## 1.3 Trois manières de définir la sémantique d'un langage

La sémantique d'un langage de programmation est une relation binaire sur l'ensemble des termes de ce langage. Maintenant que nous avons défini la notion de langage et introduit des outils pour définir des relations, nous pouvons commencer à présenter les trois grandes manières de définir la sémantique d'un langage : comme une fonction, par une définition inductive et comme la fermeture réflexive-transitive d'une relation explicitement définie. On parle alors de la sémantique *dénotationnelle*, de la sémantique *opérationnelle à grands pas* et de la sémantique *opérationnelle à petits pas*.

### 1.3.1 La sémantique dénotationnelle

La sémantique dénotationnelle est surtout utile pour les programmes déterministes. Dans ce cas, pour chaque programme `p`, la relation qui lie entrées et sorties du programme est une fonction, que l'on note $[\![\texttt{p}]\!]$. La relation $\hookrightarrow$ se définit donc par

$$\texttt{p, e} \hookrightarrow \texttt{s} \text{ si et seulement si } [\![\texttt{p}]\!]\ \texttt{e} = \texttt{s}$$

Bien entendu, cela ne fait que repousser le problème un peu plus loin : sur la manière dont on définit la fonction $[\![\texttt{p}]\!]$. Pour définir cette fonction, nous utiliserons deux outils : les définitions explicites de fonctions et le théorème du point fixe... mais laissons cela pour le moment.

### 1.3.2 La sémantique opérationnelle à grands pas

La sémantique opérationnelle à grands pas s'appelle aussi *sémantique opérationnelle structurée* — S.O.S. — ou *sémantique naturelle*. Elle consiste à donner une définition inductive de la relation $\hookrightarrow$.

### 1.3.3 La sémantique opérationnelle à petits pas

La sémantique opérationnelle à petits pas s'appelle aussi *sémantique par réécriture*. Elle consiste à définir la relation $\hookrightarrow$ à partir d'une autre relation $\triangleright$ qui décrit les étapes élémentaires qui mènent du terme de départ `t` à la sortie `s`.

Par exemple, quand on exécute le programme `fun x -> (x * x) + x` sur l'entrée `4`, on obtient le résultat `20`. Mais le terme `(fun x -> (x * x) + x) 4` ne se transforme pas en `20` en une seule étape, il se transforme d'abord en `(4 * 4) + 4`, puis en `16 + 4`, et enfin en `20`.

La relation importante n'est pas la relation qui lie le terme `(fun x -> (x * x) + x) 4` au terme `20`, mais la relation $\triangleright$ qui lie le terme `(fun x -> (x * x) + x) 4` au terme `(4 * 4) + 4`, puis le terme `(4 * 4) + 4` au terme `16 + 4` et enfin le terme `16 + 4` au terme `20`.