Sémantiques pour un langage impératif

Dominique Duval

XLIM, colloquium du LACO
Université de Limoges, 23 février 2007

Outline

Introduction

Syntaxe

Sémantique dénotationnelle

Sémantique opérationnelle

SémantiqueS

Syntaxe et sémantiques

Un langage de programmation possède :

- une syntaxe (ou grammaire),
- et des sémantiques (pour le “sens”).

Une sémantique peut être (entre autres) :

- opérationnelle,
- dénotationnelle,
- axiomatique, ...

Pour un langage donné,
ces sémantiques ne sont pas toujours équivalentes !

Paradigmes de langages de programmation

On peut classer les langages de programmation :

- impératif (FORTRAN, Pascal, C, Ada)
- fonctionnel (Lisp, Scheme, ML, Caml, Haskell)
- logique (Prolog)

et orienté objet (Smalltalk, Eiffel, Java, C++, OCaml)

Le langage IMP est un langage impératif particulièrement simple, mais aussi puissant que les autres langages.

Toutes les sémantiques de IMP sont équivalentes.

Référence : Glynn WINSKEL,
The formal semantics of programming languages.

Syntaxe et sémantiques d'un langage impératif

Le programme s'exécute pas à pas, en écrasant la valeur d'une zone mémoire pour la remplacer par une nouvelle valeur.

L'état de la mémoire jour donc un rôle primordial dans la description de l'exécution d'un programme.

- Dans la syntaxe l'état est implicite (caché)
- Dans les sémantiques l'état est explicite (visible)

Travail en cours [Dumas, Duval, Reynaud] :

- Définir une sémantique avec état implicite

Des idées voisines :
les sortes cachées [Goguen], les monades [Moggi]

Un peu de calculabilité

Thèse de Church-Turing.
Toutes les définitions des fonctions calculables sont équivalentes.

Enoncé par Kleene en 1943.
Impossible à énoncer précisément, donc à prouver,
mais s'appuie sur des théorèmes d'équivalence (1936) :

- machines de Turing,
- fonctions récursives de Gödel,
- lambda calcul de Church, ...

"*Tous les langages de programmation ont la même puissance.*"

Indécidabilité.
Le problème de l'arrêt est indécidable (Turing 1936).

"*Tous les langages permettent d'écrire des programmes qui ne terminent pas.*"

Outline

Introduction

Syntaxe

Sémantique dénotationnelle

Sémantique opérationnelle

SémantiqueS

Syntaxe : les constantes

Nombres : $n \in N = \{\ldots, -1, 0, 1, \ldots\}$

Valeurs de vérité : $t \in T = \{\ \texttt{true}\ ,\ \texttt{false}\ \} = \{\ \texttt{t}\ ,\ \texttt{f}\ \}$

Variables : $v \in V = \{X, Y, Z, X_1, Y_1, \ldots\}$

On note $U = \{*\}$ (singleton),
et on identifie $0 = (* \mapsto 0) : U \to N$, etc

$$\begin{tikzcd} V \\ U \arrow[u, "X,\ldots"] \end{tikzcd} \qquad \begin{tikzcd} N \\ U \arrow[u, "\ldots,0,\ldots"] \end{tikzcd} \qquad \begin{tikzcd} T \\ U \arrow[u, "\texttt{t}\ ,\ \texttt{f}"] \end{tikzcd}$$

Syntaxe : les expressions

Définition inductive.
Notation "à la BNF" (Backus-Naur Form).

Expressions arithmétiques : $a \in A$.

$$a ::= n \mid v \mid -a \mid a + a' \mid a \times a'$$

Expressions booléennes : $b \in B$.

$$b ::= t \mid \neg b \mid b \wedge b' \mid b \vee b' \mid a = a' \mid a < a' \mid a \leq a'$$

