

TD n°6 - Correction

Sémantique du langage IMP

Exercice 1 Les expressions booléennes suivantes sont des abréviations. Écrivez-les en forme étendue.

1. $(5 > x) \vee (y = 3)$
2. $(7 \leq 3) \rightarrow (x = y)$
3. $(x \leq y) \vee (x \neq z)$

Interprétez ces expressions par rapport à l'affectation $\sigma = [x \mapsto 4, y \mapsto 6, z \mapsto -2]$

Correction :

1. $(x + 1 \leq 5) \vee \neg(\neg(y \leq 3) \vee \neg(3 \leq y))$, interprétation 1.
2. $\neg(7 \leq 3) \vee \neg(\neg(x \leq y) \vee \neg(y \leq x))$, interprétation 1.
3. $(x \leq y) \vee (x + 1 \leq z) \vee (z + 1 \leq x)$, interprétation 1.

Exercice 2 Le langage IMP contient l'instruction **if** b **then** p_1 **else** p_2 **fi**. Une instruction similaire existe en Java, mais sans **then** ni **fi**. Pourquoi ?

Écrivez en Java un fragment de programme qui corresponde au programme IMP suivant :

```
z := 0;
y1 := 0;
while y1 ≠ y do
  z := z + x;
  y1 := y1 + 1;
od
```

Correction : L'instruction conditionnelle en Java a la syntaxe suivante : **if** (C) I_1 **else** I_2 , où C est une expression booléenne (ne pas oublier les parenthèses qui l'entourent) et où I_1 et I_2 sont formées soit d'une instruction soit d'un bloc d'instructions à l'intérieur d'une paire d'accolades.

La présence des parenthèses autour de la condition C rend superflue le mot-clé **then**.

fi n'est pas nécessaire grâce aux accolades (ou au point-virgule si I_2 est une seule instruction).

```
int x,y,y1,z;
...
z = 0;
y1 = 0;
while (y1!=y) {
  z = z+x;
  y1 = y1 + 1;
}
```

Exercice 3 Exécutez à la main les configurations suivantes.

1. $\langle y := x; x := 2; z := x * y, [x \mapsto 4] \rangle$
2. $\langle x := 0; y := 1; \text{if } x > y \text{ then } z := x \text{ else } z := y \text{ fi}, [] \rangle$
3. $\langle \text{while } x > 0 \text{ do } x := x + y; y := y + 1 \text{ od}, [x \mapsto 3, y \mapsto 5] \rangle$

Correction :

$$\begin{aligned}
& \langle y := x; x := 2; z := x * y, [x \mapsto 4] \rangle \\
\Rightarrow & \langle x := 2; z := x * y, [x \mapsto 4; y \mapsto 4] \rangle \\
\Rightarrow & \langle z := x * y, [x \mapsto 2; y \mapsto 4] \rangle \\
\Rightarrow & \langle \epsilon, [x \mapsto 2; y \mapsto 4; z \mapsto 8] \rangle
\end{aligned}$$

$$\begin{aligned}
& \langle x := 0; y := 1; \text{if } x > y \text{ then } z := x \text{ else } z := y \text{ fi}, [] \rangle \\
\Rightarrow & \langle y := 1; \text{if } x > y \text{ then } z := x \text{ else } z := y \text{ fi}, [x \mapsto 0] \rangle \\
\Rightarrow & \langle \text{if } x > y \text{ then } z := x \text{ else } z := y \text{ fi}, [x \mapsto 0; y \mapsto 1] \rangle \\
\Rightarrow & \langle z := y, [x \mapsto 0; y \mapsto 1] \rangle \\
\Rightarrow & \langle \epsilon, [x \mapsto 0; y \mapsto 1; z \mapsto 1] \rangle
\end{aligned}$$

$$\begin{aligned}
& \langle \text{while } x > 0 \text{ do } x := x + y; y := y + 1 \text{ od}, [x \mapsto 3, y \mapsto 5] \rangle \\
\Rightarrow & \langle x := x + y; y := y + 1; \text{while } x > 0 \text{ do } x := x + y; y := y + 1 \text{ od}, [x \mapsto 3, y \mapsto 5] \rangle \\
\Rightarrow & \langle y := y + 1; \text{while } x > 0 \text{ do } x := x + y; y := y + 1 \text{ od}, [x \mapsto 8, y \mapsto 5] \rangle \\
\Rightarrow & \langle \text{while } x > 0 \text{ do } x := x + y; y := y + 1 \text{ od}, [x \mapsto 8, y \mapsto 6] \rangle \\
\Rightarrow & \langle x := x + y; y := y + 1; \text{while } x > 0 \text{ do } x := x + y; y := y + 1 \text{ od}, [x \mapsto 8, y \mapsto 6] \rangle \\
\Rightarrow & \dots
\end{aligned}$$

Le calcul ne s'arrête pas.

Exercice 4 On imagine une nouvelle instruction "Arrêt l'exécution" :

abort

Donnez la règle (ou les règles) de transition de cette instruction

Correction : Une solution possible est

$$\langle \text{abort}; p, \sigma \rangle \Rightarrow \langle \epsilon, \sigma \rangle$$

Exercice 5 On imagine une nouvelle instruction "Répète un certain nombre de fois" :

do e times p od

Donnez la règle (ou les règles) de transition de cette instruction

Correction : Une solution possible est :

$$\langle \text{do } e \text{ times } p' \text{ od}; p, \sigma \rangle \Rightarrow \begin{cases} \langle p, \sigma \rangle & \text{si } \llbracket e \rrbracket \sigma \leq 0 \\ \langle p' @ \text{do } n \text{ times } p' \text{ od } @ p, \sigma \rangle & \text{si } \llbracket e \rrbracket \sigma = n + 1 \end{cases}$$

Une autre solution possible est :

$$\langle \text{do } e \text{ times } p' \text{ od}; p, \sigma \rangle \Rightarrow \begin{cases} \langle p, \sigma \rangle & \text{si } \llbracket e \rrbracket \sigma \leq 0 \\ \langle p' @ \text{do } e - 1 \text{ times } p' \text{ od } @ p, \sigma \rangle & \text{si } \llbracket e \rrbracket \sigma > 0 \end{cases}$$

Quelle est la différence entre ces solutions ?

Exercice 6 Qu'est-ce qu'il faudrait changer si on voulait ajouter au langage une instruction **affiche** (e) ?

Correction : Il faudrait ajouter une sortie standard dans la définition d'une configuration (Définition 22). On considérerait l'ensemble des chaînes de caractères Str muni de l'opération de concaténation $@$. Le nouvel ensemble des configurations serait $Conf = Imp \times Aff \times Str$. Une configuration serait alors un triplet $\langle p, \sigma, s \rangle$ où s désigne la sortie standard.

Il resterait alors à modifier la définition de la relation \Rightarrow (Définition 23) en ajoutant la règle

$$\langle \mathbf{affiche}(e); p, \sigma, s \rangle \Rightarrow \langle p, \sigma, s @ \llbracket e \rrbracket \sigma \rangle$$

et en adaptant les règles précédentes pour tenir compte du rajout de la sortie standard dans les configurations.

Note : si on avait mis $s @ e$ au lieu de $s @ \llbracket e \rrbracket \sigma$ dans la règle, le programme $x := 0; \mathbf{affiche}(x)$ afficherait la chaîne de caractères x et non 0.