Asupra unor abordări semantice ale unor limbaje concurente

Conf.dr. Lucian Luca, Asist. Adela Ionescu,
Universitatea „Tibiscus" Timişoara
Mat.ec. Sorina-Carmen Luca
Anglo Romanian Bank Ldt.- Timişoara

ABSTRACT. The study presents some semantic approaches of two concurrent languages (notations), CSP and CCS, which have had a significant influence on the concurrent pragramming languages that have followed. These approaches have transformed the two concurrent notations in process algebra, a starting point for the temporal theoretical approach in the domain of the real time systems.

1 Introducere

În [LIL04] sunt amintite mai multe încercări pentru a defini un calcul pentru procese paralele. Algebrele de proces introduse, ce au dat o înţelegere mai bună asupra comportamentelor de bază pe care le pot arăta sistemele concurente, au pornit de la diverse încercări de a defini limbaje concurente.

Astfel, limbajul CSP (*Communicating Sequential Processes*), care a fost propus de către C.A.R. Hoare încă din 1978, specificaţiile finale fiind publicate în 1985 [Hoa85], a avut o influienţă deosebită asupra tuturor limbajelor de programare concurentă care i-au urmat. Limbajul face parte din familia celor orientate pe mesaje şi utilizează transferul sincron de mesaje şi comunicaţiile selective. Aşa cum a arătat şi Hoare, CSP nu este un limbaj complet, scopul său principal fiind acela de a propune o notaţie pentru o serie de concepte noi de programare.

CSP a constituit punctul de plecare al unor serii de cercetări privind programarea sistemelor distribuite şi demonstrarea corectitudinii

programelor. Vom prezenta câteva aspecte semantice legate de CSP, în strânsă legătură cu CCS ([Mil89])

2 Structura unui program CSP

Un program CSP este alcătuit dintr-o serie de procese concurente descrise prin mecanismul clasic *cobegin-coend*. Procesele comunică prin mesaje transmise sincron, cu ajutorul operațiilor clasice *send* şi *receive*.

O caracteristică generală a limbajului este folosirea unor caractere unice în locul diverselor cuvinte cheie prezente în mod obişnuit în alte limbaje. Astfel, cuvintele cheie ce intervin în alte limbaje (*begin ... end, loop ... end loop, cobegin ... coend*, etc.) se redau prin caracterele "[", respectiv "]".

În ce priveşte caracteristicile de concurență, specificarea execuției concurente a proceselor se face printr-o variantă a instrucțiunii *cobegin*. Astfel,

[P1::S1 || P2::S2 ||... || Pn::Sn]

înseamnă execuția concurentă a instrucțiunilor S1, S2, ..., Sn, care constituie corpul proceselor P1, P2, ..., Pn. Nu există de fapt definiții de procese, astfel că P1, P2, ..., Pn trebuie considerate doar ca nume ale unor secvențe de instrucțiuni, servind pentru identificarea acestora în diverse locuri din program. Execuția unei instrucțiuni paralele de forma celei de mai înainte se termină când toate procesele componente sunt terminate.

Este permis ca procesele să aibă şi variabile comune, dar acestea nu pot fi modificate de către procese. Pentru comunicarea şi sincronizarea proceselor se folosesc în exclusivitate comenzile de introducere/extragere.

O comandă de introducere are sintaxa:

sursă ? variabilă

unde *sursă* este numele unui proces, iar *variabila* este locală procesului care conține comanda. Rezultă că o asemenea comandă este echivalentă cu *receive*.

Comanda de extragere are forma:

destinație ! expresie

fiind echivalentă cu un *send.* O comunicare între două procese, P1 şi P2, se realizează când acestea execută comenzile:

P2 ! expresie ,respectiv **P1 ? variabilă**

Rezultatul comunicării este faptul că în procesul P2 se atribuie variabilei prevăzute în comanda de introducere valoarea expresiei inclusă în comanda de extragere din P1, după care cele două procese continuă independent.

Observăm că atât operația de emisie cât şi operația de recepție explicitează numele procesului destinație, respectiv sursă (denumire directă), iar expresia din operația de emisie trebuie să corespundă ca tip cu variabila specificată în operația de recepție corespunzătoare.

Instrucțiunile cu gardă, alternative (*if*) şi repetitive (*loop*), sunt mijloace de control al fluxului acțiunilor şi a gradului de nedeterminism propriu programului concurent.

