

INF889J Semaine 4

Le parallélisme et le non déterminisme

PROBLÈME 1. Ajoutez une composition parallèle de commandes au langage impératif de l'annexe A :

$$c ::= \dots \\ | \text{ par}(c_1; c_2) \qquad c_1 \text{ par } c_2$$

Cet opérateur exécute les commandes c_1 et c_2 en parallèle selon un modèle de cohérence stricte. En d'autres termes, $\text{par}(c_1; c_2)$ exécute les commandes c_1 et c_2 en parallèle en sorte que leurs accès mémoire (lecture ou écriture) sont ordonnés par une relation d'ordre total. Donnez-y une sémantique à petites étapes dans le style de la sémantique de l'annexe. Expliquez pourquoi il est plus difficile d'y donner une sémantique à grosses étapes.

Pour vous aider, voici deux exemples à valider :

— Exécuter la commande

$$(a := 2; b := a) \text{ par } a := 3$$

à partir de l'état $[a \mapsto 0, b \mapsto 0]$ peut produire les états $[a \mapsto 2, b \mapsto 2]$, $[a \mapsto 3, b \mapsto 2]$, ou $[a \mapsto 3, b \mapsto 3]$ selon l'ordre des opérations.

— Exécuter la commande

$$a := b + 1 \text{ par } b := a + 2$$

à partir de l'état $[a \mapsto 0, b \mapsto 0]$ peut seulement produire les états $[a \mapsto 1, b \mapsto 3]$ ou $[a \mapsto 3, b \mapsto 2]$, mais jamais $[a \mapsto 1, b \mapsto 2]$.

PROBLÈME 2. Ajoutez un opérateur de choix non déterministe de commandes au langage impératif de l'annexe A :

$$c ::= \dots \\ | \text{ or}(c_1; c_2) \qquad c_1 \text{ or } c_2$$

Cette structure fait un choix non déterministe entre les commandes c_1 et c_2 , et exécute la commande choisie. Par exemple, exécuter la commande

$$(a := 2; b := a) \text{ or } (a := 3; b := a)$$

à partir de l'état $[a \mapsto 0, b \mapsto 0]$ produit de façon non déterministe l'état $[a \mapsto 2, b \mapsto 2]$ ou l'état $[a \mapsto 3, b \mapsto 3]$.

Donnez-y une sémantique à petites étapes et une sémantique à grosses étapes.

Est-ce que vos deux sémantiques captent les mêmes exécutions de commandes ? (Indice : Examinez l'interaction du choix non déterministe avec la non-terminaison).

L'équivalence des sémantiques

PROBLÈME 3. Pour le langage de l'annexe, la sémantique à grosses étapes et la sémantique à petites étapes captent les mêmes exécutions qui terminent avec succès : $\langle c, \sigma \rangle \Downarrow \sigma'$ si et seulement si $\langle c, \sigma \rangle \mapsto \langle c', \sigma' \rangle$ et $\langle c', \sigma' \rangle$ final.

Est-ce que les deux sémantiques captent les mêmes phénomènes dans le cas des exécutions de commandes qui ne terminent pas avec succès ? (Identifiez les situations où une exécution pourrait ne pas terminer avec succès, et comparez l'expressivité des sémantiques dans ces situations.)

A Le langage impératif

Soient les métavariabes $n \in \mathbb{Z} = \{ \dots, -2, -1, 0, 1, 2, \dots \}$ pour les nombres entiers ; $a, b, c, \dots \in \mathcal{A}$ pour les noms d'affectables ; et $x, y, z, \dots \in \mathcal{V}$ pour les noms de variables. Les arbres de syntaxe abstraite pour notre langage impératif sont donnés par les grammaires suivantes (colonne de gauche), où les arbres e sont des expressions et les arbres c sont des commandes.

