Az ErlVer és a modális μ-kalkulus vizsgálata[1]

Horváth Zoltán
email:hz@inf.elte.hu

2002. december 20.

[1]OTKA T037742

Tartalomjegyzék

Bevezetés

Az OTKA T037742 sz. pályázat célja az, hogy elosztott funkcionális programok helyességének vizsgálatához, (fél)automatikus helyességbizonyításhoz eszközöket hozzon létre. A pályázatban modern, erősen típusos, tisztán funkcionális nyelvekkel foglalkozunk. Ez Erlang nyelv ugyan nem tisztán funkcionális, gyengén típusos, de a nyelv dedikált helyességbizonyító rendszere, az EVT, vagy korábbi nevén az ErlVer segítségével elosztott Erlang programok biztonságossági és haladási tulajdonságai megfogalmazhatók és bizonyíthatók. Az ErlVer specifikációs nyelve a modális μ-kalkulus. A modális μ-kalkulus az elágazó idejű temporális logikából is ismert temporális operátorokat és a legkisebb, illetve legnagyobb fixpont műveletét (μ, ν) használja. A tanulmány célja, hogy ezen fogalmakat, a fogalmak közötti összefüggéseket feltárja és bemutassa. Különösen érdekes számunkra az ErlVer specifikációs nyelvének kifejező ereje azért, hogy ismereteket szerezzünk a Clean nyelv dedikált helyességbizonyító rendszere, a Sparkle specifikációs nyelvének alkalmas kiterjesztéséhez. A tanulmány megértéséhez feltételezzük, hogy az olvasó rendelkezik programozási ismeretekkel, ismeri programnyelvek szemantikájának alapfogalmait és jártas programok formális specifikációjában is.

1. fejezet

A modális μ-kalkulus

1.1. Temporális logika

A temporális logikák a klasszikus logika [Pász93] 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 [Ben88]. Időstruktúráról beszélünk, ha az időpillanatok halmaza felett definiált reláció tulajdonságai megfelelnek az időről alkotott alapvető elképzeléseinknek, azaz a reláció[4]

[1] Az időpont fogalmát absztrakt értelemben használjuk. Nem foglalkozunk az időpontok halmaza felett metrika definiálásával.

[2] A temporális logikák a modális logikák körébe tartoznak [Rácz92]. 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. 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

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 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. *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. 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 [EmeSri88] 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 elágazó idejű temporális logika operátorait.[5]

Az egyes temporális logikák között a leglényegesebb különbség a kifejezőerőben van. Általában minél nagyobb a logika kifejezőereje, annál bo-

[5]Egy rögzített program programállapotai által definiált időstruktúra esetén egy P formula az állapottér felett értelmezett logikai függvényt definiál. Az alábbi temporális logikai műveletek segítségével tehát rögzített S absztrakt program esetén P, Q logikai függvényekből új logikai függvényeket konstruálhatunk.

nyolultabb a logika eldöntési problémája.[6]

1.1.1. Elágazó idejű temporális logika

Az elágazó idejű temporális logikában az időpillanatok halmaza felett olyan parciális rendezés (1.12. 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 [EmeSri88] 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 (1.1., 1.6. 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 [EmeSri88].

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 (1.1. def.) betartásával egymásbaágyazhatóak.

- AFP - P elkerülhetetlen,

[6] 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 [Pász93].

- FGP - majdnem mindenütt P, jelölésben: $\overset{\infty}{_G}P$
- 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 az elsőrendű klasszikus logika formalizált nyelvének atomi formula fogalmát[7]. Jelölje $\mathcal{A}$ az atomi formulák halmazát.

1.1. 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.

1.1. Megjegyzés. *Az alábbi formulák rövidítések:*
$AP ::= \neg E \neg P$
$P \rightarrow Q ::= \neg P \vee Q$
$FP ::= \uparrow \mathcal{U} P$

1.2. 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, $L \subseteq A \times \mathcal{A}$.

1.3. Definíció. $R \subseteq A \times B$. *Az R* reláció értelmezési tartománya*:*
$\mathcal{D}_R ::= \{ a \in A \mid \exists b \in B : (a, b) \in R \}$.

[7] 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 [Pász93].

1.4. Definíció. $L_R(a_0, \ldots\ldots)$ *az* $R \subseteq A \times A$ *reláció* végtelen pontlánc*a, ha* $\forall i \in N : a_i \in R(a_{i-1})$.

1.5. Definíció. *Legyen* $n \in N_0$. $L_R(a_0, \ldots, a_n)$ *az* $R \subseteq A \times A$ *reláció* véges pontlánc*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, $x^i ::= x_i, x_{i+1}, \ldots$.

1.6. 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* $\forall j : 0 \leq j < i : x^j \models P$.

1.7. 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, amelyek alapján temporális logikai műveletek fixpont műveletekkel történő meghatározása is lehetséges (1.3. fejezet):

1.1. Lemma. .

- $EFP = P \vee EXEFP$

- $EGP = P \wedge EXEGP$
- $EX(P \vee Q) = EXP \vee EXQ$
- $AFP = P \vee AXAFP$
- $AGP = P \wedge AXAGP$
- $A(P \wedge Q) = AP \wedge AQ$

További érvényes formulákra mutat példákat [EmeSri88].

Bevezetjük az egyszerű útkifejezések fogalmát. Egészítsük ki a 1.1. 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.

1.8. Definíció. *A P0,P2,P3 szabályok véges sokszori alkalmazásával kapjuk az egyszerű útkifejezéseket.*

1.9. 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.*

1.1. Példa. *megszorított útkifejezésre:*
$\neg(P \wedge Q)\ \mathcal{U}\ (P \rightarrow Q) \wedge (XP \vee F(P \leftrightarrow Q))$

Különböző pártatlan ütemezési feltételeket ([EmeSri88, Hor96]) 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.

1.10. Definíció. *Azt mondjuk, hogy az ütemezés*

- feltétlenül pártatlan, ha $\bigwedge_{j \in J} {}^{\infty}_{F}\, exec(j)$
- gyengén pártatlan, ha $\bigwedge_{j \in J} ({}^{\infty}_{G}\, enabled(j) \rightarrow {}^{\infty}_{F}\, exec(j))$

- szigorúan pártatlan, ha $\bigwedge_{j \in J} (\overset{\infty}{F} enabled(j) \rightarrow \overset{\infty}{F} exec(j))$

1.2. 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* $\bigcup_{j \in [1..m]} L_a(B_j) = R$. *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. Kiegészítjük a 1.6. 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.

