

Sémantique des langages

Programmation raisonnée

ENSIIE S5

Pourquoi de la sémantique ?

On cherche à accroître la confiance en la sûreté d'un programme en raisonnant dessus

- ▶ il faut donc être capable de **dire ce que le programme fait**

Sans sémantique, seulement intuition, bonne impression

Que fait ce programme ?

En C :

```
while (*a++ = *b++);
```

Que fait ce programme ?

En C :

```
while (*a++ = *b++);
```

- ▶ copie de chaînes de caractère, sous certaines conditions

L'intuition est importante

- ▶ Essentielle pour écrire de bons programmes
- ▶ Suffisante pour comprendre la plupart des codes
- ▶ Pas toujours suffisante pour s'assurer de leur correction
- ▶ Il s'agit d'aller au delà de l'intuition

L'intuition n'est pas suffisante

- ▶ Quand on transforme des programmes (compilateurs, outils de refactoring, ...)
- ▶ Pour des applications critiques (transports, nucléaire, ...)
- ▶ Dès qu'on veut pouvoir raisonner (formellement) sur les programmes

Comment définir la sémantique d'un programme ?

En le faisant tourner ?

- ▶ Un compilateur permet de définir une sémantique pour le langage :
Un programme de ce langage fait ce que fait l'exécutable obtenu par compilation

Problèmes :

- ▶ Comment définir la sémantique de l'exécutable ?
- ▶ Difficile de faire des raisonnements formels dessus (trop de détails)
- ▶ Plusieurs (version de) compilateurs → plusieurs sémantiques

Sémantique

Besoin de décrire une **sémantique abstraite** du langage,
indépendamment du compilateur

- ▶ Peu de langages ont une telle sémantique
- ▶ Le plus souvent **informelle** (anglais)
- ▶ Ne couvre qu'une partie du langage

Sémantique : Quoi ?

“Sémantique : Étude du sens des unités linguistiques et de leur composition.” [Petit Larousse, 1994]

Associer une “**signification**” à un programme.

Sémantique d'un langage : définition **précise, non ambiguë et indépendante de l'implantation**, de la “signification” des constructions de ce langage

- ▶ quelle **valeur** nous **décidons** de donner à une expression ?
- ▶ quel **effet** nous **décidons** d'attribuer à une instruction ?

Sémantique formelle d'un langage : exprimée dans un **formalisme mathématique**

Sémantique formelle

Premier langage réel avec une sémantique formelle :

[Standard ML \(SML\)](#)

The Definition of Standard ML, Robin Milner, Mads Tofte, Robert Harper, MIT Press 1990

Sémantique : Pourquoi (1) ?

Garantir certaines propriétés vérifiées par les programmes ...
augmenter la confiance que l'on peut avoir dans les programmes.

- ▶ **spécification d'un compilateur** pour un langage donné
 - définition d'un **standard**
 - **portabilité** des programmes
- ▶ **raisonnement sur les propriétés attendues du langage** –
comme par exemple le **déterminisme** de l'exécution des instructions

Sémantique : Pourquoi (2) ?

- ▶ **raisonnement sur les propriétés des programmes**
 - **Equivalence** de programmes ... utile pour transformer un programme en un programme équivalent mais plus efficace
 - **Terminaison** de programmes
 - Non-modification par un programme des valeurs contenues à des adresses sensibles de la mémoire
 - ...
- ▶ génération d'outils
 - interpréteur
 - prouveurs spécialisés
 - ...

Sémantique : Comment ? (1)

Sémantique dynamique : description du comportement (i.e., l'exécution) de tous les programmes, y compris ceux dont l'exécution provoque une "erreur".

- ▶ *sémantique dénotationnelle* (vision fonctionnelle)

programme = fonction

Associer un ensemble à chaque famille de données et une fonction sur l'ensemble dénotant son paramètre à chaque procédure.

On s'intéresse à l'effet d'un programme et non à la manière avec laquelle il a été exécuté.

peu développé ici

- ▶ *sémantique axiomatique* (vision déclarative)
programme = "transformateur" de propriétés (sur les états)

Logique de Hoare

Sémantique : Comment ? (2)

- ▶ *sémantique opérationnelle* (vision impérative)
programme = “transformateur” d'états de la mémoire
Définition d'une relation de transition entre états décrivant les changements produits par l'exécution d'une instruction.
Abstraction : on ignore les détails sur l'utilisation de registres, l'adressage des variables ... indépendance vis à vis de l'architecture des machines.
 - Expressions arithmétiques et booléennes
 - Constructions impératives
 - Expressions fonctionnelles
 - ...

Sémantique statique : le *typage* par exemple ... sans avoir recours à l'exécution ... *voir aussi module d'analyse statique*

Outline

- Motivations
- Rappels
 - Systèmes d'inférence
- Expressions arithmétiques
- Expressions booléennes
- IMP
- MiniML

Rappels : Systèmes d'inférence

Permet la *définition* d'ensembles, de relations, ...

\mathcal{J} : ensemble de *jugements*

$\Phi[\mathcal{J}]$: *système d'inférence* = ensemble de *règles* portant sur des jugements de \mathcal{J} :

$$(R) \frac{j_1 \quad j_2 \quad \cdots \quad j_n}{j}$$

$\{j_1, \dots, j_n\}$: *prémises* qui doivent être “satisfaites” pour que la *conclusion* j le soit.

axiome ($n = 0$) : $(R)_{\bar{j}}$ Le jugement j est toujours satisfait.

Exemples : règles de la déduction naturelle, règles de typage du λ -calcul

Arbres d'inférence (de preuve, de dérivation)

Un **arbre d'inférence** d'un jugement $j \in \mathcal{J}$ pour un système d'inférence $\Phi[\mathcal{J}]$ est un **arbre fini** dont la racine est j et tel que pour chaque noeud \dot{j}_k dont les fils sont $\dot{j}_{k_1}, \dot{j}_{k_2}, \dots, \dot{j}_{k_n}$, il existe une règle de $\Phi[\mathcal{J}]$:

$$(\tau) \frac{\dot{j}_{k_1} \quad \dot{j}_{k_2} \quad \dots \quad \dot{j}_{k_n}}{\dot{j}_k}$$

Théorèmes

$\Phi[\mathcal{J}]$ caractérise un ensemble de **théorèmes** $\text{Th}(\Phi[\mathcal{J}]) \subseteq \mathcal{J}$ contenant les jugements qui admettent un **arbre d'inférence**.
 $j \in \mathcal{J}$ est un **théorème** s'il existe dans $\Phi[\mathcal{J}]$ une règle :

$$(R) \frac{}{j} \quad \text{ou} \quad (R) \frac{j_1 \quad j_2 \quad \cdots \quad j_n}{j}$$

telle que chaque j_i ($1 \leq i \leq n$) soit un théorème.

Systèmes d'inférence et Définitions inductives

Etant donné un **système d'inférence** $\Phi[\mathcal{J}]$, un ensemble $A \subseteq \mathcal{J}$ est dit **$\Phi[\mathcal{J}]$ -clos** si pour toute règle :

$$(R) \frac{j_1 \quad \cdots \quad j_n}{j} \in \Phi[\mathcal{J}]$$

$$\{j_1, \dots, j_n\} \subseteq A \Rightarrow j \in A.$$

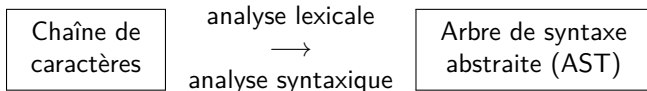
L'**ensemble défini inductivement** par $\Phi[\mathcal{J}]$ est l'intersection de tous les ensembles $\Phi[\mathcal{J}]$ -clos :

$$\text{Ind}(\Phi[\mathcal{J}]) = \bigcap \{A \subseteq \mathcal{J} \mid A \text{ est } \Phi[\mathcal{J}]\text{-clos}\}$$

Théorème : $\text{Ind}(\Phi[\mathcal{J}]) = \text{Th}(\Phi[\mathcal{J}])$

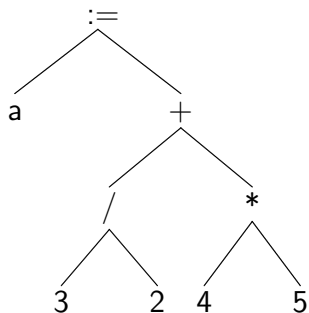
PREUVE : exercice ...

Syntaxe abstraite



a := 3 / 2 + 4 * 5;

\longrightarrow



syntaxe concrète

syntaxe abstraite

Outline

- Motivations
- Rappels
- Expressions arithmétiques
 - Définition inductive
 - Sémantique dénotationnelle
 - Sémantique opérationnelle à grands pas
 - Sémantique opérationnelle à petits pas
- Expressions booléennes
- IMP

Définition inductive des expressions arithmétiques

Définition inductive de l'ensemble E_A par un **système d'inférence**

Jugements : $(\mathbb{Z} \cup V \cup \{+, -, \times, /\})^*$

$$\begin{array}{c}
 \hline
 (\mathbb{A}_1) \frac{\quad}{n} \quad (n \in \mathbb{Z}) \qquad (\mathbb{A}_2) \frac{\quad}{x} \quad (x \in V) \\
 (\mathbb{A}_3) \frac{a_1 \quad a_2}{a_1 + a_2} \quad (\mathbb{A}_4) \frac{a_1 \quad a_2}{a_1 - a_2} \quad (\mathbb{A}_5) \frac{a_1 \quad a_2}{a_1 \times a_2} \quad (\mathbb{A}_6) \frac{a_1 \quad a_2}{a_1 / a_2} \\
 \hline
 \end{array}$$

La règle \mathbb{A}_1 peut s'appliquer pour tout $n \in \mathbb{Z}$: elle dénote en fait un ensemble de règles :

$$Inst(\mathbb{A}_1) = \left\{ \dots, (\mathbb{A}_1) \frac{\quad}{-2}, (\mathbb{A}_1) \frac{\quad}{-1}, (\mathbb{A}_1) \frac{\quad}{0}, (\mathbb{A}_1) \frac{\quad}{1}, (\mathbb{A}_1) \frac{\quad}{2}, \dots \right\}$$

n est une **méta-variable**, \mathbb{A}_1 est une **méta-règle** dont les instances sont les règles contenues dans l'ensemble $Inst(\mathbb{A}_1)$ qui sont obtenues en remplaçant les méta-variables par des éléments de \mathbb{Z} .

Syntaxe abstraite – Implantation

$37 + v \in E_A ?$

$$(\mathbb{A}_3) \frac{(\mathbb{A}_1) \frac{}{37} \quad (\mathbb{A}_2) \frac{}{v}}{37 + v}$$

Arbre d'inférence = arbre de syntaxe abstraite

Expressions arithmétiques en OCAML :

```
type 'a exp_arith =
| Ent of int
| Var of 'a
| Plus of 'a exp_arith * 'a exp_arith
| Moins of 'a exp_arith * 'a exp_arith
| Fois of 'a exp_arith * 'a exp_arith
| Div of 'a exp_arith * 'a exp_arith
```

Induction

Prouver par **induction** une propriété P sur tous les éléments de l'ensemble $\text{Ind}(\Phi[\mathcal{J}])$,
 c'est prouver que pour toute règle de $\Phi[\mathcal{J}]$, si chacune des prémisses satisfait P (**hypothèse d'induction**), alors la conclusion satisfait aussi P .

Théorème

Si $(\forall \frac{j_1 \dots j_n}{j} \in \Phi[\mathcal{J}] (\forall k \in \{j_1, \dots, j_n\} P(k)) \text{ implique } P(j))$
 alors $\forall x \in \text{Ind}(\Phi[\mathcal{J}]) P(x)$.

PREUVE : exercice ...

Exemple Définition inductive de \mathbb{N} : $(N_1) \bar{0}$, $(N_2) \frac{n}{n+1}$

Induction sur \mathbb{N} :

Si $P(0)$ et si pour tout $n \in \mathbb{N}$, $P(n) \Rightarrow P(n+1)$, alors
 $\forall n \in \mathbb{N} P(n)$.

Induction sur les expressions arithmétiques

- si $\forall n \in \mathbb{Z} P(n)$
- et $\forall x \in V P(x)$
- et $\forall a_1, a_2 \in E_A (P(a_1) \text{ et } P(a_2)) \Rightarrow P(a_1 + a_2)$
- et $\forall a_1, a_2 \in E_A (P(a_1) \text{ et } P(a_2)) \Rightarrow P(a_1 - a_2)$
- et $\forall a_1, a_2 \in E_A (P(a_1) \text{ et } P(a_2)) \Rightarrow P(a_1 \times a_2)$
- et $\forall a_1, a_2 \in E_A (P(a_1) \text{ et } P(a_2)) \Rightarrow P(a_1 / a_2)$
- alors $\forall a \in E_A P(a)$

Sémantique dénotationnelle

On associe à chaque mot du langage une **dénotation** (\simeq interprétation) qui abstrait son comportement

Exemples

- ▶ ensembles pour les données
- ▶ fonctions transformant des états pour les procédures
- ▶ etc.

Ici on cherche à interpréter des expressions arithmétiques

- ▶ candidat naturel : **entiers relatifs**

Interprétation des expressions arithmétiques

- ▶ Pour **interpréter** les expressions arithmétiques :
 - on associe une “**signification**” à chacun des **symboles** pouvant apparaître dans une expression
 - puis à chacune des **constructions** possibles.
- ▶ Interpréter une expression arithmétique, c'est lui donner une **valeur** appartenant à un certain ensemble :

$$\mathbb{V} = \mathbb{Z} \cup \{\text{Err}\}$$

Err dénote une erreur lors de l'interprétation ... lors d'une division par 0, par exemple.

