Plan

1. Introduction

2. Langage des expressions arithmétiques

3. Langage de programmation simple

Dans cette section, on va :

- définir la syntaxe d'un langage de programmation simple
- donner un sémantique opérationnelle
- esquisser la démonstration de son déterminisme
- donner une sémantique dénotationnelle

Un langage de programmation simple

Soit le langage dont la syntaxe (en forme BNF) est donnée ci-dessous.

$$
\begin{array}{rcl}
B \in \mathit{Bool} & ::= & \texttt{true} \mid \texttt{false} \mid E = E \mid E < E \mid \ldots \\
 & & \mid B \& B \mid \neg B \mid \ldots \\
E \in \mathit{Exp} & ::= & x \mid \texttt{n} \mid (E + E) \mid \ldots \\
C \in \mathit{Com} & ::= & x := E \mid \texttt{if } B \texttt{ then } C \texttt{ else } C \\
 & & \mid C; C \mid \texttt{skip} \mid \texttt{while } B \texttt{ do } C
\end{array}
$$

Un langage de programmation simple

Remarques :

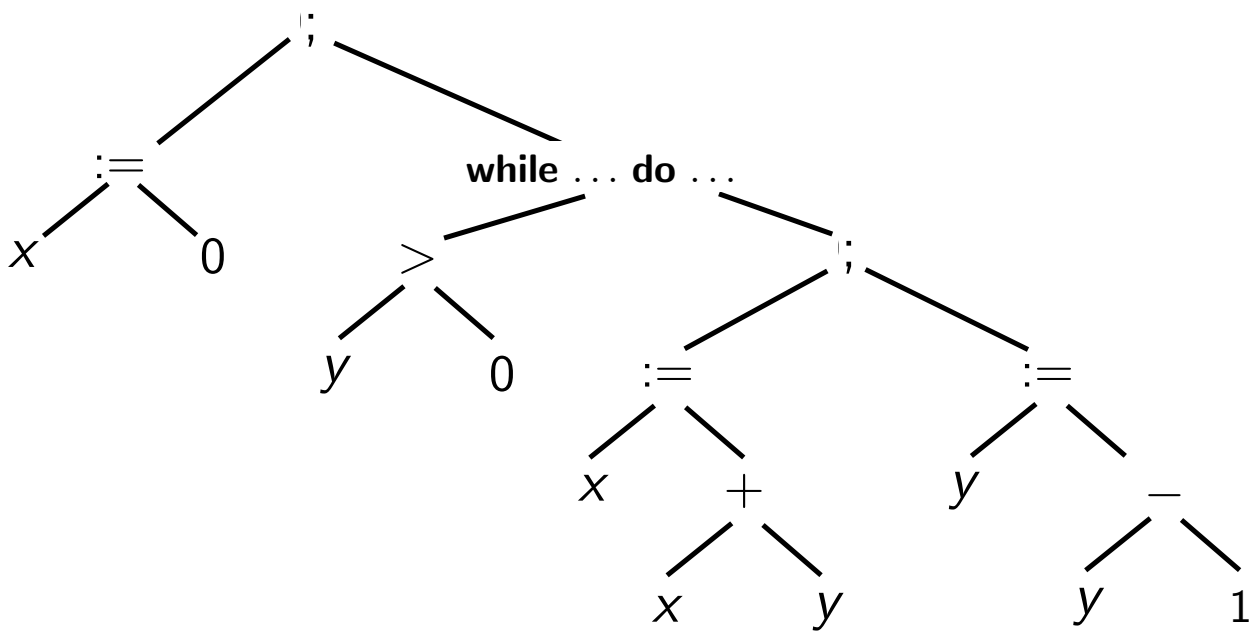
- *Exp* contient les expressions arithmétiques, *Bool* les expressions à valeurs booléennes, *Com* est l'ensemble des programmes.
- Ces trois ensembles sont définies de manière récursive et croisée.
- $\mathtt{n}$ et x sont des méta-variables de la définition. $\mathtt{n}$ désigne le numéral correspondant au naturel $n \in \mathbb{N}$. x est un identifiant de variable quelconque.
- La sémantique est la sémantique habituelle des langages impératifs.
- Comme pour les expressions, on travaillera toujours sur base de la syntaxe abstraite.

