HOCore en Coq : résumé

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

14 février 2016

1 Introduction

L'article résumé ici traite de la formalisation en Coq de HOCore, un calcul de processus (similaire au lambda-calcul). L'enjeu est de contourner les problèmes qui se posent lors de la formalisation d'un tel calcul, puis de construire des preuves sur ce calcul. Certaines de ces preuves ayant déjà été faites à la main avant, on pourra ainsi comparer les deux méthodes.

On présentera donc une introduction à HOCore, accompagnée d'exemples et de propriétés, comme la décidabilité de l'équivalence de processus. On présentera ensuite les problèmes qui se posent pour formaliser le calcul et ses réductions en Coq. Enfin, on s'intéressera aux effets de ce travail.

2 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

-> il faudrait peut etre des exemples d'applications

2.1 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.

2.2 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. Elle permet la communication entre processus sur les canaux.

En effet, 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 de réduction précédent, x était donc une variable liée puisqu'elle est remplacée à la lecture sur le canal a. Par contre, dans le processus $P \| x$, x est une variable libre : elle ne va pas être remplacée.

Enfin, dans le processus $\bar{a}\langle P\rangle \| a(x).Q \| x$, qui se réduit donc en $[P/x]Q \| x$, il faut distinguer x dans Q qui est une variable liée puisque remplacée par P, et x éxécuté en parallèle, libre.

3 Exemples de processus en HOCore

3.1 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.

3.2 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 après lecture que le processus émis sur a_1 : P.

3.3 Turing Completude

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

4 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.

5 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.

5.1 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 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.

5.2 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.

5.3 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.

6 Conclusion

Le travail réalisé a donc consisté à formaliser la syntaxe d'HOCore en Coq, puis sa sémantique en permettant à l'assistant de preuve d'effectuer des réductions. On doit alors utiliser des procédés comme par exemple l'approche canonique locale des noms pour reconnaître les variables locales identiques. On arrive ainsi à avoir une procédure pour montrer l'équivalence de deux processus. Ensuite, de nombreuses preuves sur ce calcul ont été faites en Coq. La vérification de Coq a permis de détecter des erreurs dans des preuves faites à la main, ou des définitions mal posées. Ce travail a également amélioré les connaissances sur HOCore et corrigé les erreurs dans les preuves.

Il reste cependant des preuves à traduire en Coq et des résultats à formaliser.

À la connaissance des auteurs, il s'agit de la première formalisation d'un π-calcul d'ordre supérieur.

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.