HOCore en Coq : résumé

Aurèle Barrière, sur un article de Petar Maskimovic & Alan Schmitt

13 février 2016

Table des matières

1 Introduction à HOCore

HOCore est un calcul de processus, semblable au lambda-calcul, utilisé en particulier pour décrire des exécutions distribuées de processus.

-> sémantique comportementale ? je ne sais pas comment tourner ça

1.1 Syntaxe

Un processus, en HOCore, suit la grammaire suivante :

$$P ::= a(x).P \mid \bar{a}\langle P\rangle \mid P \| P \mid x \mid 0$$

On va distinguer 3 catégories : des canaux sur lesquels émettre et recevoir des messages (dans la grammaire ci-dessus, a est un canal), des variables (remplacées lors de la lecture d'un message, x dans la grammaire) et des processus (P dans la grammaire).

Pour deux processus P et Q, le processus $P \| Q$ correspond à l'éxecution en parallèle de P et Q.

1.2 Sémantique et simplifications

0 est un processus qui ne fait rien.

En HOCore, l'éxécution en parallèle est associative et commutative, et le processus 0 en est l'élément neutre.

Lorsqu'un processus émet un autre processus ou une variable sur un canal a et qu'en parallèle un autre processus lit sur le même canal a, alors on peut faire la réduction suivante :

$\bar{a}\langle P\rangle \| a(x).Q \to [P/x]Q$, qui signifie que les instances de x dans Q sont remplacées par P.

Parmi les variables, il faut donc distinguer celles qui sont dites *libres* et celles dites *liées*. Une variable est liée lorsqu'elle peut être changée par la lecture sur un canal. On peut rapprocher ces notions à celles de variables *globales* et *locales*. Dans l'exemple précédent, x était donc une variable liée puisqu'elle est remplacée à la lecture sur le canal a.

2 Exemples de processus en HOCore

2.1 Récursivité

On peut se demander s'il est possible de répliquer un processus. Et en effet, HOCore permet de décrire des exécutions infinies.

En se donnant un processus P, on cherche donc un processus $!P$ tel que $!P \to {}^* P \| !P$, où $\to {}^*$ signifie que le processus $!P$ se réduit en un certain nombres d'opérations à $P \| !P$.

Si on prend

$$!P = (r(X).(X \| \bar{r}\langle X\rangle)) \parallel \bar{r}\langle P \| r(X).(X \| \bar{r}\langle X\rangle)\rangle$$

À droite, on émet un processus sur le canal r, et donc on va remplacer à gauche X par tout ce processus. Après son émission, le processus de droite se transforme en 0 puisqu'il a fini d'émettre. On a donc

$$!P \to P \| r(X).(X \| \bar{r}\langle X \rangle) \parallel \bar{r}\langle P \| r(X).(X \| \bar{r}\langle X \rangle)\rangle \| 0$$

Et on constate qu'on a $!P \to P \| !P \| 0$, et donc $!P \to P \| !P$ comme on le souhaitait, comme 0 est l'élément neutre pour la parallèlisation.

2.2 Choix de processus

On aimerait également disposer d'un processus qui choisisse entre deux processus P et Q selon si il est exécuté en parallèle d'un processus $\hat{a_1}$ ou $\hat{a_2}$.

On souhaite donc avoir le processus $(a_1.P \oplus a_2.Q)$ et les processus $\hat{a_1}$ et $\hat{a_2}$ tels que

$$(a_1.P \oplus a_2.Q) \| \hat{a_1} \to^* P$$

$$(a_1.P \oplus a_2.Q) \| \hat{a_2} \to^* Q$$

En HOCore, cela peut se faire ainsi :

$$(a_1.P \oplus a_2.Q) = \bar{a_1}\langle P \rangle \parallel \bar{a_2}\langle Q \rangle$$

$$\hat{a_1} = a_1(X).(a_2(Y).(X))$$

$$\hat{a_2} = a_1(X).(a_2(Y).(Y))$$

Ainsi, si on a en parallèle les processus $(a_1.P \oplus a_2.Q)$ et $\hat{a_1}$, le premier va émettre P et Q sur les canaux a_1 et a_2, qui seront lus par le second, qui lui même ne va exécuter aprs lecture que le processus émis sur a_1 : P.

2.3 Turing Completude

On a en particulier que HOCore est un calcul Turing-Complet. Cela est montré dans l'article ... en encodant les mchines de Minsky dans HOCore.

3 Équivalence de processus

En HOCore, l'équivalence de processus est décidable. Pour comprendre cela, il convient de définir ce que sont deux processus équivalents.

Deux processus P et Q sont dits équivalents si :

— S'il existe une réduction de P à un processus P', il existe Q' tel qu'il y a une réduction de Q à Q'.

— P et Q ont les mêmes *observables* : ils émettent des messages sur les mêmes canaux.

— Pour tout contexte C (un processus avec un trou), le contexte complété avec P, $C[P]$, est équivalent à $C[Q]$.

Une des particularités de HOCore (contrairement au π-calcul par exemple), est que les processus ne peuvent pas *cacher* des variables. On peut donc choisir un contexte qui explore le processus et ses actions sur les canaux et les variables.