Exemple

Un programme :

$$
\begin{array}{l}
x := 0; \\
\textbf{while } y > 0 \textbf{ do} \\
\qquad x := x + y\,; \\
\qquad y := y - 1
\end{array}
$$

L'arbre syntaxique correspondant :



Sémantique opérationnelle : état

L'état de la machine évaluant un programme doit inclure :

- Le programme P ($\in Com \cup Exp \cup Bool$) restant à exécuter
- L'état de la mémoire contenant les valeurs assignées aux variables

L'état de la mémoire est modélisé par une fonction partielle s associant une valeur numérique $s(x) \in \mathbb{N}$ à un nombre fini d'identifiants x.

Soit s un état de la mémoire. L'état de la mémoire noté $s[x \mapsto n]$ est défini par :

$$s[x \mapsto n](y) = \begin{cases} \mathtt{n} & \text{si } y = x, \\ s(y) & \text{sinon.} \end{cases}$$

La sémantique opérationnelle définit des transitions du type :

$$\langle P, s \rangle \rightarrow \langle P', s' \rangle.$$

Sémantique opérationnelle : expressions

Comme précédemment, sauf pour la première règle :

$$\frac{}{\langle x, s\rangle \longrightarrow \langle \mathtt{n}, s\rangle}\ \texttt{(W-EXP.NUM)} \qquad (\text{où } s(x) = \mathtt{n})$$

$$\frac{}{\langle (\mathtt{n}_1 + \mathtt{n}_2), s\rangle \longrightarrow \langle \mathtt{n}_3, s\rangle}\ \texttt{(W-EXP.ADD)} \qquad (\text{où } n_3 = n_1 + n_2)$$

$$\frac{\langle E_1, s\rangle \longrightarrow \langle E_1', s'\rangle}{\langle (E_1 + E_2), s\rangle \longrightarrow \langle (E_1' + E_2, s'\rangle}\ \texttt{(W-EXP.LEFT)}$$

...

Note : ces règles n'ont pas d'effet sur la mémoire

(*Exercice : ajouter les autres règles pour les expression et les règles pour les booléens*)

Sémantique opérationnelle : assignation

Evaluation de la valeur à assigner :

$$\frac{\langle E, s\rangle \to \langle E', s'\rangle}{\langle x := E, s\rangle \to \langle x := E', s'\rangle} \text{ (W-ASS.EXP)}$$

Assignation proprement dite :

$$\frac{}{\langle x := \mathtt{n}, s\rangle \to \langle \mathtt{skip}, s[x \mapsto \mathtt{n}]\rangle} \text{ (W-ASS.NUM)}$$

Sémantique opérationnelle : composition

Exécution de l'instruction à gauche :

$$\frac{\langle C_1, s\rangle \to \langle C_1', s'\rangle}{\langle C_1; C_2, s\rangle \to \langle C_1'; C_2, s'\rangle} \text{ (W-SEQ.LEFT)}$$

Passage à la seconde commande, une fois la première exécutée :

$$\frac{}{\langle \texttt{skip}; C_2, s\rangle \to \langle C_2, s\rangle} \text{ (W-SEQ.RIGHT)}$$

Sémantique opérationnelle : conditions

On évalue d'abord le gardien :

$$\frac{\langle B, s\rangle \to \langle B', s'\rangle}{\langle \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, s\rangle \to \langle \texttt{if } B' \texttt{ then } C_1 \texttt{ else } C_2, s'\rangle} \text{ (W-COND.B)}$$

En fonction de sa valeur, on évalue la première ou la deuxième commande :

$$\frac{}{\langle \texttt{if true then } C_1 \texttt{ else } C_2, s\rangle \to \langle C_1, s\rangle} \text{ (W-COND.TRUE)}$$

$$\frac{}{\langle \texttt{if false then } C_1 \texttt{ else } C_2, s\rangle \to \langle C_2, s\rangle} \text{ (W-COND.FALSE)}$$

Sémantique opérationnelle : while

Une première version incorrecte :

$$\frac{\langle B, s\rangle \to \langle B', s'\rangle}{\langle \texttt{while}\ B\ \texttt{do}\ C, s\rangle \to \langle \texttt{while}\ B'\ \texttt{do}\ C, s'\rangle}$$

$$\frac{}{\langle \texttt{while false do}\ C, s\rangle \to \langle \texttt{skip}, s\rangle}$$

$$\frac{}{\langle \texttt{while true do}\ C, s\rangle \to\ ?}$$

Une version correcte :

$$\frac{}{\langle \texttt{while}\ B\ \texttt{do}\ C, s\rangle \to \langle \texttt{if}\ B\ \texttt{then}(C; \texttt{while}\ B\ \texttt{do}\ C)\ \texttt{else skip}, s\rangle}\ \texttt{(W-WHILE)}$$