En OCAML : **Err** est une exception

Interprétation des symboles

- Interprétation des **variables** par une **valuation** : $\mathcal{V}[\mathbb{Z}] = \{V \rightarrow \mathbb{Z}\}$ (mémoire, environnement, ...)
- Interpréter des symboles de $\mathbb{Z} \cup \{+, -, \times, /\}$:
 n est interprété par lui-même : $\llbracket n \rrbracket = n$

$v_1 \left(\begin{array}{c} \llbracket + \rrbracket \\ \llbracket - \rrbracket \\ \llbracket \times \rrbracket \end{array} \right) v_2$	$v_2 \in \mathbb{Z}$	Err
$v_1 \in \mathbb{Z}$	$v_1 \left(\begin{array}{c} + \\ - \\ \times \end{array} \right) v_2$	Err
Err	Err	Err

$v_1 \llbracket / \rrbracket v_2$	$v_2 \in \mathbb{Z} \setminus \{0\}$	0	Err
$v_1 \in \mathbb{Z}$	v_1 / v_2	Err	Err
Err	Err	Err	Err

Schéma d'interprétation des expressions arithmétiques

$$\mathcal{A}[_]_ : E_A \rightarrow \mathcal{V}[\mathbb{Z}] \rightarrow \mathbb{Z} \cup \{\text{Err}\}$$

$$\mathcal{A}[e]_\sigma = \begin{cases} n & \text{si } e = n \\ \sigma(x) & \text{si } e = x \\ \mathcal{A}[e_1]_\sigma [+]\mathcal{A}[e_2]_\sigma & \text{si } e = e_1 + e_2 \\ \mathcal{A}[e_1]_\sigma [-]\mathcal{A}[e_2]_\sigma & \text{si } e = e_1 - e_2 \\ \mathcal{A}[e_1]_\sigma [\times]\mathcal{A}[e_2]_\sigma & \text{si } e = e_1 \times e_2 \\ \mathcal{A}[e_1]_\sigma [/\]\mathcal{A}[e_2]_\sigma & \text{si } e = e_1/e_2 \end{cases}$$

Ce schéma correspond exactement à celui de l'interprétation d'un ensemble de termes.

Evaluation des expressions arithmétiques :

Implantation en OCAML

```
exception Err
```

```
let rec eval_arith s e = match e with  
  Ent n -> n  
| Var x -> (s x)  
| Plus(e1,e2) -> (eval_arith s e1) + (eval_arith s e2)  
| Moins(e1,e2) -> (eval_arith s e1) -(eval_arith s e2)  
| Fois(e1,e2) -> (eval_arith s e1) * (eval_arith s e2)  
| Div(e1,e2) -> let n1 = (eval_arith s e1) in  
                 let n2 = (eval_arith s e2) in  
                 if n2=0 then raise Err else n1/n2
```

```
eval_arith : ('a -> int) -> 'a exp_arith -> int
```

Sémantique opérationnelle d'évaluation à grands pas

Décrire le processus d'**évaluation** à l'aide de systèmes d'inférence dont les règles portent sur des jugements exprimant qu'une valeur $v \in \mathbb{Z}$ est le résultat de l'évaluation d'une expression $a \in E_A$ étant donnée une valuation $\sigma \in \mathcal{V}[\mathbb{Z}]$:

$$\langle a, \sigma \rangle \rightsquigarrow v$$

Caractériser un sous-ensemble des jugements de la forme $\langle a, \sigma \rangle \rightsquigarrow v$ qui sont "**corrects d'un point de vue sémantique**" : il s'agit de l'ensemble des théorèmes d'un système d'inférence.

Exemples

$\langle 3 + 2, \sigma \rangle \rightsquigarrow 5$ est un théorème

$\langle 3 + 2, \sigma \rangle \rightsquigarrow 1$ n'est pas un théorème, il s'agit d'un jugement "syntaxiquement" correct mais "sémantiquement" erroné

Règles d'évaluation à grands pas

$$(A_1) \frac{}{\langle n, \sigma \rangle \rightsquigarrow n} \quad (n \in \mathbb{Z}) \quad (A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow \sigma(x)} \quad (x \in V)$$

$$(A_3) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 + a_2, \sigma \rangle \rightsquigarrow n_1 \llbracket + \rrbracket n_2} \quad (A_4) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 \times a_2, \sigma \rangle \rightsquigarrow n_1 \llbracket \times \rrbracket n_2}$$

$$(A_5) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 - a_2, \sigma \rangle \rightsquigarrow n_1 \llbracket - \rrbracket n_2} \quad (A_6) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 / a_2, \sigma \rangle \rightsquigarrow n_1 \llbracket / \rrbracket n_2}$$

Évaluation : exemple

σ : valuation telle que $\sigma(x) = 2$

$\langle (2 \times 3) + x, \sigma \rangle \rightsquigarrow 8$?

$$(A_3) \frac{(A_4) \frac{(A_1) \frac{\langle 2, \sigma \rangle \rightsquigarrow 2}{(A_1) \frac{\langle 3, \sigma \rangle \rightsquigarrow 3}{(A_2) \frac{\langle x, \sigma \rangle \rightsquigarrow 2}}{\langle 2 \times 3, \sigma \rangle \rightsquigarrow 6}}{\langle (2 \times 3) + x, \sigma \rangle \rightsquigarrow 8}}{\langle (2 \times 3) + x, \sigma \rangle \rightsquigarrow 8}}{\langle (2 \times 3) + x, \sigma \rangle \rightsquigarrow 8}}$$

Déterminisme de l'évaluation des expressions arithmétiques (1)

Dans la suite on pose $\mathbb{VA} = \mathbb{Z} \cup \{Err\}$.

Proposition :

$$\forall a \in E_A \forall \sigma \in \mathcal{V}[\mathbb{Z}] \forall v_1, v_2 \in \mathbb{VA} \\ (\langle a, \sigma \rangle \rightsquigarrow v_1 \text{ et } \langle a, \sigma \rangle \rightsquigarrow v_2) \Rightarrow v_1 = v_2$$

PREUVE : Par **induction** sur a .

Déterminisme de l'évaluation des expressions arithmétiques (2)

Si $a = z \in \mathbb{Z}$, alors seule la règle A_1 a pu être appliquée et on a donc $v_1 = z$ et $v_2 = z$ ce qui permet d'obtenir $v_1 = v_2$.

Déterminisme de l'évaluation des expressions arithmétiques (3)

Si $a = x \in V$, alors seule la règle A_2 a pu être appliquée et on a donc $v_1 = \sigma(x)$ et $v_2 = \sigma(x)$ ce qui permet d'obtenir $v_1 = v_2$.

Déterminisme de l'évaluation des expressions arithmétiques (4)

Si $a = a_1 + a_2$, alors, par **hypothèse d'induction**, on a :

$$\begin{aligned} \forall \sigma \in \mathcal{V}[\mathbb{Z}] \forall v_1, v_2 \in \mathbb{V}\mathbb{A} (\langle a_1, \sigma \rangle \rightsquigarrow v_1 \text{ et } \langle a_1, \sigma \rangle \rightsquigarrow v_2) &\Rightarrow v_1 = v_2 \\ \forall \sigma \in \mathcal{V}[\mathbb{Z}] \forall v_1, v_2 \in \mathbb{V}\mathbb{A} (\langle a_2, \sigma \rangle \rightsquigarrow v_1 \text{ et } \langle a_2, \sigma \rangle \rightsquigarrow v_2) &\Rightarrow v_1 = v_2 \end{aligned}$$

Puisque, par hypothèse, $\langle a_1 + a_2, \sigma \rangle \rightsquigarrow v_1$, c'est que par la règle du $+$, il existe n_1 et n_2 tels que

$$\langle a_1, \sigma \rangle \rightsquigarrow n_1, \langle a_2, \sigma \rangle \rightsquigarrow n_2 \text{ et } v_1 = n_1 \llbracket + \rrbracket n_2$$

Par **hypothèse d'induction**, on a $n_1 = n'_1$ et $n_2 = n'_2$ ce qui permet d'obtenir $v_1 = v_2$.

Déterminisme de l'évaluation des expressions arithmétiques (7)

Raisonnement similaire pour $a = a_1 - a_2$, $a = a_1 \times a_2$ et $a = a_1 / a_2$

Cas de la division : à faire

Exercice : ajouter les expressions conditionnelles au langage des expressions arithmétiques. Spécifier les règles nécessaires et faire la preuve que le langage reste déterministe.

Équivalence des sémantiques opérationnelle et dénotationnelle

Théorème :

$$\forall \sigma \in \mathcal{V}[\mathbb{Z}] \quad \forall e \in E_A \quad \forall v \in \mathbb{V}_A \quad \langle e, \sigma \rangle \rightsquigarrow v \Leftrightarrow \mathcal{A}[e]_\sigma = v$$

Évaluation des expressions arithmétiques : variables

Le résultat de l'évaluation d'une expression arithmétique a , étant donnée une valuation σ , ne dépend que de la valeur associée par σ aux variables dans $\vartheta(a)$.

$$\vartheta(a) = \begin{cases} \emptyset & \text{si } a = n \\ \{x\} & \text{si } a = x \\ \vartheta(a_1) \cup \vartheta(a_2) & \text{si } a = a_1 + a_2 \text{ ou } a = a_1 - a_2 \\ & \text{ou } a = a_1 \times a_2 \text{ ou } a = a_1 / a_2 \end{cases}$$

Proposition :

$$\forall a \in E_A, \forall \sigma_1, \sigma_2 \in \mathcal{V}[\mathbb{Z}], \\ \sigma_1 =_{\vartheta(a)} \sigma_2 \Rightarrow (\forall v \in \mathbb{V}\mathbb{A} \langle a, \sigma_1 \rangle \rightsquigarrow v \Leftrightarrow \langle a, \sigma_2 \rangle \rightsquigarrow v)$$

où $\forall \sigma_1, \sigma_2 \in \mathcal{V}[\mathbb{Z}] \quad \sigma_1 =_X \sigma_2$ ssi $\forall x \in X, \quad \sigma_1(x) = \sigma_2(x)$

Exercice Montrer que si $X_1 \subseteq X_2$ et $\sigma_1 =_{X_2} \sigma_2$, alors $\sigma_1 =_{X_1} \sigma_2$.

Évaluation des expressions arithmétiques : variables

PREUVE : par **induction** sur a

Si $a = n \in \mathbb{Z}$, alors $\vartheta(a) = \emptyset$ et seule la règle A_1 a pu être utilisée pour établir $\langle a, \sigma_1 \rangle \rightsquigarrow v$ avec $n = v$ et en utilisant cette même règle, il vient $\langle a, \sigma_2 \rangle \rightsquigarrow v$.

Si $a = x \in V$, alors $\vartheta(a) = \{x\}$, et seule la règle A_2 a pu être utilisée pour établir $\langle a, \sigma_1 \rangle \rightsquigarrow \sigma_1(x)$. Par hypothèse, $\sigma_1 =_{\vartheta(a)} \sigma_2$, et donc $\sigma_1(x) = \sigma_2(x) = n$ et il suffit d'appliquer la règle A_2 pour établir $\langle a, \sigma_2 \rangle \rightsquigarrow n$.

Évaluation des expressions arithmétiques : variables

Si $a = a_1 + a_2$, alors si $\langle a, \sigma_1 \rangle \rightsquigarrow v$ a été obtenu :

À partir de la règle A_3 :

$$(A_3) \frac{\begin{array}{c} \vdots \\ (A_i) \frac{}{\langle a_1, \sigma_1 \rangle \rightsquigarrow n_1} \\ \vdots \end{array} \quad \begin{array}{c} \vdots \\ (A_j) \frac{}{\langle a_2, \sigma_1 \rangle \rightsquigarrow n_2} \\ \vdots \end{array}}{\langle a_1 + a_2, \sigma_1 \rangle \rightsquigarrow v}$$

avec $v = n_1 \llbracket + \rrbracket n_2$. Par **hypothèse d'induction**, puisque $\vartheta(a_1) \subseteq \vartheta(a)$ et $\vartheta(a_2) \subseteq \vartheta(a)$ (et donc $\sigma_1 =_{\vartheta(a_1)} \sigma_2$ et $\sigma_1 =_{\vartheta(a_2)} \sigma_2$), on a $\langle a_1, \sigma_2 \rangle \rightsquigarrow n_1$ et $\langle a_2, \sigma_2 \rangle \rightsquigarrow n_2$, et donc en appliquant la règle A_5 , on a $\langle a_1 + a_2, \sigma_2 \rangle \rightsquigarrow v$.

Si $a = a_1 - a_2$, $a = a_1 \times a_2$ ou $a = a_1 / a_2$ alors le raisonnement est similaire au cas précédent

Expressions arithmétiques équivalentes (1)

Caractériser les expressions arithmétiques qui s'évaluent à la même valeur.

$$a_1 \sim a_2 \Leftrightarrow (\forall v \in \mathbb{VA} \forall \sigma \in \mathcal{V}[\mathbb{Z}] \quad \langle a_1, \sigma \rangle \rightsquigarrow v \Leftrightarrow \langle a_2, \sigma \rangle \rightsquigarrow v)$$

Proposition : \sim est une **congruence**.

Une congruence est une relation d'équivalence compatible avec la structure des expressions :

si $a \sim b$ alors pour tout c on a $a + c \sim b + c$,

$$c + a \sim c + b,$$

$$a - c \sim b - c,$$

...

Expressions arithmétiques équivalentes (2)

Exemple $x + x \sim 2 \times x$

Si $\langle x + x, \sigma \rangle \rightsquigarrow n$, alors on a :

$$(A_3) \frac{(A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow \sigma(x)} \quad (A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow \sigma(x)}}{\langle x + x, \sigma \rangle \rightsquigarrow \sigma(x) \llbracket + \rrbracket \sigma(x)}$$

où $\sigma(x) + \sigma(x) = n$. D'autre part, on peut construire l'arbre :

$$(A_4) \frac{(A_1) \frac{}{\langle 2, \sigma \rangle \rightsquigarrow 2} \quad (A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow \sigma(x)}}{\langle 2 \times x, \sigma \rangle \rightsquigarrow 2 \llbracket \times \rrbracket \sigma(x)}$$

Puisque $2 \llbracket \times \rrbracket \sigma(x) = \sigma(x) \llbracket + \rrbracket \sigma(x) = n$, on a bien $\langle 2 \times x, \sigma \rangle \rightsquigarrow n$.
De même, on montre que si $\langle 2 \times x, \sigma \rangle \rightsquigarrow n$, alors $\langle x + x, \sigma \rangle \rightsquigarrow n$.