1.11. Definíció.
- $\phi = \forall j \in [1..m] : GF \text{ exec}(j)$,
- $A_\phi P = A(\phi \Rightarrow P)$,
- $E_\phi P = E(\phi \wedge P)$

1.3. 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 1.6. alakú szemantikamegadás nem lehetséges, ún. általánosított szemantikadefinícióra van szükség [EmeSri88]. Általánosított szemantikát egy M=(A,X,L) struktúra felett adhatunk meg, ahol X az utak halmaza. A 1.6. 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* ($a \in A, x_1 a y_1, x_2 a y_2 \in X, \Rightarrow x_1 a y_2 \in X$[8]) *és limit zárt* ($x_0 y_0, x_0 x_1 y_1, x_0 x_1 x_2 y_2 ... \in X \Rightarrow x_0 x_1 x_2 ... \in X$). *Könnyen belátható, hogy a feltétlenül pártatlan ütemezésnek megfelelő utak gráfja nem generálható egy relációval, azaz nem* R-generálható, *mert nem zárt utak egyesítésére.*

[8] 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 [EmeSri88].

1.1.2. 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.

- $(P \text{ atnext } Q)(t_i) ::=$
  $((\forall j > i : \neg Q(t_i)) \vee (Q(t_k) \wedge P(t_k) \wedge \forall j \in (i,k) : \neg Q(t_j)))$ [Krö87].
- $\Box P ::= P \wedge (\downarrow \text{ atnext } \neg P) \;\equiv\; G\ P$ [Krö87].
- $\Diamond P ::= \neg\, \Box\neg \equiv P \vee \neg(\downarrow \text{ atnext } P)$ [Krö87]. *(not never)*
- $\rightsquigarrow P ::= F\ P$. *(sometimes, eventually).* Megj.: $\Diamond = \rightsquigarrow$ [EmeSri88].
- $P\ \mathcal{U}_w\ Q ::= Q \text{ atnext } (P \rightarrow Q)$ *(weak until)* [Krö87].

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* [EmeSri88]: $(A(FP \vee G\neg P) \not\equiv (AFP \vee AG\neg P))$![9].

Hasonló a helyzet a processzalgebrából ismert $a(b+c) \neq ab+ac$ (időben elágazó szemantikát kifejező) összefüggés temporális logikai megfelelője esetén: $F(a \wedge (Fb \vee Fc)) \not\equiv F(a \wedge b) \vee F(a \wedge c)$ [Ben88].

[9] „Sometime" is Sometimes „Not Never" (Lamport)

1.2. Leképezések fixpontja

1.2.1. Parciális rendezés, irányított halmaz

1.12. 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.

1.2.2. Teljes hálók

A $(D, \leq)$ rendezett párt *teljes háló*nak 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:

1.13. 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.*

1.4. Megjegyzés (Parciális rendezés logikai relációk felett). *A 1.13. def. kiterjeszthető logikai relációkra is, ebben az esetben a definiált $\leq$ reláció* preorder*, amely egy parciális rendezést generál .*

1.2.3. 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 *fixpontj*ának nevezzük, ha $F(X) = X$.

A Knaster-Tarski tétel szerint teljes háló felett minden monoton funkcionálnak egyértelműen létezik legkisebb és legnagyobb fixpontja.

1.2. Tétel. *Teljes háló felett minden monoton függvénynek van* legkisebb és legnagyobb fixpontja *[Par79].*

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: $\nu X : G(X) = \bigcup\{X | X \subseteq G(X)\}$, *(röviden:* νG*),*

e) fixpont indukció legnagyobb fixpontra: ha $Z \subseteq G(Z)$, *akkor* $Z \subseteq \nu G$,

f) $G(\nu G) = \nu G$.

1.14. Definíció (Folytonosság). *Egy F leképezés*

$\cup$ *-folytonos, ha minden monoton növő láncra megőrzi a legkisebb felső korlátot, azaz* $\forall\ P_1 \subseteq P_2 \subseteq \ldots : F(\bigcup_i P_i) = \bigcup_i F(P_i)$,

$\cap$ *-folytonos, ha minden csökkenő láncra megőrzi a legnagyobb alsó korlátot, azaz* $\forall\ P_1 \supseteq P_2 \supseteq \ldots : F(\bigcap_i P_i) = \bigcap_i F(P_i)$,

Ha egy F leképezés $\cup$-folytonos, akkor a leképezés legkisebb fixpontja $\mu Z : F(Z) = \bigcup_{i \in N} F^i(Hamis)$, ill. ha egy F leképezés $\cap$-folytonos, akkor a leképezés legnagyobb fixpontja $\nu Z : F(Z) = \bigcap_{i \in N} F^i(Igaz)$.

(Ha a leképezés nem folytonos de monoton, akkor tovább kell lépni a határrendszámokra [Mor90].)

1.3. Fixpontos temporális logika

A CTL műveleteit az 1.1. összefüggések alapján leírhatjuk, mint monoton leképezések legkisebb, illetve legnagyobb fixpontjait [Em90].

Ha minden állapotnak csak véges sok rákövetkezője van (nincs végtelen nemdeterminisztikusság), akkor a fixpont műveletek hatáskörében álló alábbi leképezések $\cap, \cup$ folytonosak is.

$EFP = \mu Z.P \vee EXZ$
$AGP = \nu Z.P \wedge AXZ$
$AFP = \mu Z.P \vee AXZ$
$EGP = \nu Z.P \wedge EXZ$

Figyeljük meg, hogy a legnagyobb fixpont kiszámításának műveletét az (AG) operátor, a "mindig igaz" [San91, Hor96] meghatározásánál, a legkisebb fixpontot pedig az (AF), az elkerülhetetlenséget kifejező temporális logikai operátor megadásakor használjuk.

Haladási tulajdonságok legkisebb fixpontokkal, biztonságossági tulajdonságok legnagyobb fixpontokkal jellemezhetőek (ha az ütemezésnél pártatlansági kikötéseket nem veszünk figyelembe).

Tfh. $\tau(Z) = P \vee EXZ$ folytonos (véges nemdeterminisztikusság mellett), ekkor $\mu Z.\tau(Z) = EFP$ meghatározható, mint $\bigcup_{i \in N} \tau^i(Hamis)$. Elegendő azt belátni, hogy $\forall i \in N : (\tau^i(Hamis))(s)$, ha van i-nél nem hosszabb olyan út s-ből valamely olyan t-be, amelyre $P(t)$. Ezt az elvet használják a modell ellenőrző rendszerek (model checker).

1.4. A modális μ-kalkulus

Felépíthetünk egy ítélet alapú (0-ad rendű, propositional) modális logikát. Elemei az atomi $P, Q, ...$ predikátumok, az $Y, Z, ...$ predikátumváltozók, a $\vee, \wedge, \neg$ logikai műveletek, az AX, EX temporális operátorok megfelelői ([]=wp, <>=wpa [Hor96]), és a μ, ν fixpont műveletek. Megköveteljük, hogy a formulák szintaktikusan monotonak legyenek, azaz fixpont operátor kötésében álló predikátumváltozóra csak páros számú tagadás vonatkozhat.