(On recopie la boucle while à l'intérieur d'une condition)

Synthèse de la sémantique opérationnelle

$$\frac{}{\langle x, s\rangle \to \langle \mathtt{n}, s\rangle}\ \text{(W-EXP.NUM)}$$

$$\frac{}{\langle (\mathtt{n}_1 + \mathtt{n}_2), s\rangle \to \langle \mathtt{n}_3, s\rangle}\ \text{(W-EXP.ADD)}$$

$$\frac{\langle E_1, s\rangle \to \langle E_1', s'\rangle}{\langle (E_1 + E_2), s\rangle \to \langle (E_1' + E_2, s'\rangle}\ \text{(W-EXP.LEFT)}$$

$$\frac{\langle E, s\rangle \to \langle E', s'\rangle}{\langle x := E, s\rangle \to \langle x := E', s'\rangle}\ \text{(W-ASS.EXP)}$$

$$\frac{}{\langle x := \mathtt{n}, s\rangle \to \langle \mathtt{skip}, s[x \mapsto \mathtt{n}]\rangle}\ \text{(W-ASS.NUM)}$$

$$\frac{\langle C_1, s\rangle \to \langle C_1', s'\rangle}{\langle C_1; C_2, s\rangle \to \langle C_1'; C_2, s'\rangle}\ \text{(W-SEQ.LEFT)}$$

$$\frac{}{\langle \mathtt{skip}; C_2, s\rangle \to \langle C_2, s\rangle}\ \text{(W-SEQ.SKIP)}$$

$$\frac{\langle B, s\rangle \to \langle B', s'\rangle}{\langle \mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2, s\rangle \to \langle \mathtt{if}\ B'\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2, s'\rangle}\ \text{(W-COND.B)}$$

$$\frac{}{\langle \mathtt{if\ true\ then}\ C_1\ \mathtt{else}\ C_2, s\rangle \to \langle C_1, s\rangle}\ \text{(W-COND.TRUE)}$$

$$\frac{}{\langle \mathtt{if\ false\ then}\ C_1\ \mathtt{else}\ C_2, s\rangle \to \langle C_2, s\rangle}\ \text{(W-COND.FALSE)}$$

$$\frac{}{\langle \mathtt{while}\ B\ \mathtt{do}\ C, s\rangle \to \langle \mathtt{if}\ B\ \mathtt{then}(C; \mathtt{while}\ B\ \mathtt{do}\ C)\ \mathtt{else\ skip}, s\rangle}\ \text{(W-WHILE)}$$

Illustration

Soit le programme P suivant calculant la factoriel de x dans a :

$$
\begin{array}{l}
y := x; a := 1; \\
\textbf{while } y > 0 \textbf{ do} \\
\qquad (a := a \times y\,; \\
\qquad y := y - 1)
\end{array}
$$

Soit s l'état initial de la mémoire tel que $s(x) = 3, s(y) = 2, s(a) = 9$. Montrez que

$$\langle P, s\rangle \rightarrow \langle \texttt{skip}, s'\rangle$$

où $s'(a) = 6$.

Illustration

Si on note l'état de la mémoire $s_{i,j,k}$ pour $x \mapsto i$, $y \mapsto j$, et $a \mapsto k$, l'exécution de la machine d'état est la suivante (où P' désigne la boucle while) :

$$
\begin{aligned}
& \langle y := x, a := 1; P', s_{3,2,9} \rangle \\
\rightarrow \quad & \langle y := 3; a := 1; P', s_{3,2,9} \rangle \\
\rightarrow \quad & \langle \texttt{skip}; a := 1; P', s_{3,3,9} \rangle \\
\rightarrow \quad & \langle a := 1; P', s_{3,3,9} \rangle \\
\rightarrow \quad & \langle \texttt{skip}; P', s_{3,3,1} \rangle \\
\rightarrow \quad & \langle P', s_{3,3,1} \rangle \\
& \ldots \\
\rightarrow \quad & \langle \texttt{skip}, s_{3,0,6} \rangle
\end{aligned}
$$

Réponse finale et normalisation

Pour un programme P et un état de la mémoire s, la *réponse finale* associée à l'exécution de P à partir de s est l'état s' tel que :

$$\langle P, s\rangle \rightarrow^* \langle \texttt{skip}, s'\rangle$$

Cette état s' n'existe cependant pas pour tout P et s.