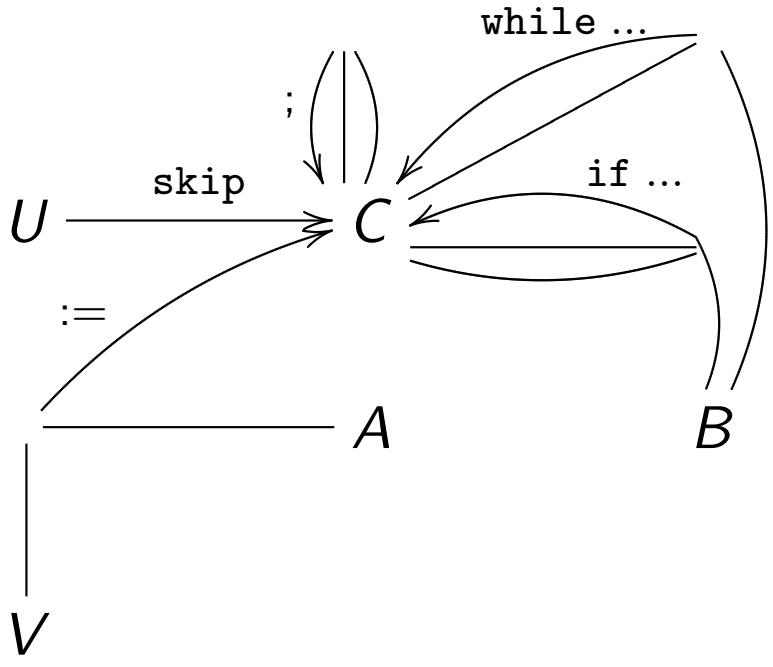
$$\begin{tikzcd}
 & A \arrow[loop left, "-,+,\times"] \arrow[r, "=,<,\leq"] & B \arrow[loop right, "\neg,\wedge,\vee"] \\
V \arrow[ur] & N \arrow[u] & T \arrow[u]
\end{tikzcd}$$

Syntaxe : les commandes

Définition inductive.
Commandes : $c \in C$.

$$c ::= \texttt{skip} \mid v := a \mid c; c' \mid \texttt{if}\ b\ \texttt{then}\ c_t\ \texttt{else}\ c_f \mid \texttt{while}\ b\ \texttt{do}\ c$$

(pas de fonctions, procédures, pointeurs,...)



Constantes et expressions

Syntaxiquement, les constantes sont aussi des expressions :

$$a ::= n \mid v \mid \ldots\ , \quad b ::= t \mid \ldots$$

On peut rendre ces conversions explicites :

$$a ::= \mathrm{n_a}(n) \mid \mathrm{v_a}(v) \mid \ldots\ , \quad b ::= \mathrm{t_b}(t) \mid \ldots$$

$$\begin{tikzcd}
 & A \arrow[loop above, "-,+,\times"] \arrow[r, "=,<,\leq"] & B \arrow[loop above, "\neg,\wedge,\vee"] \\
V \arrow[ur, "\mathrm{v_a}"] & N \arrow[u, "\mathrm{n_a}"] & T \arrow[u, "\mathrm{t_b}"]
\end{tikzcd}$$

Cela permet de distinguer la variable X (à gauche) de l'expression X (à droite) dans la commande

$$X := -X$$

en notant éventuellement

$$X := -\mathrm{v_a}(X)$$

Exemples : affectation

Une commande est un "chemin" de source U et de but C (ou terme clos de type C) engendré par le "graphe" (ou signature) décrivant la syntaxe.

$X := 0$

$$\begin{tikzcd}
U \arrow[rr, "x"] & & V \arrow[dr, dash] & & \\
 & & & & C \\
U \arrow[r, "0"] & N \arrow[r] & A \arrow[ur, dash] \arrow[r, ":="] & & C
\end{tikzcd}$$

