő és ismert hatásrelációjú feltételes értékadást módosítunk.

3.20. Definíció (Értékadás kiegészítése feltétellel). *(s_j), ha π ::= $\underset{i\in[1,n]}{\|}$ $(v_i :\in F_{j_i}(v_1,..,v_n)$, ha $\pi_{j_i} \wedge \pi)$.*

3.4. Lemma. *$p((s_j)$, ha $\pi)$ = $(p(s) \cap (\lceil\pi\rceil \times A)) \cup (id_A \cap (\lceil\neg\pi\rceil \times A))$.*
Biz.: A 3.20. def. közvetlen következménye.□

3.21. Definíció (Feltételes értékadások szuperpozíciója). *Legyen s_1 és s_2 azonos állapottéren adott két feltételes értékadás és legyen $VL(s_2) \cap V(s_1) = \emptyset$. Ekkor $s_1 \| s_2$::=*
$\underset{v_i\notin VL(s_2)}{\|}(v_i :\in F_{1i}(v_1,..,v_n)$, *ha* $\pi_{1i})$ $\underset{v_i\in VL(s_2)}{\|}(v_i :\in F_{2i}(v_1,..,v_n)$, *ha* $\pi_{2i})$.

3.5. Megjegyzés. *Tegyük fel, hogy $u : A \mapsto A_{i'}$ nem szerepel az s_j értékadás baloldalán (3.6. def.).*
Ekkor az előző definíció értelmében speciális esetként definiálhatjuk az s_j feltételes értékadás és a $(u := F_{i'}$, ha $\pi_{j_{i'}})$ egyszerű értékadás szuperpozícióját: $s_j \| (u := F_{i'}$, ha $\pi_{j_{i'}})$ ::=
$\underset{i\in([1,n]\setminus\{i'\})}{\|}(v_i :\in F_{j_i}(v_1,..,v_n)$, *ha* $\pi_{j_i} \wedge \pi) \| (u := F_{i'}$, *ha* $\pi_{j_{i'}})$.

3.5. Lemma. (Szuperpozíció hatásrelációja) *Legyen s_1 és s_2 azonos állapottéren adott két feltételes értékadás és legyen $VL(s_2) \cap V(s_1) = \emptyset$. Ekkor $p(s_1 \| s_2) = p(s_1) \circ p(s_2)$.*

Biz.: Jelöljük $id \subseteq A \times A$-val azt a relációt, amelyik minden ponthoz önmagát rendeli ($id_i \subseteq A_i \times A_i$). Legyen $\forall a \in A : p(s)_i(a)$::=$(p(s)(a))_i$.
$\forall v_i \notin VL(s_2) : p(s_2)_i = id_i$, így $\forall v_i \notin VL(s_2) : (p(s_1) \circ p(s_2)(a))_i = (p(s_1) \circ id)_i = p(s_1)_i$. $\forall v_i \in VL(s_2) : v_i \notin V(s_1)$, tehát $p(s_1)_i = id_i$, így $\forall v_i \in VL(s_2) : (p(s_1) \circ p(s_2)(a))_i = (id \circ p(s_2))_i = p(s_2)_i$. □

3.2.2. Program kiterjesztése

Definiáljuk az S_1 program A-ra való kiterjesztettjét utasításonként.

3.22. Definíció (Program kiterjesztése). *Legyen az A_1 és A_2 tér az A állapottér két egymást kiegészítő altere. Legyen S program az A_1 altér, S' az A tér felett definiálva. Az S' programot az S program A-ra való kiterjesztésének nevezzük, ha minden utasítása kölcsönösen egyértelműen megfeleltethető az S program egy utasítása kiterjesztésének.*

3.6. Megjegyzés. *A 3.3. lemma és az absztrakt program definíciója alapján könnyen belátható, hogy*
$\forall a \in A : pr_{A_2}(E(S')(a)) = \{pr_{A_2}(a)\}$ *és* $pr_{A_1}(UPG(S')) = UPG(S)$.

3.3. Pártatlan ütemezés fogalma

A megoldás definíciója kimondja majd, hogy a feladatban megfogalmazott feltételeknek csak azokra a végrehajtási utakra kell teljesülni, amelyekre teljesül a feltétlenül pártatlan ütemezés axiómája.

3.23. Definíció (Feltétlenül pártatlan ütemezés). *Egy végrehajtási útról azt mondjuk, hogy teljesül rá a* feltétlenül pártatlan ütemezés *axiómája, ha az út mentén a feltételes értékadások halmazából minden utasítás végtelen sokszor kerül kiválasztásra, azaz a címkefüggvény az út mentén a J indexhalmaz minden elemét végtelen sokszor rendeli az élekhez*[5].

Ha a konkrét, adott architektúrára leképezett programra teljesül, hogy feltétlenül pártatlan ütemezés mellett kerül végrehajtásra, akkor a többi utat valóban nem kell figyelembe venni a megoldás helyessége szempontjából. Ebben az esetben a program működése nemdeterminisztikus módon kiválasztott transzformációk iterációjával írható le, ahol a nemdeterminisztikusság véges de nem korlátos hasonló értelemben, ahogy relációk nem korlátos lezártjáról beszéltünk [Fót 83, Hor 90]. Több utasítás közül ugyanaz az utasítás véges, de nem korlátos sokszor nem kerül kiválasztásra közvetlenül egymás után.

[5]Megkövetelhetünk azonban kevesebbet is, pl.: hogy a feladatban megfogalmazott feltételeknek csak azokra a végrehajtási utakra kell teljesülni, amelyekre teljesül az utófeltételekre vonatkozóan pártatlan ütemezés axiómája.

Utasítások pártatlan kiválasztására sokféle feltételt lehet megfogalmazni [Fra 86, And 91], ezek közül az egyik leggyengébb a pártatlan ütemezés axiómája. Így feltétlenül pártatlan ütemezés mellett jól működő programok az általában szigorúbb ütemezési feltételek esetén is megoldják a feladatot.

Az ún. *őrfeltétel*ek [Dij 75, Hoa 78, And 91] hasonlítanak a feltételes értékadásokban szereplő π_i feltételekhez. A feltételes értékadások hatásrelációi azonban akkor is definiáltak, ha az adott állapotra a feltétel nem teljesül. Ezért a generált fában mindig megjelenik az értékadás hatásrelációjának megfelelő él. A hamis őrfeltételű műveletek azonban nem generálnak éleket.

- *Feltétlenül pártatlan*nak nevezünk egy ütemezést, ha minden őrfeltételhez nem kötött és végrehajtásra váró elemi művelet előbb-utóbb végrehajtásra kerül (3.23. def.).

- *Gyengén pártatlan* egy ütemezés, ha feltétlenül pártatlan és minden olyan művelet, amelynek őrfeltétele igazzá válik és igaz is marad, előbb-utóbb végrehajtásra kerül.

- *Szigorúan pártatlan* egy ütemezés, ha feltétlenül pártatlan és minden olyan elemi művelet, amely végrehajtásra vár és őrfeltétele végtelen sokszor igaz, előbb-utóbb végrehajtásra kerül.

3.24. Definíció (Utófeltételekre pártatlan ütemezés). *Nem teljesül egy végrehajtási útra az* utófeltételekre vonatkozóan pártatlan[6] ütemezés *axiómája, ha*
- egy adott pontjától kezdődően minden pontjában kiválasztható olyan utasítás, amely az adott pontnak megfelelő állapotból egy adott logikai függvény igazsághalmazába visz és
- sohasem kerül ilyen utasítás kiválasztásra.

A feltételes értékadások halmazaként definiált absztrakt program nem tartalmaz őrfeltételeket. A továbbiakban feltételezzük, hogy az implementált program végrehajtása feltétlenül pártatlan.

3.7. Megjegyzés. *Belátható, hogy feltétlenül pártatlan ütemezéssel nem pártatlan ütemezés is modellezhető [Cha Mis 89].*

[6] gyengén pártatlan

3.4. Az absztrakt program tulajdonságai

Az absztrakt programok tulajdonságait az állapottér hatványhalmaza felett értelmezett relációkkal írjuk le. A könyv első részében már ismertetett leggyengébb előfeltétel fogalmát általánosítjuk és ennek segítségével határozzuk meg a programtulajdonságokat. Leggyengébb előfeltételt a program szövege alapján tudunk számolni, a tulajdonságok tehát statikusan, a program működésének vizsgálata nélkül meghatározhatóak.

3.4.1. A leggyengébb előfeltétel és általánosítása

Az absztrakt programok jellemzésekor támaszkodunk a leggyengébb előfeltétel [Dij 76, Fót 83], a legszigorúbb utófeltétel [Lam 90], ill. a monoton leképezések fixpontjának (11. fejezet) fogalmára.

3.25. Definíció (Leggyengébb előfeltétel, legszigorúbb utófeltétel). *Legyen s egy utasítás, Q,R pedig logikai függvények az A állapottér felett. A $lf(s,R) : A \longmapsto \mathcal{L}$ logikai függvény az R utófeltétel s utasításra vonatkozó* leggyengébb előfeltétele*, ahol*
$\lceil lf(s,R)\rceil ::= \{a \in \mathcal{D}_{p(s)} \mid p(s)(a) \subseteq \lceil R\rceil\}$.
Az $sp(s,Q) : A \longmapsto \mathcal{L}$ logikai függvény a Q előfeltétel legszigorúbb utófeltétel*e az s-re nézve, ahol $\lceil sp(s,Q)\rceil ::= p(s)(\lceil Q\rceil)$.*

3.6. Lemma. (Leggyengébb előfeltétel alaptulajdonságai)

(1) $lf(s,Hamis) = Hamis$ (csoda kizárásának elve),

(2) Ha $\mathcal{D}_{p(s)} = \lceil Igaz\rceil$, akkor $lf(s,Igaz) = Igaz$,

(3) $\lceil lf(s,R)\rceil = \lceil R\circ p(s)\rceil$ (utófeltételbe helyettesítés módszere),

(4) Ha $P \Rightarrow Q$, akkor $lf(s,P) \Rightarrow lf(s,Q)$ (monotonitás),

(5) $lf(s,Q)\vee lf(s,R) \Rightarrow lf(s,Q\vee R)$ (gyenge additivitás),

(6) $lf(s,Q)\wedge lf(s,R) = lf(s,Q\wedge R)$ (multiplikativitás).

Biz.: Az állítások közvetlenül a 3.25. definícióból következnek. ((1), (3), (4), (5), (6) bizonyítása megtalálható [Fót 83, Fót Hor 91]-ben.) □

3.8. Megjegyzés. *A lemma (2)-es állítása az absztrakt programban előforduló utasításokra, a feltételes értékadásokra (3.9. def.) mindig teljesül.*

3.7. Lemma. (Kiterjesztés és leggyengébb előfeltétel) *Legyen R az A_1 altéren definiált logikai függvény, R' pedig az R logikai függvény kiterjesztése A-ra. s' jelölje az s utasítás A-ra vonatkozó kiterjesztését. Legyen a' egy tetszőleges olyan pont, amelyre $a = pr_{A_1}(a')$, Ekkor: $a \in lf(s,R) \iff a' \in (lf(s',R'))$ és $a \in sp(s,R) \iff a' \in (sp(s',R'))$*

Biz.: $a' \in lf(s',R') \iff a' \in \mathcal{D}_{p'(s)}$ és $p(s')(a') \subseteq R' \iff$ (a 3.3. lemma alkalmazásával) $\iff a \in \mathcal{D}_{p(s)}$ és $p(s)(a) \subseteq R \iff a \in lf(s,R)$. A legszigorúbb utófeltételre vonatkozó állítás ugyanígy bizonyítható. □

3.1. Következmény. *$lf(s,R)' = (lf(s',R'))$ és $sp(s,Q)' = (sp(s',Q'))$.*

3.8. Lemma. (Kiegészítés és leggyengébb előfeltétel) *Ha $P \Rightarrow lf(s,Q)$, akkor $P \vee \neg\pi \Rightarrow lf((s, \text{ha } \pi), Q \vee \neg\pi)$ és $P \wedge \pi \Rightarrow lf((s, \text{ha } \pi), Q)$. Ha $P \Rightarrow lf(s,P)$, akkor $P \Rightarrow lf((s, \text{ha } \pi), P)$.*

Biz.: Ha $a \in P \wedge \pi$, akkor $p(s, \text{ha } \pi)(a) = p(s)(a) \subseteq \lceil Q \rceil$ (3.4. lemma). Ha $a \in \neg\pi$, akkor $p(s, \text{ha } \pi)(a) = id_A(a) \subseteq \neg\pi$. □

3.9. Lemma (Szuperpozíció és leggyengébb előfeltétel). *Legyen Q, R egy-egy logikai függvény és $VR(Q) \cap VL(s_1) = \emptyset$, $VR(R) \cap VL(s_1) = \emptyset$. Ekkor $lf(s_1,Q) = Q$, ill. ha $R \Rightarrow lf(s,Q)$, akkor $R \Rightarrow lf(s \| s_1, Q)$.*

Biz.: A 3.6. lemmát többször alkalmazva bizonyítunk. A feltétel szerint $Q \circ p(s_1) = Q$, azaz $lf(s_1,Q) = Q \circ p(s_1) = Q$. A feltétel szerint $p(s_1)(\lceil R \rceil) = \lceil R \rceil$ és $\lceil R \rceil \subseteq \lceil Q \circ p(s) \rceil$, így $p(s_1)(\lceil R \rceil) \subseteq \lceil Q \circ p(s) \rceil$, azaz $p(s) \circ p(s_1)(\lceil R \rceil) \subseteq \lceil Q \rceil$. A 3.5. lemma szerint $p(s \| s_1, Q) = p(s) \circ p(s_1)$, így éppen a lemma állítását kaptuk. □

A továbbiakban jelöljön S egy absztrakt programot (3.15. def.), az állapottér legyen $A ::= \times_{i \in [1..n]} A_i$. $S = (s_0, \{s_1, ... s_m\})$, ahol s_0 és $\forall s_j \in S$ egy (szimultán, nemdeterminisztikus) feltételes értékadás.

$s_j : \|_{i \in [1,n]} (v_i :\in F_{j_i}(v_1, .., v_n), \text{ ha } \pi_{j_i})$.

Általánosítjuk a leggyengébb előfeltétel fogalmát:

3.26. Definíció (Leggyengébb előfeltétel általánosítása).
$lf(S,R) ::= \forall s \in S : lf(s,R)$.

$lfa(S,R)$::=$\exists s \in S : lf(s,R)$ *($lfa(S,R)$ az ún. "angyali" leggyengébb előfeltétel [Mor 90]).*[7]

3.10. Lemma. (Általánosított leggyengébb előfeltétel alaptulajdonságai)

(1) $lf(S,Hamis) = Hamis$*,*

(2) $lf(S,Igaz) = Igaz$*,*

(3) Ha $P \Rightarrow Q$*, akkor* $lf(S,P) \Rightarrow lf(S,Q)$*,*

(4) $lf(S,Q) \vee lf(S,R) \Rightarrow lf(S,Q \vee R)$*,*

(5) $lf(S,Q) \wedge lf(S,R) = lf(S,Q \wedge R)$*.*

Biz.: Az állítások az a 3.26. definícióból, a 3.6. lemmából és a logikai és művelet asszociativitásából és kommutativitásából következnek. □

3.4.2. Invariánsok és elérhető állapotok

Jelöljük $\mathrm{inv}_S(\lceil Q \rceil)$-val azon P logikai függvények igazsághalmazainak halmazát, amelyek az S programra nézve invariánsok[8], ha a program $\lceil Q \rceil$-beli állapotból indul. A $\lceil P \rceil \in \mathrm{inv}_S(\lceil Q \rceil)$-t röviden $P \in \mathrm{inv}_S(Q)$-val jelöljük. Jelölje $\mathrm{INV}_S(Q)$ azon P logikai függvények konjunkcióját, amelyekre $P \in \mathrm{inv}_S(Q)$[9].

3.27. Definíció (Invariáns tulajdonság).
$inv_S : \mathcal{P}(A) \longmapsto \mathcal{P}(\mathcal{P}(A))$. $inv_S(\lceil Q \rceil) \subseteq \mathcal{P}(A)$.
$inv_S(\lceil Q \rceil)$::=$\{\lceil P \rceil | sp(s_0,Q) \Rightarrow P \text{\textit{és}} P \Rightarrow lf(S,P)\}$.

3.9. Megjegyzés ($\mathbf{inv}_S(Q)$ nem üres). *A definíció szerint* $\forall S,Q : Igaz \in inv_S(Q)$.

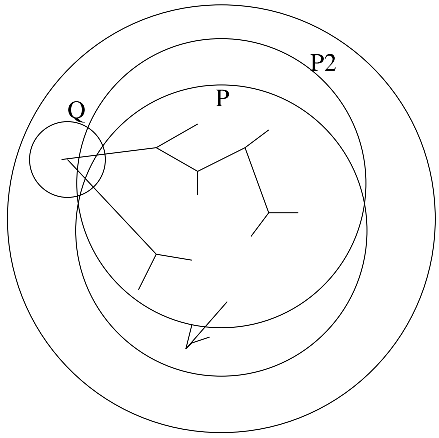
3.11. Lemma. (Invariánsok konjunkciója) $inv_S(Q)$ *zárt a* $\wedge$ *műveletre nézve [Pra 94].*

Biz.: Legyen $P,K \in inv_S(Q)$. 3.27. def. felhasználásával: $sp(s_0,Q) \Rightarrow P$ és $sp(s_0,Q) \Rightarrow K \Longrightarrow sp(s_0,Q) \Rightarrow P \wedge K$. $P \Rightarrow lf(S,P)$ és $K \Rightarrow lf(S,K) \Longrightarrow P \wedge K \Rightarrow lf(S,P) \wedge lf(S,K)$. A 3.10. lemma szerint: $lf(S,P) \wedge lf(S,K) = lf(S,P \wedge K)$, így $P \wedge K \Rightarrow lf(S,P \wedge K)$. □

[7] A lfa leképezés definíciója hasonlít a J.R. Rao által egyes utasításokra definiált *wpp* operátor definíciójához, de attól eltérően absztrakt programra vonatkozik. A *wpp* operátort a UNITY valószínűségi alapon nemdeterminisztikus kiegészítése során használják [Rao 95].

[8] Szigorú invariáns [Pra 94].

[9] $\mathrm{INV}_S(Q)$ a legszigorúbb invariáns [Pra 94].



3.2. ábra. Invariáns (P) és mindig igaz ($P2$) tulajdonság.

3.2. Következmény (Legszigorúbb invariáns). *Az* $inv_S(Q)$ *halmaznak egyértelműen létezik legkisebb (11.13. def.) eleme és az éppen* $INV_S(Q)$.

3.28. Definíció (Legszigorúbb invariáns). *Az* $inv_S(Q)$ *halmaz legkisebb elemét,* $INV_S(Q)$*-t* a legszigorúbb invariáns*nak nevezzük.*

3.12. Tétel. (Invariáns konjunkciója kezdetben igaz állítással) *Ha* $sp(s_0,Q) \Rightarrow J$, $I \in inv_S(Q)$ *és* $I \wedge J \Rightarrow lf(S,J)$*, akkor* $I \wedge J \in inv_S(Q)$ *(*$I \wedge J$ *invariáns).*

Biz.: Ha $I \in \mathrm{inv}_S(Q)$, akkor $sp(s_0,Q) \Rightarrow I$. Így $sp(s_0,Q) \Rightarrow I \wedge J$ az első feltétel szerint. Ha $I \in inv_S(Q)$, akkor $I \Rightarrow lf(S,I)$, a harmadik feltétel és 3.10. lemma felhasználásával: $I \wedge J \Rightarrow lf(S,I) \wedge lf(S,J) = lf(S,I \wedge J)$. $\square$

3.10. Megjegyzés (Invariáns tulajdonság felbontása). *A 3.11. lemma nem megfordítható. Ha* $I \wedge J$ *invariáns, akkor nem feltétlenül igaz, hogy akár* I*, akár* J *invariáns lenne. Annak bizonyítása, hogy egy* $P = \bigwedge_{i \in [1..n]} P_i$ *állítás invariáns tulajdonság, a 3.12. tétel segítségével azonban sok esetben részekre bontható pl. úgy, hogy*

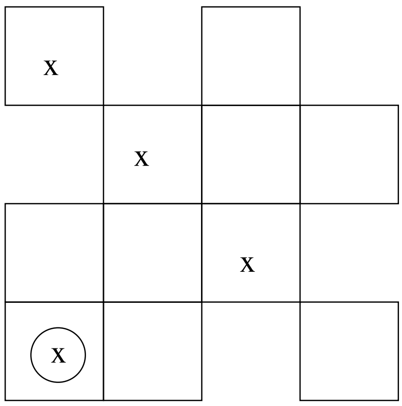
- *belátjuk, hogy* P_1 *invariáns,*
- *megmutatjuk, hogy* $\forall i : P_i$ *kezdetben igaz,*
- *igazoljuk, hogy* $\forall i : P^i \Rightarrow lf(S,P_i)$*, ahol* $P^i ::= \bigwedge_{j \in [1..i-1]} P_j$.

3.29. Definíció (Mindig igaz). $true_S : \mathcal{P}(A) \longmapsto \mathcal{P}(\mathcal{P}(A))$. $true_S(\lceil Q \rceil) \subseteq \mathcal{P}(A)$. $true_S(\lceil Q \rceil) ::= \{\lceil P \rceil | INV_S(Q) \Rightarrow P\}$.

3.11. Megjegyzés ($true_S(Q)$ nem üres). *A definíció szerint* $\forall S, Q : Igaz \in true_S(Q)$.

Azokat a logikai függvényeket, amelyek igazsághalmaza eleme a $true_S(Q)$ halmaznak, a Q-ból elérhető állapotok felett a program futása során *mindig igaz* állításoknak nevezzük[10].

A 3.3. ábra segítségével szemléltetjük, hogy van olyan mindig igaz állítás, amelyik nem invariáns. Tegyük fel, hogy a körrel jelölt mezőből indul egy huszár, amely lólépésben haladhat. Az állapottér az összes mező, a hiányos "sakktábláról" lelépni nem szabad. Könnyen ellenőrizhetjük, hogy a huszár mindig "x"-szel jelölt mezőkön marad. Mégsem invariáns tulajdonsága a huszárnak az, hogy "x"-szel jelölt mezőn áll, mert van olyan "x"-szel jelölt mező amelyről jelöletlen mezőre is léphet. Ilyen a bal felső sarokban lévő mező. Ez a mező a huszár számára nem elérhető, ezért sohasem tapasztaljuk az invariáns sérülését. Megállapíthatjuk, hogy a mindig igaz és az invariáns állítások között a lényegi különbség a nem elérehető állapotok esetén jelentkezik. Egy invariáns állítás még nem elérhető állapotokból is megmarad, egy mindig igaz állítás csak az elérhető állapotok felett teljesül. Az invariáns állítások azért fontosak, mert két komponens együttműködése könnyen eredményezheti azt, hogy korábban el nem érhető állapotok elérhetővé válnak. Több komponensből álló elosztott programok esetén tehát csak az invariáns tulajdonságokra támaszkodhatunk (ld. 6 fejezet).



3.3. ábra. Egy mindig igaz állítás nem mindig invariáns.

[10] A mindig igaz állításokat gyenge invariánsoknak is nevezik [San 91, Pra 94]

3.13. Lemma. (Az invariáns mindig igaz) $inv_S(Q) \subseteq true_S(Q)$.
Biz.: A 3.2. következmény szerint, ha $P \in inv_S(Q) \Longrightarrow INV_S(Q) \Rightarrow P$. $\square$

3.14. Lemma. (Mindig igaz állítások konjunkciója mindig igaz) *Ha* $J \in true_S(Q)$ *és* $I \in true_S(Q)$, *akkor* $I \wedge J \in true_S(Q)$.

Biz.: Ha $J, I \in true_S(Q)$, akkor $sp(s_0, Q) \Rightarrow J$ és $sp(s_0, Q) \Rightarrow I$. Így $sp(s_0, Q) \Rightarrow I \wedge J$. Ha $I, J \in true_S(Q)$, akkor $\mathrm{INV}_S(Q) \Rightarrow J$ és $\mathrm{INV}_S(Q) \Rightarrow I$. Így $\mathrm{INV}_S(Q) \Rightarrow I \wedge J$. $\square$

3.3. Következmény. *Mindig igaz és invariáns konjunkciója mindig igaz.*

3.4. Következmény (A legszigorúbb mindig igaz). *A* $true_S(Q)$ *halmaznak egyértelműen létezik legkisebb eleme és az éppen* $INV_S(Q)$, a legszigorúbb invariáns.

$INV_S(Q)$ tehát az a legszűkebb igazsághalmazú logikai függvény, amelyiknek igazsághalmazát a program soha nem hagyja el a $\lceil Q \rceil$-ból indulva. Így kimondhatjuk az alábbi tételt:

3.15. Tétel. ($\mathrm{INV}_S(Q)$ és a Q-ból elérhető állapotok) $INV_S(Q)$ *igazsághalmaza éppen a* $\lceil Q \rceil$*-ból* elérhető állapot*ok (3.17. def.) halmaza [Pra 94].*

Nem minden esetben lesz egy mindig igaz állítás és egy invariáns konjunkciója invariáns, hiszen pl. az Igaz is invariáns és konjunkciója egy mindig igaz, de nem invariáns állítással nem eredményezhet invariáns állítást.

3.16. Lemma. (Mindig igaz és invariáns konjunkciója) *Ha* $J \in true_S(Q)$, $I \in inv_S(Q)$ *és* $I \wedge J \Rightarrow lf(S, J)$, *akkor* $I \wedge J \in inv_S(Q)$.
Biz.: Ha $J \in true_S(Q)$, *akkor* $sp(s_0, Q) \Rightarrow J$. *Így az állítás következik a 3.12. tételből.* $\square$

3.4.3. Biztonságossági tulajdonságok

Jelöljük $\rhd_S$-sel azon P, Q logikai függvények igazsághalmazai rendezett párjainak halmazát, amelyekre az S program végrehajtása során igaz, hogy P *stabil feltéve, hogy* nem Q. Jelölés: $P \rhd_S Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \rhd_S$.

3.30. Definíció (Stabil feltéve, hogy – tulajdonság). $\rhd_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$.
$\rhd_S ::= \{(\lceil P \rceil, \lceil Q \rceil) | (P \wedge \neg Q) \Rightarrow lf(S,(P \vee Q))\}$[11]

3.12. Megjegyzés (Stabil tulajdonság).
Azt mondjuk, hogy S rendelkezik a P stabil tulajdonsággal, ha $P \rhd_S Hamis$.

3.17. Lemma. ($\rhd_S$ és a stabil tulajdonságok) *Ha* $P \rhd_S Q$ *és* $K \rhd_S Hamis$, *akkor* $P \wedge K \rhd_S Q \wedge K$.
Biz.: A 3.30. def. alapján $P \wedge \neg Q \Rightarrow lf(S, P \vee Q)$ *és* $K \wedge Igaz \Rightarrow lf(S,K)$. *Ebből a 3.10. lemma alkalmazásával:* $P \wedge K \wedge \neg Q \Rightarrow lf(S, P \vee Q) \wedge lf(S,K) = lf(S,(P \vee Q) \wedge K) = lf(S,(P \wedge K) \vee (Q \wedge K))$. *A 3.30. def. alkalmazásával a kívánt állításhoz jutunk.* □

3.18. Lemma. (Az invariánsok stabil tulajdonságok) *Ha* $\exists Q : K \in inv_S(Q)$, *akkor* $K \rhd_S Hamis$.
Biz.: A 3.27. és 3.30. definíciók közvetlen következménye. □

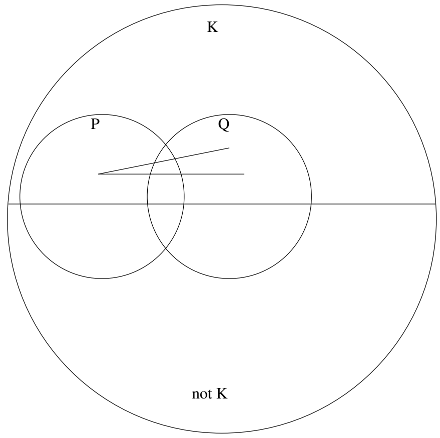
3.19. Tétel. ($\rhd_S$ és az invariánsok szigoríthatósága) *Ha* $(P \wedge J) \rhd_S (Q \wedge J)$ *és* $J \in inv_S(Q)$ *és* $K \in inv_S(Q)$, *akkor* $J \wedge K \in inv_S(Q)$ *és* $(P \wedge J \wedge K) \rhd_S (Q \wedge J \wedge K)$[12].
Biz.: Az állítás első része következik a 3.11. lemmából. Az állítás második részét pedig a 3.18. és 3.17. lemma alkalmazásával kapjuk. □

3.20. Tétel. ($\rhd_S$ és a legszigorúbb invariáns) *Ha* $(P \wedge J) \rhd_S (R \wedge J)$ *és* $J \in inv_S(Q)$, *akkor*
$P \wedge INV_S(Q) \rhd_S R \wedge INV_S(Q)$.
Biz.: $INV_S(Q) \in inv_S(Q)$ *miatt alkalmazható a 3.19. tétel.* $INV_S(Q)$ *def. alapján viszont* $INV_S(Q) \wedge J = INV_S(Q)$. □

3.4.4. Haladási tulajdonságok

Jelöljük $\mapsto_S$-sel azon P, Q logikai függvények igazsághalmazai rendezett párjainak halmazát, amelyekre az S program végrehajtása során igaz, hogy P stabil feltéve, hogy nem Q és van egy olyan $s_j \in S$ feltételes értékadás, amely garantálja, hogy a $\lceil P \rceil$-ből $\lceil Q \rceil$-ba jutunk. Jelölés: $P \mapsto_S Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \mapsto_S$.

[11] A $\rhd_S$ definíciója megfelel a [Cha Mis 89]-ben adott *unless* fogalmának.
[12] A tétel Prasetya tételének relációs átfogalmazása [Pra 94].



3.4. ábra. $P \rhd_S Q$ tulajdonság és K invariáns kapcsolata.

3.31. Definíció (Biztosítja tulajdonság). $\hookrightarrow_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$.
$\hookrightarrow_S ::= \{(\lceil P \rceil, \lceil Q \rceil) | (P, Q) \in \rhd_S \wedge \exists j \in J : (P \wedge \neg Q \Rightarrow lf(s_j, Q))\}$[13]

3.21. Lemma. ($\hookrightarrow_S$ és a stabil tulajdonság) *Ha $P \hookrightarrow_S Q$ és $K \rhd_S$ Hamis, akkor $P \wedge K \hookrightarrow_S Q \wedge K$.*

Biz.: A 3.31. def. és a 3.17. lemma alapján elegendő azt bizonyítani, hogy $\exists s \in S : (P \wedge K \wedge \neg(Q \wedge K)) \Rightarrow lf(s, Q \wedge K)$. A feltételekből tudjuk, hogy $\exists s \in S : P \wedge \neg Q \Rightarrow lf(s, Q)$. Válasszunk egy ilyen s utasítást. K stabil, ezért erre az $s \in S$-re is: $K \Rightarrow lf(s, K)$. Az s-re vonatkozó két állításból logikai és művelet és egyszerűsítés után a $P \wedge K \wedge \neg Q \Rightarrow lf(s, Q \wedge K) \wedge lf(s, K)$ eredményre jutunk. A leggyengébb előfeltétel ismert tulajdonsága alapján (3.6. lemma) $lf(s, Q \wedge K) \wedge lf(s, K) = lf(S, Q \wedge K \wedge K) = lf(s, Q \wedge K)$. $\square$

3.22. Tétel. ($\hookrightarrow_S$ és az invariánsok szigoríthatósága) *Ha $(P \wedge J) \hookrightarrow_S (Q \wedge J)$ és $J \in inv_S(Q)$ és $K \in inv_S(Q)$, akkor $J \wedge K \in inv_S(Q)$ és $(P \wedge J \wedge K) \hookrightarrow_S (Q \wedge J \wedge K)$*[14].
Biz.: Az állítás első része következik a 3.11. lemmából. Az állítás második részét pedig a 3.18. és 3.21. lemma alkalmazásával kapjuk. $\square$

3.23. Tétel. ($\hookrightarrow_S$ és a legszigorúbb invariáns) *Ha $(P \wedge J) \hookrightarrow_S (R \wedge J)$ és $J \in inv_S(Q)$, akkor*

[13] A $\hookrightarrow_S$ definíciója megfelel a [Cha Mis 89]-ben adott *ensures* fogalmának, ha az ütemezés megfelel a feltétlenül pártatlan ütemezés axiómájának.

[14] A tétel Prasetya tételének relációs átfogalmazása [Pra 94].

$P \wedge INV_S(Q) \mapsto_S R \wedge INV_S(Q)$.
Biz.: 3.20. tételhez hasonlóan. □

3.32. Definíció (Elkerülhetetlen tulajdonság). *Legyen* $\hookrightarrow_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$ *a* $\mapsto_S$ *reláció tranzitív diszjunktív lezártja (1.16. def.), vagyis az a legkisebb reláció*[15]*, amelyre teljesül, hogy*

(1) $\mapsto_S \subseteq \hookrightarrow_S$.

(2) Tranzitivitás: ha $(\lceil P \rceil, \lceil Q \rceil) \in \hookrightarrow_S$ *és* $(\lceil Q \rceil, \lceil R \rceil) \in \hookrightarrow_S$, *akkor* $(\lceil P \rceil, \lceil R \rceil) \in \hookrightarrow_S$.

(3) Diszjunkció: ha bármely W megszámlálható halmazra: $\forall m : (m \in W :: (\lceil P(m) \rceil, \lceil Q \rceil) \in \hookrightarrow_S)$, *akkor* $((\lceil \exists m : m \in W :: P(m) \rceil), \lceil Q \rceil) \in \hookrightarrow_S$.

Jelölés: $P \hookrightarrow_S Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \hookrightarrow_S$.

A 3.32. def. alapján $P \hookrightarrow_S Q$ pontosan akkor, ha a 3.32. def. (1),(2),(3) szabályainak alkalmazásával $P \hookrightarrow_S Q$ levezethető[16].

3.13. Megjegyzés (Egyértelműen létezik legkisebb adott tulajdonságú reláció). *A* $\mathcal{P}(A) \times \mathcal{P}(A)$ *rendelkezik a megadott tulajdonságokkal. Ha X és Y rendelkezik a megadott tulajdonságokkal, akkor* $X \cap Y$ *is rendelkezik velük. Így* $\hookrightarrow_S$ *egyértelműen definiált.*

3.14. Megjegyzés.
A UNITY különböző relációs kiterjesztéseiben [Pac 92, Jut Kna Rao 89] nem a feladat specifikációs feltételeinek, hanem a program által definiált inv_S, $\mapsto_S$, $\hookrightarrow_S$ *relációknak* megfelelő *relációkat definiálnak. Pachl [Pac 92] az* $\mapsto_S$ *reláció értelmezési tartományát az elérhető állapotok halmazára (3.17. def.) korlátozza.*

3.24. Lemma. ($\Rightarrow$ és $\hookrightarrow_S$) *Ha* $\lceil P \rceil \subseteq \lceil Q \rceil$, *akkor* $(P,Q) \in \hookrightarrow_S$ *tetszőleges S programra.*
Biz.: $(P,P) \in \mapsto_S$, *így* $(P,Q) \in \mapsto_S$ *a 3.31. definíció szerint.* □

3.25. Lemma. ($\hookrightarrow_S$ és a stabil tulajdonság) *Ha* $P \hookrightarrow_S Q$ *és* $K \rhd_S Hamis$, *akkor* $P \wedge K \hookrightarrow_S Q \wedge K$.

[15] A $\hookrightarrow_S$ definíciója megfelel a [Cha Mis 89]-ben adott *leads-to* fogalmának, ha az ütemezés megfelel a feltétlenül pártatlan ütemezés axiómájának.

[16] A (3)-as szabály egyetlen lépésben is végtelen sok elemre alkalmazható.

Biz.: Struktúrális indukcióval az induktív 3.32. def. alapján.
Alapeset: $P \hookrightarrow_S Q$-t közvetlenül $P \mapsto_S Q$-ból kaptuk. Ekkor a 3.21. lemma szerint $P \wedge K \mapsto_S Q \wedge K$. 3.32. def. (1) pontja szerint ekkor $P \wedge K \hookrightarrow_S Q \wedge K$.
Indukciós feltevés: a) eset: az utolsó lépésben a 3.32. def. (2) pontját, a tranzitivitást alkalmaztuk $P \hookrightarrow_S Q$ előállításakor, azaz: $P \hookrightarrow_S Q_1$ és $Q_1 \hookrightarrow_S Q$. Az indukciós feltétel szerint: $P \wedge K \hookrightarrow_S Q_1 \wedge K$ és $Q_1 \wedge K \hookrightarrow_S Q \wedge K$. A 3.32. def. (tranzitivitás) alapján: $P \wedge K \hookrightarrow_S Q \wedge K$.
b) eset: az utolsó lépésben a 3.32. def. (3) pontját, a diszjunktivitást alkalmaztuk $P \hookrightarrow_S Q$ előállításakor, azaz: $P = \exists m : m \in W :: P(m)$ és $\forall m : m \in W :: (P(m) \hookrightarrow_S Q)$. Az indukciós feltétel szerint: $\forall m : (m \in W :: (P(m) \wedge K \hookrightarrow_S Q \wedge K))$, amiből a 3.32. def. (diszjunktivitás alapján) $P \wedge K \hookrightarrow_S Q \wedge K$. □

3.15. Megjegyzés. *A tétel általánosítható a következő alakban*[17]*: Ha $P \hookrightarrow_S Q$ és $R \rhd_S B$, akkor $P \wedge R \hookrightarrow_S (Q \wedge R) \vee B$.*
Bizonyítás struktúrális indukcióval.

3.26. Tétel. ($\hookrightarrow_S$ és az invariánsok szigoríthatósága) *Ha $(P \wedge J) \hookrightarrow_S (Q \wedge J)$ és $J \in inv_S(Q)$ és $K \in inv_S(Q)$, akkor $J \wedge K \in inv_S(Q)$ és $(P \wedge J \wedge K) \hookrightarrow_S (Q \wedge J \wedge K)$*[18].

Biz.: Az állítás első része következik a 3.11. lemmából. Az állítás második részét pedig a 3.18. és 3.25. lemma alkalmazásával kapjuk. □

3.27. Tétel. ($\hookrightarrow_S$ és a legszigorúbb invariáns) *Ha $(P \wedge J) \hookrightarrow_S (R \wedge J)$ és $J \in inv_S(Q)$, akkor*
$P \wedge INV_S(Q) \hookrightarrow_S R \wedge INV_S(Q)$.
Biz.: 3.20. tételhez hasonlóan. □

3.28. Tétel. ($\hookrightarrow_S$ egyelemű részhalmazokra) $(X,Y) \in \hookrightarrow_S \Longleftrightarrow \forall x \in X : (\{x\},Y) \in \hookrightarrow_S$.