Exemples :

- Soit s associant x à 3 et indéfini pour tout autre identifiant :

$$\langle y := y + 1, s\rangle \rightarrow ?$$

- Il n'y a pas d'états s et s' tels que :

$$\langle \texttt{while true do skip}, s\rangle \rightarrow^* \langle \texttt{skip}, s'\rangle$$

Déterminisme

Lemme (déterminisme) : Pour toute configuration $\langle P, s\rangle$, il y a au plus une configuration $\langle P', s'\rangle$ telle que $\langle P, s\rangle \rightarrow \langle P', s'\rangle$.
Démonstration : La démonstration fonctionne par induction structurelle.
Soit le prédicat : $Q(P)$="Si $\langle P, s\rangle \rightarrow \langle P', s'\rangle$ et $\langle P, s\rangle \rightarrow \langle P'', s''\rangle$, alors $\langle P', s'\rangle = \langle P'', s''\rangle$".

- Cas : `skip`, `n`, `true`, `false` : il n'y a pas de transition possible à partir d'un de ces cas de base. Le prédictat est donc vrai.
- Cas : $C_1; C_2$. Supposons $Q(C_1)$ et $Q(C_2)$ vrais et montrons que $Q(C_1; C_2)$ est vrai.
  Soit C', s', C'', s'' tels que

$$\langle C_1; C_2, s\rangle \rightarrow \langle C', s'\rangle,$$

$$\langle C_1; C_2, s\rangle \rightarrow \langle C'', s''\rangle.$$

  $C' = C_1'; C_2$ et $C'' = C_1''; C_2$.

En examinant les règles, les seules possibilités sont :

- $C_1 = \texttt{skip}$: La seule règle possible est `(W-SEQ.SKIP)`. On a donc $\langle C', s' \rangle = \langle C_2, s \rangle = \langle C'', s'' \rangle$
- C_1 n'est pas une valeur (booléen, numéral, ou `skip`) : la règle appliquée est dans les deux cas `(W-SEQ.LEFT)`. On a alors il existe C_1' et C_1'' tels que $\langle C_1, s \rangle \rightarrow \langle C_1', s' \rangle$ et $\langle C_1, s \rangle \rightarrow \langle C_1'', s'' \rangle$.
  Par hypothèse inductive, on a $\langle C_1', s' \rangle = \langle C_1'', s'' \rangle$ et donc $\langle C', s' \rangle = \langle C_1'; C_2, s' \rangle = \langle C_1''; C_2, s'' \rangle = \langle C'', s'' \rangle$.

- Cas : ...

□

Exercice : complétez la démonstration pour les autres commandes et les booléens.

Boucle infinie

Théorème : Pour aucun s, il n'existe de s' tel que :

$$\langle \texttt{while true do skip}, s\rangle \rightarrow^* \langle \texttt{skip}, s'\rangle$$

Démonstration : Calculons les trois premières étapes de l'évaluation de la boucle :

$$\begin{aligned}
& \langle \texttt{while true do skip}, s\rangle \\
\rightarrow\ & \langle \texttt{if true then (skip; while true do skip) else skip}, s\rangle \\
\rightarrow\ & \langle \texttt{skip; while true do skip}, s\rangle \\
\rightarrow\ & \langle \texttt{while true do skip}, s\rangle
\end{aligned}$$

Par contradiction, supposons qu'il existe s' tel que

$$\langle \texttt{while true do skip}, s\rangle \rightarrow \langle \texttt{skip}, s'\rangle$$

et soit n le nombre d'étapes nécessaires à cette évaluation.

Vu le déterminisme de la machine, n est bien défini et les 3 premières étapes de cette évaluation sont les trois étapes reprises ci-dessus.
Les $n - 3$ étapes restantes de l'évaluation montrent que

$$\langle \texttt{while true do skip}, s \rangle \rightarrow \langle \texttt{skip}, s' \rangle,$$

ce qui n'est pas possible puisque cette évaluation requière n étapes.

□

Effets de bord

Dans notre langage, le choix de l'ordre d'évaluation des arguments des opérateurs ($+$, &, etc.) n'a pas d'importance.