$X := Y$

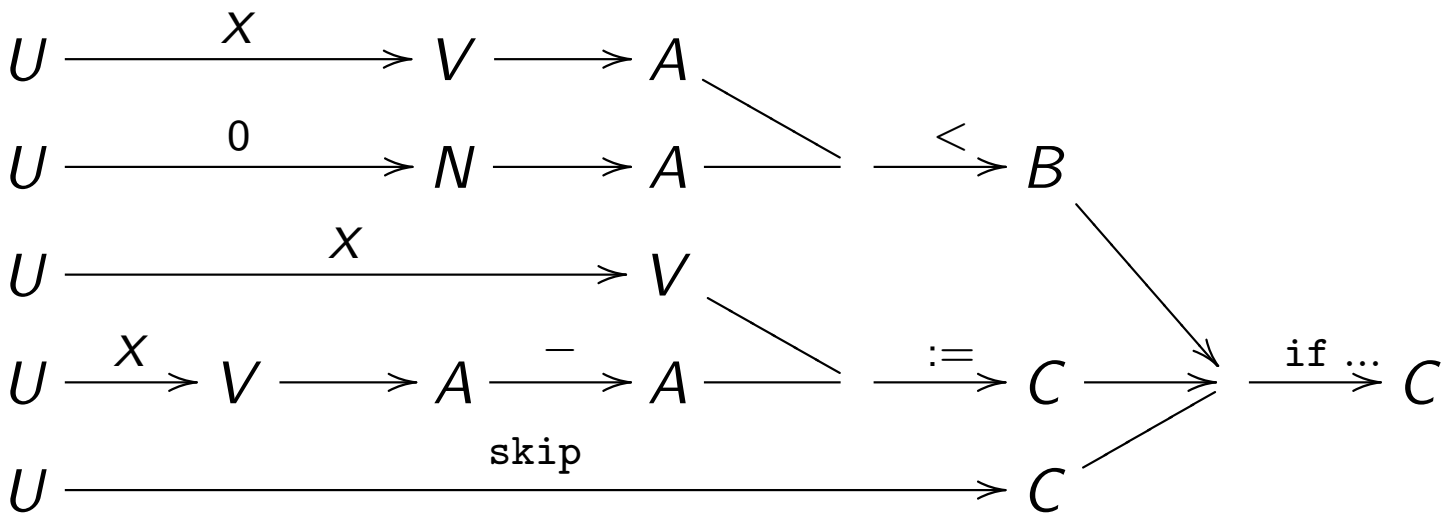
$$\begin{tikzcd}
U \arrow[rr, "x"] & & V \arrow[dr, dash] & & \\
 & & & & C \\
U \arrow[r, "Y"] & V \arrow[r] & A \arrow[ur, dash] \arrow[r, ":="] & & C
\end{tikzcd}$$

Semblable aux diagrammes usuels, mais plus précis :

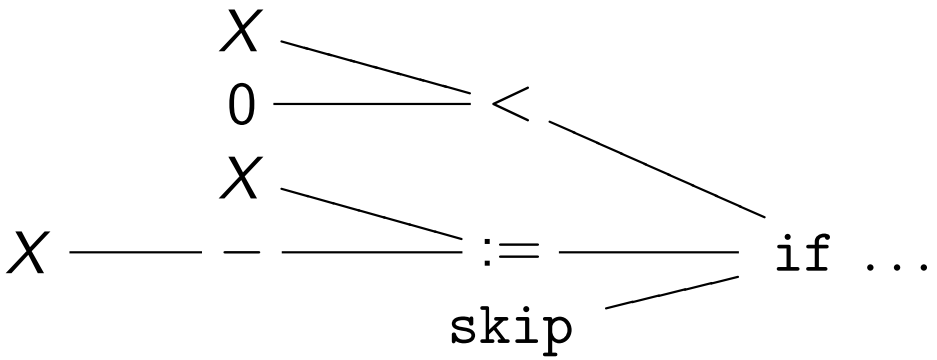
$$\begin{tikzcd}
X \arrow[dr, dash] & \\
 & := \\
0 \arrow[ur, dash] &
\end{tikzcd}
\qquad
\begin{tikzcd}
X \arrow[dr, dash] & \\
 & := \\
Y \arrow[ur, dash] &
\end{tikzcd}$$

Exemple : conditionnelle

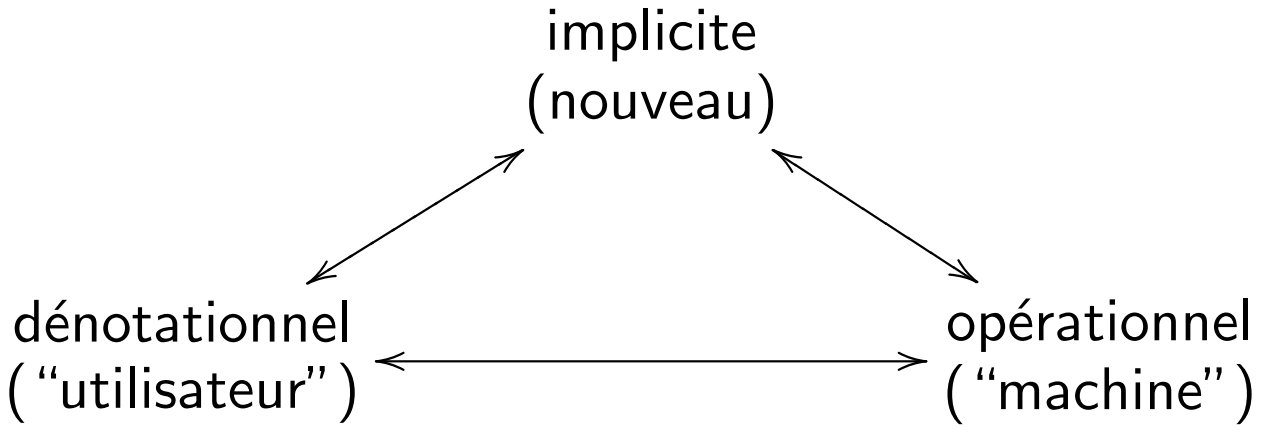
$\texttt{if}\ X < 0\ \texttt{then}\ X := -X\ \texttt{else skip}$