Expressions arithmétiques équivalentes (3)

On peut donc remplacer dans toute expression l'expression $x + x$ par $2 \times x$

Puisque \sim est une congruence, on a
 $(x + x) + (x + x) \sim (2 \times x) + (2 \times x)$.

Dans la suite on allège la sémantique en ne donnant que les règles d'inférence qui conduisent à des valeurs entières.

L'interprétation du symbole $+$ devient alors l'addition entière. On notera dorénavant $\llbracket + \rrbracket$ par $+$. C'est donc le contexte qui permet de dire si on parle du symbole $+$ ou de son interprétation. (Si il y a ambiguïté, on précisera.)

Les règles en sémantique à grands pas sont alors les suivantes :

$$(A_1) \frac{}{\langle n, \sigma \rangle \rightsquigarrow n} \quad (n \in \mathbb{Z}) \quad (A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow \sigma(x)} \quad (x \in V)$$

$$(A_3) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 + a_2, \sigma \rangle \rightsquigarrow n_1 + n_2} \quad (A_4) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 \times a_2, \sigma \rangle \rightsquigarrow n_1 \times n_2}$$

$$(A_5) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 - a_2, \sigma \rangle \rightsquigarrow n_1 - n_2} \quad (A_6) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 / a_2, \sigma \rangle \rightsquigarrow n_1 / n_2} \quad (n_2 \neq 0)$$

Grands pas vs petits pas

Sémantique opérationnelle à **grands pas** :

- ▶ on a accès uniquement à la **valeur finale**, pas la manière dont elle est calculée
 - **ordre d'évaluation** ?
 - états intermédiaires ?
 - calculs en parallèle, **terminaison** ?

Besoin d'une sémantique à **grain plus fin** (moins abstraite)

- ▶ sémantique opérationnelle à **petits pas**

Sémantique opérationnelle d'évaluation à petits pas (1)

On dit aussi *sémantique de réduction*.

Rendre compte des étapes qui ont conduit au résultat d'un calcul de manière "plus fine" que la sémantique à grands pas.

Relation \hookrightarrow de transition entre configurations $\langle e, \sigma \rangle$:

$\langle e, \sigma \rangle \hookrightarrow \langle e', \sigma \rangle$ est un jugement exprimant le fait que l'évaluation d'une expression e à partir d'une valuation σ conduit à évaluer une expression e' "plus simple" à partir de la même valuation.

Exemples $\langle 3 + (2 \times 8), \sigma \rangle \hookrightarrow \langle 3 + 16, \sigma \rangle$
 $\langle 2/x, \sigma \rangle \hookrightarrow \langle 2/0, \sigma \rangle$ si $\sigma(x) = 0$

Configuration : paire $\langle e, \sigma \rangle$ où $e \in E_A$ et $\sigma \in \mathcal{V}[\mathbb{Z}]$

Configuration terminale : $\langle v, \sigma \rangle$ où $v \in \mathbb{Z}$

Sémantique opérationnelle d'évaluation à petits pas (2)

Séquence de calcul : $\langle e_0, \sigma \rangle \hookrightarrow \langle e_1, \sigma \rangle \hookrightarrow \langle e_2, \sigma \rangle \hookrightarrow \dots$ telle que $\forall i \geq 0, \langle e_i, \sigma \rangle \hookrightarrow \langle e_{i+1}, \sigma \rangle$ admette un arbre d'inférence à partir des règles définissant \hookrightarrow .

$\langle e, \sigma \rangle \xrightarrow{\star} \langle e', \sigma \rangle$ ssi il existe une séquence de calcul de longueur finie $\langle e_0, \sigma \rangle \hookrightarrow \langle e_1, \sigma \rangle \hookrightarrow \langle e_2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle e_k, \sigma \rangle$ avec $e = e_0$ et $e' = e_k$.

$\langle e, \sigma \rangle \xrightarrow{k} \langle e', \sigma \rangle$ ssi il existe une séquence de calcul de longueur k $\langle e_0, \sigma \rangle \hookrightarrow \langle e_1, \sigma \rangle \hookrightarrow \langle e_2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle e_k, \sigma \rangle$ avec $e = e_0$ et $e' = e_k$.

De plus, si $e_k = v \in \mathbb{Z}$ (configuration terminale), alors $\langle e, \sigma \rangle \xrightarrow{\star} v$.
 e s'évalue en la valeur v ($\in \mathbb{Z}$) si et seulement il existe une séquence finie $\langle e, \sigma \rangle \xrightarrow{\star} \langle v, \sigma \rangle$

Sémantique opérationnelle d'évaluation à petits pas (3)

Deux sortes de règles :

- règles de réduction élémentaires

$$(AS_1) \frac{}{\langle x, \sigma \rangle \hookrightarrow \langle \sigma(x), \sigma \rangle} \text{ (si } x \in V \text{)}$$

$$(AS_4) \frac{}{\langle n_1 \text{ op}_2 n_2, \sigma \rangle \hookrightarrow \langle n, \sigma \rangle} \left(\begin{array}{l} \text{si } n_1, n_2 \in \mathbb{Z} \text{ et } \text{op}_2 \in \{+, -, \times\} \\ \text{et } n = n_1 \llbracket \text{op}_2 \rrbracket n_2 \end{array} \right)$$

$$(AS_5) \frac{}{\langle n_1/n_2, \sigma \rangle \hookrightarrow \langle n, \sigma \rangle} \text{ (si } n_1 \in \mathbb{Z} \text{ et } n_2 \in \mathbb{Z} \setminus \{0\} \text{ et } n = n_1/n_2 \text{)}$$

Sémantique opérationnelle d'évaluation à petits pas (4)

- Etapes de simplification (passage au contexte, propagation)
- Choisissons un mode d'évaluation "de gauche à droite"

$$(AS_7) \frac{\langle e_1, \sigma \rangle \hookrightarrow \langle e'_1, \sigma \rangle}{\langle e_1 \text{ op}_2 e_2, \sigma \rangle \hookrightarrow \langle e'_1 \text{ op}_2 e_2, \sigma \rangle} \text{ (si } \text{op}_2 \in \{+, -, \times, /\})$$

$$(AS_8) \frac{\langle e_2, \sigma \rangle \hookrightarrow \langle e'_2, \sigma \rangle}{\langle n_1 \text{ op}_2 e_2, \sigma \rangle \hookrightarrow \langle n_1 \text{ op}_2 e'_2, \sigma \rangle} \left(\begin{array}{l} \text{si } n_1 \in \mathbb{Z} \\ \text{et } \text{op}_2 \in \{+, -, \times, /\} \end{array} \right)$$

Sémantique opérationnelle d'évaluation à petits pas : Exemple 1

σ : valuation telle que $\sigma(x) = 1$, $\sigma(y) = 3$ et $\sigma(z) = 2$

Séquence de calcul qui permet de décrire l'évaluation de l'expression

$$\begin{aligned} & \langle (z + (x - y)) \times (x + 7), \sigma \rangle \\ \hookrightarrow & \langle (2 + (x - y)) \times (x + 7), \sigma \rangle \\ \hookrightarrow & \langle (2 + (1 - y)) \times (x + 7), \sigma \rangle \\ \hookrightarrow & \langle (2 + (1 - 3)) \times (x + 7), \sigma \rangle \\ \hookrightarrow & \langle (2 + -2) \times (x + 7), \sigma \rangle \\ \hookrightarrow & \langle 0 \times (x + 7), \sigma \rangle \\ \hookrightarrow & \langle 0 \times (1 + 7), \sigma \rangle \\ \hookrightarrow & \langle 0 \times 8, \sigma \rangle \\ \hookrightarrow & \langle 0, \sigma \rangle \end{aligned}$$

Sémantique opérationnelle d'évaluation à petits pas : Exemple 2

σ : valuation telle que $\sigma(x) = 1$, $\sigma(y) = 3$ et $\sigma(z) = 2$

$$\begin{aligned} & \langle (z + (x - y)) / (x - 1), \sigma \rangle \\ \hookrightarrow & \langle (2 + (x - y)) / (x - 1), \sigma \rangle \\ \hookrightarrow & \langle (2 + (1 - y)) / (x - 1), \sigma \rangle \\ \hookrightarrow & \langle (2 + (1 - 3)) / (x - 1), \sigma \rangle \\ \hookrightarrow & \langle (2 + -2) / (x - 1), \sigma \rangle \\ \hookrightarrow & \langle 0 / (x - 1), \sigma \rangle \\ \hookrightarrow & \langle 0 / (1 - 1), \sigma \rangle \\ \hookrightarrow & \langle 0 / 0, \sigma \rangle \end{aligned}$$

Séquence bloquée : configuration terminale qui ne correspond pas à une valeur.

\hookrightarrow est déterministe

Proposition :

$$\forall a \in E_A \forall \sigma \in \mathcal{V}[\mathbb{Z}] \forall v_1, v_2 \in \mathbb{VA} \\ (\langle a, \sigma \rangle \hookrightarrow \langle a_1, \sigma_1 \rangle \text{ et } \langle a, \sigma \rangle \hookrightarrow \langle a_2, \sigma_2 \rangle) \Rightarrow (a_1 = a_2 \text{ et } \sigma_1 = \sigma_2)$$

PREUVE : Par induction sur a et étude de cas

Autre présentation : Contexte d'évaluation (1)

Autre présentation plus synthétique (Wright Felleisen)

Toutes les règles de simplification sont réduites à une seule : la règle de passage au contexte

Contexte d'évaluation : expression dont une feuille unique est remplacée par un trou (métavariable, placeholder)

Les contextes sont décrits par une grammaire, ils spécifient l'ordre d'évaluation :

$$C ::= \bullet \mid C \text{ op } e \mid v \text{ op } C$$

avec $op \in \{\times, /, +, -\}$, e une expression et v un entier (valeur)

Exemples : $(\bullet + x) \times 2$ est un contexte d'évaluation valide (C_1)

$\bullet + \bullet$ n'est pas un contexte

$(y \times x) + \bullet$ n'est pas un contexte

Contexte d'évaluation (2)

$C[e]$: expression obtenue en remplissant le trou avec e

$$C_1[x + y] = ((x + y) + x) \times 2$$

La sémantique à réduction est spécifiée par les règles de réduction élémentaires et la règle suivante :

$$(\text{contexte}) \frac{\langle e_1, \sigma \rangle \hookrightarrow \langle e_2, \sigma \rangle}{\langle C[e_1], \sigma \rangle \hookrightarrow \langle C[e_2], \sigma \rangle}$$

Equivalence des sémantiques (1)

Proposition $\forall e \in E_A \forall \sigma \in \mathcal{V}[\mathbb{Z}] \forall v \in \mathbb{V} \langle e, \sigma \rangle \rightsquigarrow v \Rightarrow \langle e, \sigma \rangle \xrightarrow{*} v$

PREUVE Induction sur e .

- si $e = n \in \mathbb{Z}$, alors l'arbre de dérivation de $\langle e, \sigma \rangle \rightsquigarrow v$ est :

$$(A_1) \frac{}{\langle n, \sigma \rangle \rightsquigarrow n} \quad (n = v)$$

On a alors clairement $\langle n, \sigma \rangle \xrightarrow{*} n$ (configuration terminale).

- si $e = x \in V$, alors l'arbre de dérivation de $\langle e, \sigma \rangle \rightsquigarrow v$ est :

$$(A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow \sigma(x)} \quad (\sigma(x) = v)$$

Dans ce cas, la règle AS_1 permet de construire un arbre de dérivation pour $\langle x, \sigma \rangle \xrightarrow{*} \langle \sigma(x), \sigma \rangle$ et on a donc $\langle x, \sigma \rangle \xrightarrow{*} v$.