Az így kapott ítélet alapú (propositional) μ-kalkulus ($L_\mu, L_{\mu+}$ [Koz83]) kifejezőerőben meghaladja a FairCTL-t és a CTL*-t is.

1.4.1. Az L_μ és az $L_{\mu+}$

Az $L_{\mu+}$ nyelv az L_μ nyelv kiterjesztése oly módon, hogy monoton műveletek α-szoros kompozíciója is megengedett (ahol α egy rendszám).

Az $L_{\mu+}$ nyelv elemei: 0 (Hamis), 1 (Igaz), P, Q, ... ítéletkonstansok; X, Y, ... ítéletváltozók; a, b, ... programkonstansok, p, q, ... formulák. A formulákat induktívan definiáljuk:

$X, P, p \vee q, \neg p, < a > p, \alpha X.pX, \mu X.pX,$

ahol pX pozitív az X-ben. (Kvantorok: α, μ. X minden szabad előfordulása páros számú tagadás hatáskörében van. $\alpha X.pX = p^\alpha(0)$)

Az $L_{\mu+}$ nyelv szemantikája: legyen M=(S,I) egy struktúra, ahol S az állapotok halmaza, I az ítéletkonstansok és a programkonstansok interpretációja. Az ítéletkonstansokat az S részhalmazainak (igazsághalmaz), a programokat az S felett értelmezett bináris relációknak feleltetjük meg (hatásreláció). Megköveteljük, hogy $I(0) = \emptyset, I(1) = S$.

Egy értéklés az S részhalmazait rendeli hozzá az ítéletkonstansokhoz. Egy p formula jelentése a benne szereplő szabad változók értékelésétől függ (p^M az értékelésekől az S részhalmazaira képez). $p(\overline{X})$-szel jelöljük, hogy p szabad változói a $\overline{X} = X_1, ..., X_n$ halmaz elemei. $p^M(\overline{A})$ jelöli p^M értékét, ha az értékelés rendre az $\overline{A} = (A_1...A_n)$ értékeket rendeli az $X_1, ..., X_n$ változókhoz. p^M-et induktívan definiáljuk:

$X_i^M(\overline{A}) = A_i$,
$P^M(\overline{A}) = I(P)$,
$(p \vee q)^M(\overline{A}) = p^M(\overline{A}) \cup q^M(\overline{A})$,
$(\neg p)^M(\overline{A}) = S \setminus p^M(\overline{A})$,
$(< a > p)^M(\overline{A}) =< a^M > (p^M(\overline{A}))$, ahol
$< a^M > (B) = \{s | \exists t \in B : (s,t) \in I(a))\}$ ($I(a)^{(-1)}$,wpa(a)).

Jelöljön pX egy X-ben pozitív formulát. pX X-ben monoton az igazsághalmazokon értelmezett részhalmaz relációra nézve. $\overline{X}$ a formula többi szabad változója.

$0X.pX^M(\overline{A}) = 0^M = 0$,
$(\alpha + 1)X.pX^M(\overline{A}) = p^M(\alpha X.pX^M(\overline{A}), \overline{A})$,
$\delta X.pX^M(\overline{A}) = \bigcup_{\beta<\delta} \beta X.pX^M(\overline{A})$, ha δ határrendszám,
$\mu X.pX^M(\overline{A}) = \bigcap_{\beta} X.pX^M(\overline{A})$.

Rögzített $\overline{A}$ kiértékelés mellett $p^M(X, \overline{A})$ monotonitása miatt, nagyobb rendszámra nagyobb vagy egyenlő értéket kapunk. Az első olyan rendszámot, amelyre kapott érték egyenlő a rákövetkezővel lezáró rendszámnak nevezzük. Ez az érték a legkisebb fixpont.

$[a]p = \neg < a > \neg p$, (biztosan p, wp(a) [Hor96]),
$([a]p)^M(\overline{A}) = [a^M](p^M(\overline{A}))$,
$[a^M](B) = \{s | \forall t(s,t) \in I(a) \rightarrow t \in B\}$.
$\nu X.pX = \neg\mu X.\neg p\neg X$, a legnagyobb fixpont.
σ-val jelöljük az általános fixpont-operátort amely lehet μ is, és ν is.

Ha minden változó csak egyszeresen kvantált, akkor a formula pozitív normál formában van.

Formula lezártja az a legkisebb formulahalmaz, amelyet úgy kapunk, hogy a formulában a szabad változók helyére fixpontoperátorral kötve behelyettesítjük a formulát magát és az így kapott formulának vesszük az összes valódi részformuláját.

1.4.2. Egy teljes levezetési rendszer

Jelölje $p \equiv q$ két formula egyenlőségét (ekvivalenciáját), $p \leq q ::= p \vee q \equiv q$. Egyenlő formulák egymás helyébe helyettesíthetőek, ha a μ formulákra tett

szintaktikai megszorítások nem sérülnek meg. Alkalmazhatóak a Boole-algebra aximómái és a modális ítéletkalkulus axiómái:

$< a > X \vee < a > Y \equiv < a > (X \vee Y)$,

$< a > X \wedge [a]Y \leq < a > (X \wedge Y)$,

$< a > 0 \equiv 0$,

$p(\mu X.pX) \leq \mu X.pX$,

$pY \leq Y \Rightarrow \mu X.pX \leq Y$, ahol Y nem fordul elő pX-ben.

Konzisztens egy formula, ha $\not\vdash p \equiv 0$. Az utolsó axióma Park fixpont indukciós szabály [Par79].

Alapvető tételek:

$\mu X.pX \equiv \mu Y.pY$,

$pX \leq qX \Rightarrow \ \sigma X.pX \leq \sigma X.qX$,

Monotonicitás: $q \leq r \Rightarrow p(q) \leq p(r)$, ha pX pozitív X-ben,

$p(\sigma X.pX) \equiv \sigma X.pX$,

$\mu X.q \equiv q$, ha X nem szabad q-ban,

$p(\mu X.q \wedge pX) \leq q \equiv \mu X.pX \leq q$.

1.5. Gentzen kalkulus

Az ErlVer, sok más helyességbizonyító rendszerhez hasonlóan, Gentzen-stílusú kalkulust használ.

A Gentzen kalkulus alapfogalma a szekvent [PászVár03]. A szekvent véges formulahalmazok rendezett párja, jele: (Γ, Δ), vagy $\Gamma \vdash \Delta$. Formulahalmazokat gyakran bővítünk egy-egy formulával, ill. egyesítünk. Mindkét műveletet ,-vel jelöljük, pld.: $\Gamma, \Delta ::= \Gamma \cup \Delta$, $\Gamma, A ::= \Gamma \cup \{A\}$.