À comparer au diagramme usuel :



Trois sémantiques



Outline

Introduction

Syntaxe

Sémantique dénotationnelle

Sémantique opérationnelle

SémantiqueS

Sémantique dénotationnelle

La sémantique dénotationnelle est une interprétation mathématique de la syntaxe utilisant uniquement les constantes (N, T, V) et les constructions ensemblistes.

La sémantique dénotationnelle représente le point de vue "de l'utilisateur".

L'état

On pose $S = N^V$ (fonctions de V vers N)

$s(v) \in N$ est la valeur de la variable v dans l'état s,

$s[n/v]$ ou `update` (s, v, n) est l'état obtenu à partir de s en remplaçant la valeur de v par n.

$$s = \begin{array}{|c|c|} \hline X & 1 \\ \hline Y & 55 \\ \hline Z & -4 \\ \hline \vdots & \vdots \\ \hline \end{array} \qquad s(X) = 1 \qquad s[9/X] = \begin{array}{|c|c|} \hline X & 9 \\ \hline Y & 55 \\ \hline Z & -4 \\ \hline \vdots & \vdots \\ \hline \end{array}$$

Remarque. $S = N^V$, donc $V \times S \to N$, donc $V \to N^S$.

Interprétation : les constantes

Les constantes sont leur propre interprétation :

$$[[n]] = n \in N\,,\ [[t]] = t \in T\,,\ [[v]] = v \in V$$

$$\begin{tikzcd} V \\ U \arrow[u, "x,\ldots"] \end{tikzcd} \qquad \begin{tikzcd} N \\ U \arrow[u, "\ldots,0,\ldots"] \end{tikzcd} \qquad \begin{tikzcd} T \\ U \arrow[u, "\mathtt{t}\ ,\ \mathtt{f}"] \end{tikzcd}$$

Interprétation : les expressions

Expressions arithmétiques : A est interprété par N^S

$[[n]] : s \mapsto n\,,\ [[v]] = [[\mathrm{v_a}(v)]] : s \mapsto s(v)\,,\ [[-a]] = -[[a]]\,,\ [[a{+}a']] = [[a]]{+}[[a']]\,,\ \ldots$

Expressions booléennes : B est interprété par T^S.

$[[t]] : s \mapsto t\,,\ [[\neg b]] = \neg[[b]]\,,\ [[b \wedge b']] = [[b]] \wedge [[b']]\,,\ [[a < a']] = [[a]] < [[a']]\,,\ \ldots$

$$\begin{tikzcd}
 & N^S \arrow[r, "{=,>,\geq}"] \arrow[loop above, "{-,+,\times}"] & T^S \arrow[loop above, "{\neg,\wedge,\vee}"] \\
V \arrow[ur] & N \arrow[u] & T \arrow[u]
\end{tikzcd}$$

Interprétation : les commandes

C est interprété par S_p^S (fonctions partielles de S vers S).

L'ensemble S_p^S est ordonné :
$c \leq c' \iff \mathrm{dom}(c) \subseteq \mathrm{dom}(c')$ et $\forall s \in \mathrm{dom}(c)\ ,\ c(s) = c'(s)$

$$[[\ \texttt{skip}\ ]] : s \mapsto s$$

$$[[v := a]] : s \mapsto s[n/v] \ \text{ où } \ n = [[a]](s)$$

$$[[c; c']] = [[c']] \circ [[c]] : s \mapsto [[c']]([[c]](s))$$

$$[[\ \texttt{if}\ b\ \texttt{then}\ c_t\ \texttt{else}\ c_f]] :$$

$$s \mapsto \ \texttt{si}\ [[b]](s)\ \texttt{alors}\ [[c_t]](s)\ \texttt{sinon}\ [[c_f]](s)$$

c'est la définition d'une fonction par cas, car
$T = \{\ \texttt{true}\ ,\ \texttt{false}\ \} = \{\ \texttt{true}\ \} + \{\ \texttt{false}\ \}$