4 Formalisation en Coq

Un des principaux travaux de l'équipe de recherche a été de formaliser HOCore en Coq (l'assistant de preuve). La grammaire relativement simple d'HOCore se traduit simplement en Coq, cependant des problèmes subsistent : il faut pouvoir reconnaître les variables liées dont le rôle est identique. En effet, deux processus peuvent s'écrire différemment mais être équivalents.

Par exemple, $a(X).(P\|X)$ et $a(Y).(P\|Y)$ sont équivalents, mais ne s'écrivent pas rigoureusement de la même manière.

4.1 Alpha-conversion et représentation canonique locale des noms

Dans l'exemple précédent, on parle d'alpha-conversion si deux processus ont des variables liées dont le nom a été changé. Cependant, il faut bien distingeur variables et liées et variables libres. On ne peut pas parler d'alpha-conversion pour des variables libres : le contexte peut utiliser ces variables libres et on pourrait alors perdre l'équivalence.

-> exemple

Si on veut pouvoir décider, avec Coq par exemple, de l'équivalence de deux processus, cela peut être un problème.

Pour le contourner, on peut entièrement se dispenser des noms de variables : on va disposer d'une fonction qui pour chaque variable va calculer une hauteur de manière identique et indépendante du nom. Cette approche est celle de *représentation canonique locale des noms* introduite dans l'article ... .

On n'a plus alors d'alpha-conversion possible : les variables ne sont plus que des indices, qui seront égaux dans le cas de processus équivalents.

4.2 Système de transitions étiquetées

Pour pouvoir analyser un processus, trouver les communications entre les différents processus en parallèles sans modifier complètement sa syntaxe, on introduit une nouvelle forme de sémantique : un système de transitions étiquetées ou LTS (*labeled transition system*).

En effet, notre réduction pour la communication se formalise ainsi :

$$\bar{a}\langle P\rangle\|a(x).Q \to [P/x]Q$$

Cependant, on sait qu'on a défini la parallèlisation comme étant transitive. On peut donc se retrouver dans le cas

$$\bar{a}\langle P\rangle\|R\|S\|T\|U\|a(x).Q$$

Comment alors repérer la communication entre processus sans changer toute la syntaxe pour se retrouver avec le processus récepteur juste après le processus émetteur ?

Le principe du LTS est d'indiquer le comportement de chaque processus et les réductions possibles avec celui-ci.

4.3 Correction de preuves

Un des avantages de formaliser avec Coq HOCore a été de repérer des fautes dans des démonstrations.

Par exemple, une des preuves raisonne inductivement sur des tailles de processus mais en utilisant une structure différente de HOCore.

Dans une autre preuve, on affirme implicitement que la décomposition première d'un processus en forme normale reste en forme normale alors qu'il y a des contre-exemples.

Certines erreurs peuvent amener à redéfinir une notion pour pouvoir rester cohérent avec le reste des travaux.

Ces erreurs sont faciles à commettre à la main lorsque la complexité de la preuve en cache les subtilités, et refaire ces preuves en Coq garantit leur validité. Il s'agit cependant d'une grande partie du travail à effectuer : si la formalisation de Coq a nécessité 4000 lignes de code, les preuves s'étendent sur 22000 lignes.

5 Conclusion

Ancienne version

6 Introduction à HOCore

6.1 Pi-calcul

Le π-calcul est un langage formel utilisé pour décrire, en particulier, les éxécutions distribuées de processus. Sa syntaxe, très simple, décrit simplement l'éxécution en parallèle.

En π-calcul , on manipule des processus, qui peuvent s'éxécuter séquentiellement ou parallèlement et terminer ou non. Des canaux sont également disponibles pour la réception et l'émission de messages ou de variables.

Le π-calcul utilise donc la grammaire suivante :

$P = 0$	fin du processus
$\|!P$	répéter le processus
$\|P\|\|P$	lancer les deux processus en parallèle
$\|x(y).P$	lire un message sur le canal x pour remplacer y, puis lancer P
$\|\bar{x}(y).P$	envoyer le message y sur le canal x, puis lancer P
$\|(\nu x)P$	réserver le nom x pour le processus P

Il s'agit d'un calcul Turing Complet.

6.2 Pi-calcul d'ordre supérieur : HOPi

Pour l'ordre supérieur, on se permet de communiquer par les canaux aussi bien des noms (variables) que des processus.

Dans la grammaire proposée plus haut, x et y peuvent donc désigner des processus.

6.3 HOCore

Il s'agit d'une restriction qui conserve le caractère Turing Complet du π-calcul d'ordre supérieur. Il s'agit d'une restriction minimale. On peut le voir aussi comme du λ-calcul autorisant les calculs parallèles.

Les travaux de l'équipe se basent sur un premier arcticle : *On the Expressiveness and Decidability of Higher-Order Process Calculi*, dans lequel est définie la syntaxe de HOCore. On y montre, entre autres, la Turing complétude.