Egy szekvent teljesül, ha a szekvent baloldali formulahalmazának minden formulája igaz, akkor a jobboldali formulahalmaz legalább egyik formulája is igaz. A levezetési szabályok szekventeket tartalmaznak, általában vonallal elválasztva. Jelentésük: ha a vonal feletti szekvent(ek) levezetése ismert, akkor a vonal alatti szekvent levezetési is megkonstruálható. A levezetési szabályok a levezetendő szekventben szereplő formulák szerkezete szerint választhatók ki. Példaként bemutatjuk a vágási szabályt:

$$\frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}$$

2. fejezet

Az Erlang nyelv és az ErlVer

2.1. Az Erlang nyelv

Az Erlang [Arm96, Erl] nem tisztán funkcionális nyelv[1], az Ericsson cég alkalmazza távközlési rendszerek programozására, valós-idejű feladatok megoldására[2]. Ezen programok megbízhatósága alapvető követelmény, így a nyelvhez készítettek egy dedikált helyességbizonyító rendszert: az EVT-t (Erlang Verification Tool), vagy korábbi nevén az ErlVer-t, amely sok szempontból meghaladja más eszközök képességeit [EVT2000].

Az Erlang nyelvnek statikus típusrendszere nincs, nagyon gyengén típusos. Alaptípusai: egész, lebegőpontos, atom. Az atomok körébe tartoznak a felsorolási típusok és a string típus is, az Erlang megközelítésében ezek mind névvel ellátott konstansok. Típuskonstrukciók: rendezett pár ({a,b}), lista ([a,1,'b',c|xs])[3]. A nyelv megengedi a mintaillesztést függvéynhívások esetén és összetett minták használatát definíciók baloldalán
(`C, [Head,Tail]} = {{222,man},[a,b,c]}`).

Erlangban a kiértékelési sorrend nem definiált. Ha a kiértékelendő kifejezésben nincsenek mellékhatással járó összetevők (üzenetküldés, folyamatindítás stb.), akkor a kiértékelési sorrendtől nem függ az eredmény (ha a kiértékelés terminál).

Megengedett függvényazonosítók túlterhelése, ha az argumentumok száma különböző. A függvények azonosítása `fv_azonosító/argumentum_szám`

[1] Az Erlang program változói legfeljebb egyszer kaphatnak értéket. Definiálatlan értékű változó nem szerepelhet kifejezésekben.

[2] A nyelv támogatására egy elosztott infrastruktúrát is (middleware) létrehoztak Open Telecom Platform néven [EVT2000].

[3] A legtöbb funkcionális nyelvtől eltérően Erlangban egymástól különböző "típusú" értékek is lehetnek ugyanabban a listában.

alakú. Az Erlang program modulokból áll, más modulból csak az `export` listán szereplő függvények használhatóak.

Erlangban folyamatok (`proc`) definiálhatóak. A folyamatok olyan azonosítóval és üzenetsorral rendelkező objektumok, amelyek egy funkcionális kifejezés kiértékelését végzik el.

A `self` függvény az éppen futó folymat azonosítóját adja meg, a `spawn(m,fn,args)` segítségével pedig új folyamtokat hozhatunk létre, amely az `fn(args)` kifejezést értékeli ki az `m` modulban megadott definíciók szerint. A `spawn` függvény értéke a létrehozott folyamat azonosítója (`pid`). A `spawn` függvénynek olyan változata is rendelkezésre áll, amelyik egy másik processzoron (node) indítja el az új folyamatot.
`pid = spawn('hz@erlang_node.inf.elte.hu',modulnév,fn,args).`
Távoli csúcspontok állapotáról a monitorozási funkció segítségével kaphatunk információt: `monitor_node(node,true)`.

A vessző a szekvenciális kompozíció jele `e1,e2`, a kifejezés értéke a második kifejezés értéke, az első kifejezés kiértékelésének eredménye kizárólag a kiértékelés mellékhatása.

Az `if g1 -> seq1; ... gn -> seqn; end.` alkalmazásával logikai feltételek használatával definiálhatunk esetszétválasztást. `case` kifejezés segítségével padig mintaillesztést alkalmazhatunk:
`case e of p1 when g1 -> e1; ...`, ahol `g1` egy őrfeltétel.
`pid!e2` az aszinkron üzenetküldés,
`receive {pid,m1} - > ...; m2 -> ...; AnyMessage -> ...; end` az üzenetfogadás nyelvi eleme, ahol a kifejezést kiértékelő folyamat `q` üzenetsorából az első olyan üzenet kerül kiválasztásra, amely megfelel valamelyik mintának (read(q,v)). A mintában megadhatjuk azt is, hogy kitől várjuk az üzenetet. Ha nem adunk meg időkorlátot (`after timeout -> ...`) és nincs ilyen üzenet, addig a `reveive` kiértékelése felfüggesztődik. Az üzenetküldés értéke az elküldött, `receive` értéke pedig a fogadott üzenet. Az üzenetküldés és fogadás ugyanúgy alkalmazható távoli folyamatok között elosztott rendszerben, mint azonos processzoron futó konkurrens folyamatok között[4]. A `spawn_link` függvény használatával úgy hozhatunk létre új folyamatot, hogy egy kétirányú csatorna is létrejön. Üzenetküldéskor a folyamat azonosítója helyett ezen kétirányú csatorna azonosítóját adhatjuk meg.

Az Erlang lehetőséget ad arra is, hogy olyan folyamattal is üzenetet tudjunk váltani, amelynek nem ismerjük az azonosítóját.

[4]A kommunikációt az Open Telecom Protocol (OTP) [EVT2000] middleware támogatja.

A `register(folyamat_név,pid)`, `unregister(név)`, `pid=whereis(név)`, `list_of_registered=registered()` függvények folyamatnevek és azonosítók egymáshozrendelését és lekérdezését támogatják egy reisztrációs rendszer alkalmazásával.

Konkurencia esetén az Erlang a folyamatok ütemezésénél megköveteli a pártatlanságot (ld. 1.11 def.) abban az értelemben, hogy minden olyan folyamat, amelyik képes arra, hogy haladjon az lehetőséget is kap erre. Egyes folyamatok esetén a programozó előírhatja, hogy azok prioritása (`process_flag(priority, prioritás_érték)`) magasabb legyen a többinél. Magasabb prioritású folyamatok gyakrabban kapnak lehetőséget arra, hogy fussanak. Az Erlang megengedi azt is, hogy modulok kódját futás közben dinamikusan szerkesszük be (`load_module(modul,kód_objektum)`, a Clean `read_dynamic`-tól [Pil98, Hor99] eltérő szemantikával, de hasonló céllal.