Dans des langages plus complexes, l'ordre peut avoir de l'importance.

Exemples :

- Dans l'expression suivante, il est essentiel de connaître l'ordre d'évaluation des arguments du $+$:

$$(x := x + 1; \texttt{return}(x)) + (x := x \times 2; \texttt{return}(x))$$

- Selon l'implémentation du &, l'expression suivante aura on non une valeur :

$$\texttt{false\&(while true do skip; return(true))}$$

Evaluation du & avec court-circuit

$$\frac{B_1 \to B_1'}{\langle (B_1 \& B_2) \to (B_1' \& B_2)}$$

$$\frac{}{(\texttt{false} \& B_2) \to \texttt{false}} \qquad \frac{}{(\texttt{true} \& B_2) \to B_2}$$

Sémantique dénotationnelle

Rappel : la dénotation est une fonction associant une valeur du domaine sémantique à un programme.

Pour évaluer un programme, on doit pouvoir évaluer aussi les expressions arithmétique et booléennes et l'évaluation dépend de l'état initial de la mémoire.

On va définir trois fonctions de dénotation :

$$\begin{aligned} \text{eval}_{Com} &: Com \times \Sigma \to \Sigma_\perp, \\ \text{eval}_{Exp} &: Exp \times \Sigma \to \mathbb{N}_\perp, \\ \text{eval}_{Bool} &: Bool \times \Sigma \to \{\texttt{true}, \texttt{false}, \perp\}, \end{aligned}$$

où

- Σ est l'ensemble des configurations possibles de la mémoire,
- Le symbole $\perp$ (*bottom*) permet de représenter une valeur *indéfinie* (boucle infinie ou variable non définie)
- $\Sigma_\perp = \Sigma \cup \{\perp\}$, $\mathbb{N}_\perp = \mathbb{N} \cup \{\perp\}$.

Sémantique dénotationnelle

Exemples :

- $\text{eval}_{Com}(x := 0, s) = s[x \mapsto 0]$
- $\text{eval}_{Exp}(x + 3, s) = s(x) + 3$ si $s(x)$ est défini, $\perp$ sinon.

Quels sont les domaines sémantiques associés ?

- *Com* : l'ensemble des fonctions totales de Σ vers $\Sigma_\perp$
- *Exp* : l'ensemble des fonctions totales de Σ vers $\mathbb{N}_\perp$
- *Bool* : l'ensemble des fonctions totales de Σ vers $\{\texttt{true}, \texttt{false}, \perp\}$

En effet, on a :

$$Com \times \Sigma \to \Sigma_\perp = Com \to (\Sigma \to \Sigma_\perp)$$

$$Exp \times \Sigma \to \mathbb{N}_\perp = Exp \to (\Sigma \to \mathbb{N}_\perp)$$

$$Bool \times \Sigma \to \{\texttt{true}, \texttt{false}, \perp\} = Bool \to (\Sigma \to \{\texttt{true}, \texttt{false}, \perp\})$$

Expressions et booléens

Les fonctions eval_{Exp} et eval_{Bool} sont définies (récursivement) comme les fonctions eval aux transparents 140 et 157.

Seules modifications :

- Recherche de la valeur d'une variable :

$$\text{eval}_{Exp}(x, s) = \begin{cases} s(x) & \text{si } s(x) \text{ est défini} \\ \bot & \text{sinon} \end{cases}$$

- Il faut faire remonter le $\bot$:

$$\text{eval}_{Exp}(E_1 + E_2, s) = \begin{cases} \text{eval}_{exp}(E_1) + \text{eval}_{exp}(E_2) & \text{si } \text{eval}_{exp}(E_1) \text{ et } \text{eval}_{exp}(E_2) \neq \bot \\ \bot & \text{sinon} \end{cases}$$

(Exercice : définissez complètement eval_{Exp} et eval_{Bool}.)