$e ::= x$	x
$\text{get}[a]$	a
$\text{num}[n]$	n
true	true
false	false
$\text{plus}(e_1; e_2)$	$e_1 + e_2$
$\text{lt}(e_1; e_2)$	$e_1 < e_2$
$\text{if}(e_1; e_2; e_3)$	if e_1 then e_2 else e_3
$\text{let}(e_1; x.e_2)$	let $x = e_1$ in e_2
$c ::= \text{skip}$	skip
$\text{set}[a](e)$	$a := e$
$\text{seq}(c_1; c_2)$	$c_1; c_2$
$\text{cif}(e; c_1; c_2)$	if e then c_1 else c_2
$\text{while}(e; c)$	while e do c
diverge	diverge
abort	abort

A.1 Système de typage

Les types τ sont générés par la grammaire suivante :

$$\tau ::= \text{int} \mid \text{bool}$$

Les contextes Γ qui donnent les types des variables sont générés par la grammaire suivante, où $\cdot$ est le contexte vide et $x : \tau$ signifie que la variable x a le type τ :

$$\Gamma ::= \cdot \mid \Gamma, x : \tau$$

On utilise un contexte Σ pour énumérer les noms d'assignables qui sont alloués :

$$\Sigma ::= \cdot \mid \Sigma, a$$

Le jugement $\Sigma \mid \Gamma \vdash e : \tau$ veut dire que l'expression e a le type τ lorsqu'il y a les affectables Σ et lorsque les variables sont typées selon Γ . Ce jugement est bien formé seulement si chaque nom d'affectable apparaît au plus une fois dans Σ et seulement si chaque variable apparaît au plus une fois dans Γ . Le jugement $\Sigma \vdash c \text{ comm}$ signifie que c est une commande bien formée lorsqu'il y a les affectables Σ .

$\boxed{\Sigma \mid \Gamma \vdash e : \tau}$ L'expression e a le type τ avec les affectables Σ et les variables typées par Γ

$$\begin{array}{c}
\frac{(x : \tau) \in \Gamma}{\Sigma \mid \Gamma \vdash x : \tau} \text{ (T-VAR)} \quad \frac{a \in \Sigma}{\Sigma \mid \Gamma \vdash \text{get}[a] : \text{int}} \text{ (T-GET)} \quad \frac{}{\Sigma \mid \Gamma \vdash \text{num}[n] : \text{int}} \text{ (T-NUM)} \\
\frac{}{\Sigma \mid \Gamma \vdash \text{true} : \text{bool}} \text{ (T-TRUE)} \quad \frac{}{\Sigma \mid \Gamma \vdash \text{false} : \text{bool}} \text{ (T-FALSE)} \\
\frac{\Sigma \mid \Gamma \vdash e_1 : \text{int} \quad \Sigma \mid \Gamma \vdash e_2 : \text{int}}{\Sigma \mid \Gamma \vdash \text{plus}(e_1; e_2) : \text{int}} \text{ (T-PLUS)} \quad \frac{\Sigma \mid \Gamma \vdash e_1 : \text{int} \quad \Sigma \mid \Gamma \vdash e_2 : \text{int}}{\Sigma \mid \Gamma \vdash \text{lt}(e_1; e_2) : \text{bool}} \text{ (T-LT)} \\
\frac{\Sigma \mid \Gamma \vdash e_1 : \text{bool} \quad \Sigma \mid \Gamma \vdash e_2 : \tau \quad \Sigma \mid \Gamma \vdash e_3 : \tau}{\Sigma \mid \Gamma \vdash \text{if}(e_1; e_2; e_3) : \tau} \text{ (T-IF)} \\
\frac{\Sigma \mid \Gamma \vdash e_1 : \tau_1 \quad \Sigma \mid \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Sigma \mid \Gamma \vdash \text{let}(e_1; x.e_2) : \tau_2} \text{ (T-LET)}
\end{array}$$

$\boxed{\Sigma \vdash c \text{ comm}}$ La commande c est bien formée sous les affectables Σ

$$\begin{array}{c}
\frac{}{\Sigma \vdash \text{skip comm}} \text{ (C-SKIP)} \quad \frac{}{\Sigma \vdash \text{abort comm}} \text{ (C-ABORT)} \quad \frac{a \in \Sigma \quad \Sigma \mid \cdot \vdash e : \text{int}}{\Sigma \vdash \text{set}[a](e) \text{ comm}} \text{ (C-SET)} \\
\frac{\Sigma \vdash c_1 \text{ comm} \quad \Sigma \vdash c_2 \text{ comm}}{\Sigma \vdash \text{seq}(c_1; c_2) \text{ comm}} \text{ (C-SEQ)} \quad \frac{\Sigma \mid \cdot \vdash e : \text{bool} \quad \Sigma \vdash c_1 \text{ comm} \quad \Sigma \vdash c_2 \text{ comm}}{\Sigma \vdash \text{cif}(e; c_1; c_2) \text{ comm}} \text{ (C-IF)} \\
\frac{}{\Sigma \vdash \text{diverge comm}} \text{ (C-DIVERGE)} \quad \frac{\Sigma \mid \cdot \vdash e : \text{bool} \quad \Sigma \vdash c \text{ comm}}{\Sigma \vdash \text{while}(e; c) \text{ comm}} \text{ (C-WHILE)}
\end{array}$$