Équivalence des sémantiques (5)

- si $e = a_1 \text{ op}_2 a_2$ avec $\text{op}_2 \in \{+, -, \times\}$

On a donc $\langle a_1, \sigma \rangle \rightsquigarrow n_1$ et $\langle a_2, \sigma \rangle \rightsquigarrow n_2$ ($n_1, n_2 \in \mathbb{Z}$)

Par **hypothèse d'induction** on a $\langle a_1, \sigma \rangle \xrightarrow{*} n_1$ et $\langle a_2, \sigma \rangle \xrightarrow{*} n_2$ et il existe donc 2 séquences de calcul :

$$\langle a_1, \sigma \rangle = \langle a_1^0, \sigma \rangle \hookrightarrow \langle a_1^1, \sigma \rangle \hookrightarrow \langle a_1^2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle a_1^{k_1}, \sigma \rangle = \langle n_1, \sigma \rangle \quad (1)$$

$$\langle a_2, \sigma \rangle = \langle a_2^0, \sigma \rangle \hookrightarrow \langle a_2^1, \sigma \rangle \hookrightarrow \langle a_2^2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle a_2^{k_2}, \sigma \rangle = \langle n_2, \sigma \rangle \quad (2)$$

La règle AS_7 permet de transformer chaque $\langle a_1^i, \sigma \rangle \hookrightarrow \langle a_1^{i+1}, \sigma \rangle$ de (1) en $\langle a_1^i \text{ op}_2 a_2, \sigma \rangle \hookrightarrow \langle a_1^{i+1} \text{ op}_2 a_2, \sigma \rangle$. On obtient alors :

$$\begin{aligned} \langle a_1 \text{ op}_2 a_2, \sigma \rangle &= \langle a_1^0 \text{ op}_2 a_2, \sigma \rangle \hookrightarrow \langle a_1^1 \text{ op}_2 a_2, \sigma \rangle \hookrightarrow \langle a_1^2 \text{ op}_2 a_2, \sigma \rangle \hookrightarrow \dots \\ &\dots \hookrightarrow \langle a_1^{k_1} \text{ op}_2 a_2, \sigma \rangle = \langle n_1 \text{ op}_2 a_2, \sigma \rangle \end{aligned} \quad (3)$$

Equivalence des sémantiques (6)

La règle AS_8 permet de transformer chaque $\langle a_2^i, \sigma \rangle \hookrightarrow \langle a_2^{i+1}, \sigma \rangle$ de (2) en $\langle n_1 \text{ op}_2 a_2^i, \sigma \rangle \hookrightarrow \langle n_1 \text{ op}_2 a_2^{i+1}, \sigma \rangle$. On obtient alors :

$$\begin{aligned} \langle n_1 \text{ op}_2 a_2, \sigma \rangle &= \langle n_1 \text{ op}_2 a_2^0, \sigma \rangle \hookrightarrow \langle n_1 \text{ op}_2 a_2^1, \sigma \rangle \hookrightarrow \langle n_1 \text{ op}_2 a_2^2, \sigma \rangle \hookrightarrow \dots \\ \dots \hookrightarrow \langle n_1 \text{ op}_2 a_2^{k_2}, \sigma \rangle &= \langle n_1 \text{ op}_2 n_2, \sigma \rangle \end{aligned} \tag{4}$$

Enfin, la règle AS_4 permet d'obtenir la transition $\langle n_1 \text{ op}_2 n_2, \sigma \rangle \hookrightarrow \langle n, \sigma \rangle$ où n est la valeur de $n_1 \text{ op}_2 n_2$, et en concaténant les séquences (3) et (4) avec cette transition on obtient finalement $\langle a_1 \text{ op}_2 a_2, \sigma \rangle \xrightarrow{*} n_1 \text{ op}_2 n_2$.

- Si $e = a_1/a_2, \dots$

Equivalence des sémantiques (7)

Pour montrer

$\forall e \in E_A \forall \sigma \in \mathcal{V}[\mathbb{Z}] \forall v \in \mathbb{V} \langle e, \sigma \rangle \xrightarrow{\star} v \Rightarrow \langle e, \sigma \rangle \rightsquigarrow v$, on montre tout d'abord :

Lemme $\forall e_1, e_2 \in E_A \forall \sigma \in \mathcal{V}[\mathbb{Z}] \forall v \in \mathbb{V}$, si $\langle e_1, \sigma \rangle \xrightarrow{\star} \langle e_2, \sigma \rangle$ et $\langle e_2, \sigma \rangle \rightsquigarrow v$, alors $\langle e_1, \sigma \rangle \rightsquigarrow v$.

PREUVE

Induction sur l'étape de réduction

Equivalence des sémantiques (8)

Proposition $\forall e \in E_A \forall \sigma \in \mathcal{V}[\mathbb{Z}] \forall v \in \mathbb{Z} \langle e, \sigma \rangle \xrightarrow{\star} v \Rightarrow \langle e, \sigma \rangle \rightsquigarrow v$

PREUVE

Puisque $\langle e, \sigma \rangle \xrightarrow{\star} v$, il existe une séquence :

$$\langle e, \sigma \rangle = \langle e^0, \sigma \rangle \hookrightarrow \langle e^1, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle e^k, \sigma \rangle$$

avec $e^k = v \in \mathbb{Z}$. La preuve s'obtient par induction sur la longueur k de cette séquence.

Si $k = 0$, alors $e = v$ et la règle A_1 permet de conclure.

Si $k = k_0 + 1$, alors on conclut en utilisant l'hypothèse de récurrence et le lemme précédent.

Outline

- Motivations
- Rappels
- Expressions arithmétiques
- Expressions booléennes
- IMP
- MiniML

Expressions booléennes

On suit exactement la même démarche qu'avec les expressions arithmétiques ...

Définition inductive de l'ensemble E_B des expressions booléennes.

Jugements :

$(\mathbb{Z} \cup \mathbb{V} \cup \{+, -, \times, /\} \cup \{\text{true}, \text{false}\} \cup \{=, \leq\} \cup \{\text{or}, \text{and}, \text{not}\})^*$

$$(\mathbb{B}_1) \frac{}{t} \quad (t \in \{\text{true}, \text{false}\}) \quad (\mathbb{B}_5) \frac{}{a_1 \leq a_2} \quad (\mathbb{B}_6) \frac{}{a_1 = a_2} \quad (a_1, a_2 \in E_A)$$

$$(\mathbb{B}_2) \frac{b}{\text{not } b} \quad (\mathbb{B}_3) \frac{b_1 \quad b_2}{b_1 \text{ or } b_2} \quad (\mathbb{B}_4) \frac{b_1 \quad b_2}{b_1 \text{ and } b_2}$$

Exemple not $(x = 0 \text{ and } x \leq (7/z))$

Interprétation des expressions booléennes

Interpréter une expression booléenne c'est lui donner une **valeur** appartenant à $\mathbb{B} \cup \{\text{Err}\}$ avec $\mathbb{B} = \{\text{true}, \text{false}\}$

... à partir de l'interprétation des expressions arithmétiques, il reste à interpréter les symboles de

$\{\text{true}, \text{false}\} \cup \{=, \leq\} \cup \{\text{or}, \text{and}, \text{not}\}$.

$t \in \mathbb{B}$ est interprété par lui-même : $\llbracket \text{true} \rrbracket = \text{true}$ et $\llbracket \text{false} \rrbracket = \text{false}$

$v_1 \left(\begin{array}{c} \llbracket = \rrbracket \\ \llbracket \leq \rrbracket \end{array} \right) v_2$	$v_2 \in \mathbb{Z}$	Err	$v_1 \left(\begin{array}{c} \llbracket \text{and} \rrbracket \\ \llbracket \text{or} \rrbracket \end{array} \right) v_2$	$v_2 \in \mathbb{B}$	Err
$v_1 \in \mathbb{Z}$	$v_1 \left(\begin{array}{c} = \\ \leq \end{array} \right) v_2$	Err	$v_1 \in \mathbb{B}$	$v_1 \left(\begin{array}{c} \wedge \\ \vee \end{array} \right) v_2$	Err
Err	Err	Err	Err	Err	Err

$\llbracket \text{not} \rrbracket v$	$v \in \mathbb{B}$	Err
	$\neg t$	Err

Schéma d'interprétation des expressions booléennes

$$\mathcal{B}[_]_ : E_B \rightarrow \mathcal{V}[\mathbb{Z}] \rightarrow \mathbb{B} \cup \{\text{Err}\}$$

$$\mathcal{B}[e]_\sigma = \begin{cases} t & \text{si } e = t \in \mathbb{B} \\ \mathcal{A}[e_1]_\sigma [\leq] \mathcal{A}[e_2]_\sigma & \text{si } e = e_1 \leq e_2 \\ \mathcal{A}[e_1]_\sigma [=] \mathcal{A}[e_2]_\sigma & \text{si } e = (e_1 = e_2) \\ \mathcal{B}[e_1]_\sigma [\text{ and }] \mathcal{B}[e_2]_\sigma & \text{si } e = e_1 \text{ and } e_2 \\ \mathcal{B}[e_1]_\sigma [\text{ or }] \mathcal{B}[e_2]_\sigma & \text{si } e = e_1 \text{ or } e_2 \\ [\text{not}] \mathcal{B}[e']_\sigma & \text{si } e = \text{not } e' \end{cases}$$

Sémantique opérationnelle à grands pas (1)

Système d'inférence définissant le jugement $\langle b, \sigma \rangle \rightsquigarrow v$: une expression booléenne $b \in E_B$ s'évalue en une valeur $v \in \mathbb{B}$ étant donnée une valuation σ .

$$(B_1) \frac{}{\langle t, \sigma \rangle \rightsquigarrow t} \quad t \in \mathbb{B} \quad (B_{14}) \frac{\langle b, \sigma \rangle \rightsquigarrow t}{\langle \text{not } b, \sigma \rangle \rightsquigarrow (\neg t)}$$

$$(B_2) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 = a_2, \sigma \rangle \rightsquigarrow (n_1 = n_2)} \quad (B_3) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 \leq a_2, \sigma \rangle \rightsquigarrow (n_1 \leq n_2)} \quad \begin{array}{l} n_1, n_2 \in \mathbb{N} \\ a_1, a_2 \in E_A \end{array}$$

$$(B_8) \frac{\langle b_1, \sigma \rangle \rightsquigarrow \text{true} \quad \langle b_2, \sigma \rangle \rightsquigarrow v}{\langle b_1 \text{ and } b_2, \sigma \rangle \rightsquigarrow v} \quad (B_9) \frac{\langle b_1, \sigma \rangle \rightsquigarrow \text{false}}{\langle b_1 \text{ and } b_2, \sigma \rangle \rightsquigarrow \text{false}}$$

$$(B_{11}) \frac{\langle b_1, \sigma \rangle \rightsquigarrow \text{false} \quad \langle b_2, \sigma \rangle \rightsquigarrow v}{\langle b_1 \text{ or } b_2, \sigma \rangle \rightsquigarrow v} \quad (B_{12}) \frac{\langle b_1, \sigma \rangle \rightsquigarrow \text{true}}{\langle b_1 \text{ or } b_2, \sigma \rangle \rightsquigarrow \text{true}}$$

Choix des règles

A-t-on $\mathcal{B}[[b]]_{\sigma} = v \Leftrightarrow \langle b, \sigma \rangle \rightsquigarrow v$?

Non

$\mathcal{B}[(2 = 2) \text{ or } (2/0 \leq x)]_{\sigma} = \text{Err}$ et
 $\langle (2 = 2) \text{ or } (2/0 \leq x), \sigma \rangle \rightsquigarrow \text{true}$

On a défini deux sémantiques différentes pour le même langage :
 La sémantique opérationnelle a donné une sémantique paresseuse
 aux opérateurs `and` et `or`.

On peut définir une sémantique dénotationnelle équivalente en
 changeant l'interprétation de `and` et `or` :

true $\mathcal{B}[[\text{and}]]$ Err = true

false $\mathcal{B}[[\text{or}]]$ Err = false

Sémantique opérationnelle : Propriétés

Expressions booléennes équivalentes

$$\forall b_1, b_2 \in E_B \quad b_1 \sim b_2 \Leftrightarrow (\forall v \in \mathbb{B} \forall \sigma \in \mathcal{V}[\mathbb{Z}] \langle b_1, \sigma \rangle \rightsquigarrow t \Leftrightarrow \langle b_2, \sigma \rangle \rightsquigarrow t)$$

Proposition \sim est une congruence.

PREUVE. Exercice ...

Proposition Unicité d'un résultat $\forall b \in E_B \forall \sigma \in \mathcal{V}[\mathbb{Z}], \forall v_1, v_2 \in \mathbb{B} (\langle b, \sigma \rangle \rightsquigarrow v_1 \text{ et } \langle b, \sigma \rangle \rightsquigarrow v_2) \Rightarrow v_1 = v_2$

PREUVE. Exercice ...

Outline

- Motivations
- Rappels
- Expressions arithmétiques
- Expressions booléennes
- IMP
 - Définition
 - Sémantique opérationnelle à grands pas
 - Sémantique opérationnelle à petits pas

Constructions impératives

Syntaxe (abstraite) d'un "petit" langage impératif :

$$c ::= \text{skip} \mid x := a \mid c_1; c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \mid \text{while } b \text{ do } c$$

... exprimée à l'aide d'un système d'inférence. E_C est l'ensemble des programmes, et est défini inductivement par :

$$x \in V, a \in E_A, b \in E_B$$

$$(\mathbb{C}_1) \frac{}{\text{skip}} \quad (\mathbb{C}_2) \frac{}{x := a}$$

$$(\mathbb{C}_3) \frac{c_1 \quad c_2}{c_1; c_2} \quad (\mathbb{C}_4) \frac{c_1 \quad c_2}{\text{if } b \text{ then } c_1 \text{ else } c_2} \quad (\mathbb{C}_5) \frac{c}{\text{while } b \text{ do } c}$$

Constructions impératives : Exemple (PGCD)

while not $(x = y)$ do if $x \leq y$ then $y := y - x$ else $x := x - y \in E_C?$

$$\begin{array}{c}
 (\mathbb{C}_2) \frac{}{y := y - x} \quad (\mathbb{C}_2) \frac{}{x := x - y} \\
 (\mathbb{C}_4) \frac{}{\text{if } x \leq y \text{ then } y := y - x \text{ else } x := x - y} \\
 (\mathbb{C}_5) \frac{}{\text{while not } (x = y) \text{ do if } x \leq y \text{ then } y := y - x \text{ else } x := x - y}
 \end{array}$$

si $x, y \in V$, not $(x = y)$, $x \leq y \in E_B$ et $y - x, x - y \in E_A$

Arbre de syntaxe abstraite du programme à partir duquel on va donner sa **sémantique**.

Etats (de la mémoire)

Construction de base d'un langage impératif : **affectation** d'une valeur à "une variable".

Pour donner une **sémantique** aux éléments de E_C , il faut commencer par donner une "signification" à chacun des symboles de variable.

On associe à $x \in V$ une cellule mémoire x_C , dans laquelle est encodée une valeur k .

Une mémoire est donc décrite par un ensemble dont les éléments sont des **adresses-mémoire**. On notera plus simplement x la cellule x_C .

Un **état** de la mémoire est la description du contenu de chacune de ses cellules. Un état peut être représenté par une valuation ... vue comme une fonction partielle qui associe à tout $x \in V$ la valeur k encodée dans la cellule désignée par x .

Changement d'état

Le rôle d'un programme c est de modifier l'état courant σ_1 de la mémoire en un état σ_2 :

$$\langle c, \sigma_1 \rangle \rightarrow \sigma_2$$

L'exécution de l'instruction c dans l'état σ_1 conduit à l'état σ_2 .
Description de l'exécution d'une instruction par le changement d'état qu'elle provoque.