Példaként bemutatjuk az `echo` folyamat Erlang nyelvű megvalósítását:

```
-module(echo).
-export([start/0,loop/0]).

start() ->
   spawn(echo,loop,[]).

loop() ->
   receive
      {From, Message} ->
        From ! Message,
        loop()
   end.
```

Az `echo:start()` függvény kiértékelésekor a `spawn` hatására a hívó függvénnyel párhuzamosan kerül kiértékelésre az `echo` függvény. Az alábbi példában a hívó saját folyamatazonosítóját küldi el és kapja vissza.

```
...
Id = echo:start();
Id ! {self(), hello);
...
```

2.1.1. Core Erlang

Az Erlang nyelv nyelvi elemeinek egy részhalmazát magában foglaló Core Erlang nyelven írt programok helyességének elemzésére alkalmas az ErlVer[5].

A Core Erlang nem tartalmaz például kommunikációs őrfeltételeket, kivételkezelést, modulokat stb. A Core Erlang szintaxisát az alábbi nyelvtannal definiáljuk. v egy értéket jelöl, V egy változót, e egy kifejezést, p mintát, m egy mintaillesztést, op pedig egy adatkonstruktort, konstanst, vagy egyszerű műveletet.

```
e ::= V | self | op(e1,...,en) | e1 e2 | e1,e2 |
     case e of m | spawn(e1,e2) | receive m end | e1!e2
m ::= p1 -> e1; ... ; pn -> en
p ::= op(p1,...,pn) | V
v ::= op(v1,...,vn)
```

A Core Erlang műveleti szemantikájának bemutatásakor szükség lesz néhány további jelölésre is:

`eps` az üres sor, `q@v` a `q`-sort kiegészíti a `v` elemmel. || jelöli a konkurens végrehajtás során létrejövő összetett rendszerállapotot, `,` pedig a szekvenciális kompozíciót.

Megjegyezzük, hogy az Erlanggal ellentétben Core Erlang-ban a kiértékelési sorrend rögzített, ez nagyban megkönnyíti a műveleti szemantika leírását.

2.2. Az ErlVer

Funkcionális nyelven írt programok helyessége könnyebben bizonyítható, mint a jelenleg általánosan elterjedt imperatív nyelveken írt programoké [Tho90, Hor99, HoTePá02, MolEek99, ButDowStr02]. Tisztán funkcionális nyelvekben érvényes az ún. hivatkozási átlátszóság, így számos programtulajdonság matematikai indukcióval bizonyítható. A modern funkcionális nyelvek (Haskell, Clean, stb.) a hivatkozási átlátszóság megsértése nélkül le tudják írni bonyolult elosztott rendszerek viselkedését, állapotváltozásait. Ezért ezen rendszerek biztonságossági tulajdonságainak, invariánsainak bizonyítása és ellenőrzése könnyebben elvégezhető.

[5]Újabban már a nyelv egy bővebb részhalmazának, az Erlang-F-nek is sikerült megadni a formális műveleti szemantikáját, így ennek elemeire alkalmazhatjuk az ErlVer/EVT-t.

Az ErlVer nyitott, elosztott rendszereket megvalósító Core Erlang nyelvű programok helyességének bizonyítását támogatja. Az ErlVer egy Gentzen stílusú (1.5. fejezet) kompozícionális helyességbizonyító rendszer. Célvezérelt rendszer, a bizonyítandó állításból visszafelé indulva építi a bizonyítási fát. Az esetlegesen előálló ciklikus következtetési láncok megszakítására is tartalmaz levezetési szabályt. A kompozícionalitást az biztosítja, hogy szokásos levezetési szabályokat kiegészíti az összetett rendszerek vágási szabálya, amelynek hipotézise a részrendszerekre vonatkozó feltételeket tartalmaz, a következmény pedig az összetett rendszer tulajdonságait írja le. A bizonyítás során figyelembe veszi a Core Erlang forrásszöveg által meghatározott állapotátmeneteket, a strukturális műveleti szemantikára épülő levezetési szabályokat is alkalmazva. Az ErlVer specifikációs nyelve a nyitott specifikációk megfogalmazására is alkalmas modális μ-kalkulus néhány Erlang nyelvi elemre vonatkozó predikátummal kiterjesztve. A tulajdonságok leírásához használhatóak a legkisebb és legnagyobb fixpont operátorok, a temporális logika alapműveletei és az Erlang nyelvre vonatkozó speciális atomi predikátumok (pl. a CSP-ből átvett jelölések a szinkron kommunikációra (?,!)). A helyességbizonyító nagymértékben automatizált. Egy vezérlő nyelv segítségével a bizonyítási lépések programozhatóak.

A rendszer képes arra, hogy eltárolja a bizonyítási lépéseket és az eltárolt bizonyítást újra végrehajtsa. Eme lehetőség segítségével elegendő minden bizonyítást csak egyszer végrehajtanunk, a következő alkalommal pedig, amikor szükségünk van ugyanazon állítás bizonyítására elegendő csak az eltárolt lépések végrehajtásával ellenőrizni, hogy az előzőleg elvégzett bizonyítás valóban igazolja-e az állításunkat.

Az ErlVer bonyolult állítások kezelésére is alkalmas, míg a Sparkle jelenleg csak viszonylag egyszerűbb leírások értelmezésére képes. Az ErlVer nagymértékben automatizált, a bizonyító rendszer SML alapú [Har01].

2.2.1. Az ErlVer specifikációs nyelve

Az ErlVer specifikációs nyelve az elsőrendű modális μ-kalkulus (ld. 1.4.1. alfejezet).

Egy s állapotra teljesül az $< \alpha > \Phi$ formula, ha van α rákövetkezői között olyan állapot, amelyre teljesül Φ. A rákövetkezési relációt az Erlang műveleti szemantikájának segítségével adjuk majd meg. Egy s állapotra teljesül az $[\alpha]\Phi$ formula, ha α rákövetkezői között minden állapotra teljesül Φ. Formulákban használhatjuk a legkisebb és legnagyobb fixpont operátorokat, amelyekkel haladási, ill. biztonságossági tulajdonságokat fejezhetünk ki.

Fixpont operátrorok egymásba ágyazásával pártatlansági feltételeket is kezelhetünk [Mor90, Par79, Rao95, Hor96, Kna90, JutKnaRao89].

A specifikációs nyelvet induktív nyelvtan segítségével definiálhatjuk:

```
F ::= tt | ff | T = T | F /\ F | F => F | F \/ F | not F |
      forall Var : Type . F | exists Var : Type . F
      | lamdaVar:Type.F | F T | T : F
      | [Action] F | <Action> F
PredicateDef ::= Name : PropType DefSymbol F
PropType ::= prop | Type -> PropType
DefSymbol ::= => | <= | =
```

Az [Action], ill. ¡Action¿ modális operátorok a műveleti szemantika átmenetrelációjának megfelelő átmenetekre hivatkoznak (2.2.2 fejezet).

