Langages concurrents - mobilité
le π-calcul

Joëlle Despeyroux

INRIA Sophia-Antipolis

Module
“Méthodes formelles et fiabilité du logiciel”
Dea d'informatique, Nice, automne 2000

Le π-calcul

Le π-calcul est un modèle de la programmation concurrente, basé sur la notion de noms. Les processus recoivent et émettent des noms, denotant des canals.

$$\text{Processus :}\quad P, Q ::= \bar{x}y.P \mid x(y:t).P \mid \nu x:t.P \mid (P \mid Q) \mid !\,P \mid 0$$

x, y sont des noms et t denote le type d'un nom.
output: $\bar{x}y.P$: processus qui envoie un nom y le long d'un canal x, puis se comporte comme P.
input: $x(y).P$ processus recevant un nom z le long d'un canal x, puis évoluant comme $P[z/y]$.
$(P \mid Q)$ denote la composition *parallèle* de P et Q où P and Q sont en concurrence.
nul: 0 denote le processus vide.
bang: $!\,P$ denote le processus infini $P \mid P \mid \cdots$.
restriction: $\nu x:t.P$ declare x canal privé de type t dans P.
Dans le terme $\nu x:t.P$, x est lié dans P. Dans le terme $x(y).P$, y est lié dans P, attendant de recevoir une valeur pour passer la main à P.

Les processus s'évaluent en produisant des *actions*:

$$\text{Actions :}\quad A ::= xy \mid \bar{x}y \mid \bar{x}(y:t) \mid \tau$$

lecture xy (lecture d'une valeur pour la variable y sur le canal x), écriture $\bar{x}y$ (écriture d'une valeur y sur le canal x), écriture liée $\bar{x}(y:t)$ (où x est lié dans y) et les actions silencieuses τ. Les communications sont silencieuses.

Exemples d'évaluation

Communication simple: y est envoyé à Q, le long du canal x:

$$\bar{x}y.P \mid x(z:t).Q \xrightarrow{\tau} P \mid Q\{y/z\}$$

Appellons $P \mid Q \mid R$ le terme:

$$\bar{x}y \mid x(u:t).\bar{u}v \mid \bar{x}z$$

Il y a 2 communications concurrentes dans ce terme: soit P envoie y à Q soit R envoie z à Q. D'où 2 résultats possibles:

$$0 \mid \bar{y}v \mid \bar{x}z \quad ou \quad \bar{x}y \mid \bar{z}v \mid 0$$

Dans le cas où P et Q partagent un canal privé x, on a:

$$\nu x:t.(\bar{x}y \mid x(u:s).\bar{u}v) \mid \bar{x}z$$

Le nom x libre dans R, n'est pas le même que le nom x privé dans P and Q. Ici, seule la 1ère communication est possible, donnant un processus où la restriction n'est plus utile, et peut être éliminée par un gc:

$$\nu x:t.(0 \mid \bar{y}v) \mid \bar{x}z$$

Le phénomène d'extrusion de variable (*scope extrusion*): quand un processus envoie un nom privé à un processus externe:

$$\nu y:t.(\bar{x}y.P) \mid x(z:t').Q \hookrightarrow \nu y:t.(P \mid Q[y/z]).$$

Le nom privé y a été passé au processus externe Q. P a "extrudé" le champ du canal privé y (procédé similaire au passage par référence dans un langage impératif).

Évaluation

Jugement $P \xrightarrow{a} Q$: P s'évalue en Q, en produisant l'action a.
$F(t)$ (resp. $B(t)$) désigne l'ensemble des var libres (resp. liées) de t.
Sémantique "*Early*":

input : $$x(y:t).P \xrightarrow{xz} P\{z/y\}$$

output : $$\bar{x}y.P \xrightarrow{\bar{x}(y:t)} P$$

com : $$\frac{P \xrightarrow{xy} P' \qquad Q \xrightarrow{\bar{x}(y:t)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \frac{P \xrightarrow{xy} P' \qquad Q \xrightarrow{\bar{x}(y:t)} Q'}{Q \mid P \xrightarrow{\tau} Q' \mid P'}$$

open : $$\frac{P \xrightarrow{\bar{x}(y:t)} P'}{\nu y:t.P \xrightarrow{\bar{x}(y:t)} P'} \quad x \neq y$$

close : $$\frac{P \xrightarrow{xy} P' \qquad Q \xrightarrow{\bar{x}(y:t)} Q'}{P \mid Q \xrightarrow{\tau} \nu y:t.(P' \mid Q')} \quad y \notin F(P) \qquad \frac{P \xrightarrow{xy} P' \qquad Q \xrightarrow{\bar{x}(y:t)} Q'}{Q \mid P \xrightarrow{\tau} \nu y:t.(Q' \mid P')} \quad y \notin F(P)$$

res : $$\frac{P \xrightarrow{a} P'}{\nu x:t.P \xrightarrow{a} \nu x:t.P'} \quad x \notin Var(a)$$

par : $$\frac{P \xrightarrow{a} P'}{P \mid Q \xrightarrow{a} P' \mid Q} \quad B(a) \cap F(Q) = \emptyset \qquad \frac{P \xrightarrow{a} P'}{Q \mid P \xrightarrow{a} Q \mid P'} \quad B(a) \cap F(Q) = \emptyset$$

bang : $$\frac{!P \mid P \xrightarrow{a} P'}{!\,P \xrightarrow{a} P'}$$

Exemples de communication

$$\frac{P \xrightarrow{\bar{x}(y:t)} P' \qquad Q \xrightarrow{xy} Q'}{\bar{x}y.P \mid x(z:t).Q \xrightarrow{\tau} P' \mid Q'\{y/z\}}$$