Biz.: Ha $\forall x \in X : (\{x\},Y) \in \hookrightarrow_S$, akkor $(X,Y) \in \hookrightarrow_S$ 3.32. def. alapján (diszjunktív lezárás). Ha $(X,Y) \in \hookrightarrow_S$, akkor $\forall x \in X : \{x\} \subseteq X$. A 3.24. lemma alapján $(\{x\},X) \in \hookrightarrow_S$. Ha $(X,Y) \in \hookrightarrow_S$, akkor a tranzitív lezárás miatt $(\{x\},Y) \in \hookrightarrow_S$.□

3.29. Lemma. ($\hookrightarrow_S$ – jobboldal gyengítése) *Ha $P \hookrightarrow_S Q$ és $Q \Rightarrow R$, akkor $P \hookrightarrow_S R$.*
Biz.: 3.24. lemma és a 3.32. def. (tranzitivitás) következménye. □

[17] A PSP tétel [Cha Mis 89] relációs alakja.
[18] A tétel Prasetya bizonyítás nélkül publikált tételének relációs átfogalmazása [Pra 94].

3.33. Definíció (Elkerülhetetlen feltétlenül pártatlan ütemezés mellett). $(P,Q) \in \rightsquigarrow_S$, *akkor és csak akkor, ha* $\forall a \in P$ *az* S *által az* a*-hoz rendelt fákban bármelyik, a feltétlenül pártatlan ütemezésnek megfelelő végrehajtási úton*[19] *véges (esetleg nem korlátos) távolságban van olyan pont, amelynek címkéje eleme* Q *halmaznak.*

3.30. Tétel. ($\rightsquigarrow_S$ egyelemű részhalmazokra) $(X,Y) \in \rightsquigarrow_S \Longleftrightarrow \forall x \in X : (\{x\},Y) \in \rightsquigarrow_S$.

Biz.: A 3.33. def. közvetlen következménye. $\square$

3.31. Tétel. ($\hookrightarrow_S$ helyessége és teljessége) $\hookrightarrow_S \; = \; \rightsquigarrow_S$[20].
Biz.:[21]
a) $\hookrightarrow_S \subseteq \rightsquigarrow_S$. *Biz.:* $\mapsto_S \subset \rightsquigarrow_S$ *és* $\rightsquigarrow_S$ *tranzitív és diszjunktív.*
b) $\rightsquigarrow_S \subseteq \hookrightarrow_S$. *A 3.28., 3.30. tételek alapján elegendő bizonyítani, hogy*
$(\{x\},P) \in \rightsquigarrow_S \Longrightarrow (\{x\},P) \in \hookrightarrow_S$.
$A(P) ::= \{y | (\{y\},P) \in \hookrightarrow_S\}$. $E(P) ::= \{y | (\{y\},P) \notin \hookrightarrow_S\}$.
Legyen $w \in E(P)$. *Jelöljük* $E(w,P)$*-vel azon pontok halmazát, amelyekre igaz, hogy* w*-ből olyan úton érhetőek el, amelynek minden pontja eleme* $E(P)$*-nek.* $\forall w \in E(P) : \forall s \in S : \exists z \in E(w,P) : p(s)(z) \not\subseteq A(P)$, *ellenkező esetben* $\exists w \in E(P) : \exists s \in S : \forall z \in E(w,P) : p(s)(z) \subseteq A(P)$, *azaz* $(E(w,P), A(P)) \in \mapsto_S$ *és* $w \in E(w,P)$ *miatt* $w \in A(P)$ *következne, ami ellentmondás. Tehát* $\forall w \in E(P)$*-re megkonstruálható egy olyan út, amelyen* $\forall s \in S$ *címkéje szerepel a feltétlenül pártatlan ütemezés axiómája szerint és az út* $E(P)$ *belsejében halad. Tegyük fel indirekt, hogy* $x \in E(P)$. *Ekkor a fentiek alapján* $(\{x\},P) \notin \rightsquigarrow_S$. *Azaz* $(\{x\},P) \in \rightsquigarrow_S \Longrightarrow x \notin E(P)$, *azaz* $x \in A(P)$. $\square$

3.5. Következmény. *Ha egy program rendelkezik a* $P \rightsquigarrow_S Q$ *tulajdonsággal, akkor és csak akkor rendelkezik a* $P \hookrightarrow_S Q$ *tulajdonsággal is, amely a tranzitivitás és a diszjunkció szabályának alkalmazásával levezethető a program* $U \mapsto_S V$ *alakú tulajdonságaiból*[22].

3.4.5. Fixpont tulajdonságok

A *konkrét program* az absztrakt program végrehajtásának prefixe. Az absztrakt program nem terminál abban az értelemben, hogy több elemi művelet nem kerül

[19] v.ö. 11.10. def.

[20] A tétel Pachl tételének általánosítása [Pac 92]

[21] [Pac 92]-ben adott bizonyítás általánosítható a teljes állapottérre (lásd 3.14 lábjegyzet) és nemdeterminisztikus feltételes értékadások esetére.

[22] A tranzitív diszjunktív lezárási tulajdonságok felhasználásával *Cook-féle relatív teljes* [Rao 95] levezetési szabályrendszert alkothatunk a program haladási tulajdonságaira nézve.

végrehajtásra. Termináltnak tekinthetünk azonban egy izolált programot akkor, ha elérte egy fixpontját, azaz a további műveletek hatására állapotváltozás már nem következhet be.

Az S programról azt mondjuk, hogy *fixpontba jutott az A altér felett*, ha az A altérhez tartozó változókra vonatkozó egyszerű értékadások mindegyikére teljesül, hogy az értékadás hatásrelációja független az altérhez nem tartozó állapottérkomponensektől és

- az értékadás determinisztikus és a jobboldalán álló függvénykompozíciók (kifejezések) értéke azonos a baloldalon álló változók értékével vagy
- az értékadás jobboldalán szereplő kapcsoszárójel függvény feltétele hamis, vagy
- az értékadás nemdeterminisztikus és az altér azon részhalmazának, amely felett az értékadás determinisztikus, és a kapcsoszárójel függvény feltétele igazsághalmazának metszete felett az értékadás jobboldalán álló függvénykompozíciók (kifejezések) értéke azonos a baloldalon álló változók értékével.

Legyen $S = (s_0, \{s_1, \ldots, s_m\})$, ahol $\forall s_j \in S$ egy (szimultán, nemdeterminisztikus) feltételes értékadás, $s_j : \underset{i\in[1,n]}{\|}(v_i :\in F_{j_i}(v_1, \ldots, v_n), \text{ ha } \pi_{j_i})$.

Jelöljük $\pi_{j_{id}}$-vel azt a logikai függvényt, amelynek igazsághalmazára leszűkítve az F_{j_i} reláció determinisztikus, azaz: $\pi_{j_{id}}(a) \Leftrightarrow (|F_{j_i}(a)| = 1)$.

3.34. Definíció (Fixpontok halmaza). $fixpont_S ::= (\bigwedge_{j\in J, i\in[1..n]} (\neg\pi_{j_i} \vee (\pi_{j_{id}} \wedge v_i = F_{j_i}(v_1, \ldots, v_n))))$. *A fixpontok halmazának jelölésére φ_S-t is használjuk a továbbiakban.*

Ha az értékadások mindegyike determinisztikus, akkor a program fixpontjait jellemző logikai függvényt a szimultán értékadások egyenlőséggé való átalakításával, feltételeik implikációs előtaggá való kiemelésével és konjunkciójával kapjuk meg: $fixpont_S = (\bigwedge_{j\in J, i\in[1..n]} (\pi_{j_i} \to v_i = F_{j_i}(a)))$.

3.2. Példa (Program fixpontjainak halmaza). $S = (SKIP, \{k := k+1, \text{ ha } k < N\})$.

$fixpont_S = (k < N \to k = k+1) \equiv k \geq N$ *[Cha Mis 89].* $\square$

3.35. Definíció (Fixpont tulajdonság). *Jelöljük FP_S-sel azon R logikai függvények igazsághalmazainak halmazát, amelyekre $fixpont_S \Rightarrow R$.*

3.16. Megjegyzés. *A megoldás definíciója szerint (4.1.) egy program teljesíti a $FP \Rightarrow R$ specifikációs feltételt, ha az R fixpontfeltétel tartalmazza a $fixpont_S$ állítás igazsághalmazának és (legalább) az elérhető állapotok halmazának metszetét.*

3.32. Lemma. (Fixpont tulajdonság gyengítése) *Ha $R \Rightarrow Q$ és $R \in FP_S$, akkor $Q \in FP_S$.*
Biz.: Az 3.35. def. közvetlen következménye. □

3.4.6. Terminálási tulajdonságok

3.36. Definíció (Biztosan fixpontba jut – tulajdonság).
Jelöljük TERM_S*-sel azon Q logikai függvények igazsághalmazainak halmazát, amelyekre $Q \hookrightarrow_S fixpont_S$.*

A program biztosan fixpontba jut, ha egy alkalmasan megválasztott *variáns függvény*[23] értéke bármely állapot elérése után a jövőben elkerülhetetlenül csökken (5.4. tétel).

3.5. Az absztrakt program viselkedési relációja

Egy program *szemantikai jelentésé*t azonosíthatjuk az általa az állapottér hatványhalmaza felett definiált - invariánsok, biztonságossági, haladási, fixpont és terminálási tulajdonságoknak megfelelő unáris és bináris - relációk együttesével. Egy ilyen szemantika leíró jellegű[24] (9. fejezet) [Jut Kna Rao 89] és absztrakciós szintje lényegében megegyezik a bevezetett megoldásfogalom absztrakciós szintjével.

3.37. Definíció (Viselkedési reláció). *Legyen S program az A állapottér felett. Az A állapottér felett adott ($\rhd_S$, $\mapsto_S$, $\hookrightarrow_S$, FP_S, inv_S,* TERM_S*) rendezett relációhatost az S* absztrakt párhuzamos program viselkedési relációjá*nak hívjuk és $p(S)$-sel jelöljük.*

[23] pl. az állapottér változóiból függvénykompozícióval alkotott nemnegatív egészértékű függvény

[24] Leíró szemantikáról csak akkor beszélhetünk, ha a szemantikus leképezés kompozícionális. A *kompozcionalitás* azonban csak részben teljesül a modellben (6. fejezet), amely általában elegendő ahhoz, hogy a megoldást részfeladatok megoldásából programkonstrukciók segítségével előállítsuk, de nem felel meg a kompozicionalitás szigorú matematikai követelményének.

3.6. Feladatok

3.1. Feladat. $A = x:\mathbb{N} \times y:\mathbb{N}$. *Számoljuk ki az S absztrakt program R utófeltételre vonatkozó leggyengébb előfeltételét,* $lf(S,R)$ *-t!*

$R = (0 < x + y < 7)$,

$$S = \left(SKIP, \{x := \begin{cases} 5, & ha\ \ y < 2 \\ x, & ha\ \ y \geq 2 \end{cases} \} \right)$$

3.2. Feladat. $A = x:\mathbb{Z}$. $S = (SKIP, \{\, x := -x,\ \text{ha}\, x < 0\, \})$, $R = (x > 0)$. *Számoljuk ki* $lf(S,R)$ *-t!*

3.3. Feladat. *Bizonyítsuk be, hogy tetszőleges S absztrakt program esetén az* $\rhd_S$ *reláció rendelkezik az alábbi tulajdonságokkal!*

$Igaz \rhd_S P \qquad P \rhd_S Igaz \qquad P \rhd_S \neg P$

$Hamis \rhd_S P \qquad P \rhd_S P$

3.4. Feladat. *Igaz-e tetszőleges S programra, P,Q logikai függvényekre, ha* $P \rhd_S Q$*, akkor és csak akkor* $P \wedge \neg Q \rhd_S Q$ *és* $P \vee Q \rhd_S Q$*?*

3.5. Feladat. *Bizonyítsuk be, ha* $P \rhd_S Hamis$*, akkor* $P \rhd_S Q$*!*

3.6. Feladat. *(Gyengítési tétel)*

Igaz-e? Ha $P \rhd_S Q$, $Q \Rightarrow R$ *, akkor* $P \rhd_S R$.

3.7. Feladat. *Igaz-e? Ha* $P \Rightarrow Q$, $Q \rhd_S R$ *, akkor* $P \rhd_S R$.

3.8. Feladat. *Igaz-e? Ha* $P \Rightarrow Q$ *, akkor* $P \rhd_S Q$.

3.9. Feladat. *Igaz-e? Ha* $\neg P \Rightarrow Q$ *, akkor* $P \rhd_S Q$.

3.10. Feladat. *(* $\rhd_S$ *tranzitivitása)*

Igaz-e? Ha $P \rhd_S Q$, $Q \rhd_S R$ *, akkor* $P \rhd_S R$.

3.11. Feladat. *(* $\rhd_S$ *diszjunktivitása)*

Igaz-e? Ha $P \rhd_S R$ *és* $Q \rhd_S R$ *, akkor* $P \vee Q \rhd_S R$.

3.12. Feladat. *Igaz-e? Ha* $P \rhd_S Q$ *és* $K stabil_S$ *, akkor* $P \wedge K \rhd_S Q \wedge K$.

3.13. Feladat. *Igaz-e? Ha* $P \rhd_S P'$ *és* $Q \rhd_S Q'$*, akkor* $P \wedge Q \rhd_S P' \vee Q'$ *és* $P \vee Q \rhd_S P' \vee Q'$.

3.14. Feladat.

Igaz-e? Ha $P \rhd_S Q$ *és* $R \Rightarrow Q$*, akkor* $P \rhd_S R$.

3.15. Feladat.

Igaz-e? Ha $P \rhd_S Q$ *és* $P \Rightarrow R$*, akkor* $R \rhd_S Q$.

3.16. Feladat.

Igaz-e? Ha $R \rhd_S Q \wedge R$ *, akkor* $R stabil_S$.

3.17. Feladat.

Igaz-e? Ha $(P \vee Q)$ $stabil_S$*, akkor* $P \rhd_S Q$.

3.18. Feladat.

Igaz-e? Ha $(P \vee Q)$ $stabil_S$, $Q \Rightarrow R$ *, akkor* $P \rhd_S R$.

3.19. Feladat.

Igaz-e? Ha $P \rhd_S Q$ *és* $Q \rhd_S R$*, akkor* $P \vee Q \rhd_S R$.

3.20. Feladat.

Igaz-e? Ha $P \rhd_S Q$, $Q \rhd_S P$ *és* $(P \wedge Q) \rhd_S (P \vee Q)$*, akkor* $(P \vee Q) stabil_S$.

3.21. Feladat. *Igaz-e? Ha* $P \rhd_S Q$, $Q \rhd_S P$ *és* $P \wedge Q \Rightarrow Hamis$*, akkor* $P \vee Q stabil_S$.

3.22. Feladat. *Igaz-e? Ha* $P \rhd_S Q$, $Q \rhd_S P$ *és* $P \wedge Q stabil_S$ *, akkor* $P \vee Q stabil_S$.

3.23. Feladat. *Igaz-e? Ha* $\neg P \rhd_S (Q \vee R)$, $\neg R \rhd_S (P \vee Q)$*, akkor* $\neg Q \rhd_S (P \vee R)$.

3.24. Feladat. *Igaz-e? Ha* $P \rhd_S Q$, $Q \rhd_S P$*, akkor* $(P \vee Q) \rhd_S (P \wedge Q)$.

3.25. Feladat. *Igaz-e? Ha* $P \rhd_S Q$, $Q \rhd_S P$*, akkor* $P \rhd_S (Q \vee R)$.

3.26. Feladat. *Igaz-e? Ha* $(P \vee Q) \rhd_S Q$, $Q \rhd_S R$*, akkor* $P \rhd_S (Q \vee R)$.

3.27. Feladat. *Igaz-e? Ha* $(P \vee Q) \rhd_S R$, $Q \rhd_S R$*, akkor* $P \rhd_S (Q \vee R)$.

3.28. Feladat. *Igaz-e? Ha* $P \rhd_S (Q \vee R)$*, akkor* $(P \wedge \neg Q) \rhd_S (Q \vee R)$.

3.29. Feladat. *Igaz-e? Ha* $P \Rightarrow Q$, $(Q \wedge \neg Z) \rhd_S R$, $R \Rightarrow Z$, *akkor* $(Q \wedge \neg P) \rhd_S (Z \wedge P)$.

3.30. Feladat. *Igaz-e? Ha* $\neg R \rhd_S (P \wedge R)$, $Q \rhd_S \neg R$, $R \Rightarrow Z$, *akkor* $Z \rhd_S (Q \vee \neg R)$.

3.31. Feladat. *Bizonyítsuk be, hogy tetszőleges S absztrakt program esetén az* $\mapsto_S$ *reláció rendelkezik a következő tulajdonságokkal:*
Hamis $\mapsto_S P$, $P \mapsto_S$ *Igaz*, $P \mapsto_S P$*!*

3.32. Feladat. *(Gyengítési tétel) Igaz-e? Ha* $P \mapsto_S Q$, $Q \Rightarrow R$ *, akkor* $P \mapsto_S R$.

3.33. Feladat. *Igaz-e? Ha* $P \Rightarrow Q$, $Q \mapsto_S R$ *, akkor* $P \mapsto_S R$.

3.34. Feladat.

Igaz-e? Ha $P \mapsto_S Q$*, és* $A \mapsto_S B$ *, akkor* $P \wedge A \mapsto_S Q \wedge B$.

3.35. Feladat. *Igaz-e? Ha* $P \Rightarrow Q$ *, akkor* $P \mapsto_S Q$.

3.36. Feladat. *Igaz-e? Ha* $P \mapsto_S Q$, $K stabil_S$ *, akkor* $P \wedge K \mapsto_S Q \wedge K$.

3.37. Feladat. *Igaz-e? Ha* $P \Rightarrow Q$ *, akkor* $\neg P \mapsto_S Q$.

3.38. Feladat. *Igaz-e? Ha* $P \mapsto_S$ *Hamis , akkor* $P =$ *Hamis.*

3.39. Feladat. *(* $\mapsto_S$ *tranzitivitása) Igaz-e? Ha* $P \mapsto_S Q$, $Q \mapsto_S R$ *, akkor* $P \mapsto_S R$.

3.40. Feladat. *(* $\mapsto_S$ *diszjunktivitása) Igaz-e? Ha* $P \mapsto_S R$ *és* $Q \mapsto_S R$ *, akkor* $P \vee Q \mapsto_S R$.

3.41. Feladat. *Igaz-e? Ha* $P \mapsto_S Q$, $Q \mapsto_S R$ *, akkor* $P \vee Q \mapsto_S R$.

3.42. Feladat. *Igaz-e? Ha* $P \mapsto_S Q$ *, akkor* $(P \vee R) \mapsto_S (Q \vee R)$.

3.43. Feladat. *Igaz-e? Ha* $(P \vee Q) \mapsto_S R$ *, akkor* $P \mapsto_S (Q \vee R)$.

3.44. Feladat. *Igaz-e? Ha* $(P \vee Q) \mapsto_S Q$ *,* $Q \mapsto_S R$*, akkor* $P \mapsto_S (Q \vee R)$.

3.45. Feladat. *Igaz-e? Ha* $(P \vee Q) \mapsto_S R$ *, akkor* $(P \vee R) \mapsto_S (Q \vee R)$.

3.46. Feladat. *Igaz-e? Ha* $P \mapsto_S (Q \vee R)$ *, akkor* $(P \wedge \neg Q) \mapsto_S (Q \vee R)$.

3.47. Feladat. *Igaz-e? Ha* $P \mapsto_S \neg P$, $Q \mapsto_S \neg Q$, $R \mapsto_S \neg R$, *akkor* $(P \wedge Q) \vee (Q \wedge R) \vee (P \wedge R) \;\mapsto_S\; \neg((P \wedge Q) \vee (Q \wedge R) \vee (P \wedge R))$.

3.48. Feladat. *Bizonyítsuk be, hogy tetszőleges S absztrakt program esetén az* $\hookrightarrow_S$ *reláció rendelkezik a következő tulajdonságokkal:*
Hamis $\hookrightarrow_S P$, $P \hookrightarrow_S$ *Igaz*, $P \hookrightarrow_S P$*!*

3.49. Feladat. *Igaz-e? Ha* $P \Rightarrow Q$, *akkor* $P \hookrightarrow_S Q$.

3.50. Feladat. *Igaz-e? Ha* $P \hookrightarrow_S Q$, $Q \Rightarrow R$, *akkor* $P \hookrightarrow_S R$.

3.51. Feladat. *Igaz-e? Ha* $P \Rightarrow Q$, $Q \hookrightarrow_S R$, *akkor* $P \hookrightarrow_S R$.

3.52. Feladat. *Igaz-e? Ha* $P \hookrightarrow_S$ *Hamis, akkor* $P =$ *Hamis.*

3.33. Tétel. *PSP tétel Igaz-e? Ha* $p \hookrightarrow_S q$, $r \rhd_S b$, *akkor* $p \wedge r \hookrightarrow_S (q \wedge r) \vee b$.

3.53. Feladat. *Igaz-e? Ha* $P \hookrightarrow_S Q$, *és* $K stabil_S$, *akkor* $(P \wedge K) \hookrightarrow_S (Q \wedge K)$.

3.54. Feladat. *Igaz-e? Ha* $\forall m \in I : (P_m \hookrightarrow_S Q_m)$, *akkor* $(\bigvee_{m \in I} P_m) \hookrightarrow_S (\bigvee_{m \in I} Q_m)$.

3.55. Feladat. *Igaz-e? Ha* $A \rhd_S B$, $A \hookrightarrow_S B$, *akkor* $A \mapsto_S B$.

3.56. Feladat. *Igaz-e? Ha* $P \hookrightarrow_S Q$, $P\ stabil_S$, *akkor* $P \mapsto_S Q$.

3.57. Feladat. *Igaz-e? Ha* $P \hookrightarrow_S Q$, *akkor és csak akkor* $P \wedge \neg Q \hookrightarrow_S Q$

3.58. Feladat. *Igaz-e? Ha* $A \hookrightarrow_S B$, $B \hookrightarrow_S A$, $\neg(A \vee B)\ stabil_S$, *akkor* $(A \vee B)\ stabil_S$.

3.59. Feladat. *Igaz-e? Ha* $P \mapsto_S Q$, $Q \mapsto_S P$, $(P \vee Q) \mapsto_S P$, *akkor* $(P \vee Q) \hookrightarrow_S (P \wedge Q)$.

3.60. Feladat. *Igaz-e? Ha* $P \hookrightarrow_S \neg P$, $P \vee Q\ stabil_S$, $((P \wedge \neg Q) \vee (Q \wedge \neg P)) \mapsto_S (P \vee Q)$, *akkor* $(P \wedge \neg Q) \hookrightarrow_S Q$.

3.61. Feladat. *Igaz-e? Ha* $P \mapsto_S Q$, $Q \mapsto_S \neg P$, $P \rhd_S B$, *akkor* $P \hookrightarrow_S B$.

3.62. Feladat. *Igaz-e? Ha* $A \mapsto_S (B \vee C)$, $B \mapsto_S (B \wedge C \wedge \neg A)$, $C \mapsto_S (B \wedge C)$, $(B \wedge C) \Rightarrow D$, *akkor* $A \hookrightarrow_S D$.

3.63. Feladat. *Igaz-e? Ha* $(P \wedge B) \hookrightarrow_S Q$, $(P \wedge \neg B) \hookrightarrow_S ((P \wedge B) \vee Q)$, *akkor* $P \hookrightarrow_S Q$.

3.64. Feladat. *Igaz-e? Ha* $P \hookrightarrow_S (Q \vee B)$, $B \hookrightarrow_S R$, *akkor* $P \hookrightarrow_S (Q \vee R)$.

3.65. Feladat. *Igaz-e? Ha* $A \rhd_S B$, $B \rhd_S C$, $A \hookrightarrow_S C$, *akkor* $A \hookrightarrow_S B$.

3.66. Feladat. *Igaz-e? Ha* $P \hookrightarrow_S Q$, $(R \wedge \neg Q)$ $stabil_S$, *akkor* $(P \wedge R) \Rightarrow Q$.

3.67. Feladat. *Igaz-e? Ha* $(P \wedge Q) \mapsto_S R$, *akkor* $P \hookrightarrow_S (\neg Q \vee R)$.

3.68. Feladat. *Igaz-e? Ha* $P \hookrightarrow_S Q$, $(P \wedge R) \rhd_S (Q \wedge R)$, *akkor* $(P \wedge R) \hookrightarrow_S (Q \wedge R)$.

3.69. Feladat. *Igaz-e? Ha* $A_0 \mapsto_S C$, $\forall n \in I : A_{n+1} \mapsto_S A_n$, *akkor* $(\bigvee A_n) \hookrightarrow_S C$. *, ahol* $I \subset \mathbb{N}$ *indexhalmaz.*

3.70. Feladat. *Igaz-e? Ha* $\forall i \in I : (P_i \hookrightarrow_S (Q_i \vee R),\ Q_i \rhd_S R)$, *akkor* $(\bigwedge_{i \in I} P_i) \hookrightarrow_S ((\bigwedge_{i \in I} Q_i) \vee R)$.

4. fejezet

A megoldás fogalma

Megadjuk, hogy egy absztrakt program mikor old meg egy feladatot. A megoldás fogalmát a leggyengébb előfeltétel *fogalmára építjük fel. A megoldás definíciója így egy olyan verifikációs kalkulus alapja, amely helyes program esetén a specifikációs feltételek és az absztrakt program utasításainak számával lineárisan arányos számú lépésben véget ér. Kimondunk néhány tételt, amelyek a verifikációt egyszerűsítik, illetve igazolják, hogy a bevezetett megoldásfogalom megfelel elvárásainknak.*

4.1. A megoldás definíciója

4.1. Definíció (Megoldás). *Azt mondjuk, hogy az S program megoldja az F feladatot (2.1. def.), ha $\forall b \in B : \exists h \in F(b)$, hogy az S program megfelel a h-ban adott $inv_h P$, $P \rhd_h U$, $P \mapsto_h U$, $P \hookrightarrow_h U$, $FP_h \Rightarrow R$, $Q \in \mathrm{TERM}_h$ alakú specifikációs feltételek mindegyikének a $Q \in INIT_h$ kezdeti feltételek mellett.*

Az alábbiakban sorra megadjuk, hogy egy S program mikor *felel meg* az egyes specifikációs feltételeknek a $Q \in \mathrm{INIT}_h$ kezdeti feltételek mellett.

4.1. Megjegyzés (Specifikációs feltételek és elérhető állapotok).
Rögzített program esetén indokolt a specifikációs feltételek vizsgálatát az elérhető állapotok halmazára korlátozni [Lam Lyn 90, San 91, Pra 94]. Ha a program megfelel egy specifikációs feltételnek az elérhető állapotok felett, akkor a specifikációs feltétel nem sérül a program futása során.
A gyakorlatban azonban általában az elérhető állapotok halmazánál tágabb halmazt választunk (v.ö.: 4.3. megjegyzés), szélső esetben akár az összes állapotot, a teljes állapotteret is figyelembe vehetjük.

4.2. Megjegyzés. *A megoldás definíciójának megadásakor az absztrakt program viselkedési relációjának 3.5 bekezdésben adott definíciójára (3.37. def.) támaszkodunk az absztrakt program (3.15. def.) definíciója helyett. A viselkedési reláció és a feladat hasonló szerkezetű, így könnyen összehasonlíthatóak.*

4.2. Átmenetfeltételek

4.2.1. Biztonságossági feltételek

4.2. Definíció (Megfelel $(\mathbf{inv}_h P)$-nek). *Az S program pontosan akkor felel meg az $(inv_h P)$ specifikációs feltételnek, ha van olyan $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ invariáns tulajdonság, amely mellett megfelel a feltételnek.*

4.3. Definíció. *Az S program pontosan akkor felel meg az $(inv_h P)$ specifikációs feltételnek a $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ mellett, ha*
$P \wedge K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$.

4.3. Megjegyzés. *A program bármely invariáns tulajdonságának igazsághalmaza tartalmazza az elérhető állapotok halmazát (3.17. def., 3.15. tétel), így a továbbiakban a 4.1. megjegyzésnek megfelelően megengedjük, hogy a program az egyes specifikációs feltételeknek csak egy-egy kiválasztott* invariáns tulajdonság *igazsághalmaza felett feleljen meg*[1]*. Az alábbiakban megmutatjuk, hogy programok egyes specifikációs feltételekre vonatkozó helyességének bizonyítása során az invariáns tulajdonságokat segédtételként felhasználhatjuk.*

4.4. Megjegyzés (Specifikációs feltételek és mindig igaz állítások). *A specifikációs feltételek vizsgálatát megszoríthatnánk egyes* mindig igaz *állítások igazsághalmazára is, amelyek igazsághalmaza az invariáns tulajdonságokhoz hasonlóan tartalmazza az elérhető állapotok halmazát. A mindig igaz állítások azonban nem használhatóak fel segédtételként az utasítások* leggyengébb előfeltételének kiszámítására épülő bizonyítások *során. (Ha J mindig igaz, de nem invariáns, akkor $J \not\Rightarrow lf(S,J)$.) A mindig igaz állításokra az sem igaz, hogy szigoríthatóak az átmenetfeltételekre nézve (3.19. 3.22. 3.26. tétel) [Pra 94].*

4.4. Definíció (Megfelel $P \rhd_h Q$-nak).
S megfelel a $P \rhd_h Q$ specifikációs feltételnek, ha van olyan $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ invariáns tulajdonság ami mellett megfelel a feltételnek.

[1] Ez a döntés megfelel a helyettesítési axiómának [Cha Mis 89, UN 88-93, Pra 94]

4.5. Definíció. *Az S program pontosan akkor felel meg az $P \rhd_h Q$ specifikációs feltételnek a $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ mellett, ha $P \wedge K \rhd_S Q \wedge K$.*

4.2.2. Haladási feltételek

4.6. Definíció (Megfelel $P \mapsto_h Q$-nak). *S megfelel a $P \mapsto_h Q$ specifikációs feltételnek, ha van olyan $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ invariáns tulajdonság ami mellett megfelel a feltételnek.*

4.7. Definíció. *Az S program pontosan akkor felel meg az $P \mapsto_h Q$ specifikációs feltételnek a $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ mellett, ha $P \wedge K \mapsto_S Q \wedge K$.*

4.5. Megjegyzés (Haladási feltételek és az ütemezés). *A definíció a 3.31. definícióra épül, amelyben erősen kihasználjuk a feltétlenül pártatlan ütemezés meglétét [Cha Mis 89].*

4.8. Definíció (Megfelel $P \hookrightarrow_h Q$-nak). *S pontosan akkor felel meg a $P \hookrightarrow_h Q$ specifikációs feltételnek, ha van olyan $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ invariáns tulajdonság ami mellett megfelel a feltételnek.*

4.9. Definíció. *S pontosan akkor felel meg a $P \hookrightarrow_h Q$ specifikációs feltételnek a $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ mellett, ha $P \wedge K \hookrightarrow_S Q \wedge K$.*

4.10. Definíció (Megfelel $P \hookrightarrow$ FP$_h$-nak). *S pontosan akkor felel meg a $P \hookrightarrow FP_h$ specifikációs feltételnek, ha van olyan $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ invariáns tulajdonság ami mellett megfelel a feltételnek.*

4.11. Definíció. *S pontosan akkor felel meg a $P \hookrightarrow FP_h$ specifikációs feltételnek a $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ mellett, ha $(sp(s_0, P) \wedge K) \in \mathrm{TERM}_S$.*

4.3. Peremfeltételek

4.3.1. Fixpont feltételek

4.12. Definíció (Megfelel FP$_h \Rightarrow R$-nek). *S pontosan akkor felel meg a $FP_h \Rightarrow R$ specifikációs feltételnek, ha van olyan $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ invariáns tulajdonság ami mellett megfelel a feltételnek.*

4.13. Definíció. *S pontosan akkor felel meg a $FP_h \Rightarrow R$ specifikációs feltételnek $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ mellett, ha $fixpont_S \wedge K \Rightarrow R$.*

4.4. Megoldás K invariáns tulajdonság mellett

4.14. Definíció (Megoldás K invariáns mellett). *Azt mondjuk, hogy az S program megoldja az F feladatot (2.1. def.) a K invariáns tulajdonság mellett, ha* $\forall b \in B : \exists h \in F(b)$, *hogy* $K \in inv_S(\bigwedge_{Q\in INIT_h} Q)$ *és az S program K mellett megfelel a h-ban adott* $inv_h P$, $P \rhd_h U$, $P \mapsto_h U$, $P \hookrightarrow_h U$, $FP_h \Rightarrow R$, $Q \in \mathrm{TERM}_h$ *alakú specifikációs feltételek mindegyikének a* $Q \in INIT_h$ *kezdeti feltételek mellett.*

4.5. A megoldás definíciójának vizsgálata

4.1. Lemma. (Megfelel $(\mathrm{inv}_h P)$-nek) *S megfelel* $(inv_h P)$ *specifikációs feltételnek, ha van olyan* $K \in inv_S(\bigwedge_{Q\in INIT_h} Q)$, *hogy*
$sp(s_0, (\bigwedge_{Q\in INIT_h} Q)) \Rightarrow P \wedge K$ *és* $P \wedge K \Rightarrow lf(S, P \wedge K)$.

Biz.: 3.27. és 4.2. definíciók közvetlen következménye. □

4.1. Következmény. *S megfelel* $(inv_h P)$ *specifikációs feltételnek, ha*
$(\exists Q \in INIT_h, \exists K) : \ sp(s_0, Q) \Rightarrow P \wedge K$ *és* $K \Rightarrow lf(S, K)$ *és* $P \wedge K \Rightarrow lf(S, P \wedge K)$.

4.2. Tétel. (Megfelel inv_h-nak INV_S mellett) *Az S program pontosan akkor felel meg a* $P \in inv_h$ *specifikációs feltételnek, ha megfelel a legszigorúbb invariáns mellett, azaz, ha P mindig igaz* ($P \in true_S(\bigwedge_{Q\in INIT_h} Q)$).

Biz.: Ha megfelel a legszigorúbb invariáns mellett, akkor van olyan invariáns, amely mellett megfelel, így a 4.2. definíció szerint megfelel a feltételnek. Ekkor $P \wedge \mathrm{INV}_S(\bigwedge_{Q\in \mathrm{INIT}_h} Q) \in \mathrm{inv}_S(\bigwedge_{Q\in \mathrm{INIT}_h} Q)$. $\mathrm{INV}_S(\bigwedge_{Q\in \mathrm{INIT}_h} Q)$ a legszigorúbb invariáns, így: $P \wedge \mathrm{INV}_S(\bigwedge_{Q\in \mathrm{INIT}_h} Q) = \mathrm{INV}_S(\bigwedge_{Q\in \mathrm{INIT}_h} Q)$, azaz P mindig igaz.
Ha megfelel a feltételnek, akkor van olyan J invariáns amely mellett megfelel, így a 3.11. lemma szerint a legszigorúbb invariáns szerint is megfelel a feltételnek. □

4.3. Lemma. (Megfelel $P \rhd_h Q$-nak)
S megfelel a $P \rhd_h Q$ *specifikációs feltételnek a* $K \in inv_S(\bigwedge_{Q\in INIT_h} Q)$ invariáns tulajdonság mellett, *ha* $(P \wedge \neg Q \wedge K \Rightarrow lf(S, (P \vee Q) \wedge K))$.
Biz: A 3.30.,4.4. definíciók közvetlen következménye. □

4.4. Tétel. (Megfelel $P \rhd_h Q$-nak INV_S mellett) *Az S program pontosan akkor felel meg a* $P \rhd_h Q$ *specifikációs feltételnek, ha megfelel a legszigorúbb invariáns mellett, azaz:*
$P \wedge INV_S(\bigwedge_{Q\in INIT_h} Q) \rhd_S Q \wedge INV_S(\bigwedge_{Q\in INIT_h} Q)$.

Biz.: Ha megfelel a legszigorúbb invariáns mellett, akkor van olyan invariáns, amely mellett megfelel, így a 4.4. definíció szerint megfelel a feltételnek. Ha megfelel a feltételnek, akkor van olyan J invariáns amely mellett megfelel, így a 3.20. tétel szerint a legszigorúbb invariáns szerint is megfelel a feltételnek. □

4.6. Megjegyzés.
Azt mondjuk, hogy S megfelel a P stabil$_h$ *feltételnek, ha megfelel a* $P \rhd_h$ *Hamis feltételnek.*

4.5. Lemma. (Megfelel $(P \mapsto_h Q)$-nak) *Az S program megfelel* $(P \mapsto_h Q)$ *specifikációs feltételnek a* $K \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$ invariáns tulajdonság mellett, *ha S megfelel a* $P \rhd_h Q$ *feltételnek K mellett, és* $\exists j \in J : (P \wedge \neg Q \wedge K \Rightarrow lf(s_j, Q \wedge K))$.

Biz.: 3.31. 4.6. definíciók és a 4.3. lemma közvetlen következménye. □

4.6. Tétel. (Megfelel $P \mapsto_h Q$-nak INV_S mellett) *Az S program pontosan akkor felel meg a* $P \mapsto_h Q$ *specifikációs feltételnek, ha megfelel a legszigorúbb invariáns mellett, azaz:*
$P \wedge INV_S(\bigwedge_{Q \in INIT_h} Q) \mapsto_S Q \wedge INV_S(\bigwedge_{Q \in INIT_h} Q)$.

Biz.: 4.4. tétel bizonyításához hasonlóan a 3.23. tétel felhasználásával. □

4.7. Tétel. (Megfelel $P \hookrightarrow_h Q$-nak INV_S mellett) *Az S program pontosan akkor felel meg a* $P \hookrightarrow_h Q$ *specifikációs feltételnek, ha megfelel a legszigorúbb invariáns mellett, azaz:*
$P \wedge INV_S(\bigwedge_{Q \in INIT_h} Q) \hookrightarrow_S Q \wedge INV_S(\bigwedge_{Q \in INIT_h} Q)$.

Biz.: 4.4. tétel bizonyításához hasonlóan a 3.27. tétel felhasználásával. □

4.8. Lemma. (Megfelel $P \hookrightarrow_h Q$-nak INV_S mellett) *Az S program pontosan akkor felel meg a* $P \hookrightarrow_h Q$ *specifikációs feltételnek, ha* $P \wedge INV_S(\bigwedge_{Q \in INIT_h} Q) \hookrightarrow_S Q$.