Az `=>` jellel a legnagyobb, az `<=` jellel a legkisebb fixpontot definiálhatjuk ($X => F(X)$ megoldása $\nu X.F(X)$, $X <= F(X)$ megoldása $\mu X.F(X)$). Az `=` jelet csak olyan definíciókban használjuk, amelyek nem rekurzívak. Rekurzív kötésben álló predikátumokra csak páros számú negáció vonatkozhat a leképezés monotonitása érdekében.

Az `T:F` állítás azt fejezi ki, hogy `T` típusa `F`. Alaptípusok:
`erlanValue, erlangExpression, werlangSystem, werlangAction`.

Az Erlang nyelvhez kötődő speciális predikátumok például a következőek:

term(pid) = e - akkor teljesül, ha a pid azonosítójú folyamat által kiértékelt kifejezés e,

queue(pid) = q - akkor teljesül, ha a pid azonosítójú sorozathoz tartozó sor q.

A specifikációs nyelv kompozícionális leíró szemantikáját a 1.4.1 fejezetben megismert módon adhatjuk meg formálisan [Dam98] az egyes formulák értékét tetszőleges η kiértékelés mellett meghatározva.

2.2.2. Core Erlang programok műveleti szemantikája

Core Erlang formális szemantikája a rendszerállapotok felett megadott struktúrális műveleti szemantika (címkézett állapotátmenetrendszer). A rendszer állapota Erlang folyamatok halmaza. A folyamatok rendezett hármasok: `<e,pid,q>`, ahol `e` a folyamat által kiértékelt kifejezés, `pid` a folyamat azonosítója, `q` pedig a folyamthoz tartozó üzenetsor (ahová a beérkezett üzenetek kerülnek). Ha a rendszer állapotának két különböző folyamat eleme, akkor folyamatazonosítójuk is különböző. A rendszer állapotát az alábbi induktív nyelvtannal definiált szintaxis szerint adják meg, ahol a párhuzamos kompozíció szokásos jele, a || most a halmazúnió jele.

```
s::=  <e,pid,q>   |    s || s
```

A nyelv teljes műveleti szemantikája nem bonyolult [Dam98], de a szabályok és segédfüggvények nagy száma miatt eltekintünk megadásától. Néhány példát mutatunk. A szabályok három csoportba sorolhatók: lokális, egy folyamatra vonatkozó, ill. folyamatok rendszerére vonatkozó szabályok.
Jelölések: m egy mintaillesztést tartalmazó kifejezést jelöl, pid (saját) folyamatazonosító, $pid <> pid'$. $\longrightarrow$ jelöl egy olyan kiértékelési lépést, amelyben nincs kommunikáció, $\stackrel{pid!v}{\longrightarrow}$, illetve $\stackrel{pid?v}{\longrightarrow}$ olyan lépéseket, ahol (szinkron) kommunkációra kerül sor. A nyelv aszinkron kommunikációs modelljéhez úgy jutunk el, hogy az üzenetküldés szemantikai leírását két lépésre bontjuk. Az első lépésben az üzenet csak a fogadó fél üzeneteket tároló sorába jut el a CSP-ből ismert `!,?` műveletpár alakalmazásával, szinkron kommunikációval. A második lépésben, az üzenetet fogadó folyamat a `receive` Erlang nyelvi elem segítségével, az üzenet küldésétől időben elválasztva, azinkron módon olvassa ki az üzenetet a saját várakozási sorából. Megjegyezzük, hogy a `?` művelet nem eleme az Erlang nyelvnek, bevezetése a műveleti szemantika leírását könnyíti meg.

SEQ: $v, e \longrightarrow e$
SEND: $< r[pid'!v], pid, q > \stackrel{pid'!v}{\longrightarrow} < r[v], pid, q >$, ha $pid \not= pid'$.
INPUT: $< e, pid, q > \stackrel{pid?v}{\longrightarrow} < e, pid, q@v >$, tetszőleges v értékre.
COM: $s_1 || s_2 \longrightarrow s_1' || s_2'$, ha $s_1 \stackrel{pid!v}{\longrightarrow} s_1'$ és $s_2 \stackrel{pid?v}{\longrightarrow} s_2'$.

2.2.3. ErlVer levezetési szabályok, szekvent kalkulus

Egy $\Gamma \vdash \Delta$ alakú, Gentzen típusú levezetési szabály érvényes, ha a szabad változók minden lehetséges η kiértékelése mellett, amennyiben Γ minden állítása teljesül, akkor Δ valamely állítása is teljesül. Az állításokban szereplő paraméterek befuthatnak valamely típusértékhalmazt (pl. üzenetek, függvények vagy folyamatok halmazát).

A levezetési szabályoknak öt csoportja van [Dam98]. A struktúrális szabályok állítások használatáról, bevezetéséről, elhagyásáról (pl. vágásási szabályok), a logikai szabályok a logikai összekötőjelek alkalmazásáról, az egyenlőségi érvelés szabályai pedig az egyenlőségek és az egyenlőtlenségek felhasználásáról szólnak. A modális szabályok a temporális operárotok monotonitási tulajdonságait, a nyelv struktúrális műveleti szamantikája által definiált állapotátmenteket leíró szabályokat és a kompozícionális szabályokat (alrendszerekre bontás) foglalják magukban. Végül az atomi formulákra vonatkozó szabályok az Erlang nyelvhez kötődő speciális predikátumok (`queue`,

`term` stb.) használatáról szólnak. Ezen levezetési szabályok közül – terjedelmi okok miatt – egyet mutatunk be, a folyamatokra vonatkozó vágási szabályt.

Paraméterek segítségével nyitott specifilációkat fogalmazhatunk meg. $x : \Psi \vdash P(x) : \Phi$ jelntése: P rendelkezik a Φ tulajdonsággal, ha az x paraméter rendelkezik a Ψ tulajdonsággal.

Nyitott specifikációk (paraméterek) segítségével kompozícionálisan érvelhetünk összetett rendszerek viselkedéséről. Legyen Q a P egy tetszőleges olyan komponense, amely rendelkezik a Ψ tulajdonsággal. A vágási szabály a következő:

ha $\Gamma \vdash Q : \Psi, \Delta$ és $\Gamma, x : \Psi \vdash P : \Phi, \Delta$, akkor $\Gamma \vdash P[Q/x] : \Phi, \Delta$.

Ha Q modul rendelkezik a P paraméterétől (x-től) megkövetelt tulajdonsággal, akkor behelyettesítve adott tulajdonságú összetett programhoz jutunk.

2.2.4. Perzisztens halmaz - példa

Példaként tekintsük a [FreGur99] cikkben bemutatott absztrakt perzisztens halmaztípus ErlVer programját és specifikációját. Perzisztens halmaztípus objektuma csak bővíthető, elemet nem lehet belőle eltávolítani.