$$\frac{0 \hookrightarrow 0 \qquad \bar{u}v \hookrightarrow \bar{u}v}{\bar{x}y \mid x(u:t).\bar{u}v \mid \bar{x}z \xrightarrow{\tau} 0 \mid \bar{y}v \mid \bar{x}z}$$

$$\frac{\ldots}{\bar{x}y \mid x(u:t).\bar{u}v \mid \bar{x}z \xrightarrow{\tau} \bar{x}y \mid \bar{z}v \mid 0}$$

$$\frac{\dfrac{\bar{x}y.P \xrightarrow{\bar{x}(y:t)} P}{\nu y:t.(\bar{x}y.P) \xrightarrow{\bar{x}(y:t)} P} \qquad x(z:t').Q \xrightarrow{xy} Q[y/z]}{\nu y:t.(\bar{x}y.P) \mid x(z:t').Q \xrightarrow{\tau} \nu y:t.(P \mid Q[y/z])}$$

Types

Le type t d'un nom de canal est l'union de sa *capacité*, représentant la manière dont le canal sera utilisé (read, write or both) et du *type* des noms qui peuvent transiter sur lui:

$$\begin{array}{ll} \text{Capacités:} & c ::= \mathit{read} \mid \mathit{write} \mid \mathit{both} \\ \text{Types:} & t ::= (c, t) \end{array}$$

Relation de *sous-typage* sur les capacités:

$$c \leq c \mid \mathit{both} \leq c$$

Relation de *sous-typage* sur les types:

$$\begin{array}{c} \langle s\rangle = \mathit{both} \ \wedge \langle t\rangle = \mathit{both} \ \wedge [s] \leq [t] \wedge [t] \leq [s] \wedge s \leq t \\ \langle s\rangle \leq \mathit{read} \ \wedge \langle t\rangle = \mathit{read} \ \wedge [s] \leq [t] \Rightarrow s \leq t \\ \langle s\rangle \leq \mathit{write} \wedge \langle t\rangle = \mathit{write} \wedge [t] \leq [s] \Rightarrow s \leq t \end{array}$$

Un processus est *typé dans un environnement particulier*, associant un type à chacun de ses noms libres.

Règles de typage

$$typ\_nil : \quad \Gamma \vdash 0$$

$$typ\_input : \quad \frac{\Gamma(x) \leq s^r \quad \langle \Gamma(x) \rangle \leq read \quad [\Gamma(x)] \leq s \quad \forall z \notin Var(P)\ \Gamma.(z : [\Gamma(x)]) \vdash P\{z/y\}}{\Gamma \vdash x(y : s).P}$$

$$typ\_output : \quad \frac{\langle \Gamma(x) \rangle \leq write \qquad \Gamma(y) \leq [\Gamma(x)] \qquad \Gamma \vdash P}{\Gamma \vdash \bar{x}y.P}$$

$$typ\_res : \quad \frac{\forall y \notin Var(P)\ \Gamma.(y : t) \vdash P\{y/x\}}{\Gamma \vdash \nu x : t.P}$$

$$typ\_par : \quad \frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q}$$

$$typ\_bang : \quad \frac{\Gamma \vdash P}{\Gamma \vdash !\, P}$$

Propriétés du typage

Weakning: $\Gamma \vdash P \Rightarrow \forall s.\ \Gamma, a : s \vdash P$

Narrowing: $(\Gamma, a : s \vdash P \wedge t \leq s) \Rightarrow \Gamma, a : t \vdash P$

Subst: $(\Gamma, b : s \vdash P \wedge \Gamma(c) \leq s) \Rightarrow \Gamma \vdash P[c/b]$

Subject Reduction Property: $(P \hookrightarrow P' \wedge \Gamma \vdash P) \Rightarrow \Gamma \vdash P'$

Extensions

Calculs à objets: σ-calculs d'Abadi-Cardelli.
Calculs concurrents et à objets: basés sur le π-calcul et le σ-calcul: Join, Pict, calcul bleu, ...
Langages concurrents et à objets: Join, Pict, OCaml, ...
Embients de Cardelli pour spécifier les processus mobiles.

Quelques problèmes d'actualité:

- dans les langages à noyau fonctionnel: polymorphisme, héritage et sous-typage,
- en général: trouver le "bon" système de types,
- trouver un "bon" langage de programmation pour les applications réparties,
- formaliser Java, Java-card et des preuves de propriétés de sécurité.

Bibliographie

References

[Des00] Joëlle Despeyroux. A higher-order specification of the pi-calculus. In *Proc. of the IFIP International Conference on Theoretical Computer Science, IFIP TCS'2000, Sendai, Japan, August 17-19, 2000.*, August 2000. A preliminary version was presented at the Modelisation and Verification seminar, Marseille, Dec 98.

[FGL+96] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *Proc. of the CONCUR conference*, 1996.

[Mil91] Robin Milner. The polyadic pi-calculus: a tutorial. Technical Report ECS-LFCS-91-180, LFCS, Edinburgh University, October 1991.

[MPW92] R. Milner, R. Parrow, and J. Walker. A calculus of mobile processes, (part I and II). *Information and Computation*, 100:1–77, 1992.

[PS95] Benjamin Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 11, 1995.

[PT97] Benjamin Pierce and David Turner. Pict: A programming language based on the pi-calculus. Technical Report ., IU, 1997.