Sémantique opérationnelle

ENSIIE- ISL - 2007/2008

Catherine Dubois (d'après les transparents de Mathieu Jaume, P6)

Sémantique : Quoi ?

"Sémantique : Etude du sens des unités linguistiques et de leur composition." [Petit Larousse, 1994]

Chaîne de caractères	analyse lexicale $\longrightarrow$ analyse syntaxique	Arbre de syntaxe abstraite (AST)

Associer une "signification" à un programme à partir de son AST.

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 : Pourquoi ?

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 ... portabilité des programmes
- raisonnement sur les propriétés attendues du langage – comme par exemple le déterminisme de l'exécution des instructions
- 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

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 cf. cours ITP

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

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 $\mathcal{J}$:

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

$\{j_1, \cdots, j_n\}$: *prémisses* qui doivent être "satisfaites" pour que la *conclusion* j le soit.

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

Arbres d'inférence (de preuve, de dérivation) – Théorèmes

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 j_k dont les fils sont j_{k_1}, j_{k_2}, ... , j_{k_n}, il existe une règle de $\Phi[\mathcal{J}]$:

$$(r)\frac{j_{k_1} \quad j_{k_2} \quad \cdots \quad j_{k_n}}{j_k}$$

Les feuilles ... ne peuvent être obtenues qu'à partir des axiomes.

$\Phi[\mathcal{J}]$ caractérise un ensemble de théorèmes $\mathsf{Th}(\Phi[\mathcal{J}]) \subseteq \mathcal{J}$ 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 : $(R)\frac{}{j}$ ou :

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

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 :

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

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

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

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

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

Preuve : exercice ...

Définition inductive des expressions arithmétiques

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

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

$$(\mathbb{A}_1)\frac{}{n}\ (n \in \mathbb{Z}) \qquad (\mathbb{A}_2)\frac{}{x}\ (x \in V)$$

$$(\mathbb{A}_3)\frac{a_1 \quad a_2}{a_1 + a_2} \qquad (\mathbb{A}_4)\frac{a_1 \quad a_2}{a_1 - a_2} \qquad (\mathbb{A}_5)\frac{a_1 \quad a_2}{a_1 \times a_2} \qquad (\mathbb{A}_6)\frac{a_1 \quad a_2}{a_1 / a_2}$$

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

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

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 $\mathbb{Z}$.

Expressions Arithmétiques : Syntaxe abstraite – Implantation

$37 + v \in E_A$?

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

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;;
```

en Coq :

```
Parameter V : Set.
Parameter eq_dec_V : forall (v1 v2:V),{v1=v2}+{~v1=v2}.

Inductive EA : Set :=
nb : Z -> EA | var : V -> EA | pls : EA -> EA -> EA
| mns : EA -> EA -> EA | mlt : EA -> EA -> EA | dv : EA -> EA -> EA.
```

Induction

Prouver par induction une propriété P sur tous les éléments de l'ensemble $\mathsf{Ind}(\Phi[\mathcal{J}])$,

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 $\left(\forall \frac{j_1 \quad \cdots \quad j_n}{j} \in \Phi[\mathcal{J}]\ (\forall k \in \{j_1, \cdots, j_n\}\ P(k)) \text{ implique } P(j)\right)$
alors $\forall x \in \mathsf{Ind}(\Phi[\mathcal{J}])\ P(x)$.

Preuve : exercice ...

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

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 (1)

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 $\forall a \in E_A\ P(a)$

Induction sur les expressions arithmétiques (2)