A.2 Sémantique à petites étapes

Soit l'ensemble $\mathbb{S}$ de fonctions partielles finies $\mathcal{A} \rightarrow_{\text{fin}} \mathbb{Z}$. Nous appelons ses éléments des **états** et nous utilisons la métavariable $\sigma \in \mathbb{S}$ pour ses éléments. On note ces éléments par la syntaxe $[a \mapsto 5, b \mapsto 6, \dots, c \mapsto 4]$, où cet état σ a l'ensemble $\{a, b, \dots, c\}$ comme domaine et est tel que $\sigma(a) = 5$, $\sigma(b) = 6$, etc. Lorsque $f : A \rightarrow B$ est une fonction, on écrit $[f \mid a \mapsto b]$ pour la fonction $A \cup \{a\} \rightarrow B \cup \{b\}$ telle que

$$[f \mid a \mapsto b](x) = \begin{cases} b & \text{si } x = a \\ f(x) & \text{sinon} \end{cases}$$

La sémantique à petites étapes pour le langage impératif est donnée par les quatre jugements suivants, tous définis par induction. En premier, on doit réduire les expressions dans la présence d'un état :

$\boxed{e \text{ val}}$ L'expression e est une valeur (peut importe l'état)

$$\frac{}{\text{num}[n] \text{ val}} \text{ (VAL-NUM)} \quad \frac{}{\text{true val}} \text{ (VAL-TRUE)} \quad \frac{}{\text{false val}} \text{ (VAL-FALSE)}$$

$\boxed{\sigma \models e \mapsto e'}$ L'expression e se réduit à e' en une étape sous l'état σ

$$\begin{array}{c}
\frac{\sigma(a) = n}{\sigma \models \text{get}[a] \mapsto \text{num}[n]} \text{ (SOS-ASN)} \\
\\
\frac{}{\sigma \models \text{plus}(\text{num}[m]; \text{num}[n]) \mapsto \text{num}[m+n]} \text{ (SOS-PLUS)} \\
\\
\frac{\sigma \models e_1 \mapsto e'_1}{\sigma \models \text{plus}(e_1; e_2) \mapsto \text{plus}(e'_1; e_2)} \text{ (SOS-PLUS-1)} \quad \frac{e_1 \text{ val } \sigma \models e_2 \mapsto e'_2}{\sigma \models \text{plus}(e_1; e_2) \mapsto \text{plus}(e_1; e'_2)} \text{ (SOS-PLUS-2)} \\
\\
\frac{m < n}{\sigma \models \text{lt}(\text{num}[m]; \text{num}[n]) \mapsto \text{true}} \text{ (SOS-LT-TRUE)} \\
\\
\frac{\neg(m < n)}{\sigma \models \text{lt}(\text{num}[m]; \text{num}[n]) \mapsto \text{false}} \text{ (SOS-LT-FALSE)} \\
\\
\frac{\sigma \models e_1 \mapsto e'_1}{\sigma \models \text{lt}(e_1; e_2) \mapsto \text{lt}(e'_1; e_2)} \text{ (SOS-LT-1)} \quad \frac{e_1 \text{ val } \sigma \models e_2 \mapsto e'_2}{\sigma \models \text{lt}(e_1; e_2) \mapsto \text{lt}(e_1; e'_2)} \text{ (SOS-LT-2)} \\
\\
\frac{}{\sigma \models \text{if}(\text{true}; e_1; e_2) \mapsto e_1} \text{ (SOS-IF-TRUE)} \quad \frac{}{\sigma \models \text{if}(\text{false}; e_1; e_2) \mapsto e_2} \text{ (SOS-IF-FALSE)} \\
\\
\frac{\sigma \models e_0 \mapsto e'_0}{\sigma \models \text{if}(e_0; e_1; e_2) \mapsto \text{if}(e'_0; e_1; e_2)} \text{ (SOS-IF)} \\
\\
\frac{e_1 \text{ val}}{\sigma \models \text{let}(e_1; x.e_2) \mapsto [e_1/x]e_2} \text{ (SOS-LET)} \quad \frac{\sigma \models e_1 \mapsto e'_1}{\sigma \models \text{let}(e_1; x.e_2) \mapsto \text{let}(e'_1; x.e_2)} \text{ (SOS-LET1)}
\end{array}$$

Ensuite, on doit expliquer comment réduire des *configurations*, c'est-à-dire, une commande à exécuter sur un état. On dit qu'une configuration est finale si sa commande a fini d'exécuter et si la configuration représente la fin d'une exécution qui réussit.