Changement d'état

$$\sigma[x \leftarrow n](y) = \begin{cases} n & \text{si } y = x \\ \sigma(y) & \text{sinon} \end{cases}$$

Exemple $\langle x := 0, \sigma \rangle \rightarrow \sigma[x \leftarrow 0]$

Sémantique opérationnelle à grands pas

Caractériser un sous-ensemble de jugements de la forme $\langle c, \sigma_1 \rangle \rightarrow \sigma_2$ à l'aide d'un système d'inférence.

$$(C_1) \frac{}{\langle \text{skip}, \sigma \rangle \rightarrow \sigma} \quad (C_2) \frac{\langle a, \sigma \rangle \rightsquigarrow n}{\langle x := a, \sigma \rangle \rightarrow \sigma[x \leftarrow n]} \quad n \in \mathbb{Z}$$

$$(C_3) \frac{\langle c_1, \sigma \rangle \rightarrow \sigma_1 \quad \langle c_2, \sigma_1 \rangle \rightarrow \sigma_2}{\langle c_1; c_2, \sigma \rangle \rightarrow \sigma_2}$$

$$(C_4) \frac{\langle b, \sigma \rangle \rightsquigarrow \text{true} \quad \langle c_1, \sigma \rangle \rightarrow \sigma_1}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma_1} \quad (C_5) \frac{\langle b, \sigma \rangle \rightsquigarrow \text{false} \quad \langle c_2, \sigma \rangle \rightarrow \sigma_2}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma_2}$$

$$(C_6) \frac{\langle b, \sigma \rangle \rightsquigarrow \text{false}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma}$$

$$(C_7) \frac{\langle b, \sigma \rangle \rightsquigarrow \text{true} \quad \langle c, \sigma \rangle \rightarrow \sigma_1 \quad \langle \text{while } b \text{ do } c, \sigma_1 \rangle \rightarrow \sigma_2}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma_2}$$

Sémantique opérationnelle à grands pas : Exemple

$$(C_3) \frac{(C_2) \frac{(A_1) \overline{\langle 0, \sigma \rangle \rightsquigarrow 0}}{\langle x := 0, \sigma \rangle \rightarrow \sigma_1} (C_7) \frac{D_1 \quad D_2 \quad D_3}{\langle \text{while } x \leq 0 \text{ do } x := x + 1, \sigma_1 \rangle \rightarrow \sigma_2}}{\langle x := 0 ; \text{while } x \leq 0 \text{ do } x := x + 1, \sigma \rangle \rightarrow \sigma_2}$$

où $\sigma_1 = \sigma[x \leftarrow 0]$, $\sigma_2 = \sigma_1[x \leftarrow 1]$, D_1 et D_2 sont respectivement des arbres d'inférence de $\langle x \leq 0, \sigma_1 \rangle \rightsquigarrow \text{true}$ et $\langle x := x + 1, \sigma_1 \rangle \rightarrow \sigma_2$ et où D_3 est l'arbre :

$$(C_6) \frac{(B_3) \frac{(A_2) \overline{\langle x, \sigma_2 \rangle \rightsquigarrow 1} \quad (A_1) \overline{\langle 0, \sigma_2 \rangle \rightsquigarrow 0}}{\langle x \leq 0, \sigma_2 \rangle \rightsquigarrow \text{false}}}{\langle \text{while } x \leq 0 \text{ do } x := x + 1, \sigma_2 \rangle \rightarrow \sigma_2}$$

Effets de bords lors de l'évaluation des expressions

Les règles :

$$(C_4) \frac{\langle b, \sigma \rangle \rightsquigarrow \text{true} \quad \langle c_1, \sigma \rangle \rightarrow \sigma_1}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma_1} \quad (C_5) \frac{\langle b, \sigma \rangle \rightsquigarrow \text{false} \quad \langle c_2, \sigma \rangle \rightarrow \sigma_2}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma_2}$$

ne sont correctes que si l'évaluation d'une expression booléenne ne modifie pas l'état dans lequel s'effectue l'évaluation.

Est-ce le cas avec le langage C? Non

```
if (i++ > 0) {i=i-1} else {i=i+1};
```

... même remarque pour toutes les règles nécessitant l'évaluation d'une expression.

Boucle et Terminaison

Contrairement aux autres, l'instruction `while` permet d'écrire des programmes dont l'exécution ne termine pas.

Puisque les arbres d'inférence sont des objets **finis**, la **sémantique opérationnelle à grands pas** n'est pas en mesure de rendre compte de l'exécution des programmes qui ne terminent pas ...

contrairement à la **sémantique opérationnelle à petits pas**.

Exemple `while true do skip`

$$\frac{\begin{array}{c} (B_1) \frac{}{\langle \text{true}, \sigma \rangle \rightsquigarrow \text{true}} \\ (C_1) \frac{}{\langle \text{skip}, \sigma \rangle \rightarrow \sigma} \\ (C_7) \frac{}{\langle \text{while true do skip}, \sigma \rangle \rightarrow \sigma'} \end{array}}{\langle \text{while true do skip}, \sigma \rangle \rightarrow \sigma'} \quad \begin{array}{c} \vdots \\ \vdots \\ \vdots \end{array}$$

Il n'existe pas d'état σ' tel que $\langle \text{while true do skip}, \sigma \rangle \rightarrow \sigma'$.

Programmes équivalents

Programmes dont l'exécution est décrite par la même transformation.

$$\forall c_1, c_2 \in EC \quad c_1 \equiv c_2 \Leftrightarrow (\forall \sigma, \sigma' \in \mathcal{V}[\mathbb{Z}] \quad \langle c_1, \sigma \rangle \rightarrow \sigma' \Leftrightarrow \langle c_2, \sigma \rangle \rightarrow \sigma')$$

Exemple : $c; (\text{if } b \text{ then } c' \text{ else } c'') \stackrel{?}{\equiv} \text{if } b \text{ then } (c; c') \text{ else } (c; c'')$

Non.

c	$x := 0$
c'	$x := x + 1$
c''	$x := x + 5$
b	$1 \leq x$
σ	$\sigma(x) = 2$

$$\langle c; (\text{if } b \text{ then } c' \text{ else } c''), \sigma \rangle \rightarrow \sigma' \quad \sigma'(x) = 5$$

$$\langle \text{if } b \text{ then } (c; c') \text{ else } (c; c''), \sigma \rangle \rightarrow \sigma'' \quad \sigma''(x) = 1$$

Programmes équivalents : Exemple (1)

$c_1 : (\text{if } b \text{ then } c \text{ else } c'); c'' \quad c_2 : \text{if } b \text{ then } (c; c'') \text{ else } (c'; c'') \quad c_1 \stackrel{?}{\equiv} c_2$

Construction un arbre d'inférence de $\langle c_1, \sigma \rangle \rightarrow \sigma'$ à partir d'un arbre de d'inférence de $\langle c_2, \sigma \rangle \rightarrow \sigma'$ et *vice versa*.

Si on dispose d'un arbre d'inférence de

$\langle (\text{if } b \text{ then } c \text{ else } c'); c'', \sigma \rangle \rightarrow \sigma'$, il a forcément été obtenu à partir de la règle C_3 :

$$(C_3) \frac{\begin{array}{c} \vdots \\ (C_i) \frac{}{\langle \text{if } b \text{ then } c \text{ else } c', \sigma \rangle \rightarrow \sigma_1} \\ \vdots \end{array} \quad \begin{array}{c} \vdots \\ (C_j) \frac{}{\langle c'', \sigma_1 \rangle \rightarrow \sigma'} \\ \vdots \end{array}}{\langle (\text{if } b \text{ then } c \text{ else } c'); c'', \sigma \rangle \rightarrow \sigma'}$$

On distingue alors deux cas (résultat de l'évaluation de b).

Programmes équivalents : Exemple (2)

(1) Si b s'évalue à $true$, alors, à partir de l'arbre :

$$(C_3) \frac{(C_4) \frac{(B_i) \frac{\vdots}{\langle b, \sigma \rangle \rightsquigarrow true} (C_j) \frac{\vdots}{\langle c, \sigma \rangle \rightarrow \sigma_1} (C_k) \frac{\vdots}{\langle c'', \sigma_1 \rangle \rightarrow \sigma'}}{\langle \text{if } b \text{ then } c \text{ else } c', \sigma \rangle \rightarrow \sigma_1}}{\langle (\text{if } b \text{ then } c \text{ else } c'); c'', \sigma \rangle \rightarrow \sigma'}$$

on peut obtenir l'arbre :

$$(C_4) \frac{(B_i) \frac{\vdots}{\langle b, \sigma \rangle \rightsquigarrow true} (C_3) \frac{(C_j) \frac{\vdots}{\langle c, \sigma \rangle \rightarrow \sigma_1} (C_k) \frac{\vdots}{\langle c'', \sigma_1 \rangle \rightarrow \sigma'}}{\langle c; c'', \sigma \rangle \rightarrow \sigma'}}{\langle \text{if } b \text{ then } (c; c'') \text{ else } (c'; c''), \sigma \rangle \rightarrow \sigma'}$$

et *vice versa*.

Programmes équivalents : Exemple (3)

(2) Si b s'évalue à **false**, alors, à partir de l'arbre :

$$(C_3) \frac{(B_i) \frac{\vdots}{\langle b, \sigma \rangle \rightsquigarrow \text{false}} \quad (C_j) \frac{\vdots}{\langle c', \sigma \rangle \rightarrow \sigma_1} \quad (C_k) \frac{\vdots}{\langle c'', \sigma_1 \rangle \rightarrow \sigma'}}{\langle (\text{if } b \text{ then } c \text{ else } c'); c'', \sigma \rangle \rightarrow \sigma'}$$

on peut obtenir l'arbre :

$$(C_5) \frac{(B_i) \frac{\vdots}{\langle b, \sigma \rangle \rightsquigarrow \text{false}} \quad (C_3) \frac{(C_j) \frac{\vdots}{\langle c', \sigma \rangle \rightarrow \sigma_1} \quad (C_k) \frac{\vdots}{\langle c'', \sigma_1 \rangle \rightarrow \sigma'}}{\langle c'; c'', \sigma \rangle \rightarrow \sigma'}}{\langle \text{if } b \text{ then } (c; c'') \text{ else } (c'; c''), \sigma \rangle \rightarrow \sigma'}$$

et *vice versa*.

Terminaison de programmes (1)

Puisque tout arbre d'inférence est fini, montrer qu'un programme c termine à partir d'un état σ revient à montrer qu'il existe un état σ' tel que $\langle c, \sigma \rangle \rightarrow \sigma'$.

Exemple Terminaison du programme Euclid :

```
while not (x = y) do if x ≤ y then y := y - x else x := x - y
```

Ce programme termine à partir de tous les états σ tels que $\sigma(x) \geq 1$ et $\sigma(y) \geq 1$.

$$\forall \sigma \in \mathcal{V}[\mathbb{Z}] \quad (\sigma(x) \geq 1 \wedge \sigma(y) \geq 1) \Rightarrow \exists \sigma' \in \mathcal{V}[\mathbb{Z}] \quad \langle \text{Euclid}, \sigma \rangle \rightarrow \sigma'$$

Terminaison de programmes (2)

Utilisation de la **récurrence bien fondée** pour prouver la propriété $P(\sigma) : (\exists \sigma' \in \mathcal{V}[\mathbb{Z}] \langle \text{Euclid}, \sigma \rangle \rightarrow \sigma')$ pour tout $\sigma \in S = \{\sigma \in \mathcal{V}[\mathbb{Z}] \mid \sigma(x) \geq 1 \wedge \sigma(y) \geq 1\}$.

Quelle **relation d'ordre bien fondée** utiliser sur S ?

Terminaison de programmes (3)

A chaque tour de boucle, au moins une valeur des deux variables x et y diminue strictement.

Définition d'une **relation d'ordre** \preceq sur S :

$$\sigma_1 \preceq \sigma_2 \Leftrightarrow \sigma_1(x) \leq \sigma_2(x) \wedge \sigma_1(y) \leq \sigma_2(y) \wedge \forall z \in V \setminus \{x, y\} \sigma_1(z) = \sigma_2(z)$$

Ordre strict associé à \preceq : $\sigma_1 \prec \sigma_2 \Leftrightarrow \sigma_1 \preceq \sigma_2 \wedge \sigma_1 \neq \sigma_2$.

$$\sigma_1 \prec \sigma_2 \Leftrightarrow \left(\begin{array}{l} (\sigma_1(x) \neq \sigma_2(x) \vee \sigma_1(y) \neq \sigma_2(y)) \\ \wedge (\sigma_1(x) \leq \sigma_2(x) \wedge \sigma_1(y) \leq \sigma_2(y)) \\ \wedge \forall z \in V \setminus \{x, y\} \sigma_1(z) = \sigma_2(z) \end{array} \right)$$

Remarque

$$\left(\begin{array}{l} (\sigma_1(x) \neq \sigma_2(x) \vee \sigma_1(y) \neq \sigma_2(y)) \\ \wedge (\sigma_1(x) \leq \sigma_2(x) \wedge \sigma_1(y) \leq \sigma_2(y)) \end{array} \right) \Rightarrow (\sigma_1(x) < \sigma_2(x) \vee \sigma_1(y) < \sigma_2(y))$$

Terminaison de programmes (4)

\preceq est un **ordre bien fondé** sur S .

S'il existait une suite infinie strictement décroissante

$\sigma_1 \succ \sigma_2 \succ \dots$, alors à partir d'un certain rang k , on aurait :

$$\forall j \geq k \quad \sigma_k(x) = \sigma_j(x) \wedge \sigma_k(y) = \sigma_j(y)$$

puisque on se place ici dans le sous-ensemble S de $\mathcal{V}[\mathbb{Z}]$ dont les états associent uniquement des valeurs strictement positives aux deux variables x et y et que toute suite infinie décroissante d'entiers strictement positifs est stationnaire à partir d'un certain rang. Or, par définition, si $\sigma_k \succ \sigma_{k+1}$ alors $\sigma_k(x) \neq \sigma_{k+1}(x)$ ou $\sigma_k(y) \neq \sigma_{k+1}(y)$ ce qui est contradictoire.