Biz.: Ha $P \wedge \mathrm{INV}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q) \hookrightarrow_S Q$, akkor a 3.25. lemma miatt (a legszigorúbb invariáns stabil): $P \wedge \mathrm{INV}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q) \hookrightarrow_S Q \wedge \mathrm{INV}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$. Az állítást az előző tétel alkalmazásával kapjuk. Ha S megfelel $P \hookrightarrow_h Q$-nak, akkor az előző tétel szerint $P \wedge \mathrm{INV}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q) \hookrightarrow_S Q \wedge \mathrm{INV}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$. A 3.29. lemma szerint a $\hookrightarrow_S$ jobboldala gyengíthető, így $P \wedge \mathrm{INV}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q) \hookrightarrow_S Q$. □

4.2. Következmény. *Az S program pontosan akkor felel meg a* $P \hookrightarrow_h Q$ *specifikációs feltételnek, ha* $P \wedge INV_S(\bigwedge_{Q \in INIT_h} Q) \rightsquigarrow_S Q$.
Biz.: A 3.31. tétel szerint $\hookrightarrow_S = \rightsquigarrow_S$. □

4.9. Lemma. (Megfelel $P \hookrightarrow \mathrm{FP}_h$-nak INV_S mellett) *Az S program pontosan akkor felel meg a* $P \hookrightarrow FP_h$ *specifikációs feltételnek, ha* $sp(s_0, P) \wedge INV_S(\bigwedge_{Q \in INIT_h} Q) \hookrightarrow_S fixpont_S$.

Biz.: A 3.36.,4.2.,4.10. def. és 4.8. lemma közvetlen következménye. □

4.10. Lemma. (Megfelel $\mathrm{FP}_h \Rightarrow R$-nek) *S megfelel a* ($FP_h \Rightarrow R$) *specifikációs feltételnek az A állapottér felett,* a $K \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$ invariáns mellett, *ha* $fixpont_S \wedge K \Rightarrow R$, *ahol a* $fixpont_S$ *logikai függvény az S program A feletti fixpontjainak halmazát adja meg (3.34. def.).*

Biz.: A 3.35. és a 4.12. def. közvetlen következménye. □

4.11. Lemma. (Megfelel $\mathrm{FP}_h \Rightarrow R$-nek INV_S mellett) *S pontosan akkor felel meg a* ($FP_h \Rightarrow R$) *kikötésnek, ha* $fixpont_S \wedge INV_S(\bigwedge_{Q \in INIT_h} Q) \Rightarrow R$.

Biz.: Az előző lemma következménye. □

4.7. Megjegyzés (Megoldás K mellett). *Ha S megoldja a feladatot egy K invariáns tulajdonság mellett, akkor S megoldja a feladatot. Ha S megoldja a feladatot, akkor megoldja a legszigorúbb invariáns mellett is.*

4.12. Tétel. (Program és feladat kiterjesztése) *Legyen az F feladat és az S program az A állapottér* A_1 *altere felett definiálva. Ha S megoldja F-et, akkor S kiterjesztése A-ra megoldja az F feladat A-ra való kiterjesztését*[2].

Biz.: A 4.1. def. szerint az S program viselkedési relációja a megoldás definíciója szerint tartalmaz minden olyan programtulajdonságot, amely ahhoz szükséges, hogy S megfeleljen az F specifikációs feltételeinek. A programtulajdonságok mindegyikének definíciója a leggyengébb előfeltételre épül, ezért 3.22. def. és a 3.7. lemma szerint a programtulajdonságok kiterjesztései tulajdonságai a kiterjesztett programnak is. Így a kiterjesztett feladat (2.3. def.) specifikációs feltételeinek kielégítéséhez szükséges valamennyi programtulajdonság szerepel a kiterjesztett program viselkedési relációjában. □

[2] A tétel a [Fót 88] cikk egyik kiterjesztési tételének általánosítása. A többi kiterjesztési tétel megfelelője is megfogalmazható.

5. fejezet

Levezetési szabályok

Ebben a fejezetben olyan tételeket bizonyítunk be, amelyeket gyakran alkalmazunk feladatok finomítása során, vagy amelyek megkönnyítik annak bizonyítását, hogy egy program megfelel egy adott specifikációs feltételnek. Ezeket a tételeket levezetési szabály*oknak nevezzük. A leggyakrabban invariánsok, elkerülhetetlenséget kifejező, ill. fixpontfeltételek finomítására van szükség.*

5.1. Biztonságossági feltételek finomítása

5.1. Lemma. (Invariáns feltétel felbontása) *Ha egy S absztrakt program megfelel az $(inv_h P_1)$, $(inv_h P_2)$ specifikációs feltételeknek, akkor megfelel a $(inv_h P_1 \wedge P_2)$ specifikációs feltételnek is.*
Biz.: a 3.11. lemma következménye.$\square$

5.2. Haladási feltételek finomítása

5.2. Lemma. ($\hookrightarrow_h$ finomítása) *S megfelel a $P \hookrightarrow_h Q$ specifikációs feltételnek, ha az alábbi szabályok alkalmazásával levezethető:*

(1) Ha S megfelel $(P \mapsto_h Q)$-nak, akkor S megfelel $(P \hookrightarrow_h Q)$-nak is.

(2) Tranzitivitás: ha S megfelel $(P \hookrightarrow_h Q)$-nak és S megfelel $(Q \hookrightarrow_h R)$-nak, akkor S megfelel $(P \hookrightarrow_h R)$-nak is.

(3) *Diszjunkció: bármely W megszámlálható halmazra: ha*
$\forall m : m \in W :: S$ *megfelel* $(P(m) \hookrightarrow_h Q)$*-nak, akkor*
S *megfelel* $((\exists m : m \in W :: P(m)) \hookrightarrow_h Q)$*-nak is.*

Biz.: 3.32. def. és a 4.6.,4.7. tételek felhasználásával: (1) A feltétel szerint $P \wedge INV_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q) \mapsto_S Q \wedge INV_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$. Ekkor 3.32. def. szerint $P \wedge INV_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q) \hookrightarrow_S Q \wedge INV_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$. (2)-es és (3)-as esetben hasonlóan. □

5.3. Lemma. ($P \hookrightarrow \mathrm{FP}_h$ feltétel bizonyítása) *Az S program pontosan akkor felel meg a* $P \hookrightarrow FP_h$ *specifikációnak, ha*
megfelel a $sp(s_0, P) \hookrightarrow_h fixpont_S$ *feltételnek, tehát S* biztosan fixpontba jut.

Biz.: A 4.9. lemma szerint $sp(s_0, P) \wedge INV_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q) \hookrightarrow_S fixpont_S$, így a 4.8. lemma felhasználásával: S megfelel a $sp(s_0, \mathrm{Igaz}) \hookrightarrow_h fixpont_S$ feltételnek. □

5.1. Definíció (Variáns függvény). *Variáns függvénynek nevezzük a* $t : A \longmapsto \mathcal{Z}$, *az állapotokhoz egészeket rendelő függvényeket. Legyen* $m \in \mathcal{Z}$ *tetszőleges egész szám. A* $t = m, t > m : A \longmapsto \mathcal{L}$ *logikai függvényeket definiáljuk az igazsághalmazukkal:*
$\lceil t = m \rceil ::= \{a \in A | t(a) = m\}$, $\lceil t > m \rceil ::= \{a \in A | t(a) > m\}$.

A következő tételben a UNITY indukciós elvét [Cha Mis 89] fogalmazzuk meg a ciklus levezetési szabályához hasonló alakban [Fót 83, WRMP 95].

5.4. Tétel. (Variánsfüggvény alkalmazása) *Legyen* $P, Q : A \longmapsto \mathcal{L}$ *logikai függvény és* $t : A \longmapsto \mathcal{Z}$ *egy olyan variáns függvény, amelyre teljesül, hogy* $(P \wedge \neg Q) \Rightarrow t > 0$. *Ha* $\forall m \in \mathcal{N} :: (P \wedge \neg Q \wedge t = m) \hookrightarrow_S ((P \wedge t < m) \vee Q)$, *akkor* $P \hookrightarrow_S Q$.

Biz.: Teljes indukcióval belátjuk, hogy $\forall m \in \mathcal{N} : (P \wedge \neg Q \wedge t = m \hookrightarrow_S Q)$. Ebből a 3.32. def. alapján, a diszjunktivitás felhasználásával
$P \wedge \neg Q \wedge (\bigvee_{m \in \mathcal{N}} (t = m)) \hookrightarrow_S Q$, azaz $P \wedge \neg Q \wedge t > 0 \hookrightarrow_S Q$ adódik.
A feltétel szerint $P \wedge \neg Q \Rightarrow t > 0$, így $(P \wedge \neg Q \wedge t > 0) \equiv P \wedge \neg Q$.
Tehát $P \wedge \neg Q \hookrightarrow_S Q$. A 3.24. lemma szerint $P \wedge Q \hookrightarrow_S Q$. A 3.32. def., a diszjunktivitás alkalmazásával: $P \hookrightarrow_S Q$.
Teljes indukció:
Alapeset: $m = 1$. A tétel feltétele szerint: $(P \wedge \neg Q \wedge t = 1) \hookrightarrow_S ((P \wedge t < 1) \vee Q)$.
Tudjuk, hogy $P \wedge \neg Q \Rightarrow t > 0$, így $(P \wedge t < 1) \vee Q \equiv (P \wedge \neg Q \wedge t < 1) \vee (P \wedge Q \wedge t <$

$1) \vee Q \equiv \text{Hamis} \vee (P \wedge Q \wedge t < 1) \vee Q \equiv Q$.
Indukciós feltevés: $\forall k : k > 0 \wedge k < m :: (P \wedge \neg Q \wedge t = k \hookrightarrow_S Q)$. A 3.32. def. alapján, a diszjunktivitás felhasználásával: $(P \wedge \neg Q \wedge t > 0 \wedge t < m) \hookrightarrow_S Q$. A 3.24. lemma alapján: $Q \hookrightarrow_S Q$. A diszjunkció ismételt alkalmazásával: $(P \wedge \neg Q \wedge t > 0 \wedge t < m) \vee Q \hookrightarrow_S Q$. A tétel feltétele szerint: $(P \wedge \neg Q \wedge t = m) \hookrightarrow_S ((P \wedge \neg Q \wedge t < m) \vee (P \wedge Q \wedge t < m) \vee Q)$. $P \wedge \neg Q \Rightarrow t > 0$, így $((P \wedge \neg Q \wedge t < m) \vee (P \wedge Q \wedge t < m) \vee Q) \equiv (P \wedge \neg Q \wedge t > 0 \wedge t < m) \vee Q$. Tehát $(P \wedge \neg Q \wedge t = m) \hookrightarrow_S (P \wedge \neg Q \wedge t > 0 \wedge t < m) \vee Q$. A 3.32. def. szerint, a tranzitivitás alkalmazásával: $(P \wedge \neg Q \wedge t = m) \hookrightarrow_S Q$. $\square$

5.5. Tétel. ($\hookrightarrow_h$ finomítása variánsfüggvény alkalmazásával) *Legyen $P, Q : A \longmapsto \mathcal{L}$ logikai függvény és $t : A \longmapsto \mathcal{Z}$ egy olyan variáns függvény, amelyre teljesül, hogy $P \wedge \neg Q \Rightarrow t > 0$. Ha $\forall m \in \mathcal{N} ::$ S megfelel a $(P \wedge \neg Q \wedge t = m) \hookrightarrow_h ((P \wedge t < m) \vee Q)$ specifikációs feltételnek, akkor S megfelel a $P \hookrightarrow_h Q$ specifikációs feltételnek is.*

Biz.: Az előző tétel (5.4.) bizonyítása alapján a bizonyítás megkonstruálható. A bizonyítás során a 3.32. def. helyett a 5.2. lemmára kell hivatkozni. A 3.24. lemma helyett pedig a lemma azon következményére van szükség, amely szerint tetszőleges S program megfelel a $Q \wedge P \hookrightarrow_h Q$ feltételnek. $\square$

5.6. Lemma. *Legyen $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$, válasszunk egy olyan t variáns függvényt, amelyre: $K \wedge \neg fixpont_S \Rightarrow t > 0$. Ha minden $t' \in \mathcal{N}$-re $(K \wedge \neg fixpont_S \wedge t = t') \hookrightarrow_S (K \wedge t < t') \vee fixpont_S$, akkor S megfelel tetszőleges Q logikai függvény esetén a $Q \in \mathrm{TERM}_h$ specifikációs feltételnek (pl. a $Igaz \in \mathrm{TERM}_h$ specifikációs feltételnek is).*

Biz.: A feltétel és a 5.4. tétel szerint: $K \hookrightarrow_S fixpont_S$. A 3.28. tétel alapján bármely P logikai függvénnyel szűkíthetjük $\hookrightarrow_S$ baloldalát: $P \wedge K \hookrightarrow_S fixpont_S$. A 4.10. def. alapján $P{::=}sp(s_0, \text{Igaz})$, ill. $P{::=}sp(s_0, Q)$ választás mellett S megfelel a $\text{Igaz} \in \mathrm{TERM}_h$, $Q \in \mathrm{TERM}_h$ specifikációs feltételeknek. $\square$

Legyen $K \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$. Ha választunk egy olyan t variáns függvényt, amelyre $K \wedge \neg fixpont_S \Rightarrow t > 0$, akkor $INV_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q) \Rightarrow K$ miatt: $INV_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q) \wedge \neg fixpont_S \Rightarrow t > 0$. Ha minden $t' \in \mathcal{N}$-re $(INV_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q) \wedge \neg fixpont_S \wedge t = t') \hookrightarrow_S (INV_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q) \wedge t < t') \vee fixpont_S$, akkor S megfelel a tetszőleges Q logikai függvény esetén a $Q \in \mathrm{TERM}_h$ specifikációs feltételnek a tétel szerint. A 3.15. tétel szerint $INV_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$ éppen a program által elérhető állapotok halmaza. Ezért megfogalmazhatjuk az alábbi tételt:

5.7. Tétel. (Biztosan fixpontba jut) *Legyen* $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$. *Válasszunk egy olyan* t *variáns függvényt, amelyre* $K \wedge \neg fixpont_S \Rightarrow t > 0$.
Ha S *megfelel a* $\neg fixpont_S \wedge t = t' \hookrightarrow_h (t < t') \vee fixpont_S$ *specifikációs feltételnek minden* $t' \in \mathcal{N}$*-re, akkor* S *megfelel a* $Igaz \in \mathrm{TERM}_h$ *specifikációs feltételnek is, azaz biztosan fixpontba jut.*

5.1. Megjegyzés. *A 5.7. tétel nem használható feladatok finomítására, mert csak akkor alkalmazható, ha a* $fixpont_S$ *állítás ismert, azaz a program adott. A 5.7. tétel helyességbizonyítási eszköz.*

5.8. Tétel. (A fixpontfeltétel finomítása) *Ha* S *megfelel a* $inv_h P$, $FP_h \Rightarrow R$ *specifikációs feltételeknek és* $P \wedge R \Rightarrow Q$*, akkor megfelel a* $FP_h \Rightarrow Q$ *specifikációs feltételnek is.*

Biz.: A 3.32. és a 4.10. lemma alapján elegendő megmutatni, hogy S megfelel a $\mathrm{FP}_h \Rightarrow P \wedge R$ feltételnek. A feltétel és a 4.10. lemma szerint van olyan I invariáns, hogy $I \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$ és $fixpont_S \wedge I \Rightarrow R$. 3.11. lemma szerint ekkor $I \wedge P \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$. Így található olyan $I' ::= I \wedge P$ invariáns, hogy $I' \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$ és $fixpont_S \wedge I' \Rightarrow P \wedge R$. A 4.10. lemma alapján ezzel beláttuk, hogy S megfelel a $\mathrm{FP}_h \Rightarrow P \wedge R$ feltételnek. $\square$

5.3. Feladatok

5.1. Feladat. *Legyen N egy természetes szám!*
$V = vector([1..N] : \mathcal{Z})$
$\mathcal{H} = set([1..N])$

$$A = \underset{a}{V} \times \underset{h}{\mathcal{H}} \times \underset{m}{\mathcal{Z}} \qquad B = \underset{a'}{V}$$

$$a = a' \in INIT_{a'} \tag{1}$$

$$Igaz \in TERM_{a'} \tag{2}$$

$$R \in FP_{a'} \tag{3}$$

$$R = \forall j \in [1..N] : ((a[j] \leq m) \ \wedge \ (a[j] = m \Leftrightarrow j \in h)) \wedge$$
$$\wedge \ \exists j \in [1..N] : (a[j] = m) \ \wedge \ a = a'$$

Vizsgáljuk meg, hogy az alábbi program megfelel-e a fenti specifikációnak?

$s_0 : m, h := -\infty, \emptyset$
{
$$S ::= \quad \underset{i=1}{\overset{N}{\Box}} h, m := \begin{cases} h \setminus \{i\}, m & , ha\ a[i] < m \\ h \cup \{i\}, a[i] & , ha\ a[i] \geq m \end{cases}$$
}

5.2. Feladat. *Legyen* $\mathcal{B} = \{0, 1\}$*, és* $n \geq 1$ *tetszőleges természetes szám.*
Legyen $f : \mathcal{B}^n \mapsto \mathcal{B}^n$ *monoton növekedő függvény.*
$f = (f_1, ..f_n)$*, ahol* $f_i : \mathcal{B}^n \mapsto \mathcal{B}$.

$$A = \underset{a}{\mathcal{B}^n}$$

$$B = \underset{a'}{\mathcal{B}^n}$$

$$Igaz \in TERM$$

$$(f(a) = a) \in FP$$

Vizsgáljuk meg, hogy az alábbi program megfelel-e a fenti specifikációnak?

$s_0 : a := \underline{0}$
{
$$\underset{i=1}{\overset{n}{\Box}} a_i := f_i(a)$$
}

5.3. Feladat. $A = \mathcal{Z}^n \;\; a_1, \ldots, a_n : \mathcal{Z}$,
$B = \mathcal{Z}^n \;\; a'_1, \ldots, a'_n : \mathcal{Z}$,
$Q ::= \forall i \in [1..n] : (a_i = a'_i \wedge a'_i > 0)$
Spec. 1.

$$Q \in INIT_{a'_1,\ldots,a'_n} \tag{5.1}$$
$$Q \hookrightarrow FP_{a'_1,\ldots,a'_n} \tag{5.2}$$
$$FP_{a'_1,\ldots,a'_n} \Rightarrow a_1 = lnko(a'_1, \ldots, a'_n) \tag{5.3}$$

Spec. 2.

$$inv_{a'_1,\ldots,a'_n}(lnko(a'_1, \ldots, a'_n) = lnko(a_1, \ldots, a_n)) \tag{5.4}$$
$$FP_{a'_1,\ldots,a'_n} \Rightarrow a_1 = a_2 = \ldots = a_n \tag{5.5}$$

$v ::= \sum_{i=1}^{n} a_i$
Igaz-e, hogy a 5.4-5.5 specifikáció finomítása a 5.2 feltételnek? Igaz-e, hogy az alábbi program megoldja a 5.4-5.5 feltételekkel és a v variánsfüggvény bevezetésével finomított feladatot?

$$s_0 : \; SKIP$$
$$S : \{ \underset{i,j \in [1..n]}{\Box} a_i := a_i - a_j, \text{ ha } a_i > a_j \}$$

5.4. Feladat. $V = vektor([1..n], \mathcal{N})$
$A = V \times V \;\; x, y : V$,
$B = V \;\; x' : V$,
$Q ::= (n > 1)$
Spec. 1.

$$Q \in INIT_{x'}$$
$$Q \hookrightarrow FP_{x'}$$
$$FP_{x'} \Rightarrow y \in perm(x') \wedge r(y),$$

ahol $perm(y)$ *azon vektorok halmaza, amelyeket y-ból az y elemeinek permutálásával kapunk,* $r(y) ::= \forall i, j \in [1..n] : i < j \rightarrow y(i) \leq y(j)$.
Spec. 2. Bővítjük az állapotteret $k : \mathcal{N}$*-nel.*

$$inv_{x'}(y(1..k) \in perm(x'(1..k)) \wedge r(y(1..k))) \tag{5.6}$$
$$FP_{x'} \Rightarrow k = n \tag{5.7}$$

$v::=n-k$

Igaz-e, hogy a 5.6-5.7 specifikáció finomítása a 5.4 feltételnek? Igaz-e, hogy az alábbi program megoldja a 5.6-5.7 feltételekkel és a v variánsfüggvény bevezetésével finomított feladatot?

$$s_0: \ k := 0$$

$$S: \{k := k+1 || y(1..k+1) := beszur(y(1..k), x(k+1)), \text{ ha } k < n\},$$

ahol $beszur(y(1..k), x(k+1))::= \underset{i\in[1..n]}{||} y(1..k+1)(i) := f(y,k,x(k+1),i)$ *és*

$$f(y,k,x(k+1),i)::=\begin{cases} y(i), & ha\ x(k+1) > y(i) \wedge i \in [1..k] \\ y(i-1), & ha\ x(k+1) \leq y(i-1) \wedge i \in [2..k+1] \\ x(k+1), & ha\ (i = 1 \vee x(k+1) > y(i-1)) \wedge \\ & \quad (x(k+1) \leq y(i)) \wedge i \in [2..k] \end{cases}$$

Igaz-e, hogy $r(x) \Rightarrow lf(y := beszur(x,a), y \in perm(x,a) \wedge r(y))$*?*

5.5. Feladat. *Adottak az* $f,g,h: N_0 \to N_0$ *függvények, melyekre:*

$$\forall t: f(f(t)) = f(t)$$

$$f(t) \geq t \wedge f(t+1) \geq f(t)$$

(Hasonlóan g-re illetve h-ra is.)

Legyen továbbá $com: N_0 \to \mathcal{L}$ *a következő:*

$$com(t) \Leftrightarrow t = f(t) = g(t) = h(t)$$

Tekintsük a következő specifikációt:

$$inv: R \geq 0 \wedge \forall i \in [0,R): \neg com(i)$$

$$FP \to com(R)$$

$$\neg FP \wedge R = k \hookrightarrow R > k$$

Igaz-e,, hogy a következő program megfelel a fenti specifikációnak:

INIT: $R = 0$

$$\{R := f(R) \sqcup R := g(R) \sqcup R := h(R)\}$$

5.6. Feladat. *Adott az $A[0..N]$ vektor, amelynek elemei természetes számok. $N \geq 0$. Rendezzük az A vektort növekvően!*

- *a) Írjuk fel a feladat specifikációját!*
- *b) Bizonyítsuk be, hogy a következő program megfelel a specifikációnak:*

$$\underset{0 \leq i < N}{\cup} \{A[i], A[i+1] := A[i+1], A[i], \text{ ha } A[i] > A[i+1]\}$$

5.7. Feladat. *Feladat: számoljuk ki a Pascal háromszög első N sorát (a 0-tól az N-ig)!*

Specifikáció:

$$FP \Rightarrow (\forall n \in [0..N]; k \in [0..n] : c[n,k] = \binom{n}{k})$$

Az invariáns finomítśához használjuk fel:

$$\binom{n}{0} = 1; \ \binom{n}{n} = 1; \ \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

Fejezzük be a specifikációt, majd lássuk be, hogy az alábbi program megoldja a feladatot, ill. megfelel a specifikációnak! A fenti specifikációhoz adott a program:

$$s_0 : \ (\forall i \in [0..N]; j \in [0..i] : \ c[i,j:] = 0)$$

$$\bigcup_{n=0}^{N} \{\| \ 0 \leq i \leq n : c[n,0], c[n,n] := 1,1\} \bigcup$$

$$\bigcup \bigcup_{n=2}^{N} \{\| \ 1 \leq k \leq N-1 : c[n,k] := c[n-1,k-1] + c[n-1,k]\}$$

6. fejezet

Programkonstrukciók

Ebben a fejezetben programok és feladatok konstrukciós módszereit ismertetjük, azaz megadjuk, hogy meglévő feladatokat hogyan bonthatunk részfeladatokra, illetve programokból milyen szabályok alapján hozhatunk létre összetett programot. Bevezetjük a kompozícionális modell fogalmát, majd megvizsgáljuk, hogy a programkonstrukciós szabályokkal kiegészített modell mennyiben felel meg a kompozícionalitás követelményének.

Feladatok és programok konstrukcióit, mint egy vagy több relációhoz egy relációt hozzárendelő leképezéseket definiáljuk.

*Megengedett konstrukciós művelet*nek nevezünk a továbbiakban minden olyan relációs műveletet, amely
- relációk uniója, metszete, különbsége,
- relációk adott pontban felvett értékének uniója, metszete, különbsége,
- relációk vetítése, kiterjesztései, rendezett reláció n-esek komponenseire bontása és komponensekből rendezett reláció n-esek előállítása,
- relációk direkt szorzata,
- relációk kompozíciója, szigorú kompozíciója és lezártjai (lezárt, korlátos lezárt, tranzitív, diszjunktív lezárt),

véges sokszori egymás utáni alkalmazásaként[1] megfogalmazható.

A modellben *feladatkonstrukció*nak nevezzük azokat a megengedett konstrukciós műveleteket, amelyek egy vagy több feladatból egy feladatot állítanak elő. *Programkonstrukció*nak nevezzük azokat a megengedett konstrukciós műveleteket, amelyek egy vagy több programból egy új, összetett programot állítanak elő.

[1] A definíció nem formális. A modell keretein túlmutat annak vizsgálata, hogy a fent felsorolt elemi műveletek valamilyen értelemben teljes rendszert alkotnak-e.

6.1. Definíció. *Kompozícionálisnak nevezünk egy programozási modellt, ha a modell minden $\mathcal{F}$ feladatkonstrukciójához létezik olyan $\mathcal{S}$ programkonstrukció, hogy S_1 megoldása F_1 és S_2 megoldása F_2 esetén $S_1 \mathcal{S} S_2$ megoldja $F_1 \mathcal{F} F_2$-t.*

6.2. Definíció. Részlegesen kompozícionális *egy programozási modell, ha egyes kiválasztott $\mathcal{F}$ feladatkonstrukciók esetén megadható olyan $\mathcal{S}$ programkonstrukció, amelyre S_1 megoldása F_1 és S_2 megoldása F_2 esetén $S_1 \mathcal{S} S_2$ megoldja $F_1 \mathcal{F} F_2$-t.*

A továbbiakban megmutatjuk, hogy a bevezetett modell részlegesen kompozícionális.

Jelöljük az S_1 program viselkedési relációját
$p(S_1)::=(\rhd_{S_1}, \mapsto_{S_1}, \hookrightarrow_{S_1}, \mathrm{FP}_{S_1}, \mathrm{TERM}_{S_1}, \mathrm{inv}_{S_1})$-vel, az S_2 program viselkedési relációját $p(S_2)::=(\rhd_{S_2}, \mapsto_{S_2}, \hookrightarrow_{S_2}, \mathrm{FP}_{S_2}, \mathrm{TERM}_{S_2}, \mathrm{inv}_{S_2})$-vel.

6.1. Unió

6.3. Definíció (Unió). *Legyen az A_1, A_2 tér az A állapottér altere. Jelölje B az A_1, A_2 terek legnagyobb közös alterét. Legyen S_1 az A_1, S_2 az A_2 altér felett definiált program kiterjesztése (3.22. def.) A-ra. $S_1 = (s_{1,0}, \{s_{1,1}, \dots, s_{1,k}\})$, $S_2 = (s_{2,0}, \{s_{2,1}, \dots, s_{2,m}\})$.*
Az $S = (s_0, \{s_1, \dots, s_k, s_{k+1}, s_{k+2}, \dots, s_{k+m}\})$ programot, amely az A állapottéren adott, az S_1 és S_2 program uniójának *nevezzük és $S_1 \cup S_2$-vel jelöljük, ha minden B altérhez tartozó v_i változóra teljesül, hogy az $s_{1,0}$ és $s_{2,0}$ értékadások azonos értéket rendelnek hozzá, azaz: $F_{1,0_i} = F_{2,0_i}$ és*
$s_0 = s_{1,0} \| s_{2,0}$,
$\forall i \in [1..k] : s_i = s_{1,i}$,
$\forall i \in [k+1..k+m] : s_i = s_{2,i-k}$.

6.1. Tétel. (Unió viselkedési relációja) *Legyen $S::=(S_1 \cup S_2)$. Ekkor*[2]*:*

(1) $\rhd_S = \rhd_{S_1} \cap \rhd_{S_2}$

(2) $\mapsto_S = \rhd_{S_1} \cap \rhd_{S_2} \cap (\mapsto_{S_1} \cup \mapsto_{S_2})$

(3) $(\rhd_{S_1} \cap \rhd_{S_2} \cap (\mapsto_{S_1} \cup \mapsto_{S_2}))^{tdl} = \hookrightarrow_S$

[2] A tétel (1)-es,(2)-es és (5)-ös pontja a UNITY unió tételének relációs alakja [Cha Mis 89].

(4) $\forall Q$*-ra, amelyre* $sp(s_{1,0} \| s_{2,0}, Q) \Rightarrow sp(s_{1,0}, Q) \wedge sp(s_{2,0}, Q)$*:*
$inv_{S_1}(Q) \cap inv_{S_2}(Q) \subseteq inv_S(Q)$[3]

(5) $\varphi_S = \varphi_{S_1} \wedge \varphi_{S_2}$

(6) $FP_{S_1} \cap FP_{S_2} \subseteq FP_S$

(7) $((\rhd_{S_1} \cap \rhd_{S_2} \cap (\mapsto_{S_1} \cup \mapsto_{S_2}))^{tdl})^{(-1)}(\varphi_{S_1} \wedge \varphi_{S_2}) = \text{TERM}_S.$

Biz.:
(1) A 3.26. és a 6.3. def. alapján: $lf(S_1 \cup S_2, P \vee Q) = \bigwedge\limits_{s \in S_1 \cup S_2} lf(s, P \vee Q) =$
$= \bigwedge\limits_{s \in S_1} lf(s, P \vee Q) \wedge \bigwedge\limits_{s \in S_2} lf(s, P \vee Q) = lf(S_1, P \vee Q) \wedge lf(S_2, P \vee Q).$

Tegyük fel, hogy $P \rhd_S Q$. Ekkor a 3.30. def. és a $lf(S_1 \cup S_2, P \vee Q) = lf(S_1, P \vee Q) \wedge lf(S_2, P \vee Q)$ egyenlőség felhasználásával:
$P \wedge \neg Q \Rightarrow lf(S_1, P \vee Q) \wedge lf(S_2, P \vee Q)$. A jobboldal gyengítésével: $P \wedge \neg Q \Rightarrow lf(S_1, P \vee Q)$ ill. $P \wedge \neg Q \Rightarrow lf(S_2, P \vee Q)$, azaz $P \rhd_{S_1} Q$ és $P \rhd_{S_2} Q$.
Ha $P \rhd_{S_1} Q$ és $P \rhd_{S_2} Q$, akkor $P \wedge \neg Q \Rightarrow lf(S_1, P \vee Q)$ és $P \wedge \neg Q \Rightarrow lf(S_2, P \vee Q)$, így $P \wedge \neg Q \Rightarrow lf(S_1, P \vee Q) \wedge lf(S_2, P \vee Q)$. A bizonyított egyenlőség és a 3.30. def. alapján: $P \rhd_S Q$.

(2) Tegyük fel, hogy $P \mapsto_S Q$.
A 3.31. def. szerint $P \rhd_{S_1 \cup S_2} Q \wedge \exists s \in S_1 \cup S_2 : P \wedge \neg Q \Rightarrow lf(s, Q)$
$\Longleftrightarrow$ { (1) és a 6.3. def. felhasználásával }
$P \rhd_{S_1} Q$, $P \rhd_{S_2} Q$ és
$(\exists s \in S_1 : P \wedge \neg Q \Rightarrow lf(s, Q) \vee \exists s \in S_2 : P \wedge \neg Q \Rightarrow lf(s, Q))$, azaz
$\Longleftrightarrow$ { 3.31. def. }
$P \rhd_{S_1} Q$ és $P \rhd_{S_2} Q$ és ($P \mapsto_{S_1} Q$ vagy $P \mapsto_{S_2} Q$).

(3) A 3.32. def. és (2) következménye.

(4) Tegyük fel, hogy $P \in inv_{S_1}(Q)$ és $P \in inv_{S_2}(Q)$. A 3.27. def. szerint ekkor $sp(s_{1,0}, Q) \Rightarrow P$ és $sp(s_{2,0}, Q) \Rightarrow P$. Ekkor $sp(s_{1,0}, Q) \wedge sp(s_{2,0}, Q) \Rightarrow P$ is teljesül, így a feltétel miatt $sp(s_{1,0} \| s_{2,0}, Q) \Rightarrow P$.
A 3.27. def. alapján $P \Rightarrow lf(S_1, P)$ és $P \Rightarrow lf(S_2, P)$. A 3.26. és 6.3. def. szerint

[3] Az állítás általánosítható:
$\forall Q, P$: ha $sp(s_{1,0} \| s_{2,0}, Q) \Rightarrow P$ and $P \in (\rhd_{S_1}^{(-1)}(\text{Hamis}) \cap \rhd_{S_2}^{(-1)}(\text{Hamis}))$, akkor $P \in \text{inv}_S(Q)$.

$P \Rightarrow lf(S_1 \cup S_2, P)$.

(5) 3.34. def. és 6.3. def. közvetlen következménye.

(6) (5) következménye. Ha $\varphi_{S_1} \Rightarrow R$ és $\varphi_{S_2} \Rightarrow R$, akkor $\varphi_{S_1} \wedge \varphi_{S_2} \Rightarrow R$.

(7) A 3.36. def. és a (3),(5) állítások következménye. □

6.1. Megjegyzés ($\exists S_1, S_2, Q : \mathbf{inv}_{S_1 \cup S_2}(Q) \neq \mathbf{inv}_{S_1}(Q) \cap \mathbf{inv}_{S_2}(Q)$). *A 6.1. tétel (4)-es pontjának bizonyításához felhasználtuk, hogy $sp(s_{1,0}, Q) \Rightarrow P$ és $sp(s_{2,0}, Q) \Rightarrow P$ esetén $sp(s_{1,0}, Q) \wedge sp(s_{2,0}, Q) \Rightarrow P$ is teljesül. Ez a segédállítás nem megfordítható. A 6.1. példa segítségével megmutatjuk, hogy az összetett program invariánsa nem invariánsa az összetevőknek.*

6.1. Példa. S_1::=$(x := 1, \{SKIP\})$. S_2::=$(y := 1, \{SKIP\})$. Q::=*Igaz*. P::=$(x = 1 \wedge y = 1)$. *Könnyen ellenőrizhető, hogy $sp(s_{1,0}, Q) = (x = 1)$, $sp(s_{2,0}, Q) = (y = 1)$. $sp(s_{1,0}, Q) \wedge sp(s_{2,0}, Q) \Rightarrow P$, de $sp(s_{1,0}, Q) \not\Rightarrow P$ ill. $sp(s_{2,0}, Q) \not\Rightarrow P$.* □

6.2. Megjegyzés ($\exists S_1, S_2 : \mathbf{FP}_{S_1} \cap \mathbf{FP}_{S_2} \neq \mathbf{FP}_{S_1 \cup S_2}$). *Az előző megjegyzésben leírt gondolatmenethez hasonlóan belátható, hogy $\varphi_{S_1} \wedge \varphi_{S_2} \Rightarrow R \not\Longrightarrow \varphi_{S_1} \Rightarrow R$ és $\varphi_{S_2} \Rightarrow R$.*

6.3. Megjegyzés ($\hookrightarrow_{S_1 \cup S_2}$ és $\mathrm{TERM}_{S_1 \cup S_2}$). *A $\varphi_{S_1 \cup S_2}$ logikai függvény, ill. a $\mapsto_{S_1 \cup S_2}$ reláció meghatározható az összetevő programok viselkedési relációjából, így 3.32. és a 3.36. def. alapján a $\hookrightarrow_{S_1 \cup S_2}$ ill. a $\mathrm{TERM}_{S_1 \cup S_2}$ reláció is kifejezhető közvetett módon az összetevő komponensek viselkedési relációjából a tétel (3) és (7) pontja szerint. Az unió viselkedési relációjának egyes komponensei, pl. $\hookrightarrow_{S_1 \cup S_2}$, ill. $\mathrm{TERM}_{S_1 \cup S_2}$ általában azonban nem határozhatóak meg az összetevő programok viselkedési relációjának* megfelelő *komponenseiből. Az alábbi példa (6.2.) szerint általában nem igaz, hogy $P \hookrightarrow_{S_1} Q$ és $P \hookrightarrow_{S_2} Q$ esetén $P \hookrightarrow_{S_1 \cup S_2} Q$ is teljesül.*

6.2. Példa ($\hookrightarrow_{S_1 \cup S_2}$). S_1::=$(SKIP, \{s_1 : x := x + 1\})$. S_2::=$(SKIP, \{s_2 : x := x - 1\})$.
P::=$(0 < x < 5)$. Q::=$(x \leq 0 \vee x \geq 5)$. *A 3.32. def. alapján könnyen igazolható, hogy $P \hookrightarrow_{S_1} Q$ és $P \hookrightarrow_{S_2} Q$. A 3.31. tétel alapján, ha van olyan feltétlenül pártatlan ütemezésnek megfelelő végrehajtási útja $S_1 \cup S_2$-nek, amely mentén P-ből elkerülhető Q, akkor $P \hookrightarrow_{S_1 \cup S_2} Q$ nem teljesül. Válasszuk kezdőállapotnak az* $x = 3$ *állapotot. Az $s_1, s_2, s_1, s_2 \ldots$ ütemezés feltétlenül pártatlan és sohasem jut a program Q-beli állapotba.* □

Jelöljük $\Gamma(F)$-fel az F feladatot megoldó programok halmazát.

6.4. Definíció (Feladatok egyesítése). *Legyen F_1 és F_2 közös állapot és paramétertéren[4] adott feladat.*
$\forall b \in B : (F_1 \sqcup F_2)(b) ::= \{ \, (\, (\rhd_{h_1} \cap \rhd_{h_2}), (\rhd_{h_1} \cap \rhd_{h_2} \cap (\mapsto_{h_1} \cup \mapsto_{h_2})),$
$(\rhd_{h_1} \cap \rhd_{h_2} \cap (\mapsto_{h_1} \cup \mapsto_{h_2}))^{tdl},$
$((\rhd_{h_1} \cap \rhd_{h_2} \cap (\mapsto_{h_1} \cup \mapsto_{h_2}))^{tdl})^{(-1)}((\bigwedge_{S \in \Gamma(F_1) \cup \Gamma(F_2)} \varphi_S)),$
$(FP_{h_1} \cap FP_{h_2}), (inv_{h_1} \cap inv_{h_2}), (INIT_{h_1} \cup INITh_2)) \; | h_1 \in F_1(b), h_2 \in F_2(b) \, \}.$

6.2. Tétel. (Unió levezetési szabálya) *Legyen F_1 és F_2 az A állapottér és a B paramétertér felett megadott feladat. S_1 és S_2 az A állapottérre kiterjesztett programok, amelyek uniója értelmezett (6.3. def.). Ha S_1 megoldja F_1-et K mellett és S_2 megoldja F_2-t K mellett és $\forall b \in B : \forall h_1 \in F_1(b) : \forall h_2 \in F_2(b) : sp(s_{1,0} \| s_{2,0}, (\bigwedge_{Q \in INIT_{h_1}} Q) \wedge (\bigwedge_{Q \in INIT_{h_2}} Q)) \Rightarrow sp(s_{1,0}, (\bigwedge_{Q \in INIT_{h_1}} Q)) \wedge sp(s_{2,0}, (\bigwedge_{Q \in INIT_{h_2}} Q))$, akkor $S_1 \cup S_2$ megoldja $F_1 \sqcup F_2$-t.*

Biz.: 4.14., 6.4. def. és 6.1. tétel következménye. □

Az unió levezetési szabálya (6.2. tétel) azt mondja ki, ha a modell szemantikája összefésüléses és az összetevők rendelkeznek egy közös *globális invariáns*sal [And 91], akkor az ezen invariáns tulajdonság mellett megoldott feladatok megfelelő kompozícióját az összetett program is megoldja. A 7.1. példán megmutatjuk, hogy az összefésüléses szemantika milyen lényeges az unió viselkedési relációjának meghatározhatóságában [Cha 90].