Interprétation : la boucle

On souhaite que : $[[\texttt{while}\ b\ \texttt{do}\ c]](s)$

$$= \texttt{si}\ [[b]](s)\ \texttt{alors}\ [[\texttt{while}\ b\ \texttt{do}\ c]]([[c]](s))\ \texttt{sinon}\ s$$

Autrement dit, en définissant la fonction $\Phi_{\beta,\gamma} : S_p^S \to S_p^S$ par

$$\Phi_{\beta,\gamma}(w) : s \mapsto \texttt{si}\ \beta(s)\ \texttt{alors}\ w(\gamma(s))\ \texttt{sinon}\ s$$

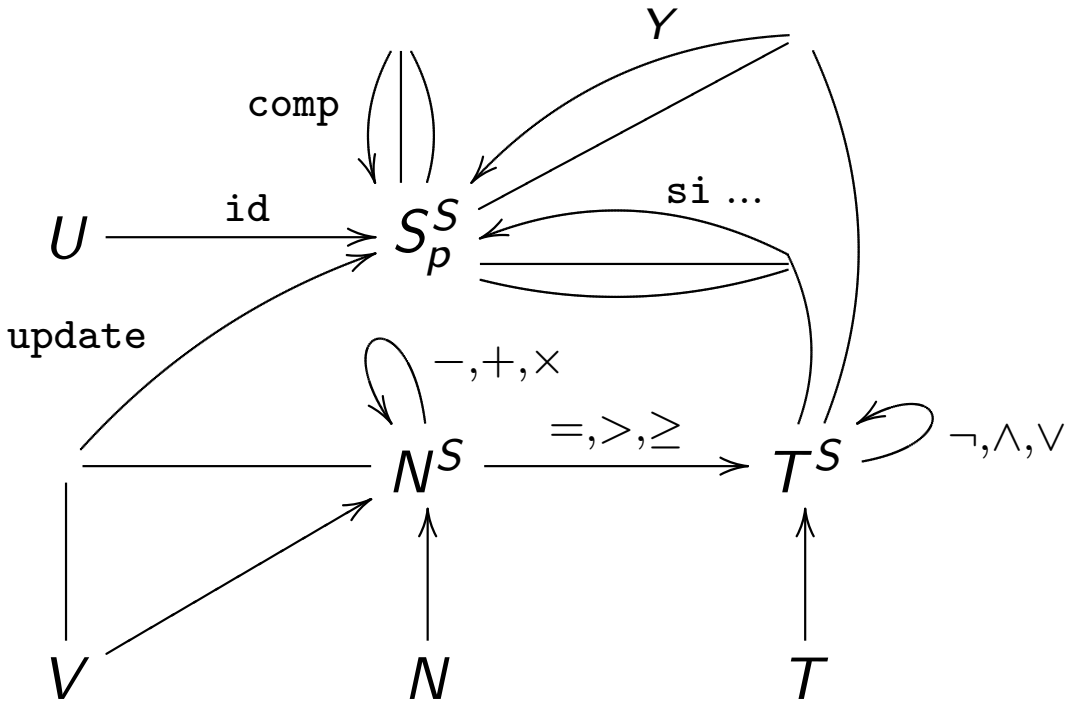
on souhaite que $[[\texttt{while}\ b\ \texttt{do}\ c]]$ soit un point fixe de $\Phi_{[[b]],[[c]]}$:

$$[[\texttt{while}\ b\ \texttt{do}\ c]] = \Phi_{[[b]],[[c]]}([[\texttt{while}\ b\ \texttt{do}\ c]])$$

Théorème. $\Phi_{\beta,\gamma}$ a au moins un point fixe,
et un unique plus petit point fixe $Y_{\beta,\gamma}$.

On pose : $[[\texttt{while}\ b\ \texttt{do}\ c]] = Y_{[[b]],[[c]]}$

L'interprétation est un modèle de la syntaxe



On dit que la sémantique dénotationnelle est compositionnelle.