Terminaison de programmes (5)

Soit $\sigma \in \mathcal{S}$, supposons que $\forall \sigma' \prec \sigma, P(\sigma')$ (**hypothèse de récurrence**) et montrons $P(\sigma)$. Notons $\sigma(x) = m$ et $\sigma(y) = n$.

Deux cas se présentent.

(1) Si $m = n$, alors on peut conclure :

$$\begin{array}{c}
 (A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow m} \quad (A_2) \frac{}{\langle y, \sigma \rangle \rightsquigarrow n} \\
 (B_2) \frac{}{\langle x = y, \sigma \rangle \rightsquigarrow \text{true}} \\
 (B_{14}) \frac{}{\langle \text{not } (x = y), \sigma \rangle \rightsquigarrow \text{false}} \\
 (C_6) \frac{}{\langle \text{Euclid}, \sigma \rangle \rightarrow \sigma}
 \end{array}$$

(2) Si $m \neq n$, alors on a l'arbre :

$$\begin{array}{c}
 (A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow m} \quad (A_2) \frac{}{\langle y, \sigma \rangle \rightsquigarrow n} \\
 (B_2) \frac{}{\langle x = y, \sigma \rangle \rightsquigarrow \text{false}} \\
 (B_{14}) \frac{}{\langle \text{not } (x = y), \sigma \rangle \rightsquigarrow \text{true}} \quad (D)
 \end{array}$$

Terminaison de programmes (6)

(2.1) Si $m \leq n$ alors on peut construire l'arbre D_1 :

$$\begin{array}{c}
 (A_2) \frac{}{\langle y, \sigma \rangle \rightsquigarrow n} \quad (A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow m} \\
 (A_7) \frac{}{\langle y - x, \sigma \rangle \rightsquigarrow n - m} \\
 D_2 \quad (C_2) \frac{}{\langle y := y - x, \sigma \rangle \rightarrow \sigma[y \leftarrow n - m]} \\
 (C_4) \frac{}{\langle \text{if } x \leq y \text{ then } y := y - x \text{ else } x := x - y, \sigma \rangle \rightarrow \sigma[y \leftarrow n - m]}
 \end{array}$$

où D_2 est l'arbre :

$$(B_3) \frac{(A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow m} \quad (A_2) \frac{}{\langle y, \sigma \rangle \rightsquigarrow n}}{\langle x \leq y, \sigma \rangle \rightsquigarrow \text{true}} \quad (D_2)$$

Or, par définition, $\sigma[y \leftarrow n - m] \prec \sigma$, et par hypothèse de récurrence, on a l'arbre D_3 :

$$(C_i) \frac{\vdots}{\langle \text{Euclid}, \sigma[y \leftarrow n - m] \rangle \rightarrow \sigma'} \quad (D_3)$$

Terminaison de programmes (7)

(2.2) Si $n \leq m$ alors on peut construire l'arbre D_1 :

$$\begin{array}{c}
 (A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow m} \quad (A_2) \frac{}{\langle y, \sigma \rangle \rightsquigarrow n} \\
 (A_7) \frac{}{\langle x - y, \sigma \rangle \rightsquigarrow m - n} \\
 D_2 \quad (C_2) \frac{}{\langle x := x - y, \sigma \rangle \rightarrow \sigma[x \leftarrow m - n]} \\
 (C_5) \frac{}{\langle \text{if } x \leq y \text{ then } y := y - x \text{ else } x := x - y, \sigma \rangle \rightarrow \sigma[x \leftarrow m - n]}
 \end{array}$$

où D_2 est l'arbre :

$$(B_3) \frac{(A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow m} \quad (A_2) \frac{}{\langle y, \sigma \rangle \rightsquigarrow n}}{\langle x \leq y, \sigma \rangle \rightsquigarrow \text{false}} \quad (D_2)$$

Or, par définition, $\sigma[y \leftarrow m - n] \prec \sigma$, et par hypothèse de récurrence, on a l'arbre D_3 :

$$(C_i) \frac{\vdots}{\langle \text{Euclid}, \sigma[x \leftarrow m - n] \rangle \rightarrow \sigma'} \quad (D_3)$$

Terminaison de programmes (8)

Dans ces deux sous-cas on peut conclure en considérant l'arbre d'inférence :

$$(C_7) \frac{D \quad D_1 \quad D_3}{\langle \text{Euclid}, \sigma \rangle \rightarrow \sigma'}$$

Déterminisme (1)

Proposition $\forall \sigma, \sigma_1, \sigma_2 \in \mathcal{V}[\mathbb{Z}] \quad \forall c \in E_C \quad (\langle c, \sigma \rangle \rightarrow \sigma_1 \text{ et } \langle c, \sigma \rangle \rightarrow \sigma_2) \Rightarrow \sigma_1 = \sigma_2$

Peut-on prouver cette proposition par **induction sur c** ?

Cas de la règle $(C_5) \frac{c}{\text{while } b \text{ do } c}$. Si b s'évalue à **true** :

$$(C_7) \frac{\begin{array}{c} \vdots \\ (B_i) \frac{}{\langle b, \sigma \rangle \rightsquigarrow \text{true}} \end{array} \quad \begin{array}{c} \vdots \\ (C_j) \frac{}{\langle c, \sigma \rangle \rightarrow \sigma_1} \end{array} \quad \begin{array}{c} \vdots \\ (C_k) \frac{}{\langle \text{while } b \text{ do } c, \sigma_1 \rangle \rightarrow \sigma_2} \end{array}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma_2}$$

$$(C_7) \frac{\begin{array}{c} \vdots \\ (B_i) \frac{}{\langle b, \sigma \rangle \rightsquigarrow \text{true}} \end{array} \quad \begin{array}{c} \vdots \\ (C_{j'}) \frac{}{\langle c, \sigma \rangle \rightarrow \sigma'_1} \end{array} \quad \begin{array}{c} \vdots \\ (C_{k'}) \frac{}{\langle \text{while } b \text{ do } c, \sigma'_1 \rangle \rightarrow \sigma'_2} \end{array}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma'_2}$$

Par hypothèse d'induction, on a seulement $\sigma_1 = \sigma'_1$. Pour prouver $\sigma_2 = \sigma'_2$, il faudrait aussi une hypothèse d'induction sur **while b do c** !

Déterminisme (2)

Schéma d'induction associé au système définissant $\langle c, \sigma \rangle \rightarrow \sigma'$
(induction sur les règles)

$$\begin{aligned}
 & \forall \sigma \in \mathcal{V}[\mathbb{Z}] \quad P(\langle \text{skip}, \sigma \rangle \rightarrow \sigma) \\
 \text{et} \quad & \forall a \in E_A \quad \forall \sigma \in \mathcal{V}[\mathbb{Z}] \quad \forall n \in \mathbb{Z} \quad \forall x \in V \quad \langle a, \sigma \rangle \rightsquigarrow n \Rightarrow P(\langle x := a, \sigma \rangle \rightarrow \sigma[x \leftarrow n]) \\
 \text{et} \quad & \forall c_1, c_2 \in E_C \quad \forall \sigma, \sigma', \sigma'' \in \mathcal{V}[\mathbb{Z}] \\
 & (P(\langle c_1, \sigma \rangle \rightarrow \sigma') \text{ et } P(\langle c_2, \sigma' \rangle \rightarrow \sigma'')) \Rightarrow P(\langle c_1; c_2, \sigma \rangle \rightarrow \sigma'') \\
 \text{et} \quad & \forall c_1, c_2 \in E_C \quad \forall \sigma, \sigma' \in \mathcal{V}[\mathbb{Z}] \quad \forall b \in E_B \\
 & (\langle b, \sigma \rangle \rightsquigarrow \text{true} \text{ et } P(\langle c_1, \sigma \rangle \rightarrow \sigma')) \Rightarrow P(\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma') \\
 \text{et} \quad & \forall c_1, c_2 \in E_C \quad \forall \sigma, \sigma' \in \mathcal{V}[\mathbb{Z}] \quad \forall b \in E_B \\
 & (\langle b, \sigma \rangle \rightsquigarrow \text{false} \text{ et } P(\langle c_2, \sigma \rangle \rightarrow \sigma')) \Rightarrow P(\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma') \\
 \text{et} \quad & \forall c \in E_C \quad \forall \sigma \in \mathcal{V}[\mathbb{Z}] \quad \forall b \in E_B \quad \langle b, \sigma \rangle \rightsquigarrow \text{false} \Rightarrow P(\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma) \\
 \text{et} \quad & \forall c \in E_C \quad \forall \sigma, \sigma', \sigma'' \in \mathcal{V}[\mathbb{Z}] \quad \forall b \in E_B \\
 & (\langle b, \sigma \rangle \rightsquigarrow \text{true} \text{ et } \boxed{P(\langle c, \sigma \rangle \rightarrow \sigma')} \text{ et } \boxed{P(\langle \text{while } b \text{ do } c, \sigma' \rangle \rightarrow \sigma'')}) \\
 & \Rightarrow P(\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma'') \\
 \Rightarrow \quad & \forall c \in E_C \quad \forall \sigma_1, \sigma_2 \in \mathcal{V}[\mathbb{Z}] \quad P(\langle c, \sigma_1 \rangle \rightarrow \sigma_2)
 \end{aligned}$$

Déterminisme (3)

PREUVE :

La propriété P à prouver peut s'exprimer par : $P(\langle c, \sigma \rangle \rightarrow \sigma_1)$ ssi
 $\forall \sigma_2 \in \mathcal{V}[\mathbb{Z}] \langle c, \sigma \rangle \rightarrow \sigma_2 \Rightarrow \sigma_1 = \sigma_2.$

A faire

Sémantique opérationnelle à grands pas

- ▶ manipule des jugements qui permettent uniquement de spécifier l'état final de la mémoire après l'exécution d'un programme
- ▶ ne donne pas d'informations précises sur les étapes qui ont conduit à cet état.
- ▶ ne permet pas de modéliser l'exécution des programmes qui ne terminent pas.
 - si un programme n'a pas de sémantique, est-ce parce qu'il boucle ou parce qu'il est bloqué ?

Définition d'une sémantique opérationnelle à petits pas

- ▶ **configuration** : paire $\langle c, \sigma \rangle$ ($c \in E_C$, $\sigma \in \mathcal{V}[\mathbb{Z}]$)
- ▶ **Relation de transition** \hookrightarrow entre configurations

De manière intuitive, $\langle c, \sigma \rangle \hookrightarrow \langle c', \sigma' \rangle$ exprime qu'exécuter une étape de c dans un état σ conduit dans un état σ' où il restera à exécuter c' .

Relation de transition

$$(S_1) \frac{\langle a, \sigma \rangle \rightsquigarrow n}{\langle x := a, \sigma \rangle \hookrightarrow \langle \text{skip}, \sigma[x \leftarrow n] \rangle}$$

$$(S_2) \frac{\langle c_1, \sigma \rangle \hookrightarrow \langle c'_1, \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \hookrightarrow \langle c'_1; c_2, \sigma' \rangle} \quad (S_3) \frac{}{\langle \text{skip}; c, \sigma \rangle \hookrightarrow \langle c, \sigma \rangle}$$

$$(S_4) \frac{\langle b, \sigma \rangle \rightsquigarrow \text{true}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \hookrightarrow \langle c_1, \sigma \rangle}$$

$$(S_5) \frac{\langle b, \sigma \rangle \rightsquigarrow \text{false}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \hookrightarrow \langle c_2, \sigma \rangle}$$

$$(S_6) \frac{\langle b, \sigma \rangle \rightsquigarrow \text{false}}{\langle \text{while } b \text{ do } c, \sigma \rangle \hookrightarrow \langle \text{skip}, \sigma \rangle}$$

$$(S_7) \frac{\langle b, \sigma \rangle \rightsquigarrow \text{true}}{\langle \text{while } b \text{ do } c, \sigma \rangle \hookrightarrow \langle c; \text{while } b \text{ do } c, \sigma \rangle}$$

Séquence de calcul

Séquence de calcul : suite, éventuellement infinie, de la forme :

$$\langle c_0, \sigma_0 \rangle \hookrightarrow \langle c_1, \sigma_1 \rangle \hookrightarrow \langle c_2, \sigma_2 \rangle \hookrightarrow \dots$$

telle que $\forall i \geq 0$, $\langle c_i, \sigma_i \rangle \hookrightarrow \langle c_{i+1}, \sigma_{i+1} \rangle$ admette un arbre de d'inférence à partir des règles définissant \hookrightarrow .

Configuration terminale : $\langle \text{skip}, \sigma \rangle$

$\langle c, \sigma \rangle \xrightarrow{\star} \langle e', \sigma' \rangle$ ssi il existe une séquence de calcul de longueur finie $\langle c_0, \sigma_0 \rangle \hookrightarrow \langle c_1, \sigma_1 \rangle \hookrightarrow \langle c_2, \sigma_2 \rangle \hookrightarrow \dots \hookrightarrow \langle c_k, \sigma_k \rangle$ avec $c = c_0$, $\sigma = \sigma_0$, $e' = c_k$ et $\sigma' = \sigma_k$.

$\langle c, \sigma \rangle \xrightarrow{k} \langle e', \sigma' \rangle$ ssi il existe une séquence de calcul de longueur k $\langle c_0, \sigma_0 \rangle \hookrightarrow \langle c_1, \sigma_1 \rangle \hookrightarrow \langle c_2, \sigma_2 \rangle \hookrightarrow \dots \hookrightarrow \langle c_k, \sigma_k \rangle$ avec $c = c_0$, $\sigma = \sigma_0$, $e' = c_k$ et $\sigma' = \sigma_k$.