Instrucțiunea alternativă are forma:

[g_1 → listă_instr_1 □ ... □ g_n → listă_instr_n]

iar instrucțiunea repetitivă:

*** [g_1 → listă_instr_1 □ ... □ g_n → listă_instr_n]**

O complicație apare datorită faptului că în gărzi nu sunt permise comenzi de extragere (echivalentele pentru *send*) şi datorită utilizării denumirii directe, deoarece atât în operația de emisie, cât şi în cea de recepție se explicitează numele procesului destinație, respectiv sursă. Din acest motiv CSP nu permite realizarea comunicarea, printr-un buffer, a unui număr oarecare de producători şi consumatori, dar permite comunicarea între procesele dintr-o familie de producători şi alta de consumatori.

Specificarea unei astfel de familii se face indicând la descrierea procesului un indice şi subdomeniul în care poate lua valori (limitele subdomeniului trebuind să fie constante). Deci, numărul proceselor unui program CSP este fixat la scrierea programului.

La comunicarea între procese prin mesaje, o problemă importantă ce trebuie rezolvată în cadrul fiecărui limbaj este situația limită, în care un proces încearcă să comunice cu un alt proces, care însă s-a terminat. În CSP Hoare a dat următoarea soluție: operația care are ca adresant un proces terminat se încheie anormal, semnalându-se o eroare. În cazul în care operația apare într-o gardă, la evaluarea gărzii va rezulta că ea este închisă. În acest fel se asigură o terminare elegantă a instrucțiunilor repetitive în care se aşteaptă mesaje din partea altor procese terminate.

3 Extensii ale limbajului CSP. Limbajul CCS

Hoare a demonstrat prin CSP că un număr foarte redus de concepte simple şi clare pot sta la baza scrierii unei game largi de programe: procesele concurente ca formă de organizare a programului, transmiterea sincronă de mesaje ca mijloc de comunicare între procese şi instrucțiunile cu gardă ca unică formă de control a fluxului acțiunilor în program. Instrucțiunile cu gardă introduc în mod explicit nedeterminismul propriu programelor

concurente. Până la această propunere, toate limbajele de programare concurentă au evitat explicitarea în instrucțiuni a evoluției nedeterministe. CSP a fost considerat de la început ca o propunere restrînsă, care să stea la baza unor dezvoltări ulterioare în vederea definirii unor limbaje de programare concurentă pentru sisteme distribuite. Două din restricțiile limbajului au constituit obiectul criticilor: denumirea directă şi statică a proceselor şi lipsa comenzilor de extragere din gărzi.

Denumirea directă face ca un proces servant, de exemplu, să poată comunica doar cu procese a căror denumire este cunoscută în momentul codificării servantului. Este imposibil să se realizeze o bibliotecă de programe care să fie utilizată în scrierea altor programe. O soluție posibilă, sugerată de altfel de către Hoare, ar fi implementarea comunicării prin porturi şi suplimentarea limbajului cu o notație care să indice, la începutul unei comenzi paralele, modul de utilizare a porturilor.

A. Silberschatz a propus o completare a limbajului CSP, prin utilizarea porturilor şi permiterea utilizării comenzilor de extragere în gărzi. În interiorul unui proces *p* pot fi declarate un număr oarecare de porturi, iar procesul *p* este "proprietarul" acestor porturi. Un alt proces poate comunica cu *p* prin intermediul unui port al acestuia. În general, două procese pot comunica prin intermediul oricărui port care aparține unuia din ele. Se elimină astfel restricția denumirii directe.

Apoi, Silberschatz a revenit cu o îmbunătățire menită să elimine acest neajuns al comunicării prin porturi, asociind fiecărui port o valoare care să identifice procesul emițător. Această valoare este atribuită apoi unei variabile de tip întreg, ce poate fi utilizată ulterior în program. Silberschatz a introdus, cu această ocazie, alte trei categorii de porturi, pe lângă cele simple:

port[!] o operație de transmisie printr-un astfel de port se adresează numai procesului corespunzător valorii momentane a portului, recepțiledesfăşurându-se normal, de la orice port;

port[?] printr-o operație de recepție se pot obține numai mesaje provenite de la procesul corespunzător valorii momentane a portului, transmisiile desfăşurându-se normal;

port[!, ?] se comportă ca un *port[!]* la transmisie şi ca un *port[?]* la recepție.

Ulterior, au apărut completări ale limbajului, unele legate de aspectele prezentate anterior, altele legate de structurarea ierarhică a programelor CSP şi de posibilitatea verificării corectitudinii programelor.

Limbajul CSP, care în familia limbajelor concurente bazate pe comunicare prin mesaje este comparabil cu Concurrent Pascal, de la

limbajele bazate pe comunicarea prin variabile comune, are doi descendenţi direcţi: limbajele Occam şi Joyce, ultimul creat de către P.Brinch Hansen în 1987, limbaje utilizate în special în domeniul programării distribuite.