```
Coq < Check EA_ind.
EA_ind
: forall (P:(EA->Prop)),
(forall (z:Z),(P (nb z)))
->(forall (v:V)(P (var v)))
->(forall (e:EA)(P e)->forall (e0:EA)(P e0)->(P (pls e e0)))
->(forall (e:EA)(P e)->forall (e0:EA)(P e0)->(P (mns e e0)))
->(forall (e:EA)(P e)->forall (e0:EA)(P e0)->(P (mlt e e0)))
->(forall (e:EA)(P e)->forall (e0:EA)(P e0)->(P (dv e e0)))
->forall (e:EA)(P e)
```

Remarque :

$$(A_1 \wedge A_2 \wedge \cdots \wedge A_{n-1} \wedge A_n) \Rightarrow B$$
$$\equiv \quad (A_1 \Rightarrow (A_2 \Rightarrow \cdots \Rightarrow (A_{n-1} \Rightarrow A_n))) \Rightarrow B$$

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 :

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

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

En OCaml : Err est une exception

En Coq : Définition d'un type pour $\mathbb{V}$

```
Inductive values : Set := Val : Z -> values | Erreur : values.
```

Interprétation des symboles

• Interprétation des variables par une valuation : $\mathcal{V}[\mathbb{Z}] = \{V \to \mathbb{Z}\}$ (mémoire, environnement, ...)

• Interpréter des symboles de $\mathbb{Z} \cup \{+, -, \times, /\}$:

n est interprété par lui-même : $[\![n]\!] = n$

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

$v_1 [\![/]\!] v_2$	$v_2 \in \mathbb{Z}\backslash\{0\}$	0	Err
$v_1 \in \mathbb{Z}$	v_1 / v_2	Err	Err
Err	Err	Err	Err

Schéma d'interprétation des expressions arithmétiques

$$\mathcal{A}[\![\_]\!]\_ : E_A \to \mathcal{V}[\mathbb{Z}] \to \mathbb{Z} \cup \{\mathsf{Err}\}$$

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

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

Approche dénotationnelle

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
```

Evaluation des expressions arithmétiques : Implantation en Coq

Définition d'une fonction récursive.

```
Definition Sigma : Set := V -> Z.

Fixpoint eval_EA_fct (s:Sigma)(a:EA){struct a} : values :=
match a with
(nb n) => (Val n)
| (var x) => (Val (s x))
| ...
| (dv a1 a2) => match ((eval_EA_fct s a1),(eval_EA_fct s a2)) with
                ((Val n1),(Val n2)) => (match n2 with
                                        `0` => Erreur
                                        |  _ => (Val (Zdiv n1 n2))
                                         end)
                | (Erreur,_) => Erreur
                | ((Val _), Erreur) => Erreur
                end)
end.
```

```
eval_EA_fct : Sigma->EA->values
```

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

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

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

$$(A_5)\ \frac{\langle a_1, \sigma\rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma\rangle \rightsquigarrow n_2}{\langle a_1 - a_2, \sigma\rangle \rightsquigarrow n_1 [\![-]\!] n_2} \qquad (A_6)\ \frac{\langle a_1, \sigma\rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma\rangle \rightsquigarrow n_2}{\langle a_1 / a_2, \sigma\rangle \rightsquigarrow n_1 [\![/]\!] n_2}$$

Evaluation : exemple

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

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

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

Evaluation à grands pas : Implantation en Coq

Définition inductive d'une relation.

```
Inductive eval_EA : EA -> Sigma -> values -> Prop :=
| eval_var : forall (s:Sigma)(v:V),(eval_EA (var v) s (s v))
| eval_nb :  forall (s:Sigma)(n : Z), (eval_EA (nb n) s (Val n))
| eval_pls : forall (s:Sigma)(a1 a2:EA)(n1 n2:values),
(eval_EA a1 s n1) -> (eval_EA a2 s n2) ->
    (eval_EA (pls a1 a2) s (ZplusErr n1 n2))
...
| eval_dv : forall (s:Sigma)(a1 a2:EA)(n1 n2: values),
(eval_EA s a1 n1) -> (eval_EA s a2 n2) ->
  (eval_EA s (dv a1 a2) (ZdivErr n1 n2)).
```

```
avec  Definition ZplusErr (v1 v2 : values) : values := match (v1, v2) with
        Val x1, Val x2 => Val (x1 + x2)
      | Erreur, _ => Erreur
      | _, Erreur => Erreur
      end.
```

Déterminisme de l'évaluation des expressions arithmétiques (1)

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

Proposition :

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

```
Lemma det_eval_EA : forall (a:EA)(s:Sigma)(v1 v2:values),
(eval_EA s a v1) -> (eval_EA s a v2) -> v1=v2.

1 subgoal

============================
forall (a:EA; s:Sigma; v1,v2:values)
(eval_EA s a v1)->(eval_EA s a v2)->v1=v2
```

PREUVE : Par induction sur a.

```
induction 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 $v_1 = v_2$.

