HOCore en Coq : résumé

Aurèle Barrière

5 février 2016

Table des matières

1 Introduction à HOCore

1.1 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.

1.2 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.

1.3 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.

1.4 Exemples de processus en HOCore

Exemple de substitution $\bar{a}(P)||a(x).Q \rightarrow [P/x]Q$, 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.

1.5 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.

2 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. $P \simeq Q$, $P \rightarrow^{\tau} P'$ et $Q \rightarrow^{\tau} Q'$ 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.

3 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.

4 Formalisation en Coq

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

4.1 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 hoisie 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.

4.2 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.

5 Bissimilarités

6 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.