Bizonyos esetekben programok uniójának viselkedési relációja könnyen kifejezhető az összetevők viselkedési relációja alapján. Ez akkor lehetséges, ha az összetevők kölcsönhatása (*interferenciája*) az unió levezetési szabályában tett megszorításokon túl további tulajdonságokkal is jellemezhető.

A kölcsönhatás jellemzésének alkalmas módja lehet a *nyitott specifikáció*, amely az eredő program egyszerűbb (általában biztonságossági) tulajdonságaira tett kikötések segítségével tesz indirekt kikötéseket az egyik vagy mindkét összetevő tulajdonságaira. A nyitott specifikáció módszerét részletesen bemutatja Chandy és Misra [Cha Mis 89].

A szekvencia programkonstrukció esetében (6.7. def.) az unió olyan speciális esetét fogalmazzuk majd meg, ahol az unióhoz tartozó értékadásokat olyan

[4] Ha az állapottér nem közös, akkor válasszunk egy olyan teret, amelynek mindkét állapottér altere. A feladatokat terjesszük ki a közös térre (2.3. def.).

diszjunkt csoportokba sorolhatjuk, hogy az állapottér egy alkalmasan megválasztott részhalmaza felett legfeljebb egyetlen csoporthoz tartozó értékadások hatásrelációi különböznek az identitástól. Az alábbi két tétel erre a speciális esetre vonatkozik.

6.3. Tétel. (Unió és az állapottér részhalmazai) *Legyen* $S = S_1 \cup S_2$*,* π *logikai függvény az A állapottéren oly módon, hogy* $\forall s \in S_2 : p(s) \cap (\lceil \pi \rceil \times A) = id_A \cap (\lceil \pi \rceil \times A)$ *és* $\pi \rhd_{S_1}$ *Hamis.*
Ekkor

(1) ha $P \rhd_{S_1} Q$*, akkor* $P \wedge \pi \rhd_S Q \wedge \pi$*,*

(2) ha $P \mapsto_{S_1} Q$*, akkor* $P \wedge \pi \mapsto_S Q \wedge \pi$*,*

(3) ha $P \hookrightarrow_{S_1} Q$*, akkor* $P \wedge \pi \hookrightarrow_S Q \wedge \pi$*.*

Biz.:
A feltétel alapján: $\forall s \in S_2 : \forall Z : A \longmapsto \mathcal{L} :: p(s)(\lceil Z \wedge \pi \rceil) = \lceil Z \wedge \pi \rceil$, így: $Z \wedge \pi \Rightarrow lf(s, (Z \wedge \pi))$.
(1) A feltételek és 3.17. lemma szerint: $\forall s \in S_1 : (P \wedge \pi \wedge \neg(Q \wedge \pi)) \Rightarrow lf(s, (P \wedge \pi) \vee (Q \wedge \pi))$ és $\forall s \in S_2 : P \wedge \pi \wedge \neg(Q \wedge \pi) \Rightarrow lf(s, P \wedge \pi \wedge \neg(Q \wedge \pi)) \Rightarrow lf(s, (P \wedge \pi) \vee (Q \wedge \pi))$.
(2) Az előző állítás szerint: $P \wedge \pi \rhd_S Q \wedge \pi$. A feltétel és 3.21. lemma szerint: $\exists s \in S_1 : P \wedge \pi \wedge \neg(Q \wedge \pi) \Rightarrow lf(s, Q \wedge \pi)$. Ha $s \in S_1$, akkor $s \in S$, így az állítást igazoltuk.
(3) Strukturális indukcióval az induktív 3.32. def. alapján.
Alapeset: $P \hookrightarrow_{S_1} Q$-t 1 lépésben vezettük le $P \mapsto_{S_1} Q$-ból. Az előző állítás szerint ekkor $P \wedge \pi \mapsto_S Q \wedge \pi$, az 3.32. def. szerint: $P \wedge \pi \hookrightarrow_S Q \wedge \pi$.
Indukciós lépés: a) eset: az utolsó lépésben a 3.32. def. (2) pontját, a tranzitivitást alkalmaztuk, azaz: $P \hookrightarrow_{S_1} Q_1$ és $Q_1 \hookrightarrow_{S_1} Q$. Az indukciós feltétel szerint: $P \wedge \pi \hookrightarrow_S Q_1 \wedge \pi$ és $Q_1 \wedge \pi \hookrightarrow_S Q \wedge \pi$. A 3.32. def. (tranzitivitás) alapján: $P \wedge \pi \hookrightarrow_S Q \wedge \pi$.
b) eset: az utolsó lépésben a 3.32. def. (3) pontját, a diszjunktivitást alkalmaztuk, azaz: $P = \exists m : m \in W :: P(m)$ és $\forall m : m \in W :: (P(m) \hookrightarrow_{S_1} Q)$. Az indukciós feltétel szerint: $\forall m : (m \in W :: (P(m) \wedge \pi \hookrightarrow_S Q \wedge \pi))$, amiből a 3.32. def. (diszjunktivitás alapján) $P \wedge \pi \hookrightarrow_S Q \wedge \pi$. $\square$

A tétel egy kicsit módosított feltételekkel is kimondható. A (3)-as állítás megfogalmazásánál alkalmazzuk a nyitott specifikáció módszerét.

6.4. Tétel. (Unió és az állapottér részhalmazai (2.)) *Legyen* $S = S_1 \cup S_2$, π *logikai függvény az* A *állapottéren oly módon, hogy* $\forall s \in S_2 : p(s) \cap (\lceil \pi \rceil \times A) = id_A \cap (\lceil \pi \rceil \times A)$, $\pi \rhd_{S_1} Q$. *Ekkor*

(1) ha $P \rhd_{S_1} Q$, *akkor* $P \wedge \pi \rhd_S Q$,

(2) ha $P \mapsto_{S_1} Q$, *akkor* $P \wedge \pi \mapsto_S Q$,

(3) ha $\neg\pi \wedge \neg Q \rhd_{S_1}$ *Hamis és* $P \hookrightarrow_{S_1} Q$, *akkor* $P \wedge \pi \hookrightarrow_S Q$.

Biz.:
A feltétel alapján: $\forall s \in S_2 : \forall Z : A \longmapsto \mathcal{L} :: p(s)(\lceil Z \wedge \pi \rceil) = \lceil Z \wedge \pi \rceil$, így: $Z \wedge \pi \Rightarrow lf(s,(Z \wedge \pi))$.
(1) A feltételek szerint: $\forall s \in S_1 : P \wedge \neg Q \Rightarrow lf(s, P \vee Q)$ és $\pi \wedge \neg Q \Rightarrow lf(s, \pi \vee Q)$. A 3.6. lemma alapján: $P \wedge \pi \wedge \neg Q \Rightarrow lf(s,(P \vee Q) \wedge (\pi \vee Q)) = lf(s,(P \wedge \pi) \vee Q)$ és $\forall s \in S_2 : P \wedge \pi \wedge \neg Q \Rightarrow lf(s, P \wedge \pi \wedge \neg Q) \Rightarrow lf(s,(P \wedge \pi) \vee Q)$.
(2) Az előző állítás szerint: $P \wedge \pi \rhd_S Q$. A feltétel szerint: $\exists s \in S_1 : P \wedge \neg Q \Rightarrow lf(s,Q)$. $P \wedge \pi \wedge \neg Q \Rightarrow P \wedge \neg Q$. Ha $s \in S_1$, akkor $s \in S$, így az állítást igazoltuk.
(3) Struktúrális indukcióval az induktív 3.32. def. alapján.
Alapeset: $P \hookrightarrow_{S_1} Q$-t 1 lépésben vezettük le $P \mapsto_{S_1} Q$-ból. Az előző állítás szerint ekkor $P \wedge \pi \mapsto_S Q$, a 3.32. def. szerint: $P \wedge \pi \hookrightarrow_S Q$.
Indukciós lépés: a) eset: az utolsó lépésben a 3.32. def. (2) pontját, a tranzitivitást alkalmaztuk, azaz: $P \hookrightarrow_{S_1} Q_1$ és $Q_1 \hookrightarrow_{S_1} Q$ A PSP tételt (6.8. lemma) a $\neg\pi \wedge \neg Q \rhd_{S_1}$ Hamis feltételre és a $Q_1 \hookrightarrow_{S_1} Q$ indukciós feltételre alkalmazva: $Q_1 \wedge \neg\pi \wedge \neg Q \hookrightarrow_{S_1} Q \wedge \neg\pi \wedge \neg Q$. A jobboldal hamis, így a 6.9. lemma alkalmazásával azt kapjuk, hogy a baloldal is hamis, azaz: $Q_1 \Rightarrow (\pi \vee Q)$. Az indukciós feltétel szerint: $P \wedge \pi \hookrightarrow_S Q_1$. Beláttuk, hogy $Q_1 \Rightarrow Q_1 \wedge (\pi \vee Q)$, így a 3.24. lemma szerint $Q_1 \hookrightarrow_S (Q_1 \wedge \pi) \vee (Q_1 \wedge Q)$, a 3.29. lemma felhasználásával $Q_1 \hookrightarrow_S (Q_1 \wedge \pi) \vee Q$. Az indukciós feltétel szerint: $Q_1 \wedge \pi \hookrightarrow_S Q$. 3.24. lemma alapján: $Q \hookrightarrow_S Q$ A 3.32. def. (diszjunktivitás) alapján: $(Q_1 \wedge \pi) \vee Q \hookrightarrow_S Q$. A 3.32. def. (tranzitivitás) kétszeri alkalmazásával: $P \wedge \pi \hookrightarrow_S Q$.
b) eset: az utolsó lépésben a 3.32. def. (3) pontját, a diszjunktivitást alkalmaztuk, azaz: $P = \exists m : m \in W :: P(m)$ és $\forall m : m \in W :: (P(m) \hookrightarrow_{S_1} Q)$. Az indukciós feltétel szerint: $\forall m : (m \in W :: (P(m) \wedge \pi \hookrightarrow_S Q))$, amiből a 3.32. def. (diszjunktivitás alapján) $P \wedge \pi \hookrightarrow_S Q$. $\square$

Az összetett program olyan programtulajdonságait is megfogalmazhatjuk, amelyek érvényessége csak olyan változóktól függ, amelyek csak az egyik összetevőben állnak a baloldalon. Az ilyen tulajdonságokat *lokálisaknak* nevezzük [Cha Mis 89].

Kimondjuk az általános lokalitás tételt ([UN 88-93]/17-90), a bizonyítás Singh, Misra és Knapp bizonyításai alapján a relációs modell eszközeivel is megkonstruálható. A tétel 4. állítása a nyitott specifikáció technikáját alkalmazza.

6.5. Tétel. (Lokalitás tétel - általános alak) *Legyen S_1 és S_2 közös állapottéren értelmezett program. X::=$V(S_1) \cap V(S_2)$. Ha $VR(P) \subseteq V(S_1)$*[5]*, akkor*

(1) $P \rhd_{S_1} Q \Longrightarrow P \wedge (X = M) \rhd_{S_1 \cup S_2} Q \vee (X \neq M)$

(2) $P \mapsto_{S_1} Q \Longrightarrow P \wedge (X = M) \mapsto_{S_1 \cup S_2} Q \vee (X \neq M)$

(3) $P \hookrightarrow_{S_1} Q \Longrightarrow P \wedge (X = M) \hookrightarrow_{S_1 \cup S_2} Q \vee (X \neq M)$

(4) *Adjunk meg egy egészértékű t variáns függvényt az X változóhalmaz érték-n-esei felett. $P \Rightarrow t(X) > 0$ és $P \hookrightarrow_{S_1} Q$ és $P \wedge (t(X) = m) \rhd_{S_1 \cup S_2} (P \wedge (t(X) < m)) \vee Q \Longrightarrow P \hookrightarrow_{S_1 \cup S_2} Q$.*

Biz.: Ha $VR(P) \subseteq V(S_1)$, akkor $VR(P) \cap V(S_2) \subseteq X$.
Lemma:[6] $P \wedge (X = M) \rhd_{S_2} (X \neq M)$.
Lemma bizonyítása[7]: Legyen $s \in S_2$. Ha $p(s)(\lceil P \wedge X = M \rceil) \subseteq \lceil X = M \rceil$, akkor $\forall a \in \lceil P \wedge X = M \rceil : \forall v \in VR(P) : v \circ p(s)(a) = v(a)$, így:
$p(s)(\lceil P(VR(P)) \wedge X = M \rceil) \subseteq \lceil P(VR(P)) \rceil$.
Ha $p(s)(\lceil P \wedge X = M \rceil) \not\subseteq \lceil X = M \rceil$, akkor
$p(s)(\lceil P(VR(P)) \wedge X = M \rceil) \cap \lceil X = M \rceil \subseteq \lceil P(VR(P)) \rceil$ és
$p(s)(\lceil P(VR(P)) \wedge X = M \rceil) \cap \lceil X \neq M \rceil \subseteq \lceil X \neq M \rceil$, azaz
$p(s)(\lceil P(VR(P)) \wedge X = M \rceil) \subseteq \lceil X \neq M \vee P \rceil$. A 3.10. lemma szerint $P \wedge X = M \Rightarrow lf(S_2, P \vee X \neq M)$, amelyből ekvivalens átalakítással: $P \wedge X = M \wedge X = M \Rightarrow lf(S_2, (P \wedge X = M) \vee X \neq M)$, azaz a 3.30. def. szerint: $P \wedge X = M \rhd_{S_2} X \neq M$. $\square$
(1), (2), (3), (4) biz. megtalálható [UN 88-93]-ban. A bizonyítás a most bizonyított lemmára és a $\rhd_S$, $\mapsto_S$, $\hookrightarrow_S$ relációk korábban bizonyított tulajdonságaira épül. $\square$

---

[5] $VR(P) \cap V(S_2) \subseteq X$

[6] Misra lokalitási axiómája

[7] $P(VR(P))$-vel azt a függvénykompozíciót jelöljük, amely a $VR(P)$ halmazhoz tartozó változókból, mint az állapottér projekciós függvényeiből, a $VR(P)$-ben nem szereplő változók helyett az identitásfüggvényből, ill. a P logikai függvényből állítható elő. Ha a $VR(P)$-hez tartozó függvények értéke változatlan, akkor $P(VR(P))$ függvénykompozíció értéke sem változik meg.

6.2. Szuperpozíció

6.5. Definíció (Szuperpozíció). *Legyen az $S = (s_0, \{s_1, \ldots, s_m\})$ program az A állapottér A_1 altere és az s feltételes értékadás az A állapottér felett definiálva oly módon, hogy $VL(s)$ az A_1 altér egyetlen változóját sem tartalmazza. Jelölje $s_j \| s$ az s_j és az s utasítás szuperpozícióját (3.21. def.). Legyen az $S' = (s'_0, \{s'_1, \ldots, s'_m\})$ az S kiterjesztése A-ra (3.22. def.). Az*
a) $(s'_0, \{s'_1, \ldots, s'_m, s\})$ ill. az
b) $S = (s'_0, \{s'_1, \ldots (s'_j \| s), \ldots s'_m\})$, ahol $j \in [1, m]$
alakú programokat az S program és az s utasítás szuperpozíciójának nevezzük.

6.6. Tétel. (Szuperpozíció viselkedési relációja) *Legyen az A állapottéren adott S'' program az A_1 alterén adott S program és az $s ::= \underset{i\in[1,n]}{\|} (v_i :\in F_i(v_1, \ldots, v_n)$, ha $\pi_i)$ utasítás egy szuperpozíciója. Jelöljük az A_1 altéren adott P, Q logikai függvények A-ra való kiterjesztését P', Q'-vel. Ekkor[8]:*

(1) $P \rhd_S Q \Longrightarrow P' \rhd_{S''} Q'$,

(2) $P \mapsto_S Q \Longrightarrow P' \mapsto_{S''} Q'$,

(3) $P \hookrightarrow_S Q \Longrightarrow P' \hookrightarrow_{S''} Q'$,

(4) $\forall Q : P \in inv_S(Q) \Longrightarrow P' \in inv_{S''}(Q')$,

(5) $\varphi_{S''} = \varphi'_S \wedge \varphi_s$,

(6) $R \in FP_S \Longrightarrow R' \in FP_{S''}$,

ahol φ'_S a φ_S logikai függvény kiterjesztése és
$\varphi_s ::= (\underset{i\in[1..n]}{\wedge} (\neg\pi \vee (\pi_{id} \wedge v_i = F_i(v_1, \ldots, v_n))))$.

Biz.:
(1), (2), (4) a 3.1. következmény és 3.9. lemma következménye.
(3) a (2)-es pontból 3.32. def. alapján ($P \hookrightarrow_S Q$ előállítása szerinti strukturális indukcióval).
(5) 2.2. és 3.34. definíciókból közvetlenül adódik.
(6) az (5) állítás és 3.35. def. következménye. □

[8] A tétel (1)-es, (2)-es, (3)-as, (4)-es pontja a UNITY szuperpozíció tételének relációs alakja [Cha Mis 89].

6.4. Megjegyzés. *Ha a szuperpozíció a) típusú, akkor a tétel (1),(2),(3) állítása a lokalitás tétel következménye.*

6.6. Definíció (Feladat gyenge kiterjesztése). *F'' az F feladat kiterjesztésének gyengítése, ha az F kiterjesztéséből, F'-ből, a $Q \in \mathrm{TERM}_h$ típusú specifikációs feltételek elhagyásával kapjuk.*

6.7. Tétel. (Szuperpozíció levezetési szabálya) *Legyen F az A állapottér A_1 altere és a B paramétertér felett megadott feladat. Ha az S program megoldja az F feladatot, akkor az S program és az s utasítás bármely szuperpozíciója megoldja az F feladat gyenge kiterjesztését.*

Biz.: A 4.1. definícióból és a 6.6. tételből következik. □

6.3. Szekvencia

6.7. Definíció (Szekvencia). *Legyen $S_1 = (s_{1,0}, \{s_{1,1}, \dots, s_{1,k}\})$ az A állapottér $A_1 = \times_{i \in I_1} A_{1i}$ altere felett, $S_2 = (s_{2,0}, \{s_{2,1}, \dots, s_{2,m}\})$ pedig az $A_2 = \times_{i \in I_2} A_{2i}$ altér felett definiált program. Legyen u egy logikai változó, amelyekhez tartozó állapottérkomponensek nem tartoznak sem az A_1 sem az A_2 altérhez.*
Jelöljük S^1-gyel az $\times_{i \in I_1} A_{1i} \times \mathcal{L}$ altéren definiált $(s_0, \{s_1, \dots, s_k\})$ programot, ahol $s_0 = (s'_{1,0} \| u := hamis)$ (→ 3.21. def.),
$\forall i \in [1..k] s_i = ((s'_{1,i})\ ha\ \neg u)$ (→ 3.20. def.).
Jelöljük S^2-vel az $\mathcal{L} \times_{i \in I_2} A_{2i}$ állapottéren definiált $(s_0, \{s_{k+2}, \dots, s_{k+m+1}\})$ programot, ahol $\forall i \in [k+2..k+m+1] : s_i = ((s'_{2,i-(k+2)+1})\ ,\ ha\ u)$.
Legyen $s_{k+1} = ((s'_{2,0} \| u := igaz),\ ha\ \neg u \wedge \varphi_{S_1})$,
Az $S = (S^{1'} \cup S^{2'} \cup (s_0, \{s_{k+1}\})')$ programot az S_1, S_2 programok szekvenciáj*ának nevezzzük, és $S_1; S_2$-vel jelöljük.*

A szekvenciát tehát feltételekkel kiegészített értékadások és unió segítségével definiáltuk.

Kimondunk néhány segédtételt:

6.8. Lemma (PSP). [9] *Ha $P \hookrightarrow_S Q$ és $R \rhd_S B$, akkor $P \wedge R \hookrightarrow_S (Q \wedge R) \vee B$.*
Biz.: strukturális indukcióval.

[9] Chandy-Misra

6.9. Lemma (Csoda kizárása és $\hookrightarrow_S$). [10] *Ha $P \hookrightarrow_S$ Hamis, akkor $P \equiv$ Hamis.*
Biz.: strukturális indukcióval.

6.10. Lemma. [11] *Ha $\lceil P \rceil \subseteq \lceil Q \rceil$, akkor $inv_S(Q) \subseteq inv_S(P)$. Hasonlóan: $true_S(Q) \subseteq true_S(P)$.*

A feltételek gyengíthetőek. Elegendő, ha $sp(s_0, P) \Rightarrow sp(s_0, Q)$, így bármely I, amelyet a Q kezdeti feltétel kezdetben igazzá tesz, P-ből indulva is teljesül. Ha $I \in inv_S$, akkor $I \rhd_S$ Hamis. A második állítás az első állításból és $INV_S(R) = \bigcap\{I | I \in inv_S(R)\}$-ból következik.

6.11. Lemma. [12] *Tetszőleges S programra, ha $(P \wedge \varphi_S) \hookrightarrow_S Q$, akkor $(P \wedge \varphi_S) \subseteq Q$.*

Az előző lemma és a PSP tétel (6.8. lemma) felhasználásával: $(\varphi_S \wedge \neg Q)$, azaz $(P \wedge \varphi_S \wedge \neg Q) \hookrightarrow_S$ Hamis, így $(P \wedge \varphi_S \wedge \neg Q) =$ Hamis a csoda kizárásának elve (6.9. lemma) miatt.

6.12. Tétel. (Szekvencia viselkedési relációjáról) *Legyen $S = S_1; S_2$. Ekkor*[13]*:*

(1) ha $P \rhd_{S_1} \varphi_{S_1}$, akkor $P' \wedge \neg u \rhd_S \varphi'_{S_1} \wedge \neg u$,

(2) ha $P \mapsto_{S_1} \varphi_{S_1}$, akkor $P' \wedge \neg u \mapsto_S \varphi'_{S_1} \wedge \neg u$,

(3) ha $P \hookrightarrow_{S_1} \varphi_{S_1}$, akkor $P' \wedge \neg u \hookrightarrow_S \varphi'_{S_1} \wedge \neg u$,

(4) ha $P \rhd_{S_2} Q$, akkor $P' \wedge u \rhd_S Q' \wedge u$,

(5) ha $P \mapsto_{S_2} Q$, akkor $P' \wedge u \mapsto_S Q' \wedge u$,

(6) ha $P \hookrightarrow_{S_2} Q$, akkor $P' \wedge u \hookrightarrow_S Q' \wedge u$,

(7) $u \wedge \varphi'_{S_2} = \varphi_S$,

(8) Ha $R \in FP_{S_2}$, akkor $R' \in FP_S$.

(9) Ha $P \in inv_{S_1}(Q)$, akkor és csak akkor $P' \wedge u \in inv_S(Q')$; valamint, ha $P \in inv_{S_2}(Q)$, akkor és csak akkor $P' \wedge \neg u \in inv_S(Q')$.

---

[10] Chandy-Misra
[11] Kozsik T.
[12] Kozsik T.
[13] (9)-(12): Kozsik T.

(10) Ha $P \in inv_{S_1}(Q)$ és $P \in inv_{S_2}(P \wedge \varphi_{S_1})$, akkor és csak akkor $P' \in inv_S(Q')$.

(11) Ha $P \hookrightarrow_{S_1} Q$, akkor $(P' \wedge \neg u) \hookrightarrow_S (Q' \wedge \neg u)$.

(12) Ha $P \hookrightarrow_{S_1} Q$ és $P \hookrightarrow_{S_2} Q$, akkor $P' \hookrightarrow_S Q'$.

Biz.:
(1), (2), (3): A 2.2., 3.19., 3.30., 3.31. def. alapján könnyen belátható, hogy $P \rhd_{S_1} Q \Longrightarrow P' \wedge \neg u \rhd_{S^1} Q' \wedge \neg u$ és $P \mapsto_{S_1} Q \Longrightarrow P' \wedge \neg u \mapsto_{S^1} Q' \wedge \neg u$. Az utóbbiból $\hookrightarrow_{S_1}$ előállítása szerinti strukturális indukcióval: $P \hookrightarrow_{S_1} Q \Longrightarrow P' \wedge \neg u \hookrightarrow_{S^1} Q' \wedge \neg u$. A szekvencia definíciója (6.7. def.) és 3.30. def. alapján ellenőrizhető, hogy: $(u \vee \varphi'_{S_1}) \wedge \neg(\varphi'_{S_1} \wedge \neg u) \rhd_{S^1}$ Hamis, $\neg u \wedge \neg\varphi'_{S_1} \rhd_{S^1} \varphi'_{S_1} \wedge \neg u$ és $p(s_{1,i}$, ha $\neg u)$=$p(s_{1,i}$, ha $\neg u \wedge \neg\varphi_{S_1})$, így alkalmazható a 6.4. tétel a $\pi ::= \neg u \wedge \neg\varphi'_{S_1}$, $Q ::= \varphi'_{S_1} \wedge \neg u$ megfeleltetéssel.
(4),(5),(6): A szekvencia (6.7. def.) és 3.30. def. alapján: $u \rhd_{S^2}$ Hamis. Alkalmazható a 6.3. tétel a $\pi ::= u$ megfeleltetéssel.
(7) Ha $\neg u \wedge \varphi_{S_1}$, akkor s_{k+1} miatt nem lehet fixpont. Ha $\neg u \wedge \neg\varphi_{S_1}$, akkor S^1, ha $u \wedge \neg\varphi_{S_2}$, akkor S^2 változtat állapotot. Ezért $\varphi_S \Rightarrow u \wedge \varphi_{S_2}$. A másik irány a 3.34. definíció következménye.
(8) A (7)-es állítás következménye. □
(9),(10) A szekvencia programjára vonatkozó leggyengébb előfeltételek kiszámításával bizonyítható.
(11)
$P \hookrightarrow_{S_1} Q$ struktúrája szerinti indukcióval. Alapesetben a 6.11. lemmát alkalmazzuk.
(12) Az első feltételből és (11)-ből: $(P' \wedge \neg u) \hookrightarrow_S (Q' \wedge \neg u)$, így $(P' \wedge \neg u) \hookrightarrow_S Q'$. Hasonlóan (6) felhasználásával: $(P' \wedge u) \hookrightarrow_S Q'$, így $P' \hookrightarrow_S Q'$.□

6.13. Tétel. (Szekvencia levezetési szabálya) *Legyen F_1 és F_2 az A állapottér A_1 ill. A_2 altere és a B paramétertér felett értelmezett determinisztikus feladat, $S1;S2$ az A_1 altéren definiált S_1 és az A_2 altéren adott S_2 szekvenciája. Tetszőleges $b \in B$-re jelöljük $F_1(b)$ komponenseit ^{F_1}, $h \in F_2(b)$ komponenseit ^{F_2} indexszel.*
Ha S_1 megfelel a $P \in \mathrm{TERM}_b^{F_1}$ és a $R \in FP_b^{F_1}$ feltételnek a $P \in INIT_b^{F_1}$ kezdeti feltétel mellett, S_2 megfelel $Q \in \mathrm{TERM}_b^{F_2}$ és $Z \in FP_b^{F_2}$ feltételeknek a $Q \in INIT_b^{F_2}$ feltétel mellett, és $R' \Rightarrow Q'$, akkor $S_1;S_2$ megfelel $P' \in \mathrm{TERM}_b$ és $Z' \in FP_b$ feltételeknek a $P' \in INIT_b$ kezdeti feltétel mellett[14].

[14] A tétel állítása Misra programszekvenciára vonatkozó - *nem bizonyított* - állításával rokon. Misra eredeti állítása a következőképpen fogalmazható meg a relációs modellben:

Biz.: A feltételekből a 4.9. tétel szerint $sp(s_{1,0},P)\wedge \mathrm{INV}_{S_1}(P) \hookrightarrow_{S_1} \varphi_{S_1}$. $sp(s_{1,0},P) \Rightarrow \mathrm{INV}_{S_1}(P)$ (3.27. def.). a 6.12. tétel (3)-as pontja és a 3.27. tétel felhasználásával: $sp(s_{1,0},P)' \wedge \neg u \wedge \neg\varphi'_{S_1} \hookrightarrow_S \varphi'_{S_1} \neg u$. $\mathrm{INV}_{S_1}(P)' \in \mathrm{inv}_{S^1}(P')$, így: $sp(s_{1,0},P)' \wedge \neg u \wedge \neg\varphi'_{S_1} \hookrightarrow_S \varphi'_{S_1} \wedge \mathrm{INV}_{S_1}(P)' \wedge \neg u$.

$sp(s_1,P') = sp(s_{1,0},P)' \wedge \neg u$ felhasználásával $sp(s_1,P') \wedge \neg\varphi'_{S_1} \hookrightarrow_S \varphi'_{S_1} \wedge \mathrm{INV}_{S_1}(P)' \wedge \neg u$.

4.11. lemma és 2.2. def. szerint $\varphi'_{S_1} \wedge \mathrm{INV}_{S_1}(P)' \Rightarrow R'$, azaz $sp(s_1,P') \wedge \neg\varphi'_{S_1} \hookrightarrow_S \varphi'_{S_1} \wedge \neg u \wedge R'$ (3.29. lemma).

Mivel $sp(s_1,P') \wedge \varphi'_{S_1} \Rightarrow INV_{S_1}(P)' \neg u \wedge \varphi'_{S_1} \Rightarrow R' \wedge \neg u \wedge \varphi'_{S_1}$, ezért a (3.24. lemma) és a 3.32. def. (diszjunktivitás) alkalmazásával: $sp(s_1,P') \hookrightarrow_S \varphi'_{S_1} \wedge \neg u \wedge R'$.

$\neg u \wedge \varphi'_{S_1} \wedge R' \mapsto_S sp(s_{k+1},R') \wedge u$.

3.32. def. (tranzitivitás) alkalmazásával: $sp(s_1,P') \hookrightarrow_S sp(s_{k+1},R') \wedge u$.

A fentihez hasonló gondolatmenet alapján $sp(s_{2,0},Q) \hookrightarrow_{S_2} \varphi_{S_2} \wedge INV_{S_2}(Q)$, amelyből a 6.12. tétel (6)-os pontja felhasználásával: $sp(s_{2,0},Q)' \wedge u \hookrightarrow_S \varphi'_{S_2} \wedge INV_{S_2}(Q)' \wedge u$. $sp(s_{k+1},R') \Rightarrow sp(s_{2,0},Q)' \wedge u$, így $sp(s_{k+1},R') \wedge u \hookrightarrow_S \varphi'_{S_2} \wedge INV_{S_2}(Q) \wedge u$. A 3.32. def. (tranzitivitás) alkalmazásával: $sp(s_1,P') \hookrightarrow_S \varphi'_{S_2} \wedge u$. 6.12. tétel (7)-es és (8)-as állításaival a tétel állítását kapjuk. □

ha $s_{2,0} = SKIP$, $P \hookrightarrow_{S_1} Q \vee R$, $R \Rightarrow \varphi_{S_1}$ és S'_2 megfelel a $P' \vee R' \hookrightarrow_h Q'$ specifikációs feltételnek a $\varphi'_{S_1} \in \mathrm{INIT}_h$ kezdeti feltétel mellett, akkor $S_1;S_2$ megfelel a $P' \hookrightarrow_h Q'$ specifikációs feltételnek ([UN 88-93]/16-90). Misra a szekvencia fogalmát nem definiálja formálisan, így a tételt sem bizonyítja. Műveleti szemantikai megfontolásokra és a helyettesítési axiómára hivatkozva indokolja az állítás helyességét és példákon keresztül mutatja meg, hogy használata helyes következtetések levonásához vezet.

6.4. Feladatok

A következő feladatok mindegyikében jelölje $S ::= S_1 \cup S_2$-t.

6.1. Feladat. *Igaz-e?*

$$\frac{A \mapsto_{S1} B,\ B \mapsto_{S1} C,\ (A \vee B) \rhd_{S2} C}{(A \vee B) \hookrightarrow_S C}$$

6.2. Feladat. *Igaz-e?*

$$\frac{A \mapsto_{S1} C,\ \ B \rhd_{S1} A,\ \ B \mapsto_{S2} C,\ \ A \rhd_{S2} B}{(A \wedge B) \hookrightarrow_S C}$$

6.3. Feladat. *Igaz-e?*

$$\frac{P \mapsto_{S1} \neg P}{P \mapsto_S \neg P}$$

6.4. Feladat. *Igaz-e?*

$$\frac{P \hookrightarrow_{S1} Q,\ \ P \hookrightarrow_{S2} Q}{P \hookrightarrow_S Q}$$

6.5. Feladat. *Igaz-e?*

$$\frac{P \hookrightarrow_{S1} Q,\ \ P \mapsto_{S2} Q\ ,\ P stabil_{S1}}{P \hookrightarrow_S Q}$$

6.6. Feladat. *Igaz-e?*

$$\frac{P \mapsto_{S1} Q,\ \ Q \Rightarrow R,\ \ P \rhd_{S2} Q}{(P \vee Q) \hookrightarrow_S R}$$

6.7. Feladat. *Igaz-e?*

$$\frac{P \hookrightarrow_{S1} Q,\ \ Q \mapsto_{S1} R,\ \ R \in inv_{S2}}{P \hookrightarrow_S R}$$

6.8. Feladat. *Igaz-e?*

$$C ::= (\bigvee_{i \in \mathcal{N}} A_i)$$

$$\frac{C stabil_{S1}\ ,\ \forall i \in \mathcal{N} : (A_i \mapsto_{S2} B}{C \hookrightarrow_S B}$$

6.9. Feladat. *Igaz-e?*

$$C::=\Big(\bigvee_{i\in\mathcal{N}} A_i\Big)$$

$$\frac{C \mapsto_{S1} B\ ,\ \forall i \in \mathcal{N} : (A_i \mapsto_{S2} B}{C \hookrightarrow_S B}$$

6.10. Feladat. *Igaz-e?*

$$\frac{(P\wedge\neg B)\mapsto_{S1} Q,\ \ (P\wedge\neg B)\mapsto_{S2} R,\quad (P\wedge B)\Rightarrow Q,\ \ Q stabil_{S1},\ \ Q\mapsto_{S2} R}{P\hookrightarrow_S R}$$

6.11. Feladat. *Igaz-e?*

$$\frac{P\mapsto_{S1} Q\vee R,\ \ P stabil_{S2},\ \ Q\hookrightarrow_S R}{(P\vee Q)\hookrightarrow_S R}$$

6.12. Feladat. *Igaz-e?*

$$\frac{P\hookrightarrow_{S1} Q,\ \ P\hookrightarrow_{S2} R,\ \ P stabil_S,\ \ Q\rhd_{S1} R}{P\hookrightarrow_S R}$$

6.13. Feladat. *Igaz-e?*

$$\frac{(P\wedge\neg B)\mapsto_{S1} Q,\ \ (P\wedge\neg B)\mapsto_{S2} R,\ \ Q\mapsto_{S1} R,\quad Q\hookrightarrow_{S2} R,\ \ (P\wedge B)\hookrightarrow_S (P\wedge\neg B)}{P\hookrightarrow_S R}$$

6.14. Feladat. *Igaz-e?*

$$\frac{(P\wedge R)\rhd_{S2} B,\ \ P\rhd_{S1} Q}{(P\wedge R)\rhd_S (Q\vee\neg R\vee B)}$$

6.15. Feladat. *Igaz-e?*

$$\frac{P\hookrightarrow_{S1} Q,\ \ (P\wedge Q\wedge R)\Rightarrow fixpont_{S2}}{P\hookrightarrow_S (Q\vee\neg R)}$$

6.16. Feladat. *Igaz-e?*

$$\frac{P\rhd_{S1}(Q\wedge R),\ \ (P\wedge\neg R)\mapsto_{S2} Q,\ \ R\Rightarrow\neg P}{P\hookrightarrow_S Q}$$

6.17. Feladat. *Igaz-e?*

$$C := \bigvee_{n\in\mathcal{N}} A_n$$

$$\frac{C\ stabil_{S2},\ \ \forall n\in\mathcal{N} : (A_n \mapsto_{S1} B)}{C \hookrightarrow_S B}$$

6.18. Feladat. *Igaz-e?*

$$P := \bigvee_{n\in\mathcal{N}} Q_n$$

$$\frac{\forall n\in\mathcal{N} : (Q_n \mapsto_{S2} R),\ \ P \rhd_{S1} R}{P \hookrightarrow_S R}$$

6.19. Feladat. *Igaz-e?*

$$\frac{P\ stabil_{S},\ \ P \rhd_{S2} Q}{P \rhd_{S1} (P \wedge Q)}$$

7. fejezet

A modell tulajdonságai

7.1. Szemantika

A 3.15. def. következményeként az absztrakt program műveleti jellegű szemantikája *elágazó idejű*, *összefésüléses* és *statikus*. Az absztrakt program viselkedési relációval megfogalmazott leíró jellegű szemantikája absztraktabb [Hen 88] a műveletinél.

Valós párhuzamosság esetén a komponensekre érvényes biztonságossági tulajdonságok sérülnek a programkompozíció (6. fejezet) során. Ennek az az oka, hogy az állapottér felett az összetett program olyan új irányokban is elmozdulhat, amelyek a komponensek egyidejű mozgásainak eredői [Cha 90].

7.1. Példa (Valós párhuzamosság és unió). S_1::=$(SKIP, \{x := x+1\})$. S_2::=$(SKIP, \{y := y+1\})$.
P::=$(0 < x < 5 \wedge 0 < y < 5)$. Q::=$((x \geq 5 \wedge y < 5) \vee (y \geq 5 \wedge x < 5))$.
$P \rhd_{S_1} Q$ és $P \rhd_{S_2} Q$, így a 6.1. tétel szerint $P \rhd_{S_1 \cup S_2} Q$.
$x = 4 \wedge y = 4$-ből ($P \wedge \neg Q$), valós párhuzamosság esetén az $S_1 \cup S_2$ programmal közvetlenül el lehet jutni az $(x = 5 \wedge y = 5) \in (\lceil \neg P \wedge \neg Q \rceil)$ állapotba. $\square$

7.2. Kifejezőerő

7.2. Példa.
$A = \mathbb{Z}\, x :: \mathbb{Z}\, B = \{b\}$. $F(b) = \{h_1, h_2\}$.
$P \equiv (x > 5), Q \equiv (x < 5), R \equiv (x = 5)$.