L'interprétation est une transformation naturelle

A chaque flèche dans la syntaxe correspond un carré commutatif, par exemple :

$$\begin{tikzcd} B \arrow[r, "\neg"] \arrow[d, "{[[-]]}"'] & B \arrow[d, "{[[-]]}"] \\ T^S \arrow[r, "\neg"] & T^S \end{tikzcd} \qquad \begin{tikzcd} A \times A \arrow[r, "+"] \arrow[d, "{[[-]]}"'] & A \arrow[d, "{[[-]]}"] \\ N^S \times N^S \arrow[r, "+"] & N^S \end{tikzcd}$$

$$\begin{tikzcd} V \times A \arrow[r, ":="] \arrow[d, "{[[-]]}"'] & C \arrow[d, "{[[-]]}"] \\ V \times N^S \arrow[r, "\mathtt{update}"] & N^S \end{tikzcd} \qquad \begin{tikzcd} B \times C \arrow[r, "\mathtt{while}\ \ldots"] \arrow[d, "{[[-]]}"'] & C \arrow[d, "{[[-]]}"] \\ T^S \times S_p^S \arrow[r, "Y"] & S_p^S \end{tikzcd}$$

Outline

Introduction

Syntaxe

Sémantique dénotationnelle

Sémantique opérationnelle

SémantiqueS

Sémantique opérationnelle

La sémantique opérationnelle est
une évaluation de la syntaxe
en utilisant uniquement les constantes (N, T, V)
et l'observation de l'état de la mémoire.

La sémantique opérationnelle représente
le point de vue "de la machine".

Pour simplifier,
on suppose encore que l'état est une fonction de V vers N :

$$S = N^V$$

(Dé)curryfication

On peut passer de la fonction d'interprétation

$$[[-]] = [[-]]_A : A \to N^S$$

à la fonction d'évaluation

$$\mathrm{ev} = \mathrm{ev}_A : A \times S \to N$$

par décurryfication :

$$\forall a \in A\,,\ \forall s \in S\,,\ \mathrm{ev}(a, s) = [[a]](s)\ (\in N)$$

De même $\mathrm{ev} = \mathrm{ev}_B : B \times S \to T$ est définie par :

$$\forall b \in B\,,\ \forall s \in S\,,\ \mathrm{ev}(b, s) = [[b]](s)\ (\in T)$$

Et $\mathrm{ev} = \mathrm{ev}_C : C \times S \rightharpoonup S$ est la fonction partielle définie par :

$$\forall c \in C\,,\ \forall s \in S\,,\ \mathrm{ev}(c, s) = [[c]](s)\ (\in S)$$

Evaluation : les constantes

Notation. On note $(a, s) \mapsto n$ pour $\mathrm{ev}(a, s) = n$, c'est-à-dire :
"dans l'état s, l'évaluation de a retourne n"
(la notation classique est $(a, s) \to n$).

Classiquement, l'évaluation est décrite par des règles,
déduites de la sémantique dénotationnelle par décurryfication.

Les constantes sont leur propre valeur :

$$\mathrm{ev}_N : N \times S \to N : (n, s) \mapsto n$$

$$\mathrm{ev}_T : T \times S \to T : (t, s) \mapsto t$$

$$\mathrm{ev}_V : V \times S \to V : (v, s) \mapsto v$$

Preuve : $\mathrm{ev}(n, s) = [[n]](s) = n$, etc.

Evaluation : les expressions arithmétiques

L'évaluation d'une expression arithmétique retourne un nombre :

$$\mathrm{ev}_A : A \times S \to N$$

$$(\mathrm{n_a}(n), s) \mapsto n\,,\ (\mathrm{v_a}(v), s) \mapsto s(v)\,,$$

$$\frac{(a, s) \mapsto n \quad (a', s) \mapsto n'}{(a + a', s) \mapsto n + n'}\,,\ \ldots$$

Preuve : $\mathrm{ev}(\mathrm{n_a}(n), s) = [[\mathrm{n_a}(n)]](s) = n$,
$\mathrm{ev}(\mathrm{v_a}(v), s) = [[\mathrm{v_a}(v)]](s) = s(v)$,
$\mathrm{ev}(a+a', s) = [[a+a']](s) = [[a]](s)+[[a']](s) = \mathrm{ev}(a, s)+\mathrm{ev}(a', s)$,
etc.