$\boxed{\langle \sigma, c \rangle \text{ final}}$ La configuration $\langle \sigma, c \rangle$ est finale et représente la terminaison avec succès

$$\frac{}{\langle \sigma, \text{skip} \rangle \text{ final}} \text{ (FIN-SKI)}$$

$\langle \sigma, c \rangle \mapsto \langle \sigma', c' \rangle$ La commande c se réduit à c' en une étape qui transforme l'état σ à σ'

$$\begin{array}{c}
\overline{\langle \sigma, \text{set}[a](\text{num}[n]) \rangle \mapsto \langle [\sigma \mid a \mapsto n], \text{skip} \rangle} \quad (\text{SOS-SET}) \\
\overline{\sigma \models e \mapsto e'} \quad (\text{SOS-SET-1}) \\
\overline{\langle \sigma, \text{set}[a](e) \rangle \mapsto \langle \sigma, \text{set}[a](e') \rangle} \\
\overline{\langle \sigma, \text{seq}(\text{skip}; c) \rangle \mapsto \langle \sigma, c \rangle} \quad (\text{SOS-SEQ}) \quad \overline{\langle \sigma, c_1 \rangle \mapsto \langle \sigma', c'_1 \rangle} \quad (\text{SOS-SEQ-1}) \\
\overline{\langle \sigma, \text{seq}(c_1; c_2) \rangle \mapsto \langle \sigma', \text{seq}(c'_1; c_2) \rangle} \\
\overline{\langle \sigma, \text{cif}(\text{true}; c_1; c_2) \rangle \mapsto \langle \sigma, c_1 \rangle} \quad (\text{SOS-CIF-TRUE}) \quad \overline{\langle \sigma, \text{cif}(\text{false}; c_1; c_2) \rangle \mapsto \langle \sigma, c_2 \rangle} \quad (\text{SOS-CIF-FALSE}) \\
\overline{\sigma \models e \mapsto e'} \quad (\text{SOS-CIF}) \\
\overline{\langle \sigma, \text{if}(e; c_1; c_2) \rangle \mapsto \langle \sigma, \text{if}(e'; c_1; c_2) \rangle} \\
\overline{\langle \sigma, \text{diverge} \rangle \mapsto \langle \sigma, \text{diverge} \rangle} \quad (\text{SOS-DIVERGE}) \\
\overline{\langle \sigma, \text{while}(e; c) \rangle \mapsto \langle \sigma, \text{cif}(e; \text{seq}(c; \text{while}(e; c)); \text{skip}) \rangle} \quad (\text{SOS-WHILE})
\end{array}$$

Une commande c termine avec succès à partir de l'état σ s'il existe un état σ' tel que $\langle c, \sigma \rangle \mapsto \langle c', \sigma' \rangle$ et $\langle c', \sigma' \rangle$ final. On dit que σ' est le résultat de l'exécution. Il n'y a pas de règle de réduction pour la commande « abort », qui représente la terminaison avec échec. Une commande c_0 a une exécution divergente à partir de l'état σ_0 s'il y a une séquence infinie d'étapes de réductions $\langle c_0, \sigma_0 \rangle \mapsto \langle c_1, \sigma_1 \rangle \mapsto \dots \mapsto \langle c_n, \sigma_n \rangle \mapsto \dots$.

A.3 Sémantique à grosses étapes

La sémantique à grosses étapes est donnée par deux jugements définis par induction. Le jugement $\sigma \models e \Downarrow v$ signifie que l'expression e s'évalue à la valeur v sous l'état σ . Le jugement $\langle \sigma, c \rangle \Downarrow \sigma'$ signifie que l'exécution de la commande c à partir de l'état σ termine avec succès et produit l'état σ' .