La grammaire utilisée est la suivante :

$P = 0$
$|x$
$|P||P$
$|a(x).P$ (à la lecture d'une variable y sur a, toutes les instances de x dans P seront remplacées par des y)
$|\bar{a}(P)$

On va distinguer 3 catégories : des canaux sur lesquels émettre et recevoir des messages, des variables (remplacées lors de la lecture d'un message) et des processus.

Parmi les variables, il faut distinguer celles qui sont dites *libres* et celles dites *liées*. Une variable est liée lorsqu'elle peut être changée par la lecture sur un canal.

En HOCore, on utilise un système de transitions labelées pour décrire l'éxécution des processus. On utilise soit une étiquette de la forme $\bar{a}(P)$ pour une émission, $a(P)$ pour une réception de processus, ou τ pour une transition interne : par exemple lorsqu'on a simultanément une émission et une réception sur un même canal.

6.4 Exemples de processus en HOCore

Exemple de substitution $\bar{a}(P)||a(x).Q \to [P/x]Q$, qui signifie que les instances de x dans Q sont ramplacées par P.

Exemple de variables liées et libres $a(x).(P||y)$. Ici, les occurences de x dans P sont liées alors que y est libre.

6.5 Réductions

Lorsqu'un processus attend un message sur un canal et qu'en parallèle, un autre processus émet un message sur ce même canal, on remplace toutes les instances de la variable.

7 Équivalence décidable

On peut montrer que le problème de décision de l'équivalence de 2 processus est décidable.

Cependant, le problème de terminaison reste indécidable.

On dit que deux processus sont équivalents si leur comportement est identique, quel que soit le contexte. On définit ainsi la *congruence barbue* : il s'agit de la plus grande relation d'équivalence (entre processus), notée $\simeq$, telle que :

- elle est stable par réduction. $P \simeq Q$, $P \to^\tau P'$ et $Q \to^\tau Q'$ impliquent $P' \simeq Q'$.
- stable par contexte. Si C est un contexte (*i.e.* un processus avec un trou) et $P \simeq Q$, on a $C[P] \simeq C[Q]$.
- Si $P \simeq Q$, P et Q ont les mêmes observables : si P émet un processus sur un canal pour devenir un autre processus, il existe pour Q une transition qui émet sur le même canal un processus.

Le fait qu'en HOCore on ne puisse pas réserver des variables à des processus rend la congruence barbue décidable : on peut explorer le comportement d'un processus avec des contextes bien choisis.

8 Alpha-conversion

Le nom donné aux variables n'importe pas dans la sémantique d'un processus, mais pose un problème pour l'équivalence de processus.

9 Formalisation en Coq

Un des principaux travaux de l'équipe de recherche a été de formaliser HOCore en Coq (l'assistant de preuve).

9.1 Axiomatisation et noms de variables

On peut facilement traduire la grammaire de HOCore en Coq. Cependant, des problèmes subsistent : il faut pouvoir reconnaître les variables liées dont le rôle est identique (alpha-conversion). Deux processus peuvent s'écrire différemment mais être équivalents s'ils utilisent des noms de variables différents.

Une première solution est d'utiliser l'indice de De Bruijn.

L'approche choisie est celle du *nom local* de Pollack et al. dans *A canonical locally named representation of binding*.

Le but est de ne pas essayer de faire de l'alpha-conversion, mais plutôt d'identifier chaque variable par un poids.

Ainsi, parmi les variables, il faut distinguer celles qui sont dites *libres* et celles dites *liées*. Une variable est liée pour un processus P lorsqu'elle peut être changée par la lecture sur un canal dans P.

9.2 Expression des transitions

HOCore utilise un systeme de transition labelées (LTS). Il utilise 3 types de transition : pour l'émission et la réception sur un canal, ou une transition interne.

Mais ce systme utilise donc le nom des variables liées : ce qui pose encore le problème de l'alpha-conversion.

10 Bissimilarités

11 Correction de preuves

Un des avantages de formaliser avec Coq HOCore a été de repérer des fautes dans des démonstrations.

Par exemple, une des preuves raisonne inductivement sur des tailles de processus mais en utilisant une structure différente de HOCore.

Dans une autre preuve, on affirme implicitement que la décomposition première d'un processus en forme normale reste en forme normale alors qu'il y a des contre-exemples.

Certines erreurs peuvent amener à redéfinir une notion pour pouvoir rester cohérent avec le reste des travaux.

Ces erreurs sont faciles à commettre à la main lorsque la complexité de la preuve en cache les subtilités, et refaire ces preuves en Coq garantit leur validité. Il s'agit cependant d'une grande partie du travail à effectuer : si la formalisation de Coq a nécessité 4000 lignes de code, les preuves s'étendent sur 22000 lignes.

Références

[1] I. Lanese, J. Pérez, D. Sangiorgi, and A. Schmitt. On the expressiveness and decidability of higher-order process calculi.

[2] Petar Maksimovic and Alan Schmitt. Hocore in coq. Aug 2015.

[3] R. Pollack, M. Sato, and W. Riciotti. A canonical locally named representation of binding.

[4] D. Pous and A. Schmitt. De la kam avec un processus d'ordre supérieur.