Commandes : assignation, `skip`, composition

La fonction eval_{Com} est définie également récursivement sur l'ensemble Com.

$$C \in Com \quad ::= \quad x := E \mid \texttt{if } B \texttt{ then } C \texttt{ else } C \mid C; C \mid \texttt{skip} \mid \texttt{while } B \texttt{ do } C$$

- Assignation :

$$\text{eval}_{Com}(x := E, s) = \begin{cases} s[x \mapsto \text{eval}_{Exp}(E, s)]] & \text{si } \text{eval}_{Exp}(E, s) \neq \bot \\ \bot & \text{sinon} \end{cases}$$

- `skip` :

$$\text{eval}_{Com}(\texttt{skip}, s) = s$$

- Composition de commandes :

$$\text{eval}_{Com}(C_1; C_2, s) = \begin{cases} \bot & \text{si } \text{eval}_{Com}(C_1, s) = \bot \\ \text{eval}_{Com}(C_2, \text{eval}_{Com}(C_1, s)) & \text{sinon} \end{cases}$$

Exemple

Montrez que $\text{eval}_{Com}(x := 0; x := x + 1, s) = s[x \mapsto 1]$.

$$
\begin{aligned}
& \text{eval}_{Com}(x := 0; y := x + 1) \\
= \quad & \text{eval}_{Com}(y := x + 1, \text{eval}_{Com}(x := 0, s)) \\
= \quad & \text{eval}_{Com}(x := x + 1, s[x \mapsto 0]) \\
= \quad & s[x \mapsto \text{eval}_{Exp}(x + 1, s[x \mapsto 0])] \\
= \quad & s[x \mapsto \text{eval}_{Exp}(x, s[x \mapsto 0]) + \text{eval}_{Exp}(1, s[x \mapsto 0])] \\
= \quad & s[x \mapsto s[x \mapsto 0](x) + 1] \\
= \quad & s[x \mapsto 0 + 1] \\
= \quad & s[x \mapsto 1]
\end{aligned}
$$

Exercices : Montrez que pour tout $s \in \Sigma$:

- $eval_{Com}(x := y; y := x, s) = eval_{Com}(x := y, s)$
- $eval_{Com}(x := z; y := z, s) = eval_{Com}(y := z; x := z, s)$
- $eval_{Com}(C_1; (C_2; C_3)) = eval_{Com}((C_1; C_2); C_3, s)$

Commandes : condition, while

- Condition :

$$\text{eval}_{Com}(\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, s) = \begin{cases} \text{eval}_{Com}(C_1, s) & \text{si } \text{eval}_{Bool}(B, s) = \texttt{true} \\ \text{eval}_{Com}(C_2, s) & \text{si } \text{eval}_{Bool}(B, s) = \texttt{false} \\ \bot & \text{sinon} \end{cases}$$

- While : Une première idée :

$$\begin{aligned} & \text{eval}_{Com}(\texttt{while } B \texttt{ do } C, s) \\ = \; & \text{eval}_{Com}(\texttt{if } B \texttt{ then } (C; \texttt{while } B \texttt{ do } C) \texttt{ else skip}, s) \\ = \; & \ldots \text{eval}_{Com}(\texttt{while } B \texttt{ do } C, s') \ldots \end{aligned}$$

On se mord la queue...

Sémantique du `while`

Définition : Soit `diverge` un programme qui rentre tout de suite dans une boucle infinie, c'est-à-dire tel que pour tout $s \in \Sigma$:

$$\mathrm{eval}_{com}(\texttt{diverge}, s) = \bot.$$

Soit un programme `while` B `do` C. Définissons (récursivement) les programmes suivants :

$$\begin{aligned} C_0 &= \texttt{diverge} \\ C_1 &= \texttt{if } B \texttt{ then } (C; \texttt{diverge}) \texttt{ else skip} \\ &\vdots \\ C_{n+1} &= \texttt{if } B \texttt{ then } (C; C_n) \texttt{ else skip}. \end{aligned}$$

Approximations du `while`

Propriétés des C_n : Etant donné un état initial s :

- Si, partant de s, la boucle doit être exécutée moins que n fois, alors C_n permet d'arriver au même état final que la boucle
- Si plus de n exécutions sont nécessaires, alors C_n donne comme état final $\perp$ qui est potentiellement différent de l'état final de la boucle
- Si $\text{eval}_{Com}(C_n, s) \neq \perp$, C_n a le même effet que la boucle sur s.

Quand n croît, C_n est identique à la boucle pour de plus en plus d'états initiaux $s \in \Sigma$.

La séquence $C_0, C_1, C_2, \ldots$ constitue une séquence d'approximateurs de qualité croissante de la boucle `while`.