```
intros.
```

```
6 subgoals
...
H : (eval_EA s (nb z) v1) H0 : (eval_EA s (nb z) v2)
============================
v1=v2
```

```
inversion H.
```

```
6 subgoals
...
H : (eval_EA s (nb z) v1) H0 : (eval_EA s (nb z) v2)
s0 : Sigma n : Z H1 : s0=s H3 : ‘n = z‘ H2 : z=v1
============================
z=v2
```

```
inversion H0.

6 subgoals
...
H : (eval_EA s (nb z) v1) H0 : (eval_EA s (nb z) v2)
s0 : Sigma n : Z H1 : s0=s H3 : ‘n = z‘ H2 :  z=v1
s1 : Sigma n0 : Z H4 : s1=s H6 : ‘n0 = z‘ H5 : z=v2
============================
z=z
```

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 $v_1 = v_2$.

```
intros. inversion H. inversion H0. trivial.
```

Déterminisme de l'évaluation des expressions arithmétiques (4)

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

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

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
$\langle a_1, \sigma\rangle \rightsquigarrow n_1$, $\langle a_2, \sigma\rangle \rightsquigarrow n_2$ et $v_1 = n_1 [\![+]\!] n_2$

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.

Equivalence des sémantiques opérationnelle et dénotationnelle (1)

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

```
Lemma eval_EA_fct_to_rel :
forall (s:Sigma)(a:EA)(v:values),
(eval_EA s a v) <-> (eval_EA_fct s a)=  v.