Evaluation : les expressions booléennes

L'évaluation d'une expr. booléenne retourne une valeur de vérité :

$$\mathrm{ev}_B : B \times S \to T$$

$$(\mathrm{t_b}(t), s) \mapsto t\,,$$

$$\frac{(b, s) \mapsto t}{(\neg b, s) \mapsto \neg t}\,, \ \ldots$$

Preuve : $\mathrm{ev}(\mathrm{t_b}(t), s) = [[\mathrm{t_b}(t)]](s) = t$,
$\mathrm{ev}(\neg b, s) = [[\neg b]](s) = \neg[[b]](s) = \neg\mathrm{ev}(b, s)$, etc.

Evaluation : les commandes

L'évaluation, ou exécution, d'une commande retourne un état, *à condition qu'elle se termine* :

$$\mathrm{ev}_C : C \times S \rightharpoonup S$$

$$(\texttt{skip}, s) \mapsto s$$

$$\frac{(a, s) \mapsto n}{(v := a, s) \mapsto s[n/v]}$$

$$\frac{(c, s) \mapsto s' \quad (c', s') \mapsto s''}{((c; c'), s) \mapsto s''}$$

Preuve : $\mathrm{ev}(c; c', s) = [[c; c']](s) = [[c']]([[c]](s))$
$= [[c']](\mathrm{ev}(c, s)) = \mathrm{ev}(c', \mathrm{ev}(c, s))$

Evaluation : la conditionnelle

$$\frac{(b,s) \mapsto \texttt{true} \quad (c_t,s) \mapsto s'}{(\texttt{ if } b \texttt{ then } c_t \texttt{ else } c_f, s) \mapsto s'}$$

$$\frac{(b,s) \mapsto \texttt{false} \quad (c_f,s) \mapsto s'}{(\texttt{ if } b \texttt{ then } c_t \texttt{ else } c_f, s) \mapsto s'}$$

Preuve :

$\mathrm{ev}(\texttt{ if } b \texttt{ then } c_t \texttt{ else } c_f, s) = [[\texttt{ if } b \texttt{ then } c_t \texttt{ else } c_f]](s)$

$= \texttt{ si } [[b]](s) \texttt{ alors } [[c_t]](s) \texttt{ sinon } [[c_f]](s)$

$= \texttt{ si } \mathrm{ev}(b,s) \texttt{ alors } \mathrm{ev}(c_t,s) \texttt{ sinon } \mathrm{ev}(c_f,s)$

Cela pourrait fournir l'unique règle :

$$\frac{(b,s) \mapsto t \quad (c_t,s) \mapsto s_t \quad (c_f,s) \mapsto s_f}{(\texttt{ if } b \texttt{ then } c_t \texttt{ else } c_f, s) \mapsto (\texttt{ si } t \texttt{ alors } s_t \texttt{ sinon } s_f)}$$

mais elle ne convient pas car non linéaire en S.

Par contre cela fournit les deux égalités conditionnelles :

si $\mathrm{ev}(b,s) = \texttt{ true}$ alors $\mathrm{ev}(\texttt{ if } b \texttt{ then } c_t \texttt{ else } c_f, s) = \mathrm{ev}(c_t,s)$

si $\mathrm{ev}(b,s) = \texttt{ false}$ alors $\mathrm{ev}(\texttt{ if } b \texttt{ then } c_t \texttt{ else } c_f, s) = \mathrm{ev}(c_f,s)$

d'où les deux règles.

Evaluation : la boucle

Comme pour la conditionnelle, il y a 2 règles.

$$\frac{(b,s) \mapsto \texttt{true} \quad (c,s) \mapsto s' \ , \quad (\texttt{while}\ b\ \texttt{do}\ c, s') \mapsto s''}{(\texttt{while}\ b\ \texttt{do}\ c, s) \mapsto s''}$$

$$\frac{(b) \mapsto \texttt{false}}{(\texttt{while}\ b\ \texttt{do}\ c, s) \mapsto s}$$

Preuve :

— Vérifier que la fonction $s \mapsto \mathrm{ev}(\texttt{while}\ b\ \texttt{do}\ c, s)$ est un point fixe de $\Phi_{[[b]],[[c]]}$.

— Vérifier que c'est le plus petit point fixe de $\Phi_{[[b]],[[c]]}$.

— Alors

$\mathrm{ev}(\texttt{while}\ b\ \texttt{do}\ c, s) = [[\texttt{while}\ b\ \texttt{do}\ c]](s) = Y_{[[b]],[[c]]}(s)$.