$R \in INIT_{h_1}$, $R \in INIT_{h_2}$, $R \hookrightarrow_{h_1} (x \neq 5)$, $R \hookrightarrow_{h_2} (x \neq 5)$, $\neg Q \hookrightarrow_{h_1} Q$, $P \in inv_{h_2}$.
$\square$

A 7.2. példában megadott feladat specifikációs feltételének *elágazó idejű* temporális logikai megfelelője: $A_\phi GP \vee A_\phi FQ$, ami *nem azonos a lineáris logiká*ban is megfogalmazható $A_\phi(GP \vee FQ)$-val (11. fejezet). A specifikációs eszközök kifejezőereje tehát meghaladja a UNITY kifejezőerejét.

7.2.1. Programhelyesség

A szekvenciális programoktól eltérően a most definiált absztrakt program helyességének igazolásához megfogalmazott programtulajdonságok nem külön-külön az egyes utasításokra, hanem a teljes utasításhalmazra vonatkoznak. Ez úgy is megfogalmazható, hogy a bizonyítás és a programszöveg szétválik. Azt is mondhatjuk, hogy a módszer a globális invariánsok módszerének általánosítása [And 91]. (Megj.: A bizonyítás és a programszöveg szétválasztása lehetséges szekvenciális programok esetén is, lásd: [Lam 90]).

8. fejezet

Programozási tételek

Ebben a fejezetben általánosan megfogalmazott programozási feladatokat oldunk meg. A kapott megoldásokat programozási tételeknek nevezzük, mert széles körben alkalmazhatóak konkrét feladatok megoldása során. Ilyen alapfeladat például:

- asszociatív művelet eredményének párhuzamos kiszámítása (8.1 pont),
- elemenként feldolgozható (8.6 pont), ill.
- sorozatokon többszörös függvénykompozícióval definiált függvény értékének kiszámítása (8.4 pont).

Példát mutatunk csatornaváltozók használatára és adatcsatornás megoldási módszerekre is. Megvizsgáljuk, hogy a kapott megoldások milyen architektúrákon implementálhatók hatékonyan. Olyan megoldásokat dolgozunk ki, amelyek osztott és aszinkron osztott memóriás rendszerekre is könnyen leképezhetőek.

8.1. Asszociatív művelet eredményének kiszámítása

Legyen H tetszőleges halmaz. $\circ : H \times H \longmapsto H$ tetszőleges kétoperandusú asszociatív alapművelet H-n.

$f : H^* \longmapsto H$ függvény. f a $\circ$ művelet egyszeri vagy ismételt alkalmazásának felel meg. $\circ$ asszociativitása miatt tetszőleges legalább 3 hosszú $x \in H^*$ sorozatra: $f(\ll x_1,\ldots,x_{|x|} \gg) = f(\ll f(\ll x_1,\ldots,x_{|x|-1} \gg), x_{|x|} \gg) = f(\ll x_1, f(\ll x_2,\ldots,x_{|x|} \gg) \gg)$. A továbbiakban a $(h_1 \circ h_2)$ helyett mindig $f(\ll h_1, h_2 \gg)-t$ írunk. f-et kiterjeszthetjük az egyetlen elemből álló sorozatokra is, legyen $f(\ll h \gg) = h$.

Adott $a \in H^*$, H-beli elemek véges, nem üres sorozata. Tegyük fel, hogy a sorozat egyes elemei közvetlenül elérhetőek: $a = \ll a_1, ..., a_n \gg, (n \geq 1)$. Számítsuk ki a $\mathcal{G}_a : [1..n] \longmapsto H$ függvény értékét minden $i \in [1..n]$-re, ahol $n \geq 1$ és $\mathcal{G}_a(i) = f(\ll a_1, ..., a_i \gg)$.

8.1.1. A feladat specifikációja

Reprezentáljuk az a sorozatot és a $\mathcal{G}_a$ függvényt egy-egy vektorral, amelyeket a-val, illetve g-vel jelölünk. A vektorok elemei H-beli értékek. Kikötjük, hogy fixpontban a g vektor i. eleme éppen $\mathcal{G}_a(i)$ legyen (8.3.), illetve a program biztosan elérje egy fixpontját (8.2.).

$$\begin{array}{llll} A = & G\times & G, \text{ ahol} & G = vektor([1..n], H),\ n \geq 1 \\ & g & a & \\ B = & G & & \\ & a' & & \end{array}$$

$$(a = a') \in INIT_{a'} \tag{8.1}$$

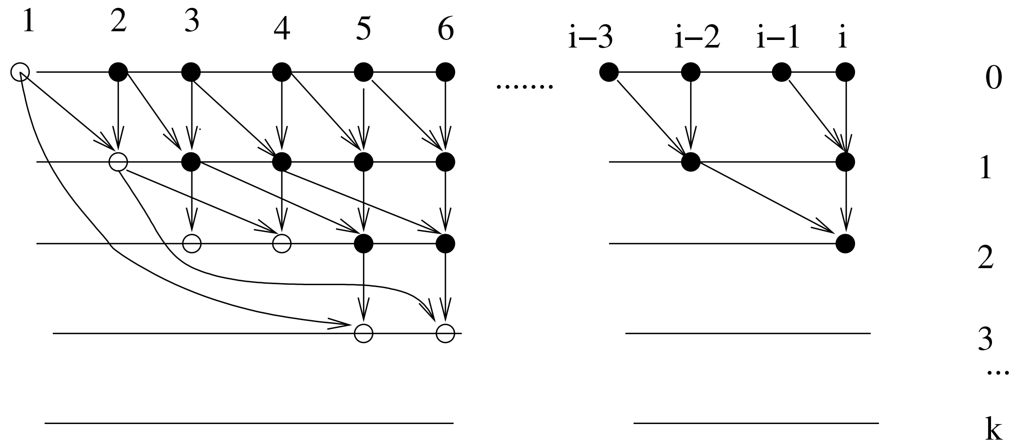
$$\text{Igaz} \hookrightarrow \text{FP}_{a'} \tag{8.2}$$

$$\text{FP}_{a'} \Rightarrow (a = a' \wedge \forall i \in [1..n] : g(i) = f(\ll a_1, ..., a_i \gg)) \tag{8.3}$$

Megállapíthatjuk, hogy a $\mathcal{G}_a$ függvény i helyen felvett értékének meghatározását megkönnyíti, ha ismerjük f értékét bármely $[u..v] \subseteq [1..i]$ intervallum elemeivel indexelt $f(\ll a_u, ..., a_v \gg)$ részsorozatra.[1] Megfigyelhetjük azt is, hogy egy részsorozatra kapott eredményt bármely, a részsorozatot tartalmazó részsorozatra vonatkozó eredmény meghatározásánál hasznosíthatjuk.

Ezen gondolatmenet alapján bővítsük a feladat állapotterét és finomítsuk a specifikációt. Vezessük be a h függvényt oly módon, hogy $h(a, i, k)$ jelentse f értékét a azon részsorozatára, amelynek utolsó eleme a_i és hossza vagy éppen 2^k vagy a_1 az első eleme, ha $i < 2^k$. A gs kétdimenziós vektort azért vezetjük be, hogy segítségével a h függvényt változóval helyettesítsük [Fót 83]. A gs, k, t változók és h kapcsolatát invariáns állítással írjuk le (8.6.)-(8.8.). Ugyanezt jelenítjük meg szemléletes alakban a 8.1. ábrán. Az ábrán szereplő vonalak a gs mátrix elemei között fennálló kapcsolatot írják le a 8.2. lemma alapján, azaz $gs(i, k) = h(a, i, k)$, ha $k \leq k(i)$. egy $gs(i, k)$ f értéke legfeljebb 2^k hosszú kezdősorozatra.

[1] A lépésenkénti finomítás során alkalmazzuk a [Qui 87] 3-2. pontjában és [Cha Mis 89] 5. fejezetében ill. 6.9. pontjában bemutatott megoldási módszerek egyes elemeit.



8.1. ábra. $\circ : k = \lceil log(i) \rceil$, asszociatív művelet kiszámításának részeredményei, a gs mátrix elemei között fennálló kapcsolatok

$$\begin{array}{llllll} A' = & G\times & G\times & GS\times & K\times & T \\ & g & a & gs & k & t \end{array}$$

$$\begin{array}{ll} G & = vektor([1..n], H), \\ GS & = vektor([1..n, 0..(\lceil \log(n) \rceil)], H) \\ K & = vektor([1..n], \mathcal{N}_0), \\ T & = vektor([1..n], \mathcal{N}_0),\ n \geq 1 \end{array}$$

Megadjuk a $h : G \times [1..n] \times \mathcal{N}_0 \longrightarrow H$ parciális függvény pontos definícióját:

$$h(a,i,k) = \begin{cases} f(\ll a_1, \ldots, a_i \gg), & \text{ha } i - 2^k + 1 \leq 1 \\ f(\ll a_{(i-2^k+1), \ldots, a_i} \gg), & \text{ha } i - 2^k + 1 \geq 1. \end{cases}$$

Válasszuk a $v : A \longmapsto \mathcal{N}_0$ variánsfüggvényt a következőképpen:

$$v = 4 * n * n - \sum_{i=1}^{n} (k(i) + \chi(k(i) = \lceil \log(i) \rceil \wedge g(i) = gs(i, k(i)))),$$

ahol $\chi : \mathcal{L} \longmapsto \{0, 1\}$. $\chi(igaz) = 1$, $\chi(hamis) = 0$. A variánsfüggvény lényegében azt adja meg, hogy a gs mátrix egyes oszlopaiban összesen hány olyan elem van, amely nem azonos a h függvény megfelelő helyen felvett értékével, illetve a g vektor értéke hány helyen különbözik a $\mathcal{G}_a$ függvény értékétől.

8.1. Lemma. (Asszociatív művelet - a feladat finomítása) *Az alábbi specifikáció finomítása az eredetinek:*

$$(a = a') \in INIT_{a'} \tag{8.4}$$

$$\text{Igaz} \hookrightarrow \text{FP}_{a'} \tag{8.5}$$

$$\text{FP}_{a'} \Rightarrow \forall i \in [1..n] : (k(i) = \lceil \log(i) \rceil) \wedge (g(i) = gs(i, \lceil \log(i) \rceil)) \tag{8.6}$$

$$\begin{aligned} &\text{inv}_{a'}(\forall i \in [1..n] : k(i) \leq \lceil \log(i) \rceil \wedge \\ &\quad \forall k : k \leq k(i) : gs(i,k) = h(a,i,k)) \end{aligned} \tag{8.7}$$

$$\text{inv}_{a'}(\forall i \in [1..n] : t(i) = 2^{k(i)}) \tag{8.8}$$

$$\text{inv}_{a'}(a = a') \tag{8.9}$$

Bizonyítás:
Fixpontban (8.6.) szerint $k(i) = \lceil \log(i) \rceil$ és $g(i) = gs(i, \lceil \log(i) \rceil)$, tehát a 5.8. lemma és (8.7.) alapján $g(i) = gs(i, \lceil \log(i) \rceil) = h(a,i, \lceil \log(i) \rceil)$. $2^{\lceil \log(i) \rceil} \geq i$, így h definícióját felhasználva $h(a,i, \lceil \log(i) \rceil) = f(\ll a_1, ..., a_i \gg)$, így a (8.9.) invariáns tulajdonság és 5.8. lemma alkalmazásával igazoltuk, hogy a (8.6.),(8.7.),(8.9.) feltételek együttesen finomítják a (8.3.) feltételt. □

8.1. Megjegyzés. *A variáns függvényre vonatkozó 5.7. tétel segítségével bizonyíthatjuk majd azt, hogy a program megfelel a (8.2.)=(8.5.) feltételnek. Ebben az értelemben a variáns függvény megválasztása is egy finomítási lépésként fogható fel.*

8.2. Megjegyzés. *A (8.8.) feltétel csak az új állapottérkomponensekre tesz kikötést, így ezt a feltételt nem kellett felhasználnunk a bizonyítás során.*

8.2. Lemma.
Ha $(i - 2^k \geq 1)$*, akkor* $f(\ll h(a, i-2^k, k), h(a,i,k) \gg) = h(a,i,k+1)$.

Bizonyítás:
Tudjuk, hogy $i - 2^k \geq 1$, tehát $h(a,i,k) = f(\ll a_{(i-2^k+1)}, \ldots, a_i \gg)$. Ha $(i-2^k) - 2^k + 1 \geq$, akkor $h(a, i-2^k, k) = f(\ll a_{(i-2^k-2^k+1)}, \ldots, a_{(i-2^k)} \gg)$. Ekkor f asszociativitása miatt $f(\ll h(a, i-2^k, k), h(a,i,k) \gg) =$
$f(\ll a_{(i-2^k-2^k+1)}, \ldots, a_{(i-2^k)}, a_{(i-2^k+1)}, \ldots, a_i \gg) = h(a,i,k+1)$. Ha $(i-2^k) - 2^k + 1 < 1$, akkor $h(a, i-2^k, k) = f(\ll a_1, \ldots, a_{(i-2^k)} \gg)$. f asszociativitása miatt $f(\ll h(a, i-2^k, k), h(a,i,k) \gg) = f(\ll a_1, ...a_{(i-2^k)}, a_{(i-2^k+1)}, , ..., a_i \gg) = h(a,i,k+1)$. □

8.1.2. A megoldás

8.3. Tétel. (Asszociatív művelet kiszámításának tétele I.) *A 8.1. program megfelel a (8.4.)-(8.9.) specifikációnak, azaz megoldja az asszociatív művelet eredménye kiszámításának feladatát.*

$$
\begin{array}{ll}
s_0: & \underset{i=[1..n]}{\Box} gs(i,0), t(i), k(i) := f(\ll a_i \gg), 1, 0 \\[2ex]
S: \{ & \underset{i=[1..n]}{\Box} gs(i,k(i)+1), t(i), k(i) := \\
& \left\{ \begin{array}{ll}
f(\ll gs(i,k(i)), & gs(i-t(i),k(i)) \gg), 2*t(i), k(i)+1, \\
 & \text{ha } (i-2*t(i)+1 \geq 1) \wedge \\
 & \wedge (k(i-t(i)) \geq k(i)) \\
f(\ll gs(i,k(i)), & gs(i-t(i),k(i-t(i))) \gg), \\
 & 2*t(i), k(i)+1, \\
 & \text{ha } (i-t(i) \geq 1) \wedge (i-2*t(i)+1 < 1) \\
 & \wedge k(i-t(i)) = \lceil \log(i-t(i)) \rceil
\end{array} \right. \\
& \underset{i=[1..n]}{\Box} g(i) := gs(i,k(i)), \text{ha}(k(i) = \lceil \log(i) \rceil) \\
\} &
\end{array}
$$

8.1. absztrakt program. Asszociatív művelet I. változat

8.3. Megjegyzés. $\underset{i=[1..n]}{\Box}$ *n utasítás rövidítése. Az egyes értékadásokat példányosítással kapjuk oly módon, hogy az általános alakban az i változót konkrét értékkel helyettesítjük.*

Bizonyítás:

(8.9.): A programban a elemeire vonatkozó értékadás nincs. Így a (8.4.) kezdeti feltétel egyben invariáns tulajdonság is.

(8.8.): $sp(s_0, \text{Igaz})$-ben: $t(i) = 1$ és $k(i) = 0$, tehát a feltétel kezdetben teljesül. Az értékadások mindegyike együtt változtatja $k(i)$ és $t(i)$ értékét, ezért a (8.8.) feltétel invariáns tulajdonság.

(8.7.): $sp(s_0, \text{Igaz}) \Rightarrow gs(i,k(i)) = h(a,i,k(i))$, mert $h(a,i,0) = f(\ll a(i) \gg)$. $sp(s_0, \text{Igaz}) \Rightarrow (k(i) \leq \lceil \log(i) \rceil)$, mert $k(i)$ kezdetben 0.

Értékadás leggyengébb előfeltételének meghatározása után elég azt megmutatni, hogy

- $(i-2*t(i)+1 \geq 1) \wedge (k(i-t(i)) \geq k(i))$-ből és $\forall k : k \leq k(i) : gs(i,k) = h(a,i,k)$-ból következik az egyenlőség $k(i)+1$-re is, azaz: $f(\ll gs(i,k(i)), gs(i-t(i),k(i)) \gg) = h(a,i,k(i)+1)$ és $k(i)+1 \leq \lceil\log(i)\rceil$
- $(i-t(i) \geq 1) \wedge (i-2*t(i)+1 < 1) \wedge (k(i-t(i)) = \lceil\log(i)\rceil)$-ből és $\forall k : k \leq k(i) : gs(i,k) = h(a,i,k)$-ból következik az egyenlőség $k(i)+1$-re is, azaz: $f(\ll gs(i,k(i)), gs(i-t(i),(\lceil\log(i-t(i))\rceil)) \gg) = h(a,i,k(i)+1)$ és $k(i)+1 \leq \lceil\log(i)\rceil$.

$(i-2*t(i)+1 \geq 1) \wedge (t(i) \geq 1) \Rightarrow (i-t(i) \geq 1) \Rightarrow k(i) \leq \log(i-1) < \log(i) \leq \lceil\log(i)\rceil$.
Az első esetben: $k(i) \leq k(i)$ miatt $gs(i,k(i)) = h(a,i,k(i))$, $(k(i-t(i)) \geq k(i))$ miatt $gs(i-t(i),k(i)) = h(a,i-t(i),k(i))$. A második esetben: $k(i) \leq k(i)$ miatt $gs(i,k(i)) = h(a,i,k(i))$, $k(i-t(i)) = \lceil\log(i-t(i))\rceil$ miatt $gs(i-t(i),(\lceil\log(i)\rceil)) = h(a,i-t(i),(\lceil\log(i)\rceil))$. Mindkét esetben a 8.2. lemma alkalmazásával kapjuk a bizonyítandó állítást.

A 3.10. megj. szerint (8.7.),(8.8.) és (8.9.) feltételeinek konjunkciója invariáns tulajdonság.

(8.5.): (8.7.),(8.8.),(8.9.) feltételek konjunkciójából következik, hogy $\forall i \in [1..n] : k(i) \leq n$, így: $v > 0$. A 5.7. tétel szerint elegendő belátni, hogy a program minden utasítására igaz, hogy vagy pontosan 1-gyel csökkenti a variáns függvényt, vagy nem okoz állapotváltozást. Ha a program nincs fixpontban, akkor van olyan $i \in [1..n]$ és megfelelő értékadás, amely $k(i)$ értékét növeli, vagy van olyan i, hogy $k(i) = \lceil\log(i)\rceil$ és $g(i)$ még nem vette fel a $gs(i,(\lceil\log(i)\rceil))$ értéket.

(8.6.): a fixpont definíciója (3.34.,4.12. def.) alapján

$$\forall i \in [1..n] :$$

$$(k(i) = \lceil\log(i)\rceil) \rightarrow g(i) = gs(i,k(i)) \wedge \tag{8.10}$$

$$((i-2*t(i)+1 < 1) \vee (k(i-t(i)) < k(i)) \wedge \tag{8.11}$$

$$(i-t(i) < 1) \vee (i-2*t(i)+1 \geq 1) \vee$$

$$\vee(k(i-t(i)) \neq \lceil\log(i-t(i))\rceil)) \tag{8.12}$$

Ebből i szerinti indukcióval $\forall i \in [1..n] : (k(i) = \lceil\log(i)\rceil)$. $i = 1$-re: (8.7.)-ből következik $(k(1) = \lceil\log(1)\rceil)$. Tegyük fel, hogy $\forall j < i : (k(j) = \lceil\log(j)\rceil)$. $t(i) \geq 1$ ezért $(k(i-t(i)) \neq \lceil\log(i-t(i))\rceil)$ ellentmond az indukciós feltételnek. Így (8.12.) az $(i-t(i) < 1) \vee (i-2*t(i)+1 \geq 1)$ feltétellel helyettesíthető.
Ha $(i-2*t(i)+1 \geq 1)$, akkor $k(i-t(i)) < k(i)$, különben (8.11.) nem teljesül. Az indukciós feltétel szerint $t(i) \geq 1$ miatt $k(i-t(i)) = \lceil\log(i-t(i))\rceil$, tehát: $\lceil\log(i-$

$t(i))\rceil < k(i)$. Ez azonban ellentmond a $(i-2*t(i)+1 \geq 1) \Rightarrow (i-t(i)-t(i)+1 \geq 1) \Rightarrow \lceil \log(i-t(i))\rceil \geq k(i)$ kezdeti feltételnek. Tehát: $(i-2*t(i)+1 < 1)$. $(i-2*t(i)+1 < 1) \Rightarrow (i-t(i) < 1)$, különben (8.12.) nem teljesül. $(i-t(i) < 1) \Rightarrow k(i) \geq \lceil \log(i)\rceil$. A (8.7.) feltétel (invariánstulajdonság része) miatt $k(i) = \lceil \log(i)\rceil$. Ekkor (8.10.) alapján: $g(i) = gs(i,k(i)) = gs(i,\lceil \log(i)\rceil)$ is. □

8.1.3. Programtranszformáció

Tegyük fel, hogy $\Theta(n)$ processzor párhuzamosan hajtja végre a kapott programot. A vektorok *i.* komponenseire vonatkozó értékadásokat az *i.* logikai processzorra képezhetjük le. A variáns függvény definíciójából és a fenti bizonyításból közvetlenül adódik, hogy a program legkésőbb $O\lceil \log(n)\rceil$ állapotváltozás után fixpontba jut. Az egyes logikai processzorok egymáshoz képest aszinkron és szinkron módon is működhetnek.

A program jelenlegi alakjában azonban még nem felel meg sem a finom atomicitás szabályának [And 91](2.4), sem az osztott változós sémának [Cha Mis 89], ezért további komponensekkel bővítjük az állapotteret. Célszerű a logaritmus függvényt is kitranszformálni.

Jelöljük gst(i)-vel $gs(i-t(i),k(i))$, $kt(i)$-vel $k(i-t(i))$, $gstk(i)$-vel $gs(i-t(i),kt(i))$ értékét, ha az szükséges és ismert az *i.* processzor számára és $kt(i)$ értéke elegendően nagy ahhoz, hogy a gs mátrix *i.* oszlopának következő ($k(i)+1$.) elemét meghatározhassuk (8.13.). Vezessük be a $ktf(i), gstf(i), gstkf(i)$ logikai változókat a segédvektorok kezelésének megkönnyítésére. A segédvektorok *i.* komponense lokális az *i.* processzorra nézve. A transzformált program esetén teljesül majd, hogy minden egyes értékadásban pontosan legfeljebb egy olyan változóra (vektorkomponensre) hivatkozunk, amely nem lokális az *i.* processzorra nézve.

8.1.4. A specifikáció finomítása

A (8.4.)-(8.9.) specifikációt bővítjük az alábbi invariánsokkal:

$$\text{inv}_{a'} \quad \forall i \in [1..n]: \ (kt(i) \leq k(i-t(i)) \land ktf(i) \to (kt(i) \geq k(i) \lor kt(i) = l(i-t(i)))) \tag{8.13}$$

$$\text{inv}_{a'} \quad \forall i \in [1..n]: (gstf(i) \to ktf(i) \land (i-2*t(i)+1 \geq 1) \land gst(i) = gs(i-t(i),k(i))) \tag{8.14}$$

$$\text{inv}_{a'} \quad \forall i \in [1..n] : (gstkf(i) \rightarrow ktf(i) \wedge (i - t(i) \geq 1) \wedge (i - 2 * t(i) + 1 < 1)$$
$$\wedge gstk(i) = gs(i - t(i), kt(i)) = gs(i - t(i), k(i - t(i)))) \tag{8.15}$$
$$\text{inv}_{a'} \quad \forall i \in [1..n] : \lceil \log(i) \rceil = l(i) \tag{8.16}$$

8.1.5. A transzformált program

8.4. Tétel. (Asszociatív művelet kiszámításának tétele II.) *A 8.2. program megfelel a (8.4.)-(8.9.),(8.13.)-(8.16.) specifikációnak.*

Bizonyítás: az (8.13.)-(8.16.) invariánsok teljesülését a leggyengébb előfeltételek és $sp(s_0, \text{Igaz})$ kiszámolásával könnyen igazolhatjuk. A (8.5.),(8.7.)-(8.8.) kikötések a transzformált programra is teljesülnek, mert a kikötésekben szereplő változókra vonatkozó értékadások a (8.13.)-(8.16.) invariáns állítások miatt ekvivalensek az eredetiekkel. A (8.6.) fixpontfeltétel teljesüléséhez azt kell megmutatni, hogy ha a transzformált program fixpontba jut, akkor az eredeti is fixpontban van és a (8.10.)-(8.12.) feltételek teljesülnek. □

8.1.6. Hatékonyság és általánosság

A fenti megoldás egyszerűen implementálható szinkron, aszinkron architektúrán is, és osztott rendszerben is [Cha Mis 89]. Szinkron architektúra esetén egyszerűsíthető a megoldás, kevesebb új változó bevezetésével is megoldható a feladat. Osztott rendszer esetén csak akkor hatékony ez a megoldás, ha elegendően sok, legalább $\Omega(\lceil \log(n) \rceil)$ csatorna áll rendelkezésre processzoronként és a kommunikációs költség alacsony. Ilyen architektúra pl. a *hiperkocka* [Qui 87]. Adatcsatornás megoldásra mutat hatékony megoldást [Loy Vor 90].

A tétel segítségével nagyon sok klasszikusnak számító feladatot oldhatunk meg egyszerű *visszavezetés*sel[2] [Fót 86]. Pl.: párhuzamos összeadás, emelkedő számsorozatok összehasonlítása [Cha Mis 89], stb.

[2]Az asszociatív függvény kiszámításának tételét eddig már kb. 200 egyetemi hallgató alkalmazta a gyakorlatban is sikeresen konkrét feladatok megoldása során. PowerXplorer típusú, 16 processzoros, párhuzamos számítógépen, PVM-ben implementált aszinkron, párhuzamos, konkrét program futási ideje a felhasznált processzorok számának emelése mellett egyre rövidebb, ha a $\circ$ művelet elvégzéséhez szükséges processzoridő elegendően nagy a kommunikációs költségekhez képest.

$s_0: \quad \underset{i=[1..n]}{\square} gs(i,0), t(i), k(i), l(i), ktf(i), gstkf(i), gstf(i), kt(i) :=$
$\quad f(\ll a_i \gg), 1, 0, \lceil \log(i) \rceil, hamis, hamis, hamis, 0$

$S: \{ \quad \underset{i=[1..n]}{\square} kt(i) := k(i - t(i)), \text{ha } \neg ktf(i) \wedge (i - t(i)) \geq 1$

$\underset{i=[1..n]}{\square} ktf(i) := igaz, \text{ha } \neg ktf(i) \wedge (i - t(i)) \geq 1 \wedge$
$\wedge (kt(i) \geq k(i) \vee kt(i) = l(i - t(i)))$

$\underset{i=[1..n]}{\square} gst(i), gstf(i) := gs(i - t(i), k(i)), igaz,$
$\text{ha } ktf(i) \wedge (i - 2 * t(i) + 1 \geq 1) \wedge (kt(i) \geq k(i)) \wedge \neg gstf(i)$

$\underset{i=[1..n]}{\square} gstk(i), gstkf(i) := gs(i - t(i), kt(i)), igaz,$
$\text{ha } ktf(i) \wedge (i - t(i) \geq 1) \wedge (i - 2 * t(i) + 1 < 1)$
$\wedge (kt(i) = l(i - t(i))) \wedge \neg gstkf(i)$

$\underset{i=[1..n]}{\square} gs(i, k(i) + 1), t(i), k(i), ktf(i), gstf(i), gstkf(i), kt(i) :=$

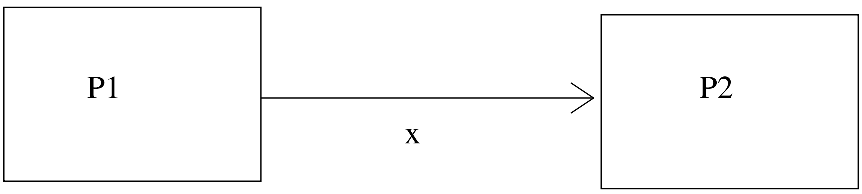
$$\begin{cases} f(\ll gs(i,k(i)), & gst(i) \gg), 2 * t(i), k(i) + 1, hamis, hamis, hamis, 0 \\ & \text{ha } gstf(i) \\ f(\ll gs(i,k(i)), & gstk(i) \gg), 2 * t(i), k(i) + 1, hamis, hamis, hamis, 0 \\ & \text{ha } gstkf(i) \end{cases}$$

$\underset{i=[1..n]}{\square} g(i) := gs(i, k(i)), \text{ ha } (k(i) = l(i))$
}

8.2. absztrakt program. Asszociatív művelet II. változat

8.2. Csatornaváltozók használata

Párhuzamos és elosztott rendszereket gyakran írunk le folyamathálózatok formájában [Hoa 78, Hoa 85]. A folyamatokat dobozokkal jelöljük, a folyamatok közötti kommunikáció formája üzenetküldés. Az üzeneteket egyirányú kommunikációra alkalmas csatornákon keresztül juttatja el a feladó a címzettnek, címzetteknek. Feltételezzük, hogy az üzenettovábbítás megbízható, üzenetek nem vesznek el, nem sérülnek meg, csak a valóban elküldött üzenetek érkeznek meg. Az üzenetküldés aszinkron, a feladó általában rögtön folytatja tevékenységét miután a csatornára elhelyezte az üzenetet, nem kell megvárnia az üzenet átvételét. A csatornák sor típusú változóként viselkednek, átmenetileg képesek tárolni a már elküldött, de még nem fogadott üzeneteket. A csatorna kapacitása határozza meg a tárolható üzenetek számát. Ha a csatorna kapacitása korlátos, akkor előfordulhat, hogy a küldő fél nem tudja rögtön elhelyezni üzenetét és várakozni kell, amíg a csatorna képes nem lesz újabb üzenet fogadására. Minden csatornához két sor típusú változó tartozik, az egyik a csatornán várakozó üzeneteket tartalmazza (a csatorna aktuális állapota), a másik a csatorna története. A csatorna története minden olyan üzenetet tartalmaz helyes sorrendben, amelyik valaha rákerült a csatornára, a történetváltozóról a fogadó fél nem távolítja el az üzeneteket. A történetváltozót egy felülvonással jelöljük. A történetváltozó tárolása a valóságban nehezen vagy egyáltalán nem megoldható, mert egy hosszú ideig futó programban az üzenetek száma idővel minden korlátot meghaladhat. Történetváltozókat ezért csak úgy használunk értékadásokban, hogy értéküket csak saját új érték meghatározásához használjuk fel. Ezek az egyszerű értékadások elhagyhatóak anélkül, hogy a program többi részének működése megváltozna.



A 8.2 ábrán két folyamatot és az őket összekötő $x : Ch(Int)$ csatornát láthatjuk. A csatornára a $P1$ folyamat helyezhet el üzenetet, egész számok formájában. A $P2$ folyamat olvas a csatornáról. Az alábbi műveletek tartoznak a csatorna típusú változókhoz:

- üzenetküldés (P1): $x := hiext(x, e)$, vagy röviden: $x := x; e$,

- üzenet eltávolítás (P2) $\quad x := lorem(x), \text{ha}\, x \neq <>$,
- csatorna inicializálása: $\quad x := <>$,
- üzenet olvasása (P2) $\quad x.lov, \text{ha}\, x \neq <>$,
- lekérdezés, hogy a csatorna üres-e:: $x = <>$, illetve hány üzenet várakozik a csatornán: $length(x)$ or $|x|$.

Az alábbiakban megadjuk az egyes műveletek pontos jelentését is. Elemi műveletek jelentését megadhatjuk hatásrelációjukkal, vagy azzal ekvivalens módon a leggyengébb előfeltételük kiszámításának meghatározásával is.

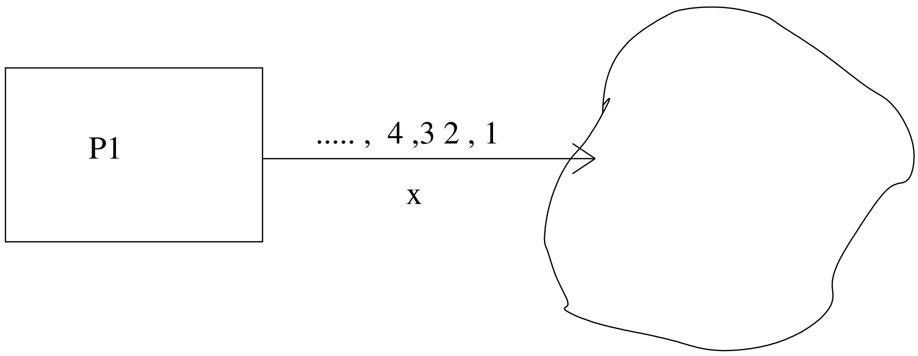
- üzenetküldés: $lf(x := x; e,\ R) = R^{x \leftarrow x;e,\ \overline{x} \leftarrow \overline{x};e}$
- üzenet eltávolítása: $lf(x := lorem(x),\ \text{ha}\, x \neq <>,\ R) =$
  $$(x \neq <> \rightarrow R^{x \leftarrow lorem(x)}) \wedge (x = <> \rightarrow R),$$
- csatorna inicializálása: $lf(x := <>, R) = R^{x \leftarrow <> \ \wedge\ \overline{x} \leftarrow <>}$.

Figyeljük meg, hogy üzenetküldés leggyengébb előfeltételének kiszámításakor a csatorna történetét is helyettesíteni kell az utófeltételben, míg az üzenet eltávolítása nem érinti a történetváltozót.

Ha a folyamatok közötti kommunikációra nem használunk osztott változókat, hanem kizárólag csatornaváltozók segítségével oldjuk meg az információ cseréjét, akkor az egyes folyamatok közötti kapcsolat jól ellenőrizhetővé válik. A lokalitás tétel állítását fogalmazhatjuk újra speciális formában:

8.5. Lemma (Lokalitás folyamathálózatokban).

- *Ha egy P állítás változói között csak $P1$ folyamat lokális változói, ill. $P1$ kimenő csatornaváltozói szerepelnek, akkor a P állítás stabil a többi folyamatban.*
- *Ha $P \Rightarrow P^{\overline{x} \leftarrow \overline{x};e}$ és $V(P) = \{\overline{x}\}$, akkor P stabil a teljes folyamathálózatban.*



8.2. ábra. Természetes számok generátora.

8.3. Természetes számok generátora

Első példánk csatornaváltozók használatára a 8.2 ábrán látható természetes számokat generáló folyamat specifikációja és az azt megvalósító program. A $P1$-gyel jelölt folyamat a folyamathálózat egy eleme, az általa generált számokat más folyamat(ok) használják fel.

A folyamat állapot- és paraméterterében egy egész típusú csatornaváltozó és ugyanezen csatorna történetváltozója jelenik meg. Mindkét változó értéke kezdetben az üres sorozat a 8.17 kikötés szerint.

$$\begin{array}{rll} A = & Ch(Int)\times & Ch(Int) \\ & x & \overline{x} \\ B = & Ch(Int)\times & Ch(Int) \\ & x' & \overline{x}' \end{array}$$

$$(x = x' = <> \wedge \overline{x} = \overline{x}' = <>) \in INIT_{x',\overline{x}'} \tag{8.17}$$

$$\overline{x} \leq [1,2,..] \in \text{inv}_{x',\overline{x}'} \tag{8.18}$$

$$\forall k \in N_0 : |\overline{x}| = k \hookrightarrow_{x',\overline{x}'} |\overline{x}| = k+1 \tag{8.19}$$

Az x csatorna történetváltozója segítségével könnyen megfogalmazhatjuk, hogy a csatornán egymás után, növekvő sorrendben a természetes számok jelennek meg (8.18): invariáns, hogy a csatorna történetének értéke a természetes számok sorozatának kezdő részsorzata. A csatornaváltozó segítségével nagyon nehéz lenne előírni hasonló követelményt. A csatornaváltozó értéke bármikor lehet az üres sorozat, így semmilyen támpontot sem ad arra nézve, hogy melyik volt az előző érték, amelyik megjelent rajta. Az invariánst a legkönnyebben úgy teljesíthetnénk, ha soha egyetlen elemet sem helyeznénk az x csatornára, az üres sorozat mindig prefixe a természetes számok sorozatának. A 8.19 kikötés azonban megköveteli, hogy az x csatorna történetének hossza elkerülhetetlenül növekedjen. Figyeljük meg, hogy a feladat nem fogalmaz meg sem terminálási, sem fixpont feltételt. Terminálás helyett végtelen működést követel meg.

A megoldás megtalálása érdekében bővítjük az állapotteret és finomítjuk a specifikációt:

$$\begin{array}{llll} A = & Ch(Int)\times & Ch(Int)\times & N_0 \\ & x & \overline{x} & i \\ B = & Ch(Int)\times & Ch(Int) & \\ & x' & \overline{x}' & \end{array}$$

$$(x = x' = <> \wedge \overline{x} = \overline{x}' = <>) \in INIT_{x',\overline{x}'} \tag{8.20}$$

$$i \in N_0 \in \text{inv}_{x',\overline{x}'} \tag{8.21}$$

$$((i = 0 \wedge \overline{x} = <>) \vee (i > 0 \wedge \overline{x} = [1,..i])) \in \text{inv}_{x',\overline{x}'} \tag{8.22}$$

$$\forall k \in N_0 : |\overline{x}| = k \mapsto_{x',\overline{x}'} |\overline{x}| = k+1 \tag{8.23}$$

Könnyen beláthatjuk a levezetési szabályok segítségével (5. fejezet), hogy a új specifikáció szigorúbb az előzőnél (8.21.) és (8.22.)-ből következik (8.18.), ill. (8.23.)-ból következik (8.19.).

A megoldó program a következő:

$$\begin{array}{ll} (& s_0 : i := 0, \\ & \\ & \{s_1 : x, i := x; (i+1), i+1\} \\ ) & \end{array}$$

Bizonyítás: Megmutatjuk, hogy a program megfelel a finomított specifikációnak.

(8.21): Megmutatjuk, hogy $i \in N_0 \in \text{inv}_S(x = x' = <> \wedge \overline{x} = \overline{x}' = <>)$

$sp(i := 0, x = x' = <> \wedge \overline{x} = \overline{x}' = <>) = i = 0 \wedge x = x' = <> \wedge \overline{x} = \overline{x}' = <> \Rightarrow i \in N_0$

$i \in N_0 \Rightarrow (lf(x, i := x; (i+1), i+1), i \in N_0) = i+1 \in N_0$.