De plus, si $c_k = \text{skip}$ (configuration terminale), alors $\langle c, \sigma \rangle \xrightarrow{\star} \sigma'$.

Sémantique opérationnelle à petits pas : Exemple

$\langle x := 0 ; \text{while } x \leq 0 \text{ do } x := x + 1, \sigma \rangle$
 $\hookrightarrow \langle \text{skip} ; \text{while } x \leq 0 \text{ do } x := x + 1, \sigma[x \leftarrow 0] \rangle$
 $\hookrightarrow \langle \text{while } x \leq 0 \text{ do } x := x + 1, \sigma[x \leftarrow 0] \rangle$
 $\hookrightarrow \langle x := x + 1 ; \text{while } x \leq 0 \text{ do } x := x + 1, \sigma[x \leftarrow 0] \rangle$
 $\hookrightarrow \langle \text{skip} ; \text{while } x \leq 0 \text{ do } x := x + 1, \sigma[x \leftarrow 1] \rangle$
 $\hookrightarrow \langle \text{while } x \leq 0 \text{ do } x := x + 1, \sigma[x \leftarrow 1] \rangle$
 $\hookrightarrow \langle \text{skip}, \sigma_2 \rangle$

Composition

Lemme Si $\langle c_0, \sigma_0 \rangle \hookrightarrow \dots \hookrightarrow \langle c_n, \sigma_n \rangle$, alors $\forall c \in E_C$, il existe une séquence $\langle c_0; c, \sigma_0 \rangle \hookrightarrow \dots \hookrightarrow \langle c_n; c, \sigma_n \rangle$.

PREUVE : Induction sur la (longueur de la) séquence.

- Pour $\langle c_0, \sigma_0 \rangle \hookrightarrow \langle c_1, \sigma_1 \rangle$, il suffit d'appliquer la règle S_2 pour obtenir la séquence $\langle c_0; c, \sigma_0 \rangle \hookrightarrow \langle c_1; c, \sigma_1 \rangle$.
- Pour une séquence de longueur $n + 1$:

$$\langle c_0, \sigma_0 \rangle \hookrightarrow \dots \hookrightarrow \langle c_n, \sigma_n \rangle \hookrightarrow \langle c_{n+1}, \sigma_{n+1} \rangle$$

Par hypothèse d'induction, il existe une séquence :

$$\langle c_0; c, \sigma_0 \rangle \hookrightarrow \dots \hookrightarrow \langle c_n; c, \sigma_n \rangle$$

et la règle S_2 permet d'obtenir la transition

$\langle c_n; c, \sigma_n \rangle \hookrightarrow \langle c_{n+1}; c, \sigma_{n+1} \rangle$ à partir de la transition

$\langle c_n, \sigma_n \rangle \hookrightarrow \langle c_{n+1}, \sigma_{n+1} \rangle$ ce qui permet de construire la séquence :

$$\langle c_0; c, \sigma_0 \rangle \hookrightarrow \dots \hookrightarrow \langle c_n; c, \sigma_n \rangle \hookrightarrow \langle c_{n+1}; c, \sigma_{n+1} \rangle$$

Equivalence sémantique (1)

Proposition

$$\forall c \in E_C \quad \forall \sigma_1, \sigma_2 \in \Sigma \quad \langle c, \sigma_1 \rangle \rightarrow \sigma_2 \Rightarrow \langle c, \sigma_1 \rangle \xrightarrow{*} \sigma_2$$

PREUVE : Induction sur $\langle c, \sigma_1 \rangle \rightarrow \sigma_2$.

$$\blacktriangleright (C_1) \frac{}{\langle \text{skip}, \sigma \rangle \rightarrow \sigma}$$

Puisque $\langle \text{skip}, \sigma \rangle$ est une configuration terminale, on a bien

$$\langle \text{skip}, \sigma \rangle \xrightarrow{*} \sigma$$

Equivalence sémantique (2)

$$\blacktriangleright (C_2) \frac{(A_i) \frac{\vdots}{\langle a, \sigma \rangle \rightsquigarrow n}}{\langle x := a, \sigma \rangle \rightarrow \sigma[x \leftarrow n]}$$

On peut construire l'arbre :

$$(S_1) \frac{(A_i) \frac{\vdots}{\langle a, \sigma \rangle \rightsquigarrow n}}{\langle x := a, \sigma \rangle \hookrightarrow \langle \text{skip}, \sigma[x \leftarrow n] \rangle}$$

On obtient la séquence $\langle x := a, \sigma \rangle \hookrightarrow \langle \text{skip}, \sigma[x \leftarrow n] \rangle$ et puisque $\langle \text{skip}, \sigma[x \leftarrow n] \rangle$ est terminale, on a bien $\langle x := a, \sigma \rangle \xrightarrow{\star} \sigma[x \leftarrow n]$.

Equivalence sémantique (3)

$$\blacktriangleright \frac{(C_i) \frac{\vdots}{\langle c_1, \sigma \rangle \rightarrow \sigma_1} \quad (C_j) \frac{\vdots}{\langle c_2, \sigma_1 \rangle \rightarrow \sigma_2}}{(C_3) \frac{\quad}{\langle c_1; c_2, \sigma \rangle \rightarrow \sigma_2}}$$

Par hyp. d'induction, on a $\langle c_1, \sigma \rangle \xrightarrow{\star} \sigma_1$ et $\langle c_2, \sigma_1 \rangle \xrightarrow{\star} \sigma_2$ et donc :

$$\langle c_1, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma_1 \rangle \quad (5)$$

$$\langle c_2, \sigma_1 \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma_2 \rangle \quad (6)$$

D'après le lemme précédent, à partir de la séquence (5), on a :

$$\langle c_1; c_2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}; c_2, \sigma_1 \rangle \quad (7)$$

La règle S_3 permet de construire la transition :

$$\langle \text{skip}; c_2, \sigma_1 \rangle \hookrightarrow \langle c_2, \sigma_1 \rangle \quad (8)$$

et à partir de (7), (8) et (6), on peut obtenir la séquence :

$$\langle c_1; c_2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma_2 \rangle$$

Equivalence sémantique (4)

$$\blacktriangleright (C_4) \frac{(B_i) \frac{\vdots}{\langle b, \sigma \rangle \rightsquigarrow \text{true}} \quad (C_j) \frac{\vdots}{\langle c_1, \sigma \rangle \rightarrow \sigma_1}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma_1}$$

Par hypothèse d'induction, $\langle c_1, \sigma \rangle \xrightarrow{\star} \sigma_1$ et on a donc $\langle c_1, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma_1 \rangle$. A partir de $\langle b, \sigma \rangle \rightsquigarrow \text{true}$ on a :

$$(S_4) \frac{\langle b, \sigma \rangle \rightsquigarrow \text{true}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \hookrightarrow \langle c_1, \sigma \rangle}$$

On a alors $\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \hookrightarrow \langle c_1, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma_1 \rangle$
ce qui permet d'établir $\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \xrightarrow{\star} \sigma_1$.

$$\blacktriangleright (C_5) \frac{(B_i) \frac{\vdots}{\langle b, \sigma \rangle \rightsquigarrow \text{false}} \quad (C_j) \frac{\vdots}{\langle c_2, \sigma \rangle \rightarrow \sigma_1}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma_1}$$

Raisonnement similaire au cas précédent.

Equivalence sémantique (5)

$$\blacktriangleright (C_6) \frac{(B_i) \frac{\vdots}{\langle b, \sigma \rangle \rightsquigarrow \text{false}}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma}$$

A partir de l'arbre de $\langle b, \sigma \rangle \rightsquigarrow \text{false}$ présent dans l'arbre en hypothèse, on peut construire l'arbre :

$$(S_6) \frac{(B_i) \frac{\vdots}{\langle b, \sigma \rangle \rightsquigarrow \text{false}}}{\langle \text{while } b \text{ do } c, \sigma \rangle \hookrightarrow \langle \text{skip}, \sigma \rangle}$$

ce qui permet d'établir $\langle \text{while } b \text{ do } c, \sigma \rangle \xrightarrow{*} \sigma$.

Equivalence sémantique (6)

$$(C_7) \frac{(B_i) \frac{\vdots}{\langle b, \sigma \rangle \rightsquigarrow \text{true}} \quad (C_j) \frac{\vdots}{\langle c, \sigma \rangle \rightarrow \sigma_1} \quad (C_k) \frac{\vdots}{\langle \text{while } b \text{ do } c, \sigma_1 \rangle \rightarrow \sigma_2}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma_2}$$

Par hypothèse d'induction on a :

$$\langle c, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma_1 \rangle \quad (9)$$

$$\langle \text{while } b \text{ do } c, \sigma_1 \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma_2 \rangle \quad (10)$$

A partir de (9), on obtient :

$$\langle c; \text{while } b \text{ do } c, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}; \text{while } b \text{ do } c, \sigma_1 \rangle \quad (11)$$

$$\text{règle } S_3 \quad \langle \text{skip}; \text{while } b \text{ do } c, \sigma_1 \rangle \hookrightarrow \langle \text{while } b \text{ do } c, \sigma_1 \rangle \quad (12)$$

$$\text{règle } S_7 \quad \langle \text{while } b \text{ do } c, \sigma \rangle \hookrightarrow \langle c; \text{while } b \text{ do } c, \sigma \rangle \quad (13)$$

A partir de (13), (11), (12) et (10), on obtient :

$$\langle \text{while } b \text{ do } c, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma_2 \rangle \text{ et donc}$$

$$\langle \text{while } b \text{ do } c, \sigma \rangle \xrightarrow{\star} \sigma_2.$$

Equivalence sémantique (7)

Lemme Si $\langle c_1; c_2, \sigma \rangle \xrightarrow{k} \sigma''$, alors $\exists \sigma' \in \mathcal{V}[\mathbb{Z}]$, $\exists k_1, k_2 \in \mathbb{N}$, tels que $\langle c_1, \sigma \rangle \xrightarrow{k_1} \sigma'$, $\langle c_2, \sigma' \rangle \xrightarrow{k_2} \sigma''$ avec $k = k_1 + k_2$.

PREUVE : Par récurrence bien fondée sur k .

Si $k = 0$ alors la propriété est triviale.

Supposons cette propriété vraie pour tout $k \leq k_0$ et montrons la pour $k_0 + 1$ (récurrence bien fondée).

On distingue deux cas :

(1). $\langle c_1; c_2, \sigma \rangle \hookrightarrow \langle c'_1; c_2, \sigma_1 \rangle \xrightarrow{k_0} \sigma''$ avec $\langle c_1, \sigma \rangle \hookrightarrow \langle c'_1, \sigma_1 \rangle$. Par hypothèse de récurrence, il existe $k'_1, k'_2 \in \mathbb{N}$ et une valuation σ' tels que $\langle c'_1, \sigma_1 \rangle \xrightarrow{k'_1} \sigma'_1$, $\langle c_2, \sigma'_1 \rangle \xrightarrow{k'_2} \sigma''$ avec $k_0 = k'_1 + k'_2$. et on peut conclure avec $k_1 = k'_1 + 1$ et $k_2 = k'_2$.

(2). $\langle \text{skip}; c_2, \sigma \rangle \hookrightarrow \langle c_2, \sigma_1 \rangle \xrightarrow{k_0} \sigma''$ et on peut conclure avec $k_1 = 0$ et $k_2 = k_0$.

Équivalence sémantique (8)

Proposition $\forall c \in E_C \quad \forall \sigma, \sigma' \in \Sigma \quad \langle c, \sigma \rangle \xrightarrow{*} \sigma' \Rightarrow \langle c, \sigma \rangle \rightarrow \sigma'$

PREUVE : On a la séquence :

$$\langle c, \sigma \rangle \hookrightarrow \langle c_1, \sigma_1 \rangle \hookrightarrow \dots \hookrightarrow \langle c_k, \sigma_k \rangle = \langle \text{skip}, \sigma' \rangle$$

Induction sur la séquence de calcul.