Az alábbi Erlang program konkurrens környezetbe helyezett *aktív adattípushoz tartozó objektumként* valósítja meg a perzisztens halmazt. A halmazt egy Erlang folyamatazonosító azonosítja. A halmazok egy interfész folyamaton keresztül érhetőek el. A halmaz objektum viselkedését folyamatok együttműködésével, folyamatlánccal adjuk meg. Ha a felhasználók a halmazokat az interfészen keresztül szólítják meg, akkor ezzel megoldással biztosítjuk, hogy a konkurrens hozzáférés ellenére se sérüljön meg a típusinvariáns.

A modul két függvényt definiál, az egyik az üres halmaz, a másik az egy elemű halmaz kezelésére szolgál. Amikor egy új halmazt létrehozunk, akkor az egyetlen egy az üres halmaznak megfelelő függvényt kiértékelő folyamatból áll. Ha új elemet adunk hozzá a halmazhoz, akkor egy új folyamat jön létre.

```
%% set_adt.erl
%% A perzisztens halmaz típus Erlang implementációja
%%    ErlVer példa
-module(set_adt).
-export([mk_empty/0, is_empty/1, is_member/2,
         add_element/2, empty_set/0]).
```

```
%% Halmaz mu"veletek:
empty_set () ->
    receive
        {is_empty, Client} ->
            Client ! {is_empty, true},
            empty_set ();
        {is_member, Element, Client} ->
            Client ! {is_member, Element, false},
            empty_set ();
        {add_element, Element} ->
            set (Element, mk_empty ())
     end.

set (Element, Set) ->
     receive
        {is_empty, Client} ->
            Client ! {is_empty, false},
            set (Element, Set);
        {is_member, SomeElement, Client} ->
             if
                SomeElement == Element ->
                Client ! {is_member, SomeElement, true},
                    set (Element, Set);
                SomeElement /= Element ->
                    Set ! {is_member, SomeElement, Client},
                    set (Element, Set)
             end;
        {add_element, SomeElement} ->
             if
                SomeElement == Element ->
                    set (Element, Set);
                SomeElement /= Element ->
                    Set ! {add_element, SomeElement},
                    set (Element, Set)
              end
     end.
```

```
%% Külso" interfész:

mk_empty () ->
    spawn (set_adt, empty_set, []).

is_empty (Set) ->
    Set ! {is_empty, self ()},
    receive
         {is_empty, Value} -> Value
    end.

is_member (Element, Set) ->
    Set ! {is_member, Element, self ()},
    receive
         {is_member, Element, Value} -> Value
    end.

add_element (Element, Set) ->
    Set ! {add_element, Element}.
```

A program legfontosabb tulajdonsága, hogy egy halmazban elhelyezett elem a halmazban örökre benn marad. Ezen tulajdonság helyett egy egyszetrűbb programtulajdonság specifikációját adjuk meg.ű

A specifikáció azt köti ki, hogy `empty SetPid` amíg `non_empty SetPid` igazzá nem válik, majd `ag_non_empty SetPid` mindig igaz. Ez azt jelenti, hogy tetszőleges `SetPid` folyamattal azonosított halmaz stabil tulajdonsága [ChaMis89, Hor96], hogy nem üres (ha egyszer igazzá válik, akkor igaz marad). A specifikációt a `set_adt.pdf` file-ban helyezhetjük el.

```
ag_nonempty: erlangPid -> erlangSystem -> prop =>
\SetPid:erlangPid. \Setsys: erlangSystem .
  ((SetSys : nonempty SetPid) /\
   (SetSys : forall Alpha:
        erlangAction.[Alpha](ag_non_empty SetPid));
persistently_non_empty : erlangPid -> erlangSystem  -> prop =>
  \SetPid:erlangPid. \SetSys:erlangSystem.
  ((SetSys: non_empty SetPid) /\ (SetSys: ag_non_empty SetPid)) \/
   (SetSys: empty SetPid) /\ (SetSys : forall Aplha:erlangaction.
    [Alpha](persistently_non:empty setPid));
```

A bizonyítandó állítás, amelyet a `proof.sml` file-ban helyezünk el a bizonyítás során alkalmazott levezetési szabályoknak megfelelő taktikák leírásával együtt:

```
prove "declare P:erlangPid in |-
     proc<empty_set(),P,eps>  : persistently_non_empty P";
```

A bizonyítás a `cevt` New Jersey SML alapú interpreter segítségével történik [EVTref00] a `use "proof.sml";` paranccsal.

Összefoglalás, további elemzés

Az ErlVer/EVT modális μ-kalkulus alapú specifikációs nyelve alkalmas elosztott rendszerek programtulajdonságainak kifejezésére. A helyességbizonyító rendszer használatához azonban a temporális logikai alapműveletek és számos szemantikai fogalom ismeretére is szükség van. Ebben a tanulmányban bemutattuk a temporális logika alapfogalmait, a fixpont műveletek tulajdonságait, a modális μ kalkulust és annak ErlVer-ben alkalmazott változatát. Ismertettük a specifikációs nyelv szintaxisát és szemantikáját, a Core Erlang műveleti szemantikáját. Az ErlVer helyességbizonyító rendszer Gentzen típusú levezetési szabályai ezekre a definíciókra támaszkodnak.

A következő időszakban az ErlVer levezetési szabályait is érdemes megvizsgálni abból a szempontból, hogy melyeket érdemes taktika formájában a Sparkle-ba átemelni. A levezetési szabályok közül különösen fontosak lehetnek a nyitott rendszerekre, illetve a modális szabályok.

Irodalomjegyzék

[Arm96] Armstrong, J., Virding, R., Wikström, C., Williams, M.: Concurrent Programming in Ergalng. Second Edition. Prentice Hall, 1996.

[Arts98] Arts, T., Dam, M., Fredlund, L., Gurov, D.: System Description: Verification of Distributed Erlang Programs. In Proc. of CADE'98, Springer-Verlag, vol 1421, pp. 38-41, 1998.

[EVT2000] Arts, T., Chugunov, G., Dam, M., Fredlund, L., Gurov, D.: A Tool for Verifying Software Written in Erlang and Thomas Noll Submitted to: Int. J. STTT, 2000. 17 pages.

[EVTref00] Arts, T., Fredlund, L., Gurov: Reference Manual for the Erlang Verification Tool, for version 2.00. December 20, 2000.

[ButDowStr02] Butterfield A., Dowse M., Strong, G.: Proving Make Correct: IO Proofs in Haskell and Clean. In: Proceedings of Implementation of Functional Programming Languages, Madrid, 2002. pp. 330-339.

[Ben88] 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.

[Car94] Carruth, A.: *Real-Time Unity.* Technical Report TR94-10, University of Texas at Austin, ftp://ftp.cs.utexas.edu. (March 29, 1994).