$\sigma \models e \Downarrow v$ L'expression e s'évalue à la valeur v sous l'état σ

$$\begin{array}{c}
\frac{}{\sigma \models \text{num}[n] \Downarrow \text{num}[n]} \text{ (BS-NUM)} \quad \frac{}{\sigma \models \text{true} \Downarrow \text{true}} \text{ (BS-TRUE)} \quad \frac{}{\sigma \models \text{false} \Downarrow \text{false}} \text{ (BS-FALSE)} \\
\\
\frac{\sigma(a) = n}{\sigma \models \text{get}[a] \Downarrow \text{num}[n]} \text{ (BS-GET)} \\
\\
\frac{\sigma \models e_1 \Downarrow \text{num}[m] \quad \sigma \models e_2 \Downarrow \text{num}[n]}{\sigma \models \text{plus}(e_1; e_2) \Downarrow \text{num}[m + n]} \text{ (BS-PLUS)} \\
\\
\frac{\sigma \models e_1 \Downarrow \text{num}[m] \quad \sigma \models e_2 \Downarrow \text{num}[n] \quad m < n}{\sigma \models \text{lt}(e_1; e_2) \Downarrow \text{true}} \text{ (BS-LT-TRUE)} \\
\\
\frac{\sigma \models e_1 \Downarrow \text{num}[m] \quad \sigma \models e_2 \Downarrow \text{num}[n] \quad \neg(m < n)}{\sigma \models \text{lt}(e_1; e_2) \Downarrow \text{false}} \text{ (BS-LT-FALSE)} \\
\\
\frac{\sigma \models e_0 \Downarrow \text{true} \quad \sigma \models e_1 \Downarrow v}{\sigma \models \text{if}(e_0; e_1; e_2) \Downarrow v} \text{ (BS-IF-TRUE)} \\
\\
\frac{\sigma \models e_0 \Downarrow \text{false} \quad \sigma \models e_2 \Downarrow v}{\sigma \models \text{if}(e_0; e_1; e_2) \Downarrow v} \text{ (BS-IF-FALSE)} \\
\\
\frac{\sigma \models e_1 \Downarrow v_1 \quad \sigma \models [v_1/x]e_2 \Downarrow v_2}{\sigma \models \text{let}(e_1; x.e_2) \Downarrow v_2} \text{ (BS-LET)}
\end{array}$$

$\langle \sigma, c \rangle \Downarrow \sigma'$ L'exécution de la commande c à partir de l'état σ termine avec succès dans l'état σ'

$$\begin{array}{c}
\frac{}{\langle \sigma, \text{skip} \rangle \Downarrow \sigma} \text{ (CBS-SKIP)} \quad \frac{\sigma \models e \Downarrow \text{num}[n]}{\langle \sigma, \text{set}[a](e) \rangle \Downarrow [\sigma \mid a \mapsto n]} \text{ (CBS-SET)} \\
\\
\frac{\langle \sigma, c_1 \rangle \Downarrow \sigma' \quad \langle \sigma', c_2 \rangle \Downarrow \sigma''}{\langle \sigma, \text{seq}(c_1; c_2) \rangle \Downarrow \sigma''} \text{ (CBS-SEQ)} \\
\\
\frac{\sigma \models e \Downarrow \text{true} \quad \langle \sigma, c_1 \rangle \Downarrow \sigma'}{\langle \sigma, \text{cif}(e; c_1; c_2) \rangle \Downarrow \sigma'} \text{ (CBS-CIF-TRUE)} \\
\\
\frac{\sigma \models e \Downarrow \text{false} \quad \langle \sigma, c_2 \rangle \Downarrow \sigma'}{\langle \sigma, \text{cif}(e; c_1; c_2) \rangle \Downarrow \sigma'} \text{ (CBS-CIF-FALSE)} \\
\\
\frac{\sigma \models e \Downarrow \text{false}}{\langle \sigma, \text{while}(e; c) \rangle \Downarrow \sigma} \text{ (CBS-WHILE-FALSE)} \\
\\
\frac{\sigma \models e \Downarrow \text{true} \quad \langle \sigma, c \rangle \Downarrow \sigma' \quad \langle \sigma', \text{while}(e; c) \rangle \Downarrow \sigma''}{\langle \sigma, \text{while}(e; c) \rangle \Downarrow \sigma''} \text{ (CBS-WHILE-TRUE)}
\end{array}$$

Il n'y a pas de règle pour les commandes « diverge » et « abort », car elles ne terminent jamais avec succès.