Limbajul CCS original ([Mil89]) a avut variabile, valori şi canale. CCS pur este o simplificare, în care comunicaţiile de-a lungul canalelor sunt simplificate în felul următor:

- valorile actuale ce se transmit pe canale se pierd după transmitere;
- de asemenea, se pierd numele variabilelor, păstrându-se numai numele canalului;
- pentru a obţine o valoare într-o variabilă ***x*** pe canalul ***a***, adică ***a?x***, se abstractizează prin ***a***;
- pentru a trimite o valoare ***n*** pe canalul ***a***, adică ***a!n***, se abstractizează prin acţiunea complementară, $\overline{a}$.

În acest mod, comunicaţia este abstractizată prin sincronizarea unei acţiuni şi a acţiunii complementare. Această sincronizare este observabilă prin apariţia unei acţiuni invizibile **τ**.

În cele urmează, **α** reprezintă acţiuni, acţiuni complementare şi τ, iar ***f*** este o funcţie pe acţiuni care comută cu $x \to \overline{x}$. Sintaxa lui CCS pur este:

$t ::= \mathbf{nil}$	(procesul nul)
$\mid t_1 + t_2$	(operatorde alegere)
$\mid t_1 \mid t_2$	(compunere paralelă)
$\mid \alpha.t$	(prefixare)
$\mid t_1 \backslash c$	(operator de restricţionare)
$\mid p[f]$	(re-etichetare)
$\mid \mathbf{rec}\ x.t(x)$	(agent recursiv)

4 Semantici în CCS şi CSP. Discuţie

Prima semantică CCS [Mil89] a fost dată în termeni de arbori de sincronizare (sisteme cu tranziţii aciclice). Adesea ea se dă folosind reguli SOS [ACS96], care descriu sistemele cu tranziţii astfel:

$$\frac{}{\alpha.t \xrightarrow{\alpha} t}$$

Această regulă spune că $\alpha.t$ poate declanşa prima dată o acţiune α.

$$\frac{t_i \xrightarrow{\alpha} t'_i}{t_1 + t_2 \xrightarrow{\alpha} t'_i}$$

Această regulă arată că t_1+t_2 se comportă ca t_1 sau ca t_2 odată şi pentru totdeauna.

$$\frac{t_1 \xrightarrow{\alpha} t'_1}{t_1|t_2 \xrightarrow{\alpha} t'_1|t_2}$$

$$\frac{t_2 \xrightarrow{\alpha} t'_2}{t_1|t_2 \xrightarrow{\alpha} t_1|t'_2}$$

Aceste reguli definesc pe $t_1|t_2$, prin întrepătrunderea lor.

$$\frac{t_1 \xrightarrow{\alpha} t'_1 \qquad t_2 \xrightarrow{\bar{\alpha}} t'_2}{t_1|t_2 \xrightarrow{\tau} t'_1|t'_2}$$

Această regulă defineşte *sincronizarea între acţiuni complementare* care, atunci când se "anihilează" între ele, produc o acţiune tăcută τ.

$$\frac{t \xrightarrow{\alpha} t' \quad \alpha \neq c \quad şi \quad \alpha \neq \bar{c}}{t \setminus c \xrightarrow{\alpha} t' \setminus c}$$

Această regulă defineşte *operatorul de restricţionare*, care se aplică atât unei acţiuni, cât şi acţiunii sale complementare.

$$\frac{t \xrightarrow{\alpha} t'}{t[f] \xrightarrow{f(\alpha)} t'[f]}$$

Această regulă defineşte într-un mod evident *operatorul de re-etichetare.*

$$\frac{\textbf{\textit{rec}}\ x.t[x] \xrightarrow{\alpha} t'}{t[\textbf{\textit{rec}}\ x.t[f]] \xrightarrow{\alpha} t'}$$

Această regulă spune că ***rec*** $x.t[x]$ este similar cu "termenul infinit" $t[t[...[\textbf{\textit{nil}}]]]$.

Semantica "denotaţională" a sistemelor ordinare cu tranziţii se poate da şi folosind proprietăţile lor categoriale[WN94]:

- produsul cartezian este o formă de produs sincronizat, în plus cu întrepătrundere (aceleaşi tranziţii, cu o tranziţie * drept una din componente);
- coprodusul fibrat este alegerea non-determinstă din CCS;
- operatorul de restricţie din CCS este obţinut printr-un lifting cartezian tare.
- operatorul de reetichetare din CCS corespunde lifting-ului cocartezian tare;

- operatorii paralel şi de prefixare din CCS nu corespund nici unui combinator categorial în categoria sistemelor cu tranziţii.

O s***emantică CCS folosind structuri de evenimente*** a fost dată în [Win89], iar în [Gup94] se dă o ***semantică CCS în termeni de spaţii Chu***.

CSP are aproximativ aceleaşi primitive ca şi CCS, însă mecanismul de sincronizare este puţin diferit.