(8.22): $sp(i := 0, x = x' = <> \wedge \overline{x} = \overline{x}' = <>) = i = 0 \wedge x = x' = <> \wedge \overline{x} = \overline{x}' = <> \Rightarrow i = 0 \wedge x' = <>$

$((i=0 \wedge \overline{x} = <>) \vee (i>0 \wedge \overline{x} = [1,..,i])) \Rightarrow$
$(lf(x, i := x;(i+1), i+1), ((i=0 \wedge \overline{x} = <>) \vee (i>0 \wedge \overline{x} = [1,..i]))) =$
$((i+1=0 \wedge \overline{x};(i+1) = <>) \vee (i+1>0 \wedge \overline{x}; i = [1,..,i+1])) =$
$(\overline{x}; i = [1,..,i+1])$

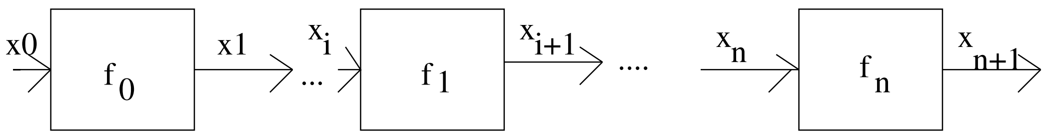
(8.23): $\forall k \in N_0 : |\overline{x}| = k \rhd_{x',\overline{x}'} |\overline{x}| = k+1 \Leftrightarrow$
$|\overline{x}| = k \Rightarrow lf(S, |\overline{x}| = k \vee |\overline{x}| = k+1) =$
$|\overline{x};(i+1)| = k \vee |\overline{x};(i+1)| = k+1$

és

$|\overline{x}| = k \Rightarrow lf(s, |\overline{x}| = k+1) = |\overline{x};(i+1)| = k+1.$ □

8.4. Adatcsatorna tétele

Adott egy $F = f_n \circ \ldots \circ f_0$ függvénykompozíció, amelynek értékét a $D =< d_1,..,d_m >$ sorozatban adott argumentumokra kell elemenként meghatározni. Tegyük fel, hogy $m >> n$, az adatok száma nagyságrendekkel nagyobb a függvénykompozícióban szereplő függvénykomponensek számánál. Feltételezzük azt is, hogy az egyes függvénykomponensek kiszámításához szükséges idő lényegében azonos. Ha elegendő processzor áll rendelkezésre, akkor megszervezhetjük az eredmény kiszámítását oly módon, hogy az egyes komponensfüggvények kiszámítását egy-egy processzorra bízzuk. Először kiszámítjuk f_0 értékét az első adatra d_1-re. Ennek eredményét $f_0(d_1)$-et továbbadjuk a következő processzornak, amelyik kiszámítja $f_1(f_0(d_1))$-et. Közben az első processzor megkapja a második adatot, d_2-t és meghatározza $f_0(d_2)$-t. n lépés után feltöltődik az adatcsatorna, az első adat eljut a n. függvénykomponenst számoló processzorig, majd minden további lépésben egy-egy újabb adatra kapunk végeredményt. Ha a kommunikációs költségeket figyelmen kívül hagyjuk, akkor arra a következtetésre juthatunk, hogy n processzorral az m adatból álló sorozatra $n+m$ lépésben elő tudjuk állítani a végeredményt, míg ugyanezt egyetlen processzoron m lépésben tudnánk megtenni. Tekintettel arra, hogy $n << m$, ezért a két lépésszám hányadosa: n, azaz n processzorral kb. n-szer gyorsabban végzünk.



8.3. ábra. Adatcsatorna

A formális specifikáció során először csak annyit fogalmazunk meg, hogy egy összetett F függvény értékét kell elemenként meghatározni a D sorozatra. A D sorozat elemei az x_0 csatornán vannak kezdetben, újabb adat nem érkezik a későbbiekben és fixpontban az eredmény, az $F(D)$ sorozat az x_{n+1} sorozat történetében található meg. Ez azt jelenti, hogy az eredmények rendre rákerültek az x_{n+1} csatornára, de esetleg már nincsenek ott.

$$\begin{array}{lllllllll} A & = & Ch(a) & \times & Ch(a) & \times & Ch(a) & \times & Ch(a) \\ & & x_0 & & \overline{x_0} & & x_{n+1} & & \overline{x_{n+1}} \\ B & = & Ch(a) & \times & Ch(a) & \times & Ch(a) & \times & Ch(a) \\ & & x_0' & & \overline{x_0}' & & x_{n+1}' & & \overline{x_{n+1}}' \end{array}$$

$$Q ::= (x_0 = \overline{x_0} = x_0' = \overline{x_0}' = D \wedge$$
$$\wedge x_{n+1} = \overline{x_{n+1}} = x_{n+1}' = \overline{x_{n+1}}' = <>)$$
$$Q \in INIT_{\underbrace{x_0'\overline{x_0}'x_{n+1}'\overline{x_{n+1}}'}_{h}} \tag{8.24}$$

$$\mathrm{FP}_h \Rightarrow \overline{x_{n+1}} = F(\overline{x_0}') = F(D) \tag{8.25}$$

$$Q \in \mathrm{TERM}_h \tag{8.26}$$

$$(\overline{x_0} = \overline{x_0}' = D) \in inv_h \text{ a teljes rendszerre} \tag{8.27}$$

A 8.24. kikötés szerint az x_0 és története kezdetben tartalmazza a D sorozatot, és a 8.27. kikötés szerint az x_0 sorozat története nem is változhat, tehát újabb elem nem kerülhet az x_0 csatornára. refpipe1. szerint az x_{n+1} csatorna kezdetben üres. Fixpontban 8.25. szerint megköveteljük, hogy az x_{n+1} története éppen $F(D)$ legyen. A 8.26. kikötés szerint a programnak terminálnia kell.

A megoldás előállításához bővítjük az állapotteret az $x_2 \dots x_n$ csatornaváltozókkal és történetváltozóikkal a 8.3. ábrának megfelelően és finomítjuk a specifikációt a fixpontfinomítás tétele alapján (8.28, 8.29), illetve variánsfüggvényt vezetünk be (8.29).

$$\mathrm{FP}_h \Rightarrow \forall i \in [0..n] : x_i = <> \tag{8.28}$$

$$\forall i \in [0..n] : (f_i(\overline{x_i} - x_i) = \overline{x_{i+1}}) \in \mathrm{inv}_h \tag{8.29}$$

$$\text{variáns függvény: } (|x_0|, ..., |x_n|) \tag{8.30}$$

A 8.30. pontban bevezetett variáns függvény értékét úgy határozzuk meg, hogy a rendezett n-es elemeit helyiértékkel súlyozzuk, az egyes csatornákon lévő elemek száma rendre egy $m+1$ alapú számrendszerben felírt szám számjegyeinek felel meg.

A fixpontfinomítás tétele alapján belátjuk, hogy az új specifikáció finomítása az eredetinek. Megmutatjuk, hogy: $(8.28) \wedge (8.29) \wedge (8.27) \Rightarrow (8.25)$.

Jelölje f^i az első $i+1$ függvény kompozícióját: $f^i ::= f_i \circ \ldots \circ f_0$. Teljes indukcióval belátható, hogy $(8.28) \wedge (8.29) \wedge (8.27) \Rightarrow (\overline{x_{i+1}}) = f^i(D)$. A lemmából $i = n$-re következik az állítás.

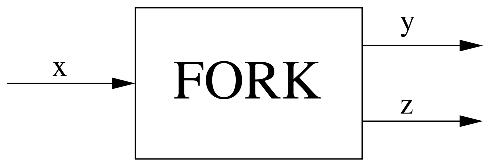
Az alábbi program megfelel a finomított specifikációnak.

$$
\begin{array}{ll}
S: & (\, \|_{i=1}^{n} \, x_i := <>, \\
\{ & \Box_{i=0}^{N} x_i, x_{i+1} := lorem(x_i), hiext(x_{i+1}, f_i(x_i.lov)), \\
 & \quad \text{ha } x \neq <> \; \} \;)
\end{array}
$$

8.3. absztrakt program. Adatcsatorna

8.5. Elágazás

Az elágazás egy olyan folyamat, melynek egy bemenő és két kimenő csatornája van.



A fenti ábrán látható elágazástól a következő négy feltétel teljesítését követeljük meg:

- adat ne vesszen el, azaz minden, ami rákerül a bemenetekre, az kerüljön feldolgozásra,
- a kimenő csatornákon csak olyan adat jelenjen meg, amelyik a bemenő csatornán érkezett, azaz legyen a külső zajoktól mentes,
- ha érkezik adat, akkor az előbb-utóbb feldolgozásra kerüljön.

Vezessünk be egy függvényt, melynek segítségével könnyebben tudjuk majd a specifikálni az elágazást:

$$split : Ch \times Ch \times Ch \longrightarrow L$$

A *split* függvényt induktívan definiáljuk:

$$split(<>,<>,<>) = igaz$$
$$split(a,b,c) \longrightarrow split(a;x,b;x,c) \wedge split(a;x,b,c;x).$$

Legyen *split* a fenti két tulajdonsággal rendelkező függvények közül a legkisebb igazsághalmazú. [3].

$A = Ch \times Ch \times Ch \times Ch \times Ch \times Ch$
$xyz\overline{x}\overline{y}\overline{z}$
$A = Ch \times Ch \times Ch \times Ch \times Ch \times Ch$
$x'y'z'\overline{x}'\overline{y}'\overline{z}'$

1.) $Q = (x = y = z = \overline{x} = \overline{y} = \overline{z} = x' = y' = z' = \overline{x}' = \overline{y}' = \overline{z}' =<>)$

$Q \in init_h$

2.) $P = (split(\overline{x} - x, \overline{y}, \overline{z})) \in inv_h$

3.) $\forall k \in N : |\overline{x}| \geq k \hookrightarrow_S |\overline{y}| + |\overline{z}| \geq k$

Az invariáns megköveteli, hogy ne vesszenek el adatok és zajmentes legyen a működés. A haladási feltétel nem követel meg "pártatlanságot" abban a tekintetben, hogy mindkét kimenő csatornára kerüljön időnként adat.

A feladatot megoldó *S* program a következő:

$$S : (SKIP, \{x,z := lorem(x), hiext(z, x.lov), \text{ ha } x \neq <> \boxdot$$
$$x,y := lorem(x), hiext(y, x.lov), \text{ ha } x \neq <>\})$$

Bizonyítás:

2.) $Q \Rightarrow lf(s_0, P) \equiv P \Rightarrow P$, mert $split(<> - <>, <>, <>) = split(<>, <>, <>) = igaz$

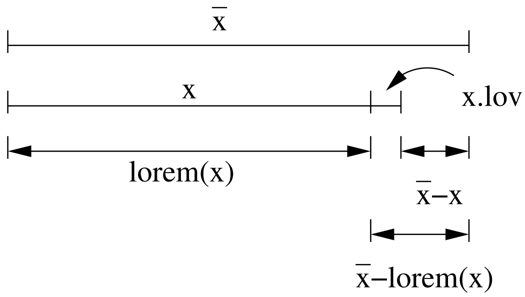
[3]Az megadott két tulajdonságnak megfelel az azonosan igaz függvény is. Bármely két, a tulajdonságoknak megfelelő függvény konjukciója is megfelel a tulajdonságoknak, így egyértelműen létezik legkisebb közöttük.

Belátjuk, hogy $P \Rightarrow lf(S,P) = lf(s_1,P) \wedge lf(s_2,P)$

Csak $lf(s_1,P)$-t bizonyítjuk. $lf(s_2,P)$ kiszámítása hasonlóan megy, ennek meggondolását az olvasóra bízzuk. $P \Rightarrow lf(s_1,P) =$
$(x \neq <> \rightarrow P^{y \leftarrow hiext(y,x.lov), \bar{y} \leftarrow \bar{y};x.lov, x \leftarrow lorem(x)}) \wedge (x = <> \rightarrow P) =$
$= (x \neq <> \rightarrow split(\bar{x} - lorem(x), \bar{y}; x.lov, \bar{z})) \wedge (x = <> \rightarrow P)$
Be kell látni, hogy

$$\bar{x} - lorem(x) = \bar{x} - x; x.lov$$

Az összefüggés teljesülését jól mutatja a következő ábra:



A *split* definícióját alkalmazva azt kapjuk, hogy ha P igaz volt, akkor igaz marad.

3.) A haladási feltétel bizonyítását az olvasóra bízzuk.

□

8.6. Elemenként feldolgozható függvények

Legyen H egy tetszőleges halmaz. Az elemenkénti feldolgozás tárgyalása során[4] az alábbi jelöléseket használjuk:

$$\begin{aligned} X &::= X_1 \times \ldots \times X_n, X_i \subseteq \mathcal{P}(H)(i \in [1..n]), \\ Y &::= Y_1 \times \ldots \times Y_m, Y_j \subseteq \mathcal{P}(H)(j \in [1..m]), \\ & x, \bar{x}, \bar{\bar{x}} \in X. \end{aligned}$$

[4] Az asszociatív művelet eredményének kiszámításakor minden egyes finomítási lépést, ill. az absztrakt program helyességét is részletes számításokkal bizonyítottuk. Az elemenkénti feldolgozás tárgyalása során a feladat részfeladatokra bontásának bemutatására és a különböző programkonstrukciók alkalmazására helyezzük a hangsúlyt. Az egyes lépések bizonyítása és az absztrakt program helyességének igazolása az előző tételben bemutatott módszerekkel könnyen elvégezhető.

8.1. Definíció (Teljesen diszjunkt felbontás). *$\overline{x}, \overline{\overline{x}}$ az $x \in X$* teljesen diszjunkt felbontás*a, ha $\forall i \in [1,n] : x_i = \overline{x}_i \cup \overline{\overline{x}}_i$ és $\forall i, j \in [1,n] : \overline{x}_i \cap \overline{\overline{x}}_j = \emptyset$.*

8.2. Definíció (Elemenként feldolgozható függvény). *Legyen $f : X \longmapsto Y$ függvény. Ha minden $x \in X$ bármely $\overline{x}, \overline{\overline{x}}$ teljesen diszjunkt felbontására*

$$f(\overline{x}) \cup f(\overline{\overline{x}}) = f(x), \tag{8.31}$$

$$f(\overline{x}) \cap f(\overline{\overline{x}}) = \emptyset, \tag{8.32}$$

akkor f elemenként feldolgozható függvény *[Fót 83].*

8.6.1. A feladat specifikációja

Kikötjük, hogy bármely fixpontban az y rendezett halmaz m-es értéke éppen $f(x')$ legyen (8.35.), ahol x' az x változó kezdeti értéke (8.33.). Megköveteljük, hogy a program biztosan elérje valamelyik fixpontját (8.34.).

$A = X \times Y \;\; x : X, y : Y, B = X \; x' : X.$

$$(x = x') \in INIT_{x'} \tag{8.33}$$

$$\text{Igaz} \hookrightarrow \text{FP}_{x'} \tag{8.34}$$

$$\text{FP}_{x'} \Rightarrow y = f(x'), \tag{8.35}$$

ahol f elemenként feldolgozható.

8.4. Megjegyzés. $\forall j \in [1..m] : y_j = f_j(x'_1, ..., x'_n)$.

Feltesszük, hogy x és $x' \setminus x$ invariáns[5] módon az x' teljesen diszjunkt felbontása és $x' \setminus x$-re már ismerjük az f függvény értékét. Felhasználva az elemenként feldolgozható függvények azon tulajdonságát, hogy értékük az argumentum teljesen diszjunkt felbontása esetén komponensenkénti diszjunkt unióval meghatározható a (8.33.)-(8.35.) specifikáció fixpont feltételét az $x_i = \emptyset$ feltétellel helyettesíthetjük.

8.6. Lemma. (Elemenkénti feldolgozás - a feladat finomítása) *Az alábbi specifikáció finomítása a (8.33.)-(8.35.) feltételekkel megadottnak.*

$$(x = x') \in INIT_{x'} \tag{8.36}$$

[5] A halmazkivonás komponensenként értendő.

$$Igaz \hookrightarrow FP_{x'} \tag{8.37}$$
$$FP_{x'} \Rightarrow \forall i \in [1..n] : (x_i = \emptyset) \tag{8.38}$$
$$inv_{x'}(\forall j \in [1,m] : (y_j \cup f_j(x_1,...,x_n) = f_j(x'_1,...,x'_n))) \tag{8.39}$$
$$inv_{x'}(\forall j \in [1,m] : (y_j \cap f_j(x_1,...,x_n) = \emptyset)) \tag{8.40}$$
$$inv_{x'}(\forall i,j \in [1,n] : (x'_i \setminus x_i) \cap x_j = \emptyset), \tag{8.41}$$

ahol f elemenként feldolgozható.

Biz.: A 5.8. lemma alapján. A bizonyításhoz szükséges matematikai meggondolások indoklása megtalálható [Fót 83]-ban.

8.6.2. A megoldás

Definiáljuk az sl függvényt a következőképpen: $sl : \mathcal{P}(\{1,\dots,n\}) \times H \longmapsto Y$,

$$sl(\{i_1,\dots,i_k\},e)_i ::= \begin{cases} \{e\}, & \text{ha } i \in \{i_1,\dots,i_k\} \\ \emptyset, & \text{ha } i \notin \{i_1,\dots,i_k\}, \end{cases}$$

ahol $i_1,\dots,i_n$ az $1,\dots,n$ számok egy permutációja.

Jelöljük a p és $\{e\}$ halmazok unióját $p\tilde{\cup}\{e\}$-vel, ha $e \notin p$. Hasonlóan, jelölje $p\tilde{-}\{e\}$ a $p \setminus \{e\}$ halmazt, ha $e \in p$. A $e : memp$ művelet egy nemdeterminisztikus feltételes értékadás, amely e-nek értékül adja p egy tetszőlegesen kiválasztott elemét, ha p nem üres.

8.7. Tétel. (Elemenkénti feldolgozás) *A 8.4. absztrakt program megoldja az elemenként feldolgozható függvény értéke kiszámításának feladatát, azaz megfelel a (8.36.)-(8.41.) specifikációnak.*

Biz.: A bizonyításhoz szükséges matematikai meggondolások indoklása megtalálható [Fót 83]-ban. □

8.3. Definíció. *Megfigyelhetjük, hogy az elemenkénti feldolgozás 8.4. programja az $U ::= \bigcup_{i\in[1..n]} \{x_i\}$ halmaz számosságával arányos számú állapotváltozás után jut fixpontba. A továbbiakban, amikor x* méretéről *beszélünk, akkor U számosságára gondolunk. Jelöljük x méretét $|x|$-szel.*

$$s_0: \quad \underset{j=[1..m]}{\|} y_j := \emptyset \parallel ch := hamis$$

$$S: \{ \quad (\underset{i=[1..n]}{\Box} e : mem(x_i) \parallel ch := igaz), \text{ ha } x_i \neq \emptyset \wedge \neg ch$$

$$ch, x_{i_1}, \ldots, x_{i_k}, y := \begin{cases} \ldots \\ hamis, \quad x_{i_1} \tilde{-} \{e\}, \ldots, x_{i_k} \tilde{-} \{e\},\ y \tilde{\cup} f(sl(\{i_1 \ldots i_k\}, e)) \\ \qquad \text{ha } e \in x_{i_1} \wedge \ldots \wedge e \in x_{i_k} \wedge e \notin x_{i_{k+1}} \wedge \ldots \wedge e \notin x_{i_n} \quad \} \\ \qquad \wedge ch \\ \ldots \end{cases}$$

Az elágazások száma $2^n - 1$.

8.4. absztrakt program. Elemenkénti feldolgozás

8.6.3. Teljesen diszjunkt felbontás párhuzamos előállítása

Reprezentáljuk a továbbiakban az x_i halmazokat olyan szigorúan növekvő monoton sorozatok formájában, amelyeknek elemei és részsorozatai közvetlenül hozzáférhetőek az indexek megadásával[6]. x_i első, legkisebb elemét jelölje $x_i(1)$. x_i hosszát $x_i.dom$-mal jelöljük. $x_i[i, j]$-vel jelöljük x_i azon részsorozatát, amely a $k : k \in (i, j]$ indexű elemeket tartalmazza.

Feltételezve, hogy $\Theta(n)$ processzor áll rendelkezésünkre, felbontjuk x-et n páronként teljesen diszjunkt részre. A felbontás kiegyensúlyozott, ha a legnagyobb és a legkisebb rész méretének különbsége legfeljebb 1. Kiegyensúlyozott felbontás esetén n processzor segítségével kb. n-szeresére gyorsíthatjuk f kiszámítását. Az egyes részekre kiszámított részeredményeket végül komponensenkénti diszjunkt unióval egyesíthetjük (8.2. def.).

Megadjuk a páronként diszjunkt felbontás feladatának formális specifikációját:

$A = X \times M \;\; x : X, m : M, B = X \; x' : X,$
$M = vektor([1..n, 0..n], \mathcal{N}_0)$

[6]Függvény típusú reprezentációt választunk [Fót 86].

$$(x = x') \in INIT_{x'} \quad (8.42)$$

$$\text{Igaz} \hookrightarrow \text{FP}_{x'} \quad (8.43)$$

$$\text{FP}_{x'} \Rightarrow cdd(m,x) \wedge (x = x'), \quad (8.44)$$

ahol $cdd(m,x)$ igaz, ha az m mátrix x páronként teljesen diszjunkt felbontását definiálja, azaz: $cdd(m,x) = \forall i, j \in [1,n] : \forall k,l \in [0,n-1] : k \neq l \rightarrow x_i[m(i,k), m(i,k+1)] \cap x_j[m(j,l), m(j,l+1)] = \emptyset$.

Ahhoz, hogy egy páronként teljesen diszjunkt felbontásról eldönthessük, hogy kiegyensúlyozott-e, meg kell határoznunk nem feltétlenül diszjunkt halmazok uniójának számosságát. Nem diszjunkt halmazok uniója azonban elemenként feldolgozható függvény. Ha az unió elemeinek számát az unió elemenkénti feldolgozással történő kiszámítása útján határozzuk meg, akkor önmagában annak eldöntése, hogy a felbontás kiegyensúlyozott-e ugyanannyi számítási lépést igényel, mint a teljes elemenkénti feldolgozás. Ez a módszer nyilvánvalóan nem vezet eredményre. Nyitott kérdés, hogy más úton lehetséges-e n (esetleg n^2) processzorral páronként teljesen diszjunkt és egyben kiegyensúlyozott felbontást hatékonyan előállítani.

A továbbiakban megmutatjuk hogyan lehet hatékonyan páronként teljesen diszjunkt felbontást előállítani anélkül, hogy a kiegyensúlyozottságot garantálnánk[7]. A felbontást a legnagyobb komponens egyenlő részekre osztásával definiáljuk, azaz ezen egyetlen komponens felosztását terjesztjük ki fokozatosan a többire oly módon, hogy a teljesen diszjunkt felbontás létrejöjjön.

Tegyük fel, hogy x_1 az x legnagyobb elemszámú komponense[8]. Bevezetjük a t vektort, amely tájékoztat arról, hogy a páronként teljesen diszjunkt felbontás mely összetevőit ismerjük. Ezen összetevők együttesét *x részlegesen meghatározott páronként teljesen diszjunkt felbontásának* nevezzük.

$t : vektor([1..n], \{0,..,n-1\})$. Jelöljük $pccd(m,t,x)$-vel, ha az m mátrix elemeivel és a t vektor értékeivel meghatározott, az $\{m(i,j) | j \leq t(i)\}$ *osztáspontokkal* megadott, részleges felbontás megfelel a páronként teljesen diszjunkt felbontás követelményeinek, azaz: $pcdd(m,t,x) = \forall i \in [1..n] : (m(i,0) = 0 \wedge m(i,n) = x_i.dom) \wedge \forall i, j \in [1,n] : \forall k,l \in [0,n-1] : k \neq l \wedge k < t(i) \wedge l < t(j) \rightarrow x_i[m(i,k), m(i,k+1)] \cap x_j[m(j,l), m(j,l+1)] = \emptyset$.

[7] A [Fót Hor Kozs 95] cikkben egy módszert mutattunk arra, hogy milyen módon oldható meg más úton az a feladat, hogy az egyes processzorok között a feladatmegosztást kiegyensúlyozzuk.

[8] A legnagyobb elemszámú komponens megtalálása visszavezethető egy maximumkeresésre, amely asszociatív művelet. A 8.3. tétel szerint ezt a feladatot $O(\log(n))$ lépésben megoldhatjuk. A későbbiekben látjuk majd, hogy $O(\log(n))$ lépés elhanyagolható a megoldáshoz szükséges összes állapotváltozáshoz képest.

A részlegesen meghatározott, páronként teljesen diszjunkt felbontás fogalmának bevezetésével finomíthatjuk a (8.42.)-(8.44.) specifikációt.

8.8. Lemma. (Páronként diszjunkt felbontás - a feladat finomítása) *Az alábbi specifikáció finomítása a (8.42.)-(8.44.) specifikációnak:*

$$(x = x') \in INIT_{x'} \tag{8.45}$$

$$Igaz \hookrightarrow FP_{x'} \tag{8.46}$$

$$FP_{x'} \Rightarrow \forall i \in [1.n] : t(i) = n - 1 \wedge (x = x'), \tag{8.47}$$

$$inv_{x'}(pccd(m,t,x)) \tag{8.48}$$

Biz.: A korábbiakhoz hasonlóan a 5.8. lemma alapján. □

Válasszuk a $v : A \longmapsto \mathcal{N}_0$ variáns függvényt a következőképpen:
$v ::= n * n - |\{m(i, j) | j \leq t(i)\}|$.

8.5. Megjegyzés. *A (8.43.) feltételt nem finomítottuk. A variáns függvényre vonatkozó 5.7. tétel segítségével bizonyíthatjuk majd azt, hogy a program megfelel a (8.43.)=(8.46.) feltételnek. Ebben az értelemben a variáns függvény megválasztása is egy finomítási lépésként fogható fel.*

A variáns függvény értéke akkor csökken, ha a $t(i)$ vektor elemeinek értéke nő. Ez azt jelenti, hogy a részlegesen meghatározott felbontást ki kell terjeszteni, az m mátrix további elemei értékének meghatározásával.

Az (8.48.) invariáns, $pccd(m,t,x)$ igaz marad, ha $m(i,t(i)+1)$-t azon x_i-beli elem indexének választjuk, amely elem kisebb vagy egyenlő $x_1(m(1,t(i)+1))$-nél, és amely elemre rákövetkező elem x_i-ben nagyobb, mint $x_1(m(1,t(i)+1))$. Jelöljük a $(j = m \vee (j \in (m,n] \wedge x_i(j) \leq h)) \wedge ((j+1 \in [m,n] \wedge x_i(j+1) > h) \vee (j = n))$ logikai függvényt $\Gamma(x_i, j, h)$-vel. A (8.48.) feltételt a

$$inv_{x'}(\forall i, j \in [1,n] : \Gamma(x_i, m(i, j), x_1(m(1, j)))) \tag{8.49}$$

feltétellel finomítjuk.

Mivel x_i monoton, a (8.49.) feltétellel definiált feladat visszavezethető szekvenciális logaritmikus keresésre [Fót 83].

Általánosítsuk a (8.49.) feltétellel definiált feladatot. Legyen H egy rendezett halmaz, $(m,n]$ az egész számok nem üres intervalluma és $f : (m,n] \longrightarrow H$ monoton növő függvény. Adott egy $h \in H$ érték. Keressük meg azt $j \in [m,n]$ egész számot, amelyre a $\gamma(j)$ tulajdonság teljesül, ahol $\gamma(j)$::=$(j = m \vee (j \in (m,n] \wedge f(j) \leq h)) \wedge ((j+1 \in [m,n] \wedge f(j+1) > h) \vee (j = n))$.

$$A = \mathcal{Z} \times \mathcal{Z} \times \mathcal{Z} \times H,\ \ m,n,j : \mathcal{Z},\ h : H.$$
$$B = \mathcal{Z} \times \mathcal{Z} \times H,\ \ m',n' : \mathcal{Z},\ h' : H.$$
$$Q ::= (m \leq n) \wedge (m = m' \wedge n = n' \wedge h = h' \wedge$$
$$\wedge \forall k,l \in (m,n] : k \leq l \Rightarrow f(k) \leq f(l))$$
$$Q \hookrightarrow \mathrm{FP}_{m',n',h'} \tag{8.50}$$
$$Q \in \mathrm{INIT}_{m',n',h'}, \tag{8.51}$$
$$\mathrm{FP}_{m',n',h'} \Rightarrow (h = h' \wedge j \in [m',n'] \wedge \gamma(j)). \tag{8.52}$$

Mivel az $[m,n]$ intervallum nem üres, ezért biztosan létezik olyan $j \in [m,n] : \gamma(j)$. Finomítsuk a specifikációt egy invariáns és egy variáns függvény bevezetésével. Az állapotteret két új komponens bevezetésével bővítjük. $u,v : \mathcal{N}_0$.

Válasszuk a $v_1 : A \longmapsto \mathcal{N}_0$ variáns függvényt a következőképpen: $v_1 ::= v - u + 1$.

$$inv_{m',n',h'}(h = h' \wedge [u,v] \subseteq [m',n'] \wedge u \leq v \wedge j \in [u,v]) \tag{8.53}$$
$$inv_{m',n',h'}(\forall k \in [m',n'] \setminus [u,v] : \neg\gamma(k)). \tag{8.54}$$

8.9. Tétel. (Logaritmikus keresés tétele) *A 8.5. absztrakt program megfelel a (8.50.)-(8.54.) specifikációnak.*

$$s_0 :\quad u,v,j := m,n,\lceil (m+n)/2 \rceil$$

$$\begin{array}{ll} S : \{ & u,j := j, \lceil (j+v)/2 \rceil, \text{ha } \neg\gamma(j) \wedge f(j) \leq h \\ & v,j := j, [(u+j)/2], \text{ha } \neg\gamma(j) \wedge f(j) > h \\ \} & \end{array}$$

8.5. absztrakt program. Szekvenciális logaritmikus keresés

Bizonyítás: A bizonyítás a [Fót 83]-ban adott bizonyítás alapján elvégezhető. □

Alkalmazzuk a logaritmikus keresés programját (8.5. prg.) n sorozatra (szuperpozíció), egyenként $n-1$-szer (explicit szekvencializálás $mod(n)$ [Lam Sin 79]).

Ezzel a módszerrel meghatározhatjuk az m mátrix értékét úgy, hogy az x egy páronként teljesen diszjunkt felbontását definiálja, azaz megkapjuk azt a programot, amely megfelel a (8.45.)-(8.48.) specifikációnak. A megoldás helyességét a szuperpozíció és a szekvencia levezetési szabályára hivatkozva igazolhatjuk. $n-1$ folyamat szekvenciáját egyszerű transzformációval ciklussá alakítjuk.

$$s_0: \quad \underset{i=[1..n]}{\square} m[i,0], m[i,n], t(i) := 0, x_i.dom, 0$$

$$\underset{i=[1..n]}{\square} m[1,i] := i * \lceil (x_1.dom \mathrm{DIV} n) \rceil$$

$$\underset{i=[1..n]}{\square} u(i), v(i), m(i,t(i)+1) := 0, x_i.dom, \lceil x_i.dom/2 \rceil$$

$$\begin{aligned}
S: \{ \quad & \underset{i=[1..n]}{\square} u(i), m(i,t(i)+1) := m(i,t(i)+1), \lceil (m(i,t(i)+1)+v(i))/2 \rceil, \\
& \text{ha } x_i(m(i,t(i)+1)) \leq x_1(m(1,t(i)+1)) \wedge \\
& \neg\Gamma(x_i, m(i,t(i)+1), x_1(m(1,t(i)+1))) \\
& \\
& \underset{i=[1..n]}{\square} v(i), m(i,t(i)+1) := m(i,t(i)+1), [(u(i)+m(i,t(i)+1))/2], \\
& \text{ha } x_i(m(i,t(i)+1)) > x_1(m(1,t(i)+1)) \wedge \\
& \neg\Gamma(x_i, m(i,t(i)+1), x_1(m(1,t(i)+1))) \\
& \\
& \underset{i=[1..n]}{\square} t(i), u(i), v(i), m(i,t(i)+1) := t(i)+1, 0, x_i.dom, \lceil x_i.dom/2 \rceil, \\
& \text{ha } \Gamma(x_i, m(i,t(i)+1), x_1(m(1,t(i)+1))) \wedge t(i) < n-1 \\
\} &
\end{aligned}$$

8.6. absztrakt program. Párhuzamos páronként teljesen diszjunkt felbontás

8.6.4. Diszjunkt halmazok uniója

Az f elemenként feldolgozható függvény értékét a páronként teljesen diszjunkt felbontással kapott szeletekre függetlenül, párhuzamosan meghatározhatjuk. A teljes eredményt a szeletekre kapott eredmény diszjunkt uniójaként állítjuk elő (8.2. def.).

Tegyük fel, hogy f értékét ismerjük x mind az n páronként teljesen diszjunkt szeletére. Jelöljük a függvényértékeket rendre $p(1),\ldots,p(n)$-nel, ahol $p(i) =$

$(p(i)_1, \ldots, p(i)_m) \in Y$. Tudjuk, hogy $\forall i \in [1..n] : \forall k,l \in [1..p(i).\underset{\sim}{dom}] : k \neq l \rightarrow p(i)_k \cap p(i)_l = \emptyset$. Tetszőleges $j \in [1..m]$-re: $y_j = f_j(x'_1, ..., x'_n) = \bigcup\limits_{i \in [1..n]} p(i)_j$. Ahhoz, hogy megkapjuk az $y = f(x')$ értéket, ki kell számítanunk n diszjunkt halmaz unióját minden $j \in [1..m]$-re. Tegyük fel, hogy a halmazok sorozatok formájában adottak, és a sorozatok elemei indexeik alapján elérhetőek. A halmazok unióját, mint a sorozatok konkatenációját állítjuk elő. $p(i)_j$ elemeit y_j azon részsorozatába kell bemásolnunk, amelyik kezdőindexe $\sum_{k=1}^{i-1} p(k)_j$, azaz meg kell határoznunk a $p(k)_j$ sorozatok hosszából kapott j szerint rendezett sorozat minden kezdőszeletének összegét. Az összeadás asszociatív művelet, így a feladat megoldható a 8.2. absztrakt program felhasználásával.

8.6.5. A párhuzamos elemenkénti feldolgozás tétele

8.10. Tétel. (A párhuzamos elemenkénti feldolgozás tétele) *A (8.33)-(8.35) specifikációs feltételek által definiált feladat megoldása a teljesen diszjunkt felbontás programjának (8.6. prg.), az elemenkénti feldolgozás programja (8.4 prg.) n-szeres szuperpozíciójának és a diszjunkt halmazok uniójára adott megoldás m-szeres szuperpozíciójának szekvenciája.*

8.6.6. Hatékonyság és általánosság

A fenti megoldás egyszerűen implementálható szinkron, aszinkron architektúrán is, és osztott rendszerben is [Cha Mis 89]. Osztott rendszer esetén csak akkor hatékony ez a megoldás, ha elegendően sok, $\Omega(\lceil \log(n) \rceil)$ (logikai) csatorna áll rendelkezésre processzoronként és a kommunikációs költség alacsony. Tegyük fel, hogy $\Theta(n)$ processzoron implementáljuk az absztrakt programot, $m = \Theta(n)$ és $|x|$ sokkal nagyobb, mint n. A párhuzamos teljesen diszjunkt felbontás eredményét $\Theta(n)$ processzor cseréli ki egymás között, hogy az egyes szeletekre megkezdődhessen az elemenkénti feldolgozás. Az elemenkénti feldolgozás eredményét ismét $\Theta(n)$ processzor cseréli ki egymás között[9]. A kommunikációs lépések száma tehát $\Theta(n)$ processzoronként. A teljesen diszjunkt felbontáshoz szükséges lépések száma $O(n * \log(|x|))$, a szelet elemenkénti feldolgozásához szükséges lépésszám: $\Omega(|x|/n)$ (kiegyensúlyozott felbontás esetén), a részeredmények konkatenációjához pedig $O(m * \log(n))$ lépés szükséges a részletösszegek kiszámítása

[9] Egyes párhuzamos gépek processzorai számára a filerendszer párhuzamosan is elérhető. Ebben az esetben a kommunikációs igény kisebb.

miatt. Kevés változó (n) és sok adat (x) és kiegyensúlyozott felbontás esetén a függvényérték meghatározásának jellemző költsége: $|x|/n$.

Elemenként feldolgozható függvény értékének kiszámítására vezethető vissza rendezett sorozatok összefésülése, halmazok uniója, az időszerűsítés [Fót Nyé 90], Conway problémája [Cha Mis 89] és még számos feladat.

8.7. Feladatok

8.1. Feladat. *Lokális minimumok száma*

Adottak az n hosszúságú egészeket tartalmazó vektor. Adjuk meg, hogy hány lokális minimum van a vektorban. (Lokális minimum egy elem, ha kisebb a baloldali és nem nagyobb a jobboldali szomszédjánál.) Oldjuk meg a feladatot legfeljebb $n+100$ *processzorral szinkron arhitekturán a lehető legkevesebb lépésben.*

- *a) Specifikáljuk a feladatot!*
- *b) Adjunk megoldó programot és mutassuk meg, hogy megfelel a specifikációnak !*

8.2. Feladat. *Feltételes összegzés*

Adott az $A : [1..N] \to Z$ *vektor, és az* $f : [1..N] \to \mathbb{L}$ *függvény.*
Számítsuk ki a

$$\sum_{i=1}^{N} \chi(f(i)) * A[i]$$

értéket!

Készítsük el a feladat specifikációját, írjunk fel megoldó programot és lássuk be a helyességét

8.3. Feladat. *Logikai mátrix sorainak egyezése egy mintával*

Adottak az $n \times m$ *logikai mátrix és az m-elemű logikai vektor. Adjuk meg, hogy a mátrix hány sora egyezik meg a vektorral. Oldjuk meg a feladatot* $m \times n$ *processzorral szinkron arhitekturán a lehető legkevesebb lépésben.*

- *a) Specifikáljuk a feladatot!*
- *b) Adjunk megoldó programot és mutassuk meg, hogy megfelel a specifikációnak*

8.4. Feladat. *Logikai mátrix szorzása*

Adottak az $n \times m$ *logikai mátrix és az m-elemű logikai vektor. Számítsuk ki a mátrix és a vektor szorzatát. Oldjuk meg a feladatot* $m \times n$ *processzorral szinkron arhitekturán a lehető legkevesebb lépésben.*

- *a) Specifikáljuk a feladatot!*

- *b) Adjunk megoldó programot és mutassuk meg, hogy megfelel a specifikációnak*

8.5. Feladat. *Első egyezés*

Adottak az $f,g,h : [1..N] \rightarrow Z$ függvények. Számítsuk ki az $l \in \mathbb{L}$ és $i \in [1..n]$ értékeket, ahol i az első olyan index, melyre a három függvény értéke megegyezik, l az a tulajdonság, hogy létezik ilyen index.