Sémantique du `while`

En conséquence de la discussion du transparent précédent, la sémantique du `while` peut être définie par :

$$\text{eval}_{Com}(\texttt{while } B \texttt{ do } C, s) = \begin{cases} s' & \text{si eval}_{Com}(C_k, s) = s' \text{ pour un } k \\ \bot & \text{sinon} \end{cases}$$

Etant donné le théorème ci-dessous, cette expression définit bien une fonction (il n'y a pas plusieurs $s' \neq \bot$ tels que $\text{eval}_{Com}(C_k, s) = s'$ pour un C_k).

Théorème : Pour tout $n \in \mathbb{N}$ et tout $s \in \Sigma$, si $\text{eval}_{Com}(C_n, s) \neq \bot$, alors $\text{eval}_{Com}(C_{n+1}, s) = \text{eval}_{Com}(C_n, s)$.
Démonstration : Par induction sur n en se basant sur la définition de eval_{Com}. *(Exercice)*.

Application

Utilisons la sémantique pour démontrer que le gardien d'une boucle est toujours faux après l'exécution de la boucle.

Lemme : Pour tout $n \in \mathbb{N}$ et $s \in \Sigma$, si $\text{eval}_{Com}(C_n, s) = s' \neq \bot$, alors $\text{eval}_{Bool}(B, s') = \texttt{false}$.

Démonstration : La preuve fonctionne par induction sur n.

Cas de base : $C_0 = \texttt{diverge}$, il n'y a donc pas de s' tel que $\text{eval}_{Com}(C_n, s) = s'$ et donc rien à prouver.

Cas inductif : Considérons le cas $n + 1$. Par définition,

$$C_{n+1} = \texttt{if } B \texttt{ then } (C; C_n) \texttt{ else skip}.$$

et donc par définition de eval_{com} :

$$\text{eval}_{Com}(C_{n+1}, s) = \begin{cases} \text{eval}_{Com}((C; C_n), s) & \text{si } \text{eval}_{Bool}(B, s) = \texttt{true} \\ s & \text{si } \text{eval}_{Bool}(B, s) = \texttt{false} \\ \bot & \text{sinon} \end{cases}$$

Si $\text{eval}_{Com}(C_{n+1}, s) = s'$, il y a deux cas possibles :

- $\text{eval}_{Bool}(B, s) = \texttt{false}$. Dans ce cas, $s = s'$ et donc $\text{eval}_{Bool}(B, s') = \texttt{false}$.
- $\text{eval}_{Bool}(B, s) = \texttt{true}$. Dans ce cas $s' = \text{eval}_{Com}((C; C_n), s)$.

  Par définition de eval_{Com} d'une composition, on a $s' = \text{eval}_{Com}(C_n, s'')$ avec $s'' = \text{eval}_{Com}(C, s)$.

  Par hypothèse inductive, on a donc $\text{eval}_{Bool}(B, s') = \texttt{false}$

□

Théorème : Si $\text{eval}_{Com}(\texttt{while } B \texttt{ do } C, s) = s' \neq \bot$, alors $\text{eval}_{Bool}(B, s') = \texttt{false}$.

Démonstration : Par la définition de la sémantique du `while`, si $\text{eval}_{Com}(\texttt{while } B \texttt{ do } C, s) = s'$, alors $s' = \text{eval}_{Com}(C_n, s)$ pour un n. Par le lemme précédent, on a donc bien $\text{eval}_{Bool}(B, s') = \texttt{false}$. □

Résumé

Ce qu'on a vu :

- Notion de syntaxe et sémantique d'un langage
- Sémantique opérationnelle (modélisation par une machine d'état)
- Sémantique dénotationnelle (définition récursive sur la syntaxe)
- Preuves de déterminisme de la sémantique opérationnelle
- Preuves de certaines propriétés des langages sur base de la sémantique

Au delà de cours

Syntaxe :

- Les langages qu'on a vu ici sont appelés des langages hors contexte. On peut définir des langages plus complexes, dépendant du contexte.
- La dérivation de la syntaxe abstraite à partir de la syntaxe concrète est non triviale pour la plupart des langages

Sémantique :

- Il existe d'autres types de sémantique. Par exemple :
  - Axiomatique : l'effet du programme est déterminé par la transformation de prédicats définis sur les variables (méthode d'invariant, triplet de Hoare, vue au chapitre 2).
  - Sémantique naturelle (ou opérationnelle à grand pas) : entre la sémantique opérationnelle et la sémantique dénotationnelle.
- Autres questions importantes en sémantique : prise en compte des fonctions, analyse de type...

(voir les cours INFO0016 (calculabilité) et INFO0085 (compilateurs))