Sintaxa unui CSP "cu valori" este dată de următoarele reguli:

$t ::= \alpha\,?\,X \rightarrow t$	(input pe canalul α şi memorează în X, apoi execută t)
$\|\alpha!a \rightarrow t$	(output valoarea a pe canalul α şi apoi execută t)
$\|t_1 \parallel t_2$	(compunere paralelă)
$\|t_1 \Box t_2$	(operator de alegere)
|l : t	(etichetare)
$\mu\, x.t(x)$	(agent recursiv, pentru t o expresie cu gardă

O semantică în termenii sistemelor cu tranziţii etichetate se poate găsi în [Win93]. În acest calcul, am ales să reprezentăm valorile şi variabilele, aşa că stările sistemului cu tranziţii vor fi de forma $\langle c,\sigma \rangle$, unde c este un program CSP şi σ este o memorare (adică o funcţie de la variabile la valori). Deoarece semantica acestui limbaj CSP este extrem de similară cu cea a limbajului CCS, o vom specifica doar pentru operatorul paralel:

$$\frac{\langle c_0,\sigma\rangle \xrightarrow{\lambda} \langle c'_0,\sigma'\rangle}{\langle c_0 \| c_1,\sigma\rangle \xrightarrow{\lambda} \langle c'_0 \| c_1,\sigma'\rangle}$$

$$\frac{\langle c_1,\sigma\rangle \xrightarrow{\lambda} \langle c'_1,\sigma'\rangle}{\langle c_0 \| c_1,\sigma\rangle \xrightarrow{\lambda} \langle c_0 \| c'_1,\sigma'\rangle}$$

CSP fără valori are sintaxa următoare:

$$\begin{array}{rccccc} P ::= & SKIP & | & STOP & | & a \rightarrow P \;| \\ & P;\, Q & | & P \cap Q & | & P \parallel Q \;| \\ & P \,|||\, Q & | & P \setminus a & | & repeat\ P \;| \end{array}$$

$P|||Q$ denotă întrepătrunderea lui P cu Q. $P\|Q$ este sincronizarea cu pas de încuiere a lui P cu Q. Aceasta înseamnă că procesele trebuie să sincronizeze fiecare acţiune a lui P şi Q cu acelaşi nume.

O semantică denotaţională (adevărata-concurenţă) a lui CSP fără valori, folosind structuri de evenimente, a fost propusă în [Win89].

Toate algebrele de proces se bazează pe o "schemă de funcționare în două faze": execuțiile lor alternează de la o parte sincronă, în care toate componentele sunt de acord pentru ca timpul să progreseze, la o parte asincronă, în care progresul timpului este blocat [LIL04]. Această abordare a influențat profund numeroasele algebre de proces care au apărut în ultimii ani (de exemplu, *TCSP* este o extensie simplă a lui CSP, iar algebrele *TeCCS, TiCCS* sunt extensii ale lui CCS)

CSP şi CCS nu s-au dorit a fi limbaje concurente complete, ci două notații pentru introducerea unor concepte noi în programarea concurentă bazată pe mesaje. Diferitele abordări semantice pe care le-am prezentat sau enumerat au transformat aceste două notații în veritabile algebre de proces, puncte de plecare pentru dezvoltările teoretice ulterioare în domeniul concurenței.

Bibliografie

[ACS96] **R. Amadio, I. Castellani, and D. Sangiorgi** - *On bisimulations for the asynchronous π-calculus*, Lecture Notes in Computer Science, 1119:147-162, Springer-Verlag, 1996.

[Gup94] **V. Gupta** - *Chu spaces*, PhD thesis, Stanford University, 1994.

[Hoa85] **C. A. R. Hoare** - *Communicating Sequential Processes*, Prentice-Hall, London, 1985.

[LIL04] **L. Luca, A. Ionescu, S.C. Luca** – *Asupra algebrelor de proces pentru sistemele în timp real,* Analele Universității "Tibiscus", Seria Informatică, vol. II, fascicula I: 102-112, Timişoara, 2004

[Luc03] **L. Luca** – *Spații de procese fuzzy*, Ed. Mirton, Timişoara, 2003.

[Mil89] **R. Milner** - *Communication and Concurrency*, International Series in Computer Science, Prentice Hall, 1989.

[Win89] **G. Winskel** - *An introduction to event structures*, Lecture Notes in Computer Science, 354:364-397, Springer-Verlag, 1989.

[Win93] **G. Winskel** - *The formal semantics of programming languages*, The MIT Press, 1993.

[WN95] **G. Winskel and M. Nielsen** - *Models for concurrency*, Handbook of Logic in Computer Science, 4:1-148, Oxford University Press, 1995.