Outline

Introduction

Syntaxe

Sémantique dénotationnelle

Sémantique opérationnelle

SémantiqueS

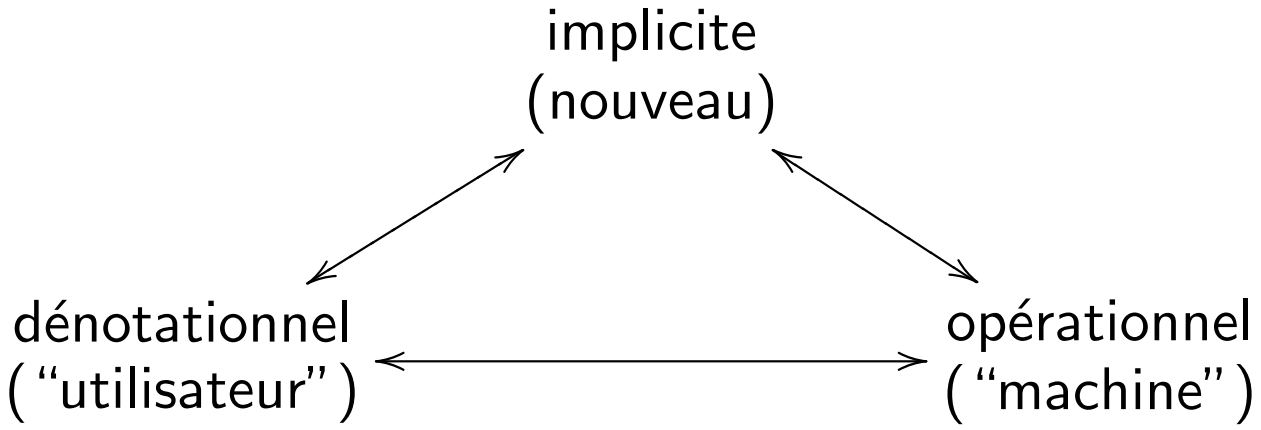
Equivalence des deux sémantiques

dénotationnelle	opérationnelle
“utilisateur”	“machine”
interprétation	évaluation
modèle	règles
exponentiel en S	linéaire en S

Théorème. La sémantique dénotationnelle et la sémantique opérationnelle de IMP sont équivalentes :
$[[a_1]] = [[a_2]]$ ssi $\forall s \in S\,,\ \mathrm{ev}(a_1, s) = \mathrm{ev}(a_2, s)$
$[[b_1]] = [[b_2]]$ ssi $\forall s \in S\,,\ \mathrm{ev}(b_1, s) = \mathrm{ev}(b_2, s)$
$[[c_1]] = [[c_2]]$ ssi $\forall s \in S\,,\ \mathrm{ev}(c_1, s) = \mathrm{ev}(c_2, s)$

Preuve : Par définition de la sémantique opérationnelle !

Vers une sémantique avec état implicite



Décorations

La syntaxe est statique.

- L'évaluation des constantes n'interagit pas avec l'état : c'est une fonction statique $\mathrm{ev}^{\mathrm{st}}$.
- L'évaluation des expressions utilise l'état : c'est une fonction inspectrice $\mathrm{ev}^{\mathrm{in}}$.
- L'évaluation des commandes utilise et modifie l'état : c'est une fonction modificatrice $\mathrm{ev}^{\mathrm{mo}}$.

Dilatation

Pour expliciter le fait que $\mathrm{ev}^{\mathrm{mo}}$ est un modificateur :
dilater la fonction $\mathrm{ev}^{\mathrm{mo}}$,
soit de façon exponentielle en S (dénotationnel)
soit de façon linéaire en S (opérationnel) :

$$\begin{tikzcd}
 & \mathrm{ev}^{\mathrm{mo}} : C \rightharpoonup U & \\
[[-]] : C \arrow[r] & S_p^S \arrow[ul, leftrightarrow] \arrow[rr, leftrightarrow] & \mathrm{ev} : C \times S \rightharpoonup S \arrow[ur, leftrightarrow]
\end{tikzcd}$$

Equivalence des trois sémantiques ?

Théorème ? La sémantique implicite de IMP est équivalente à ses sémantiques dénotationnelle et opérationnelle.

Preuve ?
Utiliser les *logiques diagrammatiques* [Duval, Lair, 2002].
Travail en cours [Dumas, Duval, Reynaud]