[ChaMis89] Chandy, K.M.-Misra, J.: *Parallel Program Design: A Foundation.* (Addison-Wesley, 1988, 1989).

[Dam98] Dam, M., Fredlund, L., Gurov,D.: Toward Parametric Verification of Open Distributed Systems. In *Compositionality: The Significant*

Difference (H. Langmaack, A. Pnueli, W.-P. De Roever (eds)), Springer-Verlag 1998.

[Em90] Emerson, E.A.: Temporal and Modal Logic. In. van Leeuven,J. ed.: Handbook of Theoretical Computer Science, pp. 995-1072., Elsevier, 1990.

[EmeSri88] 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.

[Erl] An Erlang Course. http://www.erlang.org/course/course.html. 2002.

[FreGur99] Fredlund, L., Gurov, D.: A Framework for Reasoning about Open Distributed Systems. In Proc. ASIAN'99, Lecture Notes in Computer Science, 1742: 87-100, 1999.

[Har01] Harper, R.: Programming in Standard ML. Working Draft. Carnegie Mellon University, Spring Semester, 2001. http://www.cs.cmu.edu/~ rwh/smlbook/

[Hor96] Horváth Z.: *A relational programming model of parallel programs.* PhD thesis, in Hungarian. PhD program in Informatics, Department Group Informatics, University Eötvös Loránd Budapest, Hungary. 1996.

[Hor99] Horváth,Z.-Achten,P.-Kozsik,T.-Plasmeijer,R.: Verification of the Temporal Properties of Dynamic Clean Processes, *Proceedings of the 11th International workshop on the Implementation of Functional Languages, IFL'99, Lochem*, The Netherlands,1999, pp. 203-218.

[HorKozs02] Horváth Z., Kozsik T.: Certified Proven Property Carrying Code (CPPCC) - Safe Functional Mobile Code, ECOOP WS 1 and 18, Malaga, 2002.

[HoTePá02] Horváth Z., Pásztor K., Tejfel M.: Funkcionális programok helyessége, Informatika a Felsőoktatásban, Debrecen, 2002.

[JutKnaRao89] 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.

[Koz83] Kozen, D.: Results on the Propositional μ-calculus. Theoretical Computer Science 27 (1983) 334-354.

[Kna90] Knapp, E.: A Predicate Transformer for Progress. *Information Processing Letters*, Vol. 33 (1989/90) 323-330.

[Kna92] Knapp, E.: Derivation of concurrent programs: two examples. *Science of Computer Programing*, Vol. 19 (Oct. 1992) 1-23.

[Koz94] Kozma L.: Synthesizing Methods of Parallel Systems. An Overview. In: *Proceedings of μP'94*, Technical University Budapest, Hungary (1994) 586-.

[Krö87] Kröger, F.: *Temporal Logic of Programs.* (Spriger, 1987).

[Lam77] Lamport, L.: Proving the Correctness of Multiprocess Programs, *IEEE Transactions on Software Engeneering*, Vol. SE-3, No., 2 (March 1977) 125-143.

[Lam90] Lamport, L.: win and sin: Predicate Transformers for Concurrency. *ACM Transactions on Programming Languages and Systems*, Vol. 12, No. 3 (July 1990) 396-428.

[Lam91] 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).

[LamLyn90] Lamport, L.-Lynch, N.: Distributed Computing Models and Methods. In: van Leeuwen, ed., *Handbook of Computer Science*, vol. B (Elsevier, Amsterdam, 1990) 1157-1199.

[LamSin79] van Lamsweerde, A., Sintzoff, M.: Formal Derivation of Strongly Correct Concurrent Programs. *Acta Informatica*, Vol. 12, No. 1 (1979) 1-31.

[LukSne92] Lukkien, J., van de Snepscheut J.,L.,A.: Weakest Preconditions for Progress. *Formal Aspects of Computing*, Vol. 4 (1992) 195-236.

[Mil97] Milner,R., Tofte, M., Harper, R., MacQueen, D.: *The Definition of Standard ML (Revised).* The MIT Press, 1997.

[MolEek99] Mol, de M.-van Eekelen,M.: A Proof Tool Dedicated to Clean, AGTIVE'99, Kerkade.

[MolEekPla02] Maarten de Mol, Marko van Eekelen, Rinus Plasmeijer. Theorem Proving for Functional Programmers - SPARKLE: A Functional Theorem Prover ¡ftp://ftp.cs.kun.nl/pub/Clean/papers/2002/molm2002-TheoremProvingFP.ps.gz¿ . In: Arts, Th., Mohnen M., eds. Proceedings of the 13th International Workshop on the Implementation of Functional Languages, IFL 2001, Selected Papers, Älvsjö, Sweden, September 24-26, 2001, Springer-Verlag, LNCS 2312, pages 55-71.

[Mor90] Morris, J.,M.: Temporal Predicate Transformers and Fair Termination. *Acta Informatica*, Vol. 26, 287-313, 1990.

[Pac92] Pachl, J.: A simple proof of a completeness result for leads-to in the UNITY logic. *Information Processing Letters*, Vol. 41 (1992) 35-38.

[Par79] Park, D.: *On the semantics of fair parallelism* In LNCS 86, pp 504-526. Springer 1980.

[Pász93] Pásztorné Varga K.: *Logikai alapozás alkalmazásokhoz. Matematikai logika - számítástudomány.* Egyetemi jegyzet, ELTE, TTK (Budapest, 1992).

[PászVár03] Pásztorné Varga K.-Várterész M.: *Logikai a számítástudományban.* Tankönyv, (Budapest, 2003).

[Pil98] M. Pil: *Dynamic Types and Type Dependent Functions*; in Proc. of the 10th International Workshop on Implementation of Functional Languages (IFL'98), London, 1998, pp. 65-84.

[Pra94] Prasetya, I.S.W.B.: Error in the UNITY Substitution Rule for Subscribed Operators. *Formal Aspects of Computing*, Vol. 6 (1994) 466-470.

[Pra86] Pratt, V.: Modeling Concurrency with Partial Orders. *International Journal of Parallel Programing*, Vol. 15, No. 1 (1986) 33-71.

[Rácz92] Rácz É.: *A Temporal Logic Specification of a Transaction Manager.* Ph.D. Thesis, ELTE (1992) /in Hungarian/

[Rao95] Rao, J.,R.: *Extensions of the UNITY Methodology*, Lecture Notes in Computer Science, Vol. 908. (Springer, 1995).

[San91] Sanders, B.A.: Eliminating the substitution axiom from the UNITY logic. *Formal Aspects of Computing*, Vol. 3 (1991) 189-205.

[Tho90] Thomson, S.: Lawful Functions and Program Verification in Miranda, Science of Computer Programming, Vol. 13, Num. 2-3, May 1990, 181-218.