- *a) Specifikáljuk a feladatot!*
- *b) Adjunk megoldó programot és mutassuk meg, hogy megfelel a specifikációnak*

8.6. Feladat. *Számoljuk ki két N bites bináris szám szorzatát. Specifikáljuk, adjunk rá programot, majd lássuk be, hogy a program megoldja a feladatot, megfelel a specifikációnak. A megengedett műveletek: léptetés, bitek egyenlőségvizsgálata és bitre vonatkozó értékadás.*

8.7. Feladat. *Adott egy irányított, véges, körmentes gráf. Döntsük el, hogy van-e a gráfnak olyan irányított útja, amely minden csúcsot pontosan egyszer érint! (A gráfot egy $n*k$-s mátrixban reprezentáljuk.) A gráf csúcsainak száma: n, a csúcsok fokszáma legfeljebb k. Rendelkezésre áll $O(n*k)$ processzor.*

8.8. Feladat. *Adott egy irányított, véges, körmentes gráf. A csúcsokat 0-val, illetve 1-gyel címkézzük. Döntsük el, hogy van-e a gráfnak olyan irányított útja, amely mentén a csúcsok címkéinek sorozata pontosan egy előre megadott természetes szám kettes számrendszerben felírt alakját adja meg! A gráf csúcsainak száma: n, a csúcsok fokszáma legfeljebb k. (A gráfot egy $n*(k+1)$-es mátrixban reprezentáljuk.) Rendelkezésre áll $O(n*k)$ processzor.*

8.9. Feladat. *Visszavezetés*

Adott egy fekete-fehér digitalizált kép egy sora az N hosszúságú v vektorban. A vektor egy eleme 0 vagy 1, a 0 a fekete, az 1 fehér képpontot jelöl. A sor minden képpontjára állapítsuk meg (azaz írjuk a d vektor megfelelő elemébe), hogy milyen messze van tőle jobbra az első fekete képpont ! (Fekete pontokra ez az érték 0)

8.10. Feladat. *Visszavezetés*

Adott egy fekete-fehér, N sorból és M oszlopból álló, digitalizált kép. A kép minden képpontjára az m mátrix tartalmazza, hogy milyen messze van tőle jobbra az első fekete képpont (Fekete pontokra ez az érték 0). A kép egy bekezdésekre tagolt szöveget tartalmaz, minden bekezdés első sora beljebb kezdődik. Meg szeretnénk keresni a bekezdések kezdetét a k'epen. Feladat : Jelöljük meg a kép első oszlopának azon pontjait (azaz az l logikai vektor megfelelő elemeit állítsuk igazra) , melyek felett van legalább h olyan sor, melynek első w képpontja feh'er.

9. fejezet

Modellek és tulajdonságaik

9.1. Szemantikai modellek

Azt, hogy egy program futása során nemkívánatos mellékhatások nem lépnek fel, csak akkor tudjuk igazolni, ha a modell a folyamatok kölcsönhatása során fellépő jelenségek minél szélesebb körének jellemzésére alkalmas. Célunk, hogy modellünk minél valósághűbben tükrözze a sokprocesszoros multikomputereken futó programok viselkedését, ahol az események a különböző processzorokon egyidejűleg mennek végbe, valamint támogassa a lépésenkénti finomítást. Ezért kívánatos lenne, hogy a modell alapfogalmai magukban foglalják a valós párhuzamosság és a valós nemdeterminisztikusság fogalmát [Bak War 91, Mak Ver 91]. Másrészt törekednünk kell arra is, hogy modellünk ne váljon kezelhetetlenül bonyolulttá.

Valós párhuzamosság-ot ír le egy szemantikai modell, ha a párhuzamos programot nem tekinti azonosnak azzal a programmal, amelyet a folyamatok eleminek tekintett összetevőinek összefésülésével kapunk ($a\ b$ + $b\ a \neq a \mid\mid b$)[1], ellenkező esetben *összefésüléses* (interleaving) szemantikáról beszélünk. Az összefésüléses szemantika legnagyobb hátránya, hogy az összefésülés feltételezi az elemi műveletek (atomi akciók) egy rögzített szintjét. Ha az elemi művelet fogalma relatív, akkor a modell már ellentmondásra vezet ($(ab)(cd) + (cd)(ab)$?=? $(ab) \mid\mid (cd)$). Ha nem alkalmazzuk azt az egyszerűsítést sem, amely szerint a programok nemdeterminisztikus viselkedése a kezdőállapotra korlátozható, akkor *időben elágazó* szemantikáról, ellenkező esetben *időben lineáris* szemantikáról beszélünk (linear time, branching time). Az időben lineáris szemantika szerint a későbbi nemdeterminisztikus viselkedés előre figyelembe vehető a kezdeti állapotra vonatkoz-

[1] $a \mid\mid b$ - az a esemény és a b esemény időben átfedi egymást.
$a\ b$ - az a esemény megelőzi a b eseményt.
a+b - az a esemény és a b esemény közül pontosan egy következik be nemdeterminisztikusan.

tatva (időben előrehozott döntések). Ebben az esetben minden lehetséges későbbi nemdeterminisztikus viselkedést **előre** figyelembe kell vennünk, ha a program helyességét vizsgáljuk. A partner folyamatok állapotának figyelembevétele nélkül (túl korán) meghozott döntés holtpont kialakulását eredményezheti ($a(b+c) \neq ab+ac$). Ha a szelektív várakozást tartalmazó programot egyszerűen ekvivalensnek tekintjük azzal, amely előre vagy az egyik vagy a másik partner mellett dönt, akkor a lehetséges jó megoldások egy részét eleve kizárjuk.

Ha folyamatok közötti kapcsolatok topológiája a program futása során változhat, új folyamatok jöhetnek létre korlátlan számban, illetve folyamatok szűnhetnek meg, akkor *dinamikus* modellről, ellenkező esetben *statikus* vagy *korlátosan dinamikus* modellről beszélünk aszerint, hogy a folyamatok száma rögzített vagy felülről korlátos. A változások matematikai leírására az állapottér kiterjesztése, projekciója [Fót 88] biztosíthat eszközt. Szemantikai modellek tehát abban különböznek, hogy mely absztrakt programokat tekintik azonosnak.

- Az ún. *leíró* szemantika minden programhoz a szemantikai tartomány (pl. az állapottéren értelmezett bináris relációk halmaza vagy valamely algebrai struktúra) egy elemét rendeli hozzá. A programkonstrukcióknak a szemantikai tartományon értelmezett műveletek (pl. relációk szigorú kompozíciója) felelnek meg. Teljesülnie kell annak, hogy összetett program megfelelője a komponensekből a programkonstrukciónak megfelelő művelettel áll elő (kompozícionális megfeleltetés).

- *Műveleti* szemantika definiálásakor pl. címkézett átmenetgráfot (LTS) használhatunk. A gráf csúcsaiban helyezkednek el az absztrakt programok, az éleket általában elemi műveletekkel címkézzük. A gráf azt definiálja, hogy egy (összetett) program egy elemi művelet (vagy komponens program) végrehajtása után mely programmal ekvivalens módon működik tovább. Azt vizsgáljuk, hogy mely absztrakt programok viselkedése azonos, azaz mely absztrakt programoknak megfelelő csúcsokból elindulva kapunk ekvivalensnek tekintett címkesorozatokat. Az ekvivalencia definíciója esetleg önmagában is bonyolult. (A *processzalgebrában* [Hen 88] definiált tesztelési ekvivalencia vizsgálatakor például gráfok direkt szorzataiból indulunk ki.)

- *Axiómatikus* szemantikáról beszélünk, ha absztrakt programok ekvivalenciáját axiómák és levezetési szabályok segítségével adjuk meg.

Amikor utasítások, szekvenciális programok hatásrelációját, mint az állapottér feletti bináris relációt definiáljuk, akkor leíró szemantikai eszközöket alkalmazunk programok ekvivalenciájának definiálására. Ebben a modellben programok helyességét statikus módon vizsgáljuk (pl.: halmazok összehasonlítására vezetjük vissza), míg műveleti szemantikát alkalmazva a program helyes működését annak dinamikus viselkedése elemzésével igazolhatjuk. Ez utóbbi módszer általában több hibalehetőséget hordoz magában, de előnye, hogy a program viselkedését szemléletes formában írja le. Az axiomatikus szemantika automatikus helyesség-bizonyításra alkalmas elsősorban.

Gyakran felvetik a kérdést, hogy három különböző formában definiált szemantika ekvivalens-e (az axiómatikus *teljes* és *ellentmondásmentes*-e illetve a műveleti *teljesen absztrakt*-e a leíróra nézve [Hen 88]), azaz pontosan ugyanazon absztrakt programokat tekintik-e ekvivalensnek. A kérdés eldöntése gyakran összetett matematikai apparátus használatát igényli, különösen, ha a leíró szemantikai tartomány egy bonyolult algebrai struktúra vagy metrikus tér [Bak War 91]. Ilyenkor kérdésessé válik az elmélet gyakorlati alkalmazhatósága is, mert szükségképpen hasonlóan bonyolult eszközökre van szükség annak eldöntéséhez is, hogy az absztrakt program megfelel-e a specifikációnak.

A gyakorlati alkalmazás szempontjából tehát elsődleges, hogy a jelenségek minél tágabb körének leírására alkalmas, de minél egyszerűbb matematikai struktúrájú szemantikai tartomány segítségével modellezzük a párhuzamos programok világát.

10. fejezet

Irodalmi áttekintés

Párhuzamos folyamatok leírására, szemantikájuk definiálására, lépésenkénti finomításukra számos modellt alkottak. Ezek a modellek különböznek céljukban, kifejezőerejükben, matematikai eszközkészletükben. Ebben a fejezetben röviden ismertetünk néhány, a dolgozatban leírt modellhez rokon elméletet. Azokra a fogalmakra, módszerekre helyezzük a hangsúlyt, amelyek megfelelőit megfogalmaztuk a relációs modellben is. Megemlítünk néhány olyan eredményt is, amely az általunk választott kutatási iránytól távolabb esik. Sem a felsorolásban, sem a kiválasztott modell elemzésében nem törekedtünk teljességre.

10.1. A Hoare logika kiterjesztései

A szekvenciális programok helyességbizonyítására Floyd, Hoare, Dijkstra és mások által kidolgozott elméletet már a 70-es évek elején kiterjesztették olyan elemekkel, amelyeket konkurens viselkedés ill. szinkronizáció leírására, holtpontmentesség és más biztonságossági tulajdonságok bizonyítására fogalmaztak meg. A párhuzamos programot, mint szekvenciális folyamatok együttesét vizsgálták, és olyan következtetési szabályok megfogalmazására törekedtek, amelyek az egyes folyamatok helyességének bizonyítása és az összetevők kölcsönhatásainak korlátozása mellett a párhuzamos program helyességét igazolták.

Interferenciamentesség bizonyítása

Owiczki és Gries fogalmazta meg 1976-ban az *interferenciamentesség* követelményét [Owi Gri 76]. Két szekvenciális folyamat interferenciamentes, ha az egyik helyességbizonyításában alkalmazott kritikus feltételek teljesülését a másik folyamat atomi műveletei nem érvénytelenítik. Lamport a *monoton predikátum* fogalmának bevezetésekor hasonló követelményt támasztott az együttműködő folyamatok kölcsönhatására [Lam 77]. Ezek a feltételek egy-egy n ill. m atomi lépésből álló folyamatpár esetén az összetevők szekvenciális helyességének igazolásához szükséges bizonyítási lépéseken túlmenően a párhuzamos program parciális helyességének belátásához további $n * m$ bizonyítási lépést tettek szükségessé. A módszer alkalmazásakor további nehézséget okozhat az is, hogy a párhuzamos program komponenseinek állapota nem egyértelműen meghatározott az összetett program változóinak értéke által, ezért a bizonyítások során ún. segédváltozókat vagy más néven *kontrollváltozó*kat is be kell vezetni. A segédváltozók a program futása során értéket kapnak, de értéküket csak a helyességbizonyítás során használjuk fel. Az alkalmasan megválasztott segédváltozók értéke alapján meghatározhatóvá válik, hogy melyik összetevők felelősek a korábbi állapotátmenetekért, az egyes folyamatok mely atomi művelet végrehajtásánál tartanak. Ugyanezen célt szolgálják a Lamport által bevezetett kontrollváltozók [Lam 90], melyek az egyes atomi műveletekhez rendelt logikai változók. Egy művelet kontrollváltozói pontosan akkor vesznek fel logikai *igaz* értéket, amikor az adott művelet végrehajtása megkezdődik, éppen folyamatban van, illetve véget ért.

van Lamsweerde és Sintzoff [Lam Sin 79] a párhuzamos program szerkezetét a folyamatok halmaza helyett *atomi akciók halmaza*ként, *iteratív programstruktúra* alakjában rögzíti. Megmutatják, hogy ún. *explicit szekvencializálás*i technikával szekvenciális összetevők is felbonthatóak atomi műveletek halmazára. Modelljükben a megoldás levezetésén és nem a kész programok helyességbizonyításán van a hangsúly. Az iteratív program ciklusinvariánsa mint a párhuzamos program globális *invariáns*a jelenik meg és nagyban megkönnyíti *biztonságossági feltétel*ek megfogalmazását és bizonyítását. Modelljükben egyes *haladási tulajdonság*ok kifejezésére és bizonyítására is eszközt adnak, pl. meghatározzák az adott végfeltétel elérésének *leggyengébb előfeltétel*ét, amelyet a Dijkstra által definiált leggyengébb előfeltételből [Dij 76] felépített *funkcionálok fixpontja*inak kiszámításával határoznak meg. Módszert adnak arra is, hogy hogyan határozzuk meg egy adott invariáns biztosítását garantáló *szinkronizációs feltétel*eket, hogyan transzformáljuk a programot olyan alakba, hogy az invariánst, és így a szinkronizációs feltételeket is a lehető leggyengébbre választhassuk meg. Megadják *holtpontmentes* és *kiéheztetésmentes* program *szintézis*ének módszerét. A UNITY[1]-ben [Cha Mis 89] és az általunk megfogalmazott modellben definiált *absztrakt*

[1] Unbounded Nondeterministic Iterative Transformations

program struktúrája megegyezik van Lamsweerde és Sintzoff párhuzamos programjainak stuktúrájával.

Párhuzamos programok haladási feltételeinek leírására alkalmas predikátumtranszformereket rajtuk kívül sokan megfogalmaztak. A leggyengébb előfeltételből felépített monoton funkcionálok legkisebb és legnagyobb fixpontjainak együttes alkalmazásával definiálja Park iteratív programszerkezetek haladási tulajdonságait *pártatlan ütemezés* feltételezése mellett. Hasonló predikátumtranszformert alkot Morris rekurzív programok haladási tulajdonságainak leírására. Számos, egyes speciális haladási tulajdonságokat különböző pártatlansági feltételek mellett kifejező predikátumtranszformert ad meg fixpontos alakban Flon és Suzuki [Flo Suz 81], Francez [Fra 86], Lukkien [Luk Sne 92].

Globális invariánsok bevezetése

van Lamsweerde és Sintzoff eredményeit alkalmazza Andrews konkurens programok *szintézis*ére [And 91] azzal a különbséggel, hogy a programszerkezetet nem iteratív formában definiálja, hanem visszatér az Owiczki-Gries modell programfogalmához. Módszere a megoldás *lépésenkénti finomítás*án alapszik. Először a megoldás szerkezetét definiálja a feladat meghatározásával egyidejűleg, azaz megadja a megoldásban szereplő folyamatokat, azok közös állapotterét és a kölcsönhatásukat korlátozó invariánst. Második lépésként definiálja az egyes folyamatok szekvenciális vázát a bizonyítás vázlatával együtt. A harmadik lépésben van Lamsweerde és Sintzoff módszerével meghatározza a szinkronizációs *őrfeltétel*eket a leggyengébb előfeltétel kalkulus alapján. Végül implementálja az *absztrakt program*ot egy konkrét nyelven és architektúrán. Andrews részletesen elemzi azokat a heurisztikus módszereket, amelyekkel biztosítható folyamatok interferenciamentessége. Könnyű garantálni, hogy két folyamat nem interferál egymással, ha azok diszjunkt *változó*kon dolgoznak, azaz amelyik változót az egyik folyamat ír, azt a másik folyamat egyáltalán nem használja. Ha a változók átfedik egymást, akkor a másik folyamat utasításainak hatását is figyelembe véve gyengíthetjük a bizonyítási vázlat kritikus feltételeit. Jól alkalmazható a *globális invariánsok módszere*, mikor arra törekszünk, hogy az egyes atomi utasítások elő- és utófeltételeit a globális invariáns és egy olyan állítás konjunkciójaként írjuk fel, amely állítás csak a folyamat lokális változóitól vagy legfeljebb csak olyan változóktól függ, amelyet csak az adott folyamat ír. Szinkronizációt is alkalmazhatunk az interferencia elkerülésére, oly módon, hogy őrfeltétellel korlátozzuk az adott kritikus állítást érvénytelenítő, interferenciát okozó utasítás végrehajtásának lehetőségét olyan állapotokra, amikor interferencia nem jön létre. Andrews definiálja a *finom atomicitás* és a *durva atomicitás* fogalmát. Az atomi akciók szintjének megválasztása kihat arra, hogy a helyességbizonyítás szempontjából mi számít kritikus állításnak és egyben meghatározza az absztrakt program implementációjának lehetséges hatékonyságát. *Programozási tétel*ek szintézise során törekedni fogunk arra, hogy a tételek végső alakját az Andrews által megfogalmazott legfinomabb atomicitás feltételezése mellett adjuk meg és minél gyengébb szinkronizációs feltételeket határozzunk meg.

10.2. Egy reláció alapú modell

E. Best 1983-ban megfogalmazta párhuzamos programok egy *reláció alapú szemantikai modell*jét [Best 83]. A szemantikai tartomány elemei olyan relációk, amelyek az *állapottér* pontjaihoz *érvényes végrehajtási sorozat*okat rendelnek hozzá. A végrehajtási sorozatok elemei felváltva állapotok (változók értékei) illetve az állapotátmenetért felelős folyamatok azonosítói. Két szomszédos állapotot mindig az állapotátmenetért felelős atomi művelet *hatásreláció*ja kapcsol össze. A végrehajtási sorozat tehát alkalmas arra, hogy egy prefixe egyértelműen azonosítsa a párhuzamos program minden egyes komponensének állapotát. A végrehajtási sorozatok tulajdonságainak elemzésével Best definiálja a *holtpont*, a *pártatlan ütemezés*, a *parciális helyesség*, a *terminál*ás fogalmát. Modelljében megjelenik a *feladat* (cél) fogalma, amely az állapottér felett értelmezett *bináris reláció* ([Fót 83]). Igazolja, hogy az Owiczki-Gries féle következtetési szabályrendszer *helyes és teljes* a definiált parciális helyesség bizonyítására nézve. Absztrakt programok relációs műveleti szemantikájának definíciójában az érvényes végrehajtási sorozat fogalmát általánosítjuk.

10.3. Folyamatok viselkedésének algebrai leírása

Trace-ek

Párhuzamos programok algebrai modelljei gyakran indulnak ki az egyes folyamatok végrehajtásánál megfigyelhető események (atomi műveletek) sorozataiból.

Mazurkiewicz definiálja a *konkurens ábc* fogalmát [Maz 89]. A konkurens ábc az események azonosítóinak halmaza és az események között fennálló szimmetrikus és reflexív bináris *függőségi reláció* rendezett párja. Két eseménysorozat *ekvivalens*, ha egymás permutáltja és két esemény csak akkor szerepel különböző sorrendben a két sorozatban, ha nem köti őket össze a függőségi reláció.

Eseménysorozatok ekvivalenciaosztályai a trace-ek. Trace-ek halmazát *nyelv*nek nevezzük. Trace-ek felett *parciális rendezés*i reláció adható meg a prefix fogalmának általánosításával. *Prefix zárt nyelv*ek folyamatok viselkedését írják le. Trace halmazok műveletei *programkonstrukciós* műveleteknek feleltethetőek meg. A műveletek algebrai tulajdonságainak elemzésével, homomorfizmusok megadásával programok és összetevő folyamatok tulajdonságait vizsgálhatjuk.

Címkézett átmenetrendszerek

Pratt a trace-ek fogalmát általánosítva *pomset*-ekkel [Pra 86] adja meg párhuzamos programok *leíró szemantiká*ját. A modell érdekessége, hogy az eddig ismertetett modellektől eltérően nem *összefésüléses szemantiká*jú. Pratt nagyszámú konstrukciós operátort definiál, kombinatorikai, logikai műveleteket, illetve algebrai lezártakat. Pratt valós párhuzamos szemantika mellett definiált konstrukciós műveleteinek megfelelő relációs műveletek definíciója nagyban növelheti az általunk megfogalmazott modell kifejezőerejét. A relációs modell ilyen irányú kiterjesztése további kutatási feladatot jelent (6 fejezet).

Mazurkiewicz és Pratt modelljében a feladatot a megengedett „eseménysorozatok" megadásával specifikálhatjuk. A modellek folyamatok viselkedésének analízisére alkalmasabbak, mint párhuzamos programok szintézisére.

Milner CCS[2] modelljében [Mil 89] a megfigyelhető események azok, amikor egy folyamat a külvilággal kommunikál. Folyamatokat összeköthetünk csatornákkal majd ezeket az összetett egységeket egyetlen egységként kezelhetjük oly módon, hogy a belső kommunikációt elrejtjük. Feladatot is CCS folyamat alakjában definiálunk, megadjuk a kívánt megoldás csatornáit és előírjuk, hogy ezen csatornákon milyen kommunikációs viselkedés legyen megfigyelhető. Párhuzamos rendszerek szemantikáját Milner műveleti szemantikával, *címkézett átmenetgráf*fal[3] adja meg. Egy összetett rendszer megold egy feladatot, ha kívülről megfigyelhető viselkedése ekvivalens (szigorúan ekvivalens, megfigyelhetően ekvivalens, megfigyelhetően kongruens) a specifikált viselkedéssel. Milner leír egy egyszerű párhuzamos programozási nyelvet, amelynek szemantikáját CCS-sel definiálja. Megad egy *modális logika* alapú specifikációs nyelvet is, amellyel CCS folyamatok viselkedésére tud előírásokat tenni.

Az általunk bemutatott relációs modellben folyamatok viselkedésére vonatkozó előírásokat a modális logikákhoz tartozó temporális logikai műveleteknek

[2]Calculus of Communicating Systems

[3]LTS - Labelled Transition System

megfelelő relációk megfogalmazásával teszünk.

Hennessy általánosan fogalmazza meg a *processzalgebra* elméletét [Hen 88], eredményei alkalmazhatóak pl. a CCS-re is.

CSP

A CCS-hez hasonló Hoare CSP[4] elmélete is [Hoa 85], amely folyamatok szemantikáját *műveleti szemantiká*val adja meg. A feladatokat a folyamat viselkedésére vonatkozó logikai állításokkal specifikálja. A *megoldás* definícióját a program struktúrája szerint alkalmazott *következtetési szabályrendszer*re építi. A következtetési rendszer szabályai adottak, illetve Hoare megad egy *leíró szemantiká*t is, amely alapján a *levezetési szabály*ok bizonyíthatóak. A specifikációs nyelv a csatornákhoz rendelt történetváltozókra vonatkozó alapállításokból felépített logika. *Csatornaváltozók*hoz rendelt történetváltozókból felépített függvénykompozíciókat a dolgozatban bemutatott modellben is használunk [Hor 93-96].

10.4. Temporális logikai modellek

Konkurens programok tulajdonságainak leírására alkalmas eszköz a temporális logika. Temporális logikában az egyes formulákat egy olyan modell felett értelmezzük amelyben a formulák igazságértéke általában *időpontról időpontra* változó. Számos olyan modellt fogalmaztak meg, amely az összetett temporális logikai eszközkészlet egy alkalmasan megválasztott részét[5] alkalmazza folyamatok specifikációjára [Cha Mis 89, Jär 92] esetleg folyamatok szemantikájának definiálására is [Lam 91].

Ezek közül a legismertebbek közé tartozik a Lamport által megfogalmazott TLA[6]. Lamport a programot is és a feladatot is TLA formulával adja meg [Lam 91], így a megoldás fogalma könnyen bevezethető.

A dolgozatban egy másik temporális logika alapú modellre támaszkodunk, a UNITY-ra [Cha Mis 89]. A UNITY biztonságossági és haladási tulajdonságokat kifejező oprátorai megadhatóak lineáris temporális logikai alakban [Sin 91]. Ez a modell alkalmas specifikációk lépésenkénti finomítására. UNITY-ban az absz-

[4] Communicating Sequential Processes, az elmélet nem azonos a Hoare által korábban bevezetett CSP nyelvvel [Hoa 78].

[5] A modell operátorainak jelentése megadható a lineáris temporális logika valamely műveletsorozata segítségével [Sin 91].

[6] Temporal Logic of Actions

trakt program struktúrája iteratív. Leggyengébb előfeltétel kalkulusra vezethető vissza annak igazolása, hogy egy program rendelkezik egy adott tulajdonsággal.

A 11. fejezetben a temporális logikákat részletesen bemutatjuk.

10.5. További modellek

Párhuzamos folyamatok leírására elterjedt automataelméleti eszköz pl. a Petri háló és az I/O automata. Számos további megközelítés lehetséges, modellezhetünk párhuzamos számításokat neurális hálókkal, sejtautomatákkal, stb. Léteznek tisztán funkcionális számítási modellek is, mint pl. a lambda kalkulus és a funkcionális programozási nyelvek más modelljei. Ebben az esetben egyes redukciós szabályok párhuzamos alkalmazhatósága alakjában jelenik meg a párhuzamosság.

Párhuzamos folyamatok modelljeit tekinti át pl. [Var 81, Lam Lyn 90, Koz 94].

11. fejezet

Matematikai eszközök

11.1. Temporális logika

A temporális logikák a klasszikus logika [Pász 93] lehetséges kiterjesztései. A temporális logikai nyelvek szemantikájának definiálásakor szükségünk lesz az időpontok halmazára[1]. Az egyes formulákat egy olyan modell felett értelmezzük amelyben a formulák igazságértéke általában *időpontról időpontra* változó [2].

A nyelv szemantikájának definiálásakor megadjuk, hogy adott időpontban mely atomi formulák teljesülnek. Az időpontok halmaza felett egy, adott tulajdonságokkal rendelkező reláció[3] definiált [Ben 88]. Időstruktúráról beszélünk, ha az időpillanatok halmaza felett definiált reláció tulajdonságai megfelelnek az időről alkotott alapvető elképzeléseinknek, azaz a reláció[4] irreflexív és tranzitív. A reláció további tulajdonságai határozzák meg az időstruktúra temporális logikai típusát. Megkülönböztethetünk pl. lineáris ($\forall x,y : x < y \vee x > y \vee x = y$), majdnem összefüggő ($\forall x,y,z : x < y \rightarrow (x < z \vee z < y)$), ill. elágazó idejű modellt, ahol

---

[1] Az időpont fogalmát absztrakt értelemben használjuk. Nem foglalkozunk az időpontok halmaza felett metrika definiálásával. Az ismertett modell egyelőre nem terjed ki a programok valós idejű végrehejtásának leírására [Mel 87].

[2] A temporális logikák a modális logikák körébe tartoznak [Rácz 92]. A klasszikus logikai formula igazságértéke a modális logikák formalizmusa szerint, a logika típusa alapján nemcsak az individumváltozóktól, hanem valamilyen más paramétertől, pl.: helytől, időtől, stb. is függ. Defniálhatnánk pl. olyan modális logikát is, amelyben a paraméter időpillanat helyett időintervallum [Mel 87]. Egy ilyen logika alkalmas lehet valós egyidejűség leírására. A modális paramétertér felett egy elérési reláció definiált. A különböző paraméterértékekhez tartozó univerzumokban egyszerre értelmezzük ugyanazon formulák igazságértékét. A formula igazságértéke egy adott paraméterérték által meghatározott univerzumban általában függ ugyanazon vagy más formulák az elérési reláció felhasználásával meghatározott univerzumokban felvett értékeitől. Az ilyen jellegű összefüggések leírására vezetik be a modális operátorokat, amelyek segítségével a modális paraméter ill. az elérési reláció explicit használata elkerülhető. A modális operátorokat tekinthetjük az egzisztenciális és az univerzális kvantor általánosításának. Az általánosított kvantor jelentése az elérési reláció tulajdonságaitól függ.

[3] Relációnak nevezzük halmazok direktszorzatának egy részhalmazát. Bináris relációról beszélünk, ha a reláció pontosan két halmaz direktszorzatának része.

[4] azaz a modális logika elérési relációja

a jövőbe vezető utak diszjunktak. Vannak véges ill. végtelen ($\forall x : \exists y : x < y$), diszkrét ($\forall x,y : x < y \rightarrow \exists z : (x < z \wedge \nexists u : (x < u \wedge u < z))$, ill. sűrű struktúrák. Megkövetelhető az idő homogenitása, azaz bármely x,y időpontpárhoz található olyan relációtartó automorfizmus, amely x-et y-ba viszi. Izotróp egy struktúra, ha izomorf azzal, amelyben a rendezési reláció fordított, azaz amelyben $a < b$-nek $b < a$ felel meg.

A nyelv szintaxisa szempontjából ugyanúgy, ahogyan a klasszikus logikában, a temporális logika esetén is megkülönböztetjük a 0-ad és magasabbrendű logikai nyelveket. 0-ad rendű esetben a klasszikus logika 0-ad rendű formuláiból és a temporális operátorokból építjük fel a nyelvet. A temporális operátorok használatára vonatkozó szintaktikus szabályok határozzák meg a nyelv temporális logikai típusát. Ennek megfelelően választható ki a nyelvet interpretáló időstruktúra. Az időpillanatok halmaza megadható, mint egy program programállapotainak halmaza.[5] *Endogenous* az a logika, amelynek időstruktúráját egyetlen program állapotai alapján definiálják, ill. *exogenous* a logika, ha programkonstrukciók is megengedettek.

Azt mondjuk, hogy egy temporális logikai formulának modellje egy időstruktúra, ha a struktúrában van olyan időpont, amelyben a formula teljesül[6]. Egy adott probléma megoldása a temporális logika terminológiája szerint a feladatot leíró formulahalmaz egy modelljének megtalálása lehet. Az automatikus programszintézishez modellkereső algoritmusokra van szükség. A szakirodalomban ismertetett eredmények [Eme Sri 88] azt mutatják, hogy ezek az algoritmusok általában nagyon rossz hatékonyságúak, a megoldás előállításához a specifikáció hosszával exponenciálisan arányos időre van szükség. Az előállított megoldás minősége szempontjából az sem közömbös, hogy a formulahalmazt kielégítő modellek melyikét találja meg az algoritmus.

A továbbiakban egy rögzített interpretációban értelmezzük a 0-ad rendű elágazó idejű temporális logika operátorait.[7]

Az egyes temporális logikák között a leglényegesebb különbség a kifejezőerőben van. Általában minél nagyobb a logika kifejezőereje, annál bonyolultabb

[5] Az elágazó idejű temporális logika formuláinak interpretációjához használt szokásos modellek [Eme Sri 88] és az általunk definiált program (3.15 def.) könnyen megfeleltethető egymásnak.

[6] Ha megadjuk, hogy a feladat (2.1. def.) átemenet- és peremfeltételeinek megfelelő temporális logikai formulákat hogyan írjuk fel, akkor a megoldás definícióját visszavezethetjük arra, hogy formulák egy halmazának matematikai logikai értelemben modellje-e a programnak megfelelő temporális logikai struktúra. Haladási feltételek lineáris temporális logikai megfogalmazására mutat példát [Lam 91] a 4.2.3. bekezdésben.

[7] Egy rögzített program programállapotai által definiált időstruktúra esetén egy P formula az állapottér felett értelmezett logikai függvényt definiál (2. fejezet). Az alábbi temporális logikai műveletek segítségével tehát rögzített S absztrakt program esetén P,Q logikai függvényekből új logikai függvényeket konstruálhatunk.

a logika eldöntési problémája.[8]

11.1.1. Elágazó idejű temporális logika

Az elágazó idejű temporális logikában az időpillanatok halmaza felett olyan parciális rendezés (11.12. def.) definiált, amely az időpillanatokat összefüggő fába rendezi. Az időstruktúra több lehetséges jövőt ír le.

Az elágazó idejű temporális logikák egyike a CTL (Computation Tree Logic). Az alábbiakban követjük [Eme Sri 88] leírásmódját és a CTL-en keresztül mutatjuk be az elágazó idejű temporális logikákat. Megadjuk a CTL egy változata, a CTL* szintaxis és szemantika formális definícióját (11.1., 11.6. def.).

Programok elágazó idejű temporális logikai jellemzése során a programot irányított fának feleltetik meg, azaz a leíró szemantikai tartomány elemei fák. A fa csúcsai állapotok, az éleket pedig az állapotátmenetet megvalósító programkomponens azonosítójával címkézik. Az irányított fa csúcsaira ill. útjaira állításokat fogalmaznak meg [Eme Sri 88].

Tekintsük példaként a következő CTL formulát: $AG(\neg CS_1 \vee \neg CS_2)$. Az AP alakú formula egy állapotra vonatkozik, ún. *állapotformula*. A $P = G(\neg CS_1 \vee \neg CS_2)$ formula egy útra vonatkozó kikötés. ún. *útformula*.

Röviden ismertetjük az AP,EP,GP,FP,XP formulák jelentését:

- $AP(e)$, ha minden e -ból induló t útra $P(t)$
- $EP(e)$, ha van olyan e -ból induló t út, hogy $P(t)$
- $GP(t)$, ha a t út minden e pontjára $P(e)$
- $FP(t)$, ha a t úton van olyan e pont, amelyre $P(e)$
- $XP(t,e)$, ha a t út e után következő $e1$ pontjára $P(e1)$

Az A,E,F,G, stb. operátorok a szintaktikus szabályok (11.1. def.) betartásával egymásbaágyazhatóak.

- AFP - P elkerülhetetlen,
- FGP - majdnem mindenütt P, jelölésben: $\overset{\infty}{G}P$

[8] A bizonyításelmélet eldöntési problémájának nevezik azt a feladatot, amely úgy szól, hogy egy adott, tetszőleges formula bizonyítható-e. Az eldöntésprobléma megoldását jelenti az automatikus tételbizonyítás algoritmusának megadása. A modellelmélet eldöntésproblémája az a feladat, amely úgy szól, hogy egy adott, tetszőleges formula érvényes-e [Pász 93].

- GFP - végtelenül gyakran P, jelölésben: $\overset{\infty}{F}P$
- EFP - P lehetséges
- XP - legközelebb P
- $P\mathcal{U}Q$ - P elvezet Q-hoz

Az alábbi definíció használja a 0-ad rendű klasszikus logika formalizált nyelvének atomi formula fogalmát[9]. Jelölje $\mathcal{A}$ az atomi formulák halmazát.

11.1. Definíció. *A CTL* szintaxisának szabályai:*
S1. Minden atomi formula állapotformula.
S2. Ha P és Q állapotformula, akkor $P \wedge Q$ és $\neg P$ is állapotformula.
S3. Ha P útformula, akkor EP állapotformula.
P1. Bármely állapotformula egyben útformula is.
P2. Ha P és Q útformula, akkor $P \wedge Q$ és $\neg P$ is útformula.
P3. Ha P és Q útformula, akkor XP és $P\mathcal{U}Q$ is útformula.

11.1. Megjegyzés. *Az alábbi formulák rövidítések:*
$AP ::= \neg E \neg P$
$P \rightarrow Q ::= \neg P \vee Q$
$FP ::= Igaz\,\mathcal{U}P$

11.2. Definíció. *Az S1,S2,S3,P1,P2,P3 szabályok véges sokszori alkalmazásával generált formulák alkotják a CTL* nyelvet.*

A CTL* szemantikáját az $M = (A,R,L)$ rendezett hármas által definiált időstruktúra felett adjuk meg, ahol A az állapotok halmaza, R bináris reláció az állapotok felett: $R \subseteq A \times A$, $\mathcal{D}_R = A$, L pedig egy címkézés, amely az állapotokhoz atomi formulákat rendel, $L \subseteq A \times \mathcal{A}$.

11.3. Definíció. $R \subseteq A \times B$. *Az* R reláció értelmezési tartománya*:*
$\mathcal{D}_R ::= \{ a \in A \mid \exists b \in B : (a,b) \in R \}$.

11.4. Definíció. $L_R(a_0, \ldots\ldots)$ *az* $R \subseteq A \times A$ *reláció* végtelen pontlánc*a, ha* $\forall i \in N : a_i \in R(a_{i-1})$.

[9] Ha P egy n-változós predikátumszimbólum és $t_1,..t_n$ termek, akkor $P(t_1,...,t_n)$ atomi formula. Az x változószimbólum term. Ha f n-változós függvényszimbólum és $t_1,..t_n$ termek, akkor $f(t_1,...,t_n)$ term. Minden term e két szabály véges számú alkalmazásával áll elő [Pász 93].

11.5. Definíció. *Legyen $n \in N_0$. $L_R(a_0,\ldots,a_n)$ az $R \subseteq A \times A$ reláció* véges pont*lánc*a*, ha $a_n \notin \mathcal{D}_R \wedge \forall i \in [1..n] : a_i \in R(a_{i-1})$.*

*Teljes út*nak nevezzük és x-szel jelöljük az R egy végtelen pontláncát. Az időpillanatok halmazát az R által generált teljes utakon elhelyezkedő pontok, programállapotok alkotják. A címkézés megadja, hogy mely időpillanatban mely atomi formulák igazak. Az időpontok felett értelmezett relációt az R definiálja.

Jelölés: $M,a \models P$, $M,x \models P$, ha az M struktúra a állapotára ill. x teljes útjára teljesül P. Ha a struktúra rögzített, akkor elhagyható a jelölésből. Ha a-t ill. x-et elhagyjuk, akkor bármely állapotra ill. útra teljesül P. x^i az x teljes út suffix-ét jelöli, $x^i ::= x_i, x_{i+1}, \ldots$.

11.6. Definíció. *A CTL* szemantikájának szabályai:*
S1. $a \models P$, ha $P \in L(a)$.
S2. $a \models P \wedge Q$, ha $a \models P$ és $a \models Q$.
$a \models \neg P$, ha nem teljesül $a \models P$.
S3. $a \models EP$, ha van olyan x teljes út, hogy $x_1 = a$ és $a,x \models P$.
P1. $x \models P$, ha $x_0 \models P$ és P állapotformula.
P2. $x \models P \wedge Q$, ha $x \models P$ és $x \models Q$.
$x \models \neg P$, ha nem teljesül $x \models P$.
P3. $x \models XP$, ha $x^1 \models P$.
$x \models (P\mathcal{U}Q)$, ha $\exists i \geq 0 : x^i \models Q$ és $\forall j : 0 \leq j < i : x^j \models P$.

