Chapitre 5

Le langage PCF avec des types

Au chapitre 2, nous avons remarqué que les fonctions de PCF, contrairement aux fonctions mathématiques n'avaient pas de domaine de définition. De ce fait, rien n'empêche d'appliquer la fonction `fun x -> x + 1` à la fonction `fun x -> x + 2`, ce qui n'a pas grand sens, il faut bien l'admettre.

Pouvoir appliquer ainsi n'importe quel objet à n'importe quel autre est parfois pratique. Par exemple, on peut appliquer la fonction identité `fun x -> x` à elle-même, ce qui donne le terme `(fun x -> x) (fun x -> x)` qui se réduit sur `fun x -> x`. Plus généralement la fonction identité de PCF est définie sur l'univers entier, alors qu'en mathématiques, on doit toujours la restreindre à un domaine particulier. Par ailleurs, l'application d'un objet à lui-même nous a été utile pour montrer que la construction `fix` pouvait être simulée en PCF avec l'application et le `fun` uniquement — voir l'exercice 2.10.

Cependant, pouvoir appliquer n'importe quel objet à n'importe quel autre pose un certain nombre de problèmes. Par exemple, nous avons vu que les termes `1 2`, `1 + (fun x -> x)`, `ifz (fun x -> x) then 1 else 2` étaient clos et irréductibles pour les règles de la sémantique opérationnelle à petits pas, bien que ce ne soient pas des valeurs.

La sémantique opérationnelle à grands pas, quant à elle, n'attribue pas de résultat à un terme comme `(fun x -> x) 1 2`. En pratique, quand on interprète un terme de la forme `t u` et que `t` s'interprète sur une constante entière et non un terme de la forme `fun`, on déclenche une erreur. Cette erreur se produit à l'exécution et il est raisonnable de se demander si on n'aurait pas pu la détecter statiquement, c'est-à-dire en examinant le programme avant de l'exécuter.

Par ailleurs, le fait que les fonctions n'aient pas de domaine de définition rend la définition d'une sémantique dénotationnelle pour PCF assez difficile.

L'objet de ce chapitre est de construire une variante de PCF dans laquelle les fonctions ont un domaine de définition, de montrer que quand un programme est bien formé dans ce langage, son interprétation ne peut pas produire les