- $\langle x := a, \sigma \rangle \hookrightarrow \langle \text{skip}, \sigma[x \leftarrow n] \rangle$ avec $\langle a, \sigma \rangle \rightsquigarrow n$. On conclut en construisant l'arbre :

$$(C_2) \frac{(A_i) \frac{\vdots}{\langle a, \sigma \rangle \rightsquigarrow n}}{\langle x := a, \sigma \rangle \rightarrow \sigma[x \leftarrow n]}$$

Equivalence sémantique (9)

- $\langle c_1; c_2, \sigma \rangle \hookrightarrow \langle c'_1; c_2, \sigma_1 \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma' \rangle$ avec $\langle c_1, \sigma \rangle \hookrightarrow \langle c'_1, \sigma_1 \rangle$. D'après le lemme précédent, il existe une valuation σ_0 et deux entiers k_1 et k_2 (avec $k_1 + k_2 = k$) tels que $\langle c_1, \sigma \rangle \xrightarrow{k_1} \sigma_0$ et $\langle c_2, \sigma_0 \rangle \xrightarrow{k_2} \sigma'$. Par hypothèse d'induction on a $\langle c_1, \sigma \rangle \rightarrow \sigma_0$ et $\langle c_2, \sigma_0 \rangle \rightarrow \sigma'$ et on conclut en construisant l'arbre :

$$(C_3) \frac{\begin{array}{c} \vdots \\ (C_i) \frac{}{\langle c_1, \sigma \rangle \rightarrow \sigma_0} \end{array} \quad \begin{array}{c} \vdots \\ (C_j) \frac{}{\langle c_2, \sigma_0 \rangle \rightarrow \sigma'} \end{array}}{\langle c_1; c_2, \sigma \rangle \rightarrow \sigma'}$$

- $\langle \text{skip}; c_2, \sigma \rangle \hookrightarrow \langle c_2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma' \rangle$ Par hypothèse d'induction on a $\langle c_2, \sigma \rangle \rightarrow \sigma'$ et on conclut en construisant l'arbre :

$$(C_3) \frac{\begin{array}{c} \vdots \\ (C_i) \frac{}{\langle \text{skip}, \sigma \rangle \rightarrow \sigma} \end{array} \quad \begin{array}{c} \vdots \\ (C_j) \frac{}{\langle c_2, \sigma \rangle \rightarrow \sigma'} \end{array}}{\langle c_1; c_2, \sigma \rangle \rightarrow \sigma'}$$

Equivalence sémantique (10)

- $\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \hookrightarrow \langle c_1, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma' \rangle$ avec $\langle b, \sigma \rangle \rightsquigarrow \text{true}$. Par hypothèse d'induction on a $\langle c_1, \sigma \rangle \rightarrow \sigma'$ et on conclut en construisant l'arbre :

$$(C_4) \frac{\begin{array}{c} \vdots \\ (B_i) \frac{}{\langle b, \sigma \rangle \rightsquigarrow \text{true}} \end{array} \quad \begin{array}{c} \vdots \\ (C_j) \frac{}{\langle c_1, \sigma \rangle \rightarrow \sigma'} \end{array}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma'}$$

- $\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \hookrightarrow \langle c_2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma' \rangle$ avec $\langle b, \sigma \rangle \rightsquigarrow \text{false}$. Raisonnement similaire au cas précédent.
- $\langle \text{while } b \text{ do } c, \sigma \rangle \hookrightarrow \langle \text{skip}, \sigma \rangle$ avec $\langle b, \sigma \rangle \rightsquigarrow \text{false}$. On conclut en construisant l'arbre :

$$(C_6) \frac{\begin{array}{c} \vdots \\ (B_i) \frac{}{\langle b, \sigma \rangle \rightsquigarrow \text{false}} \end{array}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma}$$

Equivalence sémantique (11)

- $\langle \text{while } b \text{ do } c, \sigma \rangle \hookrightarrow \langle c; \text{while } b \text{ do } c, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma' \rangle$ avec $\langle b, \sigma \rangle \rightsquigarrow \text{true}$. Par hypothèse d'induction on a $\langle c; \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma'$.

L'arbre d'inférence en grands pas finit obligatoirement par

$$(C_3) \frac{\begin{array}{c} \vdots \\ (C_{j_1}) \frac{}{\langle c, \sigma \rangle \rightarrow \sigma_1} \end{array} \quad \begin{array}{c} \vdots \\ (C_{j_2}) \frac{}{\langle \text{while } b \text{ do } c, \sigma_1 \rangle \rightarrow \sigma'} \end{array}}{\langle c; \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma'}$$

On conclut en construisant l'arbre :

$$(C_4) \frac{\begin{array}{c} \vdots \\ (B_i) \frac{}{\langle b, \sigma \rangle \rightsquigarrow \text{true}} \end{array} \quad \begin{array}{c} \vdots \\ (C_{j_1}) \frac{}{\langle c, \sigma \rangle \rightarrow \sigma_1} \end{array} \quad \begin{array}{c} \vdots \\ (C_{j_2}) \frac{}{\langle \text{while } b \text{ do } c, \sigma_1 \rangle \rightarrow \sigma'} \end{array}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma'}$$

Exercice

Montrer que $\text{while } b \text{ do } c \equiv \text{if } b \text{ then } c; \text{while } b \text{ do } c \text{ else skip}$

Outline

- Motivations
- Rappels
- Expressions arithmétiques
- Expressions booléennes
- IMP
- MiniML
 - Appel par valeur
 - Appel par nom

Syntaxe

Syntaxe (abstraite) d'un "petit" langage fonctionnel :

$$e ::= n \mid x \mid \text{fun } x \rightarrow e \mid e e \mid e + e$$

... exprimée à l'aide d'un système d'inférence. E_F est l'ensemble des expressions fonctionnelles, et est défini inductivement par :

$$\frac{}{n \in \mathbb{N} \quad x \in V}$$

$$(\mathbb{F}_1) \frac{}{n} \quad (\mathbb{F}_2) \frac{}{x} \quad (\mathbb{F}_3) \frac{e}{\text{fun } x \rightarrow e}$$

$$(\mathbb{F}_4) \frac{e_1 \quad e_2}{e_1 \quad e_2} \quad (\mathbb{F}_5) \frac{e_1 \quad e_2}{e_1 + e_2}$$

Appel par nom vs appel par valeur

Il existe plusieurs manière d'évaluer

$$(\text{fun } x \rightarrow e) e'$$

- ▶ Appel par valeur : on évalue e' , et on remplace x par sa valeur dans e avant de l'évaluer
on parle d'évaluation stricte
OCaml, SML, C, Java, ...
- ▶ Appel par nom : on remplace x par e' dans e et on évalue e
on parle d'évaluation paresseuse, car e' n'est évalué qu'au moment où on en a besoin
Haskell

Sémantique opérationnelle à grands pas

$$(F_1) \frac{}{n \rightarrow_v n} \quad (F_2) \frac{}{\text{fun } x \rightarrow e \rightarrow_v \text{fun } x \rightarrow e}$$

$$(F_{v3}) \frac{e_1 \rightarrow_v \text{fun } x \rightarrow e \quad e_2 \rightarrow_v v \quad \{v/x\}e \rightarrow_v v'}{e_1 e_2 \rightarrow_v v'}$$

$$(F_4) \frac{e_1 \rightarrow_v v_1 \quad e_2 \rightarrow_v v_2}{e_1 + e_2 \rightarrow_v v_1 \llbracket + \rrbracket v_2}$$

Remarque : pas besoin de garder d'état (langage fonctionnel)

Substitution

Opération du **méta-langage** (celui dans lequel est définie la sémantique), ne fait pas partie de MiniML

Définie récursivement sur les expressions fonctionnelles

$$\{t/x\}x = t$$

$$\{t/x\}y = y \quad \text{si } y \neq x$$

$$\{t/x\}(\text{fun } y \rightarrow t') = \text{fun } z \rightarrow \{t/x\}\{z/y\}t' \quad z \text{ fraîche}$$

$$\{t/x\}(t_1 \ t_2) = \{t/x\}t_1 \ \{t/x\}t_2$$

$$\{t/x\}(t_1 + t_2) = \{t/x\}t_1 + \{t/x\}t_2$$

Exemple

$$(\text{fun } x \rightarrow x + x) (3 + 2)$$

$$(F_2) \frac{}{\text{fun } x \rightarrow x + x \rightarrow_v \text{fun } x \rightarrow x + x}$$

$$(F_4) \frac{(F_1) \frac{}{3 \rightarrow_v 3} \quad (F_1) \frac{}{2 \rightarrow_v 2}}{3 + 2 \rightarrow_v 5}}$$

$$(F_4) \frac{(F_1) \frac{}{5 \rightarrow_v 5} \quad (F_1) \frac{}{5 \rightarrow_v 5}}{5 + 5 \rightarrow_v 10}}$$

$$(F_{v3}) \frac{\vdots \quad \quad \quad \vdots \quad \quad \quad \vdots}{(\text{fun } x \rightarrow x + x) (3 + 2) \rightarrow_v 10}$$

Propriétés

Une **valeur** est soit un **entier**, soit une **abstraction**
(`fun x -> e`)

Proposition : Si $e \rightarrow_v v$ alors v est une valeur.

PREUVE : Par induction sur la dérivation.

Proposition : Si $e \rightarrow_v v$ et $e \rightarrow_v v'$ alors $v = v'$ (à renommage près).

PREUVE : Par induction sur les deux dérivation.

Exercice

Montrer que $(\text{fun } x \rightarrow x x)(\text{fun } x \rightarrow x x)$ n'a pas de sémantique

Sémantique opérationnelle à petits pas

On notera v une valeur (entier ou abstraction)

$$(FS_{v1}) \frac{}{(\text{fun } x \rightarrow e)v \hookrightarrow_v \{v/x\}e}$$

$$(FS_{v2}) \frac{e_1 \hookrightarrow_v e'_1}{e_1 e_2 \hookrightarrow_v e'_1 e_2}$$

$$(FS_{v3}) \frac{e_2 \hookrightarrow_v e'_2}{v e_2 \hookrightarrow_v v e'_2}$$

$$(FS_{v4}) \frac{}{v_1 + v_2 \hookrightarrow_v v_1 \llbracket + \rrbracket v_2}$$

$$(FS_{v5}) \frac{e_1 \hookrightarrow_v e'_1}{e_1 + e_2 \hookrightarrow_v e'_1 + e_2}$$

$$(FS_{v6}) \frac{e_2 \hookrightarrow_v e'_2}{v + e_2 \hookrightarrow_v v + e'_2}$$

Exemple

$$\begin{aligned} & \left(\left(\text{fun } x \rightarrow (\text{fun } y \rightarrow x + y) \right) 1 \right) \left(\left((\text{fun } x \rightarrow 3) 1 \right) + 2 \right) \\ \hookrightarrow_v & (\text{fun } y \rightarrow 1 + y) \left(\left((\text{fun } x \rightarrow 3) 1 \right) + 2 \right) \\ \hookrightarrow_v & (\text{fun } y \rightarrow 1 + y) (3 + 2) \\ \hookrightarrow_v & (\text{fun } y \rightarrow 1 + y) 5 \\ \hookrightarrow_v & 1 + 5 \\ \hookrightarrow_v & 6 \end{aligned}$$

Déterminisme

Proposition : Si $e \hookrightarrow_v v$ et $e \hookrightarrow_v v'$ alors $v = v'$.

PREUVE : Par induction sur e : pour une expression donnée, une seule règle de dérivation peut s'appliquer.

Équivalence

Proposition :

- ▶ Si $e \rightarrow_v v$ alors $e \xrightarrow{*}_v v$
- ▶ Si $e \xrightarrow{*}_v v$ et v est une valeur alors $e \rightarrow_v v$

Sémantique opérationnelle à grands pas

$$(F_1) \frac{}{n \rightarrow_n n} \quad (F_2) \frac{}{\text{fun } x \rightarrow e \rightarrow_n \text{fun } x \rightarrow e}$$

$$(F_{n3}) \frac{e_1 \rightarrow_n \text{fun } x \rightarrow e \quad \{e_2/x\}e \rightarrow_n v}{e_1 e_2 \rightarrow_n v}$$

$$(F_4) \frac{e_1 \rightarrow_n v_1 \quad e_2 \rightarrow_n v_2}{e_1 + e_2 \rightarrow_n v_1 \llbracket + \rrbracket v_2}$$

Exemple

$$(\text{fun } x \rightarrow x + x) (3 + 2)$$

$$(F_2) \frac{}{\text{fun } x \rightarrow x + x \rightarrow_n \text{fun } x \rightarrow x + x}$$

$$(F_4) \frac{(F_4) \frac{(F_1) \frac{}{3 \rightarrow_n 3} \quad (F_1) \frac{}{2 \rightarrow_n 2}}{3 + 2 \rightarrow_n 5} \quad (F_4) \frac{\vdots}{3 + 2 \rightarrow_n 5}}{(3 + 2) + (3 + 2) \rightarrow_n 10}}$$

$$(F_{n3}) \frac{\vdots}{(\text{fun } x \rightarrow x + x) (3 + 2) \rightarrow_n 10}$$

Propriétés

Une **valeur** est soit un **entier**, soit une **abstraction**
(`fun x -> e`)

Proposition : Si $e \rightarrow_n v$ alors v est une valeur.

PREUVE : Par induction sur la dérivation.

Proposition : Si $e \rightarrow_n v$ et $e \rightarrow_n v'$ alors $v = v'$ (à renommage près).

PREUVE : Par induction sur les deux dérivation.

Sémantique opérationnelle à petits pas

$$(FS_{n1}) \frac{}{(\mathbf{fun} \ x \ \rightarrow \ e_1) e_2 \hookrightarrow_n \{e_2/x\} e_1}$$

$$(FS_{n2}) \frac{e_1 \hookrightarrow_n e'_1}{e_1 \ e_2 \hookrightarrow_n e'_1 \ e_2}$$

$$(FS_{n4}) \frac{}{v_1 + v_2 \hookrightarrow_n v_1 \llbracket + \rrbracket v_2}$$

$$(FS_{n5}) \frac{e_1 \hookrightarrow_n e'_1}{e_1 + e_2 \hookrightarrow_n e'_1 + e_2}$$

$$(FS_{n6}) \frac{e_2 \hookrightarrow_n e'_2}{v + e_2 \hookrightarrow_n v + e'_2}$$

Exemple

$$\begin{aligned} & \left(\left(\text{fun } x \rightarrow (\text{fun } y \rightarrow x + y) \right) 1 \right) \left(\left((\text{fun } x \rightarrow 3) 1 \right) + 2 \right) \\ \hookrightarrow_n & (\text{fun } y \rightarrow 1 + y) \left(\left((\text{fun } x \rightarrow 3) 1 \right) + 2 \right) \\ \hookrightarrow_n & 1 + \left(\left((\text{fun } x \rightarrow 3) 1 \right) + 2 \right) \\ \hookrightarrow_n & 1 + (3 + 2) \\ \hookrightarrow_n & 1 + 5 \\ \hookrightarrow_n & 6 \end{aligned}$$

Déterminisme

Proposition : Si $e \hookrightarrow v$ et $e \hookrightarrow v'$ alors $v = v'$.

PREUVE : Par induction sur e : pour une expression donnée, une seule règle de dérivation peut s'appliquer.

Équivalence

Proposition :

- ▶ Si $e \rightarrow_n v$ alors $e \xrightarrow{n}^* v$
- ▶ Si $e \xrightarrow{n}^* v$ et v est une valeur alors $e \rightarrow_n v$

Non-équivalence des appels par nom et par valeur

Proposition : $e \rightarrow_v v$ n'est pas équivalent à $e \rightarrow_n v$

Contre exemple :

`(fun x -> 1)((fun x -> x x) (fun x -> x x))`