11.7. Definíció. *A P állapotformuláról azt mondjuk, hogy érvényes, ha $\forall M,a : M,a \models P$. P kielégíthető, ha $\exists M,a : M,a \models P$. Ha $M,a \models P$, akkor M modellje P-nek. A P útformuláról azt mondjuk, hogy érvényes, ha $\forall M,x : M,x \models P$.*

Ha a 0-ad rendű klasszikus logika tautológiáiba állapotformulákat helyettesítünk, akkor érvényes formulákhoz jutunk. Érvényesek az alábbi összefüggések is:

- $EFP = P \vee EXEFP$
- $EGP = P \wedge EXEGP$
- $E(P \vee Q) = EP \vee EQ$
- $AFP = P \vee AXAFP$
- $AGP = P \wedge AXAGP$

- $A(P \wedge Q) = AP \wedge AQ$

További érvényes formulákra mutat példákat [Eme Sri 88].

Bevezetjük az egyszerű útkifejezések fogalmát. Egészítsük ki a 11.1. definíció szabályhalmazát az alábbi szabállyal:
„P0. bármely atomi formula útformula.”
Ekkor a P0,P2 szabályok véges sokszori alkalmazásával kapjuk az egyszerű 0-ad rendű formulákat.

11.8. Definíció. *A P0,P2,P3 szabályok véges sokszori alkalmazásával kapjuk az egyszerű útkifejezéseket.*

11.9. Definíció. *Megszorított útkifejezés egy egyszerű útkifejezés, ha minden temporális operátor argumentuma egyszerű 0-ad rendű formula és minden 0-ad rendű formula egy temporális operátor hatáskörében van.*

11.1. Példa. *megszorított útkifejezésre:*
$\neg(P \wedge Q)\,\mathcal{U}(P \rightarrow Q) \wedge (XP \vee F(P \leftrightarrow Q))$

Különböző pártatlan ütemezési feltételeket (3. fejezet) definiálhatunk a bevezetett operátorok, az $exec(j)$ és az $enabled(j)$ atomi formulák segítségével. $exec(j)$ akkor igaz, ha az adott állapotot közvetlenül megelőzően a j. programkomponens került végrehajtásra. Az $enabled(j)$ pedig akkor, ha a megelőző állapotban a j. programkomponens őrfeltétele igaz volt, vagy másképp: a j. programkomponens végrehajtásra kész volt.

11.10. Definíció. *Azt mondjuk, hogy az ütemezés*

- feltétlenül pártatlan, ha $\bigwedge_{j \in J} \overset{\infty}{F} exec(j)$
- gyengén pártatlan, ha $\bigwedge_{j \in J} (\overset{\infty}{G} enabled(j) \rightarrow \overset{\infty}{F} exec(j))$
- szigorúan pártatlan, ha $\bigwedge_{j \in J} (\overset{\infty}{F} enabled(j) \rightarrow \overset{\infty}{F} exec(j))$

11.2. Megjegyzés. *Az $exec(i)$ feltétel ún. atomi élfeltétel. Az élfeltételek szemantikájának megadásához bevezetjük az ún. multiprocessz struktúrákat $M = (A,R,L,L_a)$. A az állapotok halmaza, $R \subseteq A \times A$, $L \subseteq A \times \mathcal{A}$, $\mathcal{A}$ az atomi formulák halmaza, $L_a : \mathcal{B} \longmapsto \mathcal{P}(R)$, ahol $\mathcal{P}(R)$ az R hatványhalmaza, $\mathcal{B} = \{B_1,...,B_m\}$ pedig az atomi élfeltételek véges nem üres halmaza. Kikötjük, hogy $\bigcup\limits_{j\in[1..m]} L_a(B_j) = R$. Jelöljük R_j-vel $L_a(B_j)$-t. Ha B_j azt jelenti, hogy az adott állapotátmenetért a j. folyamat felelős, akkor R_j ezen folyamatot jellemzi (v.ö. 3.2. def.). Kiegészítjük a 11.6. szemantikus szabályokat a*
P0′ : $x \models B_j$, ha $(x_0,x_1) \in L_a(B_j)$
szabállyal, ahol B egy atomi élfeltétel.

Ha csak a *feltétlenül pártatlan ütemezés*nek megfelelő végrehajtási utakra akarunk kikötéseket tenni, akkor a Fair Computation Tree Logic műveleteit kell alkalmazni. A ϕ formula akkor teljesül egy végrehajtási útra, ha minden folyamatra igaz, hogy a végrehajtását azonosító $exec(j)$ atomi élfeltétel az út mentén végtelen sokszor szerepel.

11.11. Definíció.

- $\phi = \forall j \in [1..m] : GF\text{exec}(j)$,
- $A_\phi P = A(\phi \Rightarrow P)$,
- $E_\phi P = E(\phi \wedge P)$

11.3. Megjegyzés. *Ha a struktúra* csak *a feltétlen pártatlan ütemezésnek megfelelő utakat tartalmazza, akkor nincs szükség az A_ϕ, E_ϕ műveletek bevezetésére. Ebben az esetben azonban a 11.6. alakú szemantikamegadás nem lehetséges, ún. általánosított szemantikadefinícióra van szükség [Eme Sri 88]. Általánosított szemantikát egy M=(A,X,L) struktúra felett adhatunk meg, ahol X az utak halmaza. A 11.6. alakú szemantikamegadás akkor lehetséges, ha az utak halmaza előállítható mint egy R reláció véges és végtelen pontláncainak halmaza, azaz az utak halmaza R-generálható. A^{**}-gal jelöljük az A elemeiből képzett véges vagy végtelen sorozatok halmazát. Az $X \subseteq A^{**}$ utak halmaza pontosan akkor R-generálható, ha suffix zárt ($x \in X \Rightarrow x^1 \in XZ$) és fúzió zárt ($a \in A, x_1ay_1, x_2ay_2 \in X, \Rightarrow x_1ay_2 \in X$[10]) és limit zárt ($x_0y_0, x_0x_1y_1, x_0x_1x_2y_2... \in X \Rightarrow x_0x_1x_2... \in X$). Könnyen belátható, hogy a feltétlenül pártatlan ütemezésnek megfelelő utak gráfja nem generálható egy relációval, azaz nem R-*generálható*, mert nem zárt utak egyesítésére.*

[10] x_i állapotok véges, y_j állapotok végtelen sorozatát jelöli.

Az elágazó idejű temporális logika operátorait az alábbi módon szokták még jelölni:

- $\circ P ::= AXP$.
- $\Box P ::= AGP$.
- $\Diamond P ::= \neg \Box \neg$.
  Megj.: $\Diamond$ megfelel EF-nek, tehát lehetőséget fejez ki.
- $\rightsquigarrow P ::= AFP$.
  A $\rightsquigarrow$ tehát nem lehetőséget fejez ki (EF), hanem bizonyosságot!
  Nem ekvivalens $\neg\Box\neg$-tal [Eme Sri 88].

11.1.2. Lineáris temporális logika alapműveletei

Lineáris temporális logika esetén az időstruktúrát egy program által generált sorozatok halmazának tekinthetjük. A sorozatok a lehetséges végrehajtási utak. A program éppen aktuális végrehajtásának megfelelő sorozatra, t-re tehetünk kikötéseket.

- $(P\text{atnext}Q)(t_i) ::=$
  $((\forall j > i : \neg Q(t_i)) \vee (Q(t_k) \wedge P(t_k) \wedge \forall j \in (i,k) : \neg Q(t_j)))$ [Krö 87].
- $\Box P ::= P \wedge (\downarrow \text{atnext} \neg P) \equiv GP$ [Krö 87].
- $\Diamond P ::= \neg \Box \neg \equiv P \vee \neg(\downarrow \text{atnext} P)$ [Krö 87]. (*not never)*
- $\rightsquigarrow P ::= FP$. (*sometimes, eventually)*. Megj.: $\Diamond = \rightsquigarrow$ [Eme Sri 88].
- $P \mathcal{U}_w Q ::= Q\text{atnext}(P \to Q)$ (*weak until*) [Krö 87].

Ha egy program működését akarjuk specifikálni, akkor úgy tekintjük, hogy minden egyes lineáris temporális logikai feltétel elé implicit módon odaírtuk azt is, hogy a feltétel **minden** a kezdőállapotból kiinduló (lehetséges) végrehajtási sorozatra teljesüljön. A P lineáris temporális logikai formulával felírt specifikációt tehát AP alakra fogalmazhatjuk át elágazó temporális logikában, de az A operátor *nem disztributív* [Eme Sri 88]: $(A(FP \vee G\neg P) \not\equiv (AFP \vee AG\neg P))$![11].

Hasonló a helyzet a processzalgebrából ismert $a(b+c) \neq ab+ac$ (időben elágazó szemantikát kifejező) összefüggés temporális logikai megfelelője esetén: $F(a \wedge (Fb \vee Fc)) \not\equiv F(a \wedge b) \vee F(a \wedge c)$ [Ben 88].

[11] „Sometime” is Sometimes „Not Never” (Lamport)

11.2. Leképezések fixpontja

11.2.1. Parciális rendezés, irányított halmaz

11.12. Definíció (Parciális rendezés). *Legyen D egy halmaz, $\leq$ pedig a halmaz felett értelmezett bináris reláció. Ha a $\leq$ reláció reflexív, tranzitív és antiszimmetrikus, akkor* parciális rendezés*nek nevezzük.*

A $d \in D$ elemet *legkisebb elem*nek nevezzük, ha $\forall d' \in D : d \leq d'$. Ha létezik legkisebb elem, akkor az egyértelmű. Legyen $Y \subseteq D$. $d \in D$ az Y felső korlátja, ha $\forall d' \in Y : d' \leq d$. Ha az $Y \subseteq D$ halmaznak létezik legkisebb felső korlátja, akkor az egyértelmű. $d \in D$ az Y alsó korlátja, ha $\forall d' \in Y : d \leq d'$. Ha az $Y \subseteq D$ halmaznak létezik legnagyobb alsó korlátja, akkor az egyértelmű.

11.2.2. Teljes hálók

A $(D, \leq)$ rendezett párt *teljes hálónak* nevezzük, ha a $\leq$ reláció parciális rendezés a D felett és D bármely Y részhalmazának van legkisebb felső és legnagyobb alsó korlátja D-ben.

Egy A alaphalmaz $\mathcal{P}(A)$ hatványhalmaza teljes háló a $\subseteq$ relációra nézve. Az alaphalmaz felett definiált logikai függvényekre is kiterjeszthető a parciális rendezés:

11.13. Definíció (Parciális rendezés logikai függvények felett). *Legyen $P, R \subseteq A \times \mathcal{L}$. $P \leq R$ pontosan akkor, ha $\lceil P \rceil \subseteq \lceil R \rceil$, ahol $\lceil P \rceil$ a P logikai függvény igazsághalmaza (1.10. def.).*

11.4. Megjegyzés (Parciális rendezés logikai relációk felett). *A 11.13. def. kiterjeszthető logikai relációkra is, ebben az esetben a definiált $\leq$ reláció* preorder*, amely egy parciális rendezést generál [Hen 88] .*

11.2.3. Monoton leképezések tulajdonságai, fixpontok

Az $F : R_n(A) \to R_n(B)$ függvény *monoton*, ha $X \subseteq Y \Rightarrow F(X) \subseteq F(Y)$. A továbbiakban F és G jelöljenek monoton függvényeket: $F, G : R_n(A) \longmapsto R_n(A)$.

X-et az F leképezés *fixpontjá*nak nevezzük, ha $F(X) = X$.

11.1. Tétel. *Teljes háló felett minden monoton függvénynek van* legkisebb és legnagyobb fixpontja *[Par 79].*

a) *F legkisebb fixpontja* $\mu Y : F(Y) = \bigcap\{Y | F(Y) \subseteq Y\}$*, (röviden:* μF*),*

b) *fixpont indukció legkisebb fixpontra: ha* $F(Z) \subseteq Z$*, akkor* $\mu F \subseteq Z$*,*

c) $F(\mu F) = \mu F$*,*

d) *F legnagyobb fixpontja:* $\eta X : G(X) = \bigcup\{X | X \subseteq G(X)\}$*, (röviden:* ηG*),*

e) *fixpont indukció legnagyobb fixpontra: ha* $Z \subseteq G(Z)$*, akkor* $Z \subseteq \eta G$*,*

f) $G(\eta G) = \eta G$*.*

12. fejezet

Összefoglalás

A dolgozat párhuzamos programok tervezésének egy olyan matematikai modelljét adja meg, amely kiterjesztése a nemdeterminisztikus szekvenciális programok relációs alapú modelljének [Fót 83] és egyben relációs szemantikai modellje a UNITY logikának [Cha Mis 89]. A modell támogatja párhuzamos programok szintézisét, eszközkészlete bővebb, mint a szekvenciális modellé vagy a UNITY-é.

A programozási feladatok megfogalmazására és megoldására korábban sikeresen alkalmazott módszereket [Dij 76, Fót Hor 91] a dolgozatban ismertetett eredmények felhasználásával párhuzamos programokra is alkalmazhatjuk. A megközelítés funkcionális, más hasonló párhuzamos programozási modellektől eltérően a feladatnak önálló szemantikai jelentése van, így a lépésenkénti finomítás a feladatok és nem a programok felett értelmezett reláció. Az absztrakt program és tulajdonságai a temporális logikával rokon UNITY logikából [Cha Mis 89] ismert programfogalom relációs alapú megfogalmazásai.

A modell definiálja a specifikációs reláció, a programozási feladat, feladat finomítása, kiterjesztése, feltételes értékadás, absztrakt párhuzamos program, feladat kiterjesztése, nyitott specifikáció, utasítás és program kiterjesztése, programtulajdonságok, viselkedési reláció, megoldás fogalomrendszerét. A modell specifikációs eszközeinek kifejezőereje meghaladja a lineáris temporális logika alapú UNITY kifejezőerejét, folyamatok alternatív viselkedésének specifikálására is alkalmas.

Formálisan definiáltuk programok szekvenciáját, bevezettük a UNITY-ből ismert programkonstrukciókat, az uniót és a szuperpozíciót. Kimondtunk levezetési szabályokat, amelyek segítségével a lépésenkénti finomítás során kapott feladatok megoldása után az eredeti feladat megoldását könnyen megadhatjuk.

A modell alapfogalmainak alkalmazását két programozási tétel szintézise során mutattuk be. Az asszociatív művelet eredményeinek párhuzamos kiszámítására levezetett tétel az egyik leggyakrabban előforduló feladatosztály esetére nyújt aszinkron architektúrán is hatékonyan implementálható, verifikált megoldást. Elemenként feldolgozható függvények eredményének párhuzamos kiszámítására alkalmas programozási tételt ismereteink szerint a szakirodalomban elsőként a dolgozatban ismertetett modellben fogalmaztunk meg.

A modell fogalomrendszere alkalmazható párhuzamos programozás oktatására[1]. A fogalomrendszerbe könnyen illeszkedik az üzenetküldés, a szinkron és aszinkron kommunikáció, a csatornaváltozó fogalma [Hor 93-96]. Az adatcsatornás megoldási módszerekre [Cha Mis 89] a modell eszközeivel levezetett programozási tételt [Hor 93-96] is sikeresen alkalmazták a gyakorlatban.

Az eredmények alkalmazása során megfogalmazásra került egy formális modell, amely absztrakt és implementált programok kapcsolatrendszerének leírására alkalmas [Hor 93-96]. Több szakdolgozat foglalkozott azzal a kérdéssel, hogy a modellben hogyan adható meg a típus fogalma, illetve a UNITY modellhez hasonlóan [Sin 91] hogyan specifikálható osztott objektumok viselkedése [Fáb 94, Győr 94].

További kutatást igényel a haladási tulajdonságok és a programkonstrukciók viszonya. Kidolgozandó osztott objektumok viselkedését megadó absztrakt programok szintézisének gyakorlatban is alkalmazható módszertana. A modell nem rendelkezik elegáns eszközökkel valós idejű problémák specifikációjára [Car 94]. Nehézséget okoz a kompozicionalitás biztosítása valós párhuzamosság esetén [Cha 90].

[1] Az asszociatív függvény kiszámításának tételét eddig már kb. 200 hallgató alkalmazta a gyakorlatban is sikeresen konrét feladatok megoldása során.

Függelék

Absztrakt programok megvalósítása C/PVM-ben

Absztrakt programok megvalósításához a PVM szolgáltatásait használhatjuk. A PVM a Parallel Virtual Machine rövidítése, egy köztes réteg, amely az operációs rendszer és a felhasználó program között helyezkedik el. A PVM feladata, hogy programozási nyelvtől és operációs rendszertől független egységes felületet biztosítson elosztott programok komponenseinek együttműködéséhez. A PVM szolgáltatásait egy függvénykönyvtáron keresztül vehetjük igénybe. A könyvtár leglényegesebb elemei a `pvm_mytid`, a `pvm_send`, a `pvm_recv`, a `pvm_spawn` függvények, amelyek segítségével egy folyamat bejelentkezhet a PVM rendszerbe, üzenetet küldhet és fogadhat, illetve folymatot indíthat. A PVM használatához ismernünk kell azt a felületet is, amelyet az operációs rendszer nyújt a programok fordításához, összeszerkesztéséhez, futtatásához. A futtatás előtt össze kell állítanunk azon számítógépek halmazát, amelyen elosztott programunk működni fog. Az alábbiakban a Linux operációs rendszerre jellemző parancsokat mutatjuk be. Először létre kell hoznunk a `pvm3/bin/LiNUX` könyvtárat, ahol a PVM rendszer a futtatható állományokat keresi. A futtatható programot az `aimk` program segítségével állíthatjuk elő egy megfelelő `Makefile` alapján. Helyezzük el a forrászöveget és a make file-t egy alkönyvtárban majd adjuk ki az `aimk` parancsot. A futtatható állományra mutató hivatkozásokat helyezzük el a `pvm3/bin/LiNUX` könyvtárba. Indítsuk el a PVM konzolt a `pvm` paranccsal[2], majd bővítsük az igénybe vett számítógépek halmazát az `add számítógépnév` paranccsal. Végül futtassuk a programot a `spawn -> programnév` utasítással. A konzolt a `halt` utasítással állíthatjuk le.

```
pvm3/src/hello:  hello.c, hello\_other.c, Makefile.aimk
aimk
aimk links
pvm
pvm> spawn -> hello
```

[2] A `pvm -nlocalhost` paranccsal indíthatjuk el a konzolt, ha nincs hálózati összeköttetés más számítógépekkel.

```
pvm> add nyl35
pvm> conf
pvm> halt
```

Példaként bemutatunk egy egyszerű C nyelvű programot, amely PVM egy `printf` függvényhívásban bejelentkezik a PVM rendszerbe és kiírja a saját folyamatazonosítóját a képernyőre, majd elindít egy másik folyamatot (`hello_other`) és üzenetet fogad tőle `pvm_recv`. A kapott üzenetet egész számként értelmezve kicsomagolja és elhelyezi a `num` változóban, majd kiírja a képernyőre.

```
hello.c

#include <stdio.h>
#include "pvm3.h"
int main() {
     int tid;
     int num;
     printf("i'm t%x\n", pvm_mytid());
     pvm_spawn( "hello_other",(char**)0, 0, "", 1, &tid);
     pvm_recv(-1, -1);
     pvm_upkint(&num,1,1);
     printf("from t%x: %d\n", tid, num);
     pvm_exit();
    return 0;
}
```

Az elindított gyernekfolyamat is bejelentkezik a PVM rendszerbe a `pvm_mytid` hívással, majd azonosítva azt őt indító folyamatot (`pvm_parent`) üzenetként elküldi ennek a folyamatnak az ő szonosítóját. Az üzenetküldés három lépésből áll, az üzenetküldő puffer inicializálásából (`pvm_initsend`), az üzenet becsaomagolásából (`pvm_pkint`) és magából az üzenetküldésből (`pvm_send`).

```
hello_other.c

#include "pvm3.h"

int main() {
     int tid = pvm_mytid();
```

```
    int ptid = pvm_parent();
    pvm_initsend(PvmDataDefault);
    pvm_pkint(&tid,1,1);
    pvm_send(ptid, 1);
    pvm_exit();
    return 0;
}
```

Második példánk az asszociatív művelet eredményét kiszámító absztarkt program egy lehetséges C/PVM megvalósítása. A példaprogramban az adatok egész számok, a művelet az összeadás. Az absztrakt program közvetlenül hivatkozhat a *gs* mátrix elemeire, osztott változókat használ. PVM-ben ez nem lehetséges, a folyamatok csak üzenetek útján cserélhetnek információt. Az alábbi megvalósítás a 8.1. ábra minden oszlopához egy-egy folyamatot rendel hozzá, amelyik rendre kiszámítja az oszlop elemeit felülről lefelé. A következő elem kiszámításához mindig szükség van az előző elemre és egy másik oszlopból (egy másik folyamattól) egy további elemre, ha azt már meghatározták. Adatvezérelt megoldást készítünk, azaz nem a szükséges adatok elkérésére kerül sor, hanem az elkészült részeredmények kérés nélkül jutnak el azokhoz a folyamatokhoz, amelyeknek szükségük van rá. Az számítást végző folyamatok elején egész számok küldését és fogadását megkönnyítő segédfüggvények találhatóak. Az *i*. folyamat első lépésben saját és a többi folyamat azonosítóját kapja meg az őt indító szülő folyamattól, majd ciklusban küld részeredményket és fogad adatokat. Végül kiírja az első *i* szám összegét.

assoc.c:

```
#include <stdio.h>
#include <stdlib.h>
#include "pvm3.h"

void sendInt( int to, int mit ){
        pvm_initsend(PvmDataDefault);
        pvm_pkint(&mit,1,1);
        pvm_send(to,0);
}

int recvInt( int from ){
        int data;
```

```
        pvm_recv(from,0);
        pvm_upkint(&data,1,1);
        return data;
}

void main(){
        int tasknum;
        int *tids;
        int id;
        int data;
        int t = 1;

        pvm_mytid();

        pvm_recv(pvm_parent(),0);
        pvm_upkint(&tasknum,1,1);
        tids = (int *)malloc(1+tasknum*sizeof(int));
        pvm_upkint(&tids[1],tasknum,1);
        pvm_upkint(&id,1,1);
        pvm_upkint(&data,1,1);

        while (   ( id+t <= tasknum ) || ( id-t >= 1 )    ) {
                if ( id-t >= 1 )
                        sendInt(tids[id-t],data);
                if ( id+t <= tasknum )
                        data += recvInt(tids[id+t]);
                t<<=1;
        }

        printf("partial sum[%d..%d] =\t%d\n",id,tasknum,data);
        pvm_exit();
}
```

A főprogram a parancssorból olvassa be az *a* vektor elemeit, majd a vektor méretének megfelelő számú folyamatot indít. Egy `for` ciklusban minden gyermekfolyamatot inicializál, elküldve annak saját és a többi folyamat azonosítóját, ill. a vektor megfelelő elemét.

```
#include <stdio.h>
#include <stdlib.h>
#include "pvm3.h"

void main( int argc, char *argv[] ){
        int tasknum = argc-1;
        if (tasknum>0) {
                int i;
                int *tids = (int *)malloc(1+tasknum*sizeof(int));
                pvm_mytid();
                pvm_spawn("assoc",(char **)NULL,
                          "",tasknum,&tids[1]);
                for (i=1;i<=tasknum;i++){
                        int data = atoi(argv[i]);
                        pvm_initsend(PvmDataDefault);
                        pvm_pkint(&tasknum,1,1);
                        pvm_pkint(&tids[1],tasknum,1);
                        pvm_pkint(&i,1,1);
                        pvm_pkint(&data,1,1);
                        pvm_send(tids[i],0);
                }
                pvm_exit();
        } else  fprintf(stderr,
"The numbers are given in the command line!\n");
}
```

Fontosabb tételek és lemmák

Tárgymutató

Irodalomjegyzék

[ALRM 83] U.S. Department of Defense: *The Programming Language Ada, Reference Manual.* American National Standards Institute, Inc. ANSI/MIL-STD-1815A-1983, Lecture Notes in Computer Science, Vol. 155 (Springer, Berlin, 1983).

[And 91] Andrews, G.R.: *Concurrent Programming, Principles and Practice.* (Benjamin/Cummings, Redwood City, 1991).

[Bac Ser 90] Back, R.J.R.-Sere, K.: Stepwise Refinement of Parallel Algorithms. *Science of Computer Programming*, Vol. 13 (1989/90) 133-180.

[Bak War 91] de Bakker, J.W.-Warmerdam, J.H.A.: Four domains for concurrency. *Theoretical Computer Science* Vol. 90 (1991) 127-149.

[Ben 88] Benthem, J.: Time, Logic and Computation. In: *Linear Time, Branching Time and Partial Order in Logics an Models for Concurrency*, Lecture Notes in Computer Science, Vol. 354. (Springer, Berlin, 1989) 1-49.

[Best 83] Best, E.: Relational Semantics of Concurrent Programs. In: *Formal Description of Programming Concepts, II.* (1983) 431-452.

[Car 94] Carruth, A.: *Real-Time Unity*. Technical Report TR94-10, University of Texas at Austin, ftp://ftp.cs.utexas.edu. (March 29, 1994).

[Cha Mis 89] Chandy, K.M.-Misra, J.: *Parallel Program Design: A Foundation.* (Addison-Wesley, 1988, 1989).

[Cha 90] Chandy, K.M.: Reasoning about continuous systems. *Science of Computer Programming*, Vol. 14 (1990) 117-132.

[Col 94] Collette, P.: Composition of assumtion-commitment specifications in a UNITY style. *Science of Computer Programming*, Vol. 23 (1994) 107-125.

[Dij 75] Dijkstra, E.W.: Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *Communications of the ACM*, Vol. 18, Num. 8 (1975) 453-457.

[Dij 76] Dijkstra, E.W.: *A Discipline of Programming.* (Prentice-Hall, 1976).

[Dij Sch 89] Dijkstra, E.W.-Scholten, C.S.: *Predicate Calculus and Program Semantics.* (Springer, New York, 1989).

[Eme Sri 88] Emerson, E.A.-Srinivasan, J.: Branching Time Temporal Logic. In: *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, Lecture Notes in Computer Science, Vol. 354 (Springer, Berlin, 1989) 123-172.

[Fáb 94] Fábián G.: *Párhuzamos algoritmusok specifikációja osztott objektumokat használó rendszerek esetén UNITY módszerrel. I. Osztott bináris fák.* Szakdolgozat, ELTE, Informatika Tanszékcsoport. (Témavezető: Horváth Z.) 1994.

[Flo Suz 81] Flon, L., Suzuki, N.: The Total Correctness of Parallel Programs. *SIAM Journal of Computing*, Vol. 10, No. 2 (May 1981) 227-246.

[Fót 83] Fóthi Á.: *Bevezetés a programozáshoz.* Egyetemi jegyzet (ELTE, TTK, Budapest, 1983).

[Fót 86] Fóthi Á.: Bevezetés a programozáshoz és Programozás c. előadásainak anyaga (1986-1987).

[Fót 88] Fóthi Á.: A Mathematical Approach to Programming. *Annales Uni. Sci. Budapest de R. Eötvös Nom. Sectio Computatorica*, Tom. IX. (1988) 105-114.

[Fót Hor 91] Fóthi Á.- Horváth Z.: The Weakest Precondition and the Theorem of the Specification. In: Koskimies,K.-Räihä,K., ed., *Proceedings* *of the Second Symposium on Programming Languages and Software Tools*, Pirkkala, Finland, August 21-23, 1991, Report A-1991-5, University of Tampere, Department of Computer Science (August, 1991) 39-47.

[Fót Hor 94] Fóthi Á.- Horváth Z.: A Parallel Elementwise Processing. In: Ferenczi Sz.-Kacsuk P., ed., *Proceedings of the 2nd Austrian-Hungarian Workshop on Transputer Applications*, September 29-October 1, 1994, Budapest, Hungary, KFKI-1995-2/M,N Report (1995) 273-282.

[Fót Hor Kozs 95] Fóthi Á.- Horváth Z.- Kozsik T.: Parallel Elementwise Processing – A Novel Version. In: Varga L., ed., *Proceedings of the Fourth Symposium on Programming Languages and Software Tools*, Visegrád, Hungary, June 9-10, 1995 (1995) 180-194. és *Annales Uni. Sci. Budapest de R. Eötvös Nom. Sectio Computatorica* (1996).

[Fót Nyé 90] Fóthi Á.-Nyékyné Gaizler J.: Some Problems of Updating Sequential Files. To appear.

[Fra 86] Franczez, N.: *Fairness.* (Springer, New York, 1986).

[Fro 96] Frohner Á.: Párhuzamos programozást támogató nyelvi eszközök összehasonlítása. Diplomamunka, ELTE, Informatika Tanszékcsoport. (Témavezető: Horváth Z.) 1996.

[Győr 94] Győrffy L.: *Párhuzamos algoritmusok specifikációja osztott objektumokat használó rendszerek esetén UNITY módszerrel. II. Hatványlisták.* Diplomamunka, ELTE, Informatika Tanszékcsoport. (Témavezető: Horváth Z.) 1995.

[Hen 88] Hennessy, M.: *Algebraic Theory of Processes.* (The MIT Press, 1988).

[Hoa 78] Hoare, C.A.R.: Communicating Sequential Processes, *Communications of the ACM* Vol. 21, Num. 8 (1978) 666-677.

[Hoa 85] Hoare, C.A.R.: *Communicating Sequential Processes*. (Prentice-Hall Int., Englewood Cliffs, NJ, 1985).

[Hor 88] Horváth Z.: On-line folyamatirányító szakértői rendszerek fejlesztése. In: Fekete I., ed., *Szakértői rendszrek az ipari folyamatirányításban*, kutatási jelentés, ELTE, TTK, Általános Számítástudományi Tanszék (1988).

[Hor 90] Horváth Z.: Fundamental relation operations in the mathematical models of programming. *Annales Uni. Sci. Budapest de R. Eötvös Nom. Sectio Computatorica*, Tom. X. (1990) 277-298. {MR 92e68113 68Q55 68Q60}.

[Hor 93] Horváth Z.: The Weakest Precondition and the the Specification of Parallel Programs. In: *Proceedings of the Third Symposium on Programming Languages and Software Tools*, Käariku, Estonia, August 21-23, 1993 (1993) 24-33.

[Hor 93-96] Horváth Z.: Párhuzamos programozás alapjai. Jegyzet. Előkészületben. (ftp://augusta.inf.elte.hu/pub/parh) 1993-1996.

[Hor 95] Horváth Z.: Parallel asynchronous computation of the values of an associative function. *Acta Cybernetica*, Vol. 12, No. 1, Szeged (1995) 83-94.

[Hor 95a] Horváth Z.: The Formal Specification of a Problem Solved by a Parallel Program – a Relational Model. In: Varga L., ed., *Proceedings of the Fourth Symposium on Programming Languages and Software Tools*, Visegrád, Hungary, June 9-10, 1995 (1995) 165-179. és *Annales Uni. Sci. Budapest de R. Eötvös Nom. Sectio Computatorica* (1996).

[Hor Koz 94] Horváth Z.- Kozma L.: Parallel Programming Methodology. In: Bogdany J.-Vesztergombi G., ed., *Workshop on Parallel Processing. Technology and Applications.* Budapest, Hungary, 10-11 February, 1994, KFKI-94-09/M,N Report (1994) 57-65.

[Ivá 03] Iványi A.: *Párhuzamos algoritmusok.* ELTE Eötvös Kiadó, 2003.

[Jär 92] Järvinen, H-M.: *The Design of a Specification Language for Reactive Systems.* Thesis for the degree of Doctor of Technology, Tampere University of Technology, Publications 95, Tampere, 1992.

[Jut Kna Rao 89] Jutla, C.S., Knapp, E., Rao, J. R.: A Predicate Transformer Approach to Semantics of Parallel Programs. In: *Proc. 8th Ann. ACM SIGACT/SIGOPS Symposyum on Principles of Distributed Computing*, Edmonton, Alberta, Canada, August 14-16, 1989 (1989) 249-263.

[Kna 90] Knapp, E.: A Predicate Transformer for Progress. *Information Processing Letters*, Vol. 33 (1989/90) 323-330.

[Kna 92] Knapp, E.: Derivation of concurrent programs: two examples. *Science of Computer Programing*, Vol. 19 (Oct. 1992) 1-23.

[Koz 94] Kozma L.: Synthesizing Methods of Parallel Systems. An Overview. In: *Proceedings of μP'94*, Technical University Budapest, Hungary (1994) 586-.

[Koz Var 03] Kozma L., Varga L.: *A szoftvertechnológia elméleti kérdései.* ELTE Eötvös Kiadó, 2003.

[Krö 87] Kröger, F.: *Temporal Logic of Programs.* (Spriger, 1987).

[Lam 77] Lamport, L.: Proving the Correctness of Multiprocess Programs, *IEEE Transactions on Software Engeneering*, Vol. SE-3, No., 2 (March 1977) 125-143.

[Lam 90] Lamport, L.: win and sin: Predicate Transformers for Concurrency. *ACM Transactions on Programming Languages and Systems*, Vol. 12, No. 3 (July 1990) 396-428.

[Lam 91] Lamport, L.: *The Temporal Logic of Actions.* Technical Report SRC Research Number TR79, Digital Equipment Corporation, Systems Research Center, Palo Alto, CA, ftp: gatekeeper.dec.com: pub/DEC/SRC/research-reports (December 1991).

[Lam Lyn 90] Lamport, L.-Lynch, N.: Distributed Computing Models and Methods. In: van Leeuwen, ed., *Handbook of Computer Science*, vol. B (Elsevier, Amsterdam, 1990) 1157-1199.

[Lam Sin 79] van Lamsweerde, A., Sintzoff, M.: Formal Derivation of Strongly Correct Concurrent Programs. *Acta Informatica*, Vol. 12, No. 1 (1979) 1-31.

[Lav 78] Laventhal, M.: *Synthesis of Synchronization Code for Data Abstractions.* Ph.D. Thesis (MIT, 1978).

[Loy Vor 90] Loyens, L.D.J.C.-van de Vorst, J.G.G.: Two Small Parallel Programming Exercises. *Science of Computer Programming*, Vol. 15 (1990) 159-169.

[Luk Sne 92] Lukkien, J., van de Snepscheut J.,L.,A.: Weakest Preconditions for Progress. *Formal Aspects of Computing*, Vol. 4 (1992) 195-236.

[Lyn 02] Lynch, N.,A.: *Osztott algoritmusok.* Kiskapu Kiadó, 2002.

[Maz 89] Mazurkiewicz, A.: Basic Notions of Trace Theory. In: *Linear Time, Branching Time and Partial Order in Logics an Models for Concurrency*, Lecture Notes in Computer Science, Vol. 354. (Springer, Berlin, 1989) 285-362.

[Mil 89] Milner, R.: *Communication and Concurrency.* (Prentice Hall, 1989).

[UN 88-93] Misra, J., et al.: *Notes on UNITY*, 1988-1993., The University of Texas, Austin, ftp://ftp.cs.utexas.edu.

[Mis 01] Misra, J.: *A Discipline of Multiprogramming - Programming Theory for Distributed Applications*, Springer, New York, 2001.

[Mor 87] Morris, J.,M.: A Theoretical Basis for Stepwise Refinement and the Programming Calculus. *Science of Computer Programming*, Vol. 9 (1987) 287-306.

[Mor 90] Morris, J.,M.: Temporal Predicate Transformers and Fair Termination. *Acta Informatica*, Vol. 26, 287-313, 1990.

[Mak Ver 91] Mak, R.H., Verhoeff, T.: Classification of Models, Lecture Notes on Process Models, TU Eindhoven, manuscript, 1991.

[Mel 87] Melliar-Smyth, P.M.: Extending Interval Logic to Real Time Systems, Lecture Notes in Computer Science, Vol. 398 (1987) 224-242.

[Owi Gri 76] Owiczki, S.S., Gries, D.: An Axiomatic Proof Technique for Parallel Programs. *Acta Informatica*, Vol. 6, 319-340, 1976.

[Pac 92] Pachl, J.: A simple proof of a completeness result for leads-to in the UNITY logic. *Information Processing Letters*, Vol. 41 (1992) 35-38.

[Par 79] Park, D.: *On the semantics of fair parallelism* In LNCS 86, pp 504-526. Springer 1980.

[Pász 93] Pásztorné Varga K.: *Logikai alapozás alkalmazásokhoz. Matematikai logika - számítástudomány.* Egyetemi jegyzet, ELTE, TTK (Budapest, 1992).

[Pra 94] Prasetya, I.S.W.B.: Error in the UNITY Substitution Rule for Subscribed Operators. *Formal Aspects of Computing*, Vol. 6 (1994) 466-470.

[Pra 86] Pratt, V.: Modeling Concurrency with Partial Orders. *International Journal of Parallel Programing*, Vol. 15, No. 1 (1986) 33-71.

[Qui 87] Quinn, M.,J.: *Designing Efficient Algorithms for Parallel Computers.* (McGraw-Hill, Inc., 1987).

[Rácz 92] Rácz É.: *A Temporal Logic Specification of a Transaction Manager.* Ph.D. Thesis, ELTE (1992) /in Hungarian/

[Rao 95] Rao, J.,R.: *Extensions of the UNITY Methodology*, Lecture Notes in Computer Science, Vol. 908. (Springer, 1995).

[San 91] Sanders, B.A.: Eliminating the substitution axiom from the UNITY logic. *Formal Aspects of Computing*, Vol. 3 (1991) 189-205.

[Sin 91] Singh, A.,K.: Specification of concurrent objects using auxiliary variables. *Science of Computer Programming*, Vol. 16 (1991) 49-88.

[Tan Ste 02] Tanenbaum, A.,S., van Steen, M.: *Distributed Systems - Proncíples and Paradigms.* Prentice Hall, 2002.

[Var 81] Varga L.: *Programok analízise és szintézise.* (Akadémiai Kiadó, Budapest, 1981).

[WRMP 95] Workgroup on Relational Models of Programming – Fóthi Á. and Fekete I.-Gregorics T.-Horváth Z.- Koncz-Nagy M.-Kozics S.-Nyéky-Gaizler J.-Sike S.-Steingart F.-Tőke P.- Vargyas M.-Venczel T.: Some Concepts of a Relational Model of Programming. In: Varga L., ed., *Proceedings of the Fourth Symposium on Programming Languages and Software Tools*, Visegrád, Hungary, June 8-14, 1995 (1995) 434-446.

Tárgymutató

A dolgozat szerkesztése és szedése
LaTeX-ben készült.