Comet – exercices 10

Apportez votre réponse au cours du 23 novembre.

1 Équivalence réductionnelle en CCS

1.1 Moins d'observables

On se place dans le même cadre que ce qui a été vu en cours pour la définition de $\simeq$ en CCS. On définit $\simeq'$ comme $\simeq$, à ceci près que l'on ne considère que les outputs comme observables (on impose les mêmes observables pour $\downarrow_{\overline{a}}$, et on ignore $\downarrow_a$).

Démontrer que l'on a $\simeq = \simeq'$ (donnez les arguments essentiels, point besoin de se lancer dans une énorme preuve avec plein de détails).

1.2 Sans opérateur de somme

On reprend à présent les mêmes définitions qu'en cours, à ceci près que l'on considère CCS *sans l'opérateur de somme*[1]. $P ::= \mathbf{0} \mid a.P \mid \overline{a}.P \mid P_1|P_2$.

Les définitions de la congruence structurelle ($\equiv$) et de la réduction ($\rightarrow$) sont du coup plus simples:

$$P|0 \equiv P \qquad P|Q \equiv Q|P \qquad P|(Q|R) \equiv (P|Q)\mathcal{R}$$

$$a.P|\overline{a}.Q|R \rightarrow P|Q|R \qquad \frac{P_0 \equiv P \qquad P \rightarrow P' \qquad P' \equiv P'_0}{P_0 \rightarrow P'_0}$$

Rédigez la preuve du fait que l'équivalence réductionnelle ($\simeq$) est incluse dans la bisimilarité ($\sim$). Vous pouvez être raisonnablement expéditifs sur les étapes de la preuve qui sont inchangées par rapport à ce qui a été vu en cours.

2 Équivalence réductionnelle dans un langage impératif

On considère un petit langage impératif, avec affectation ($X_1 := e$), séquence ($c_1; c_2$), conditionnelle (if $e_1 \geq e_2$ then p_1 else p_2) et boucle (while $e_1 \geq e_2$ do p).

Les variables utilisées dans les programmes sont appelées $X_1, X_2, \ldots$. On utilise e, e_1, e_2 pour désigner les *expressions arithmétiques* (de la forme $X_1 + 3$, $X_1 - X_2 + 10$, ...), et p, p_1, p_2 pour les programmes.

Un tel langage est décrit dans le livre de G. Winskel, *The Formal Semantics of Programming Languages* (disponible à la bibliothèque). Il devrait être cependant possible de répondre à cette partie en se contentant d'une compréhension intuitive de la sémantique opérationnelle de ce langage, et de la notion d'équivalence réductionnelle, notée $\simeq$.

En particulier :

- Les programmes sont exécutés dans un état mémoire de départ où toutes les variables sont égales à zéro.
- Le programme while $2 = 2$ do $X := X + 0$ est un exemple de programme non terminant.
- Pour définir $\simeq$, on procède de manière similaire à ce qui a été fait en cours pour le λ-calcul, en testant $C[p] \Downarrow$, C étant un contexte de ce langage, et $\Downarrow$ désignant la terminaison du programme.

Comme d'habitude, les contextes sont des programmes avec des trous. On définit un seul observable : $p \downarrow$ signifie que l'exécution du programme p termine.

Démontrer que $p_1 \not\simeq p_2$, où p_1 est $X_1 := 2; X_2 := X_1 + X_1$, et p_2 est $X_1 := 4; X_2 := X_1$.

[1] *Au vu des difficultés rencontrées en cours à définir $\rightarrow$ avec la somme, c'est même préférable...*