1 subgoal
============================
forall (s:Sigma) (a:EA) (v:values),
     (eval_EA s a v)<->(eval_EA_fct sa)=v
```

Evaluation des expressions arithmétiques : variables (1)

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 :

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

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 $\sigma_1 =_{X_1} \sigma_2$.

Evaluation des expressions arithmétiques : variables (2)

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 $\langle a, \sigma_2 \rangle \rightsquigarrow n$.

Evaluation des expressions arithmétiques : variables (3)

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 :

$$(A_3)\dfrac{(A_i)\dfrac{\vdots}{\langle a_1, \sigma_1 \rangle \rightsquigarrow n_1} \quad (A_j)\dfrac{\vdots}{\langle a_2, \sigma_1 \rangle \rightsquigarrow n_2}}{\langle a_1 + a_2, \sigma_1 \rangle \rightsquigarrow v}$$

avec $v = n_1 [\![+]\!] n_2$. 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* $\mathcal{R}$ sur $E \times E$, où E est muni d'une loi de composition interne $\odot$, est une relation d'équivalence compatible avec $\odot$:

$$\forall x, x', y, y' \in E \quad \text{si } (x\,\mathcal{R}\,x' \text{ et } y\,\mathcal{R}\,y') \text{ alors } (x \odot y)\,\mathcal{R}\,(x' \odot y')$$

Expressions arithmétiques équivalentes (2)

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

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

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

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

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

Puisque $2 \times \sigma(x) = \sigma(x) + \sigma(x) = n$, 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 (1)

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

De plus, 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 $[\![+]\!]$ par +. C'est donc le contexte qui permet de dire si on parle du symbole + ou de son interprétation. (Si il y a ambiguité, on précisera.)

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

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

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

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

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 : $\langle e_0, \sigma\rangle \hookrightarrow \langle e_1, \sigma\rangle \hookrightarrow \langle e_2, \sigma\rangle \hookrightarrow \cdots$ 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 $\hookrightarrow$.

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

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

De plus, si $e_k = v \in \mathbb{Z}$ (configuration terminale), alors $\langle e, \sigma\rangle \overset{\star}{\hookrightarrow} v$.

e s'évalue en la valeur v $(\in \mathbb{Z})$ si et seulement il existe une séquence finie $\langle e, \sigma\rangle \overset{\star}{\hookrightarrow} \langle v, \sigma\rangle$

Sémantique opérationnelle d'évaluation à petits pas (3)

Deux sortes de règles :

- règles de réduction élémentaires

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

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

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

Sémantique opérationnelle d'évaluation à petits pas (4)

• Etapes de simplification (passage au contexte, propagation)

Choisissons un mode d'évaluation "de gauche à droite"

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

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

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

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

Sémantique opérationnelle d'évaluation à petits pas : Exemple 2

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

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

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

$\hookrightarrow$ est déterministe

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 :

$$C \ ::= \ \bullet \mid C \ op \ e \mid v \ op \ C$$

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 :

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

Equivalence des sémantiques (1)

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

PREUVE Induction sur e.

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

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

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

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

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

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

Equivalence des sémantiques (5)

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

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 $\langle a_1, \sigma\rangle \overset{*}{\hookrightarrow} n_1$ et $\langle a_2, \sigma\rangle \overset{*}{\hookrightarrow} n_2$ et il existe donc 2 séquences de calcul :

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

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

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 $\langle a_1^i \ \mathsf{op}_2 \ a_2, \sigma\rangle \hookrightarrow \langle a_1^{i+1} \ \mathsf{op}_2 \ a_2, \sigma\rangle$. On obtient alors :

$$\langle a_1 \ \mathsf{op}_2 \ a_2, \sigma\rangle = \langle a_1^0 \ \mathsf{op}_2 \ a_2, \sigma\rangle \hookrightarrow \langle a_1^1 \ \mathsf{op}_2 \ a_2, \sigma\rangle \hookrightarrow \langle a_1^2 \ \mathsf{op}_2 \ a_2, \sigma\rangle \hookrightarrow \cdots$$
$$\cdots \hookrightarrow \langle a_1^{k_1} \ \mathsf{op}_2 \ a_2, \sigma\rangle = \langle n_1 \ \mathsf{op}_2 \ a_2, \sigma\rangle \qquad (3)$$

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 $\langle n_1\ \mathsf{op}_2\ a_2^i, \sigma\rangle \hookrightarrow \langle n_1\ \mathsf{op}_2\ a_2^{i+1}, \sigma\rangle$. On obtient alors :

$$\begin{array}{l}\langle n_1\ \mathsf{op}_2\ a_2, \sigma\rangle = \langle n_1\ \mathsf{op}_2\ a_2^0, \sigma\rangle \hookrightarrow \langle n_1\ \mathsf{op}_2\ a_2^1, \sigma\rangle \hookrightarrow \langle n_1\ \mathsf{op}_2\ a_2^2, \sigma\rangle \hookrightarrow \cdots \\ \cdots \hookrightarrow \langle n_1\ \mathsf{op}_2\ a_2^{k_2}, \sigma\rangle = \langle n_1\ \mathsf{op}_2\ n_2, \sigma\rangle\end{array} \quad (4)$$

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

- Si $e = a_1/a_2$, ...

Equivalence des sémantiques (7)

Pour montrer $\forall e \in E_A \ \forall \sigma \in \mathcal{V}[\mathbb{Z}] \ \forall v \in \mathbb{V} \ \langle e, \sigma \rangle \overset{\star}{\hookrightarrow} v \Rightarrow \langle e, \sigma \rangle \rightsquigarrow v$, 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 $\langle e_1, \sigma \rangle \hookrightarrow \langle e_2, \sigma \rangle$ 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 (9)

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

Preuve

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

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

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.

Expressions booléennes

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

Définition inductive de l'ensemble E_B des expressions booléennes.

Jugements : $(\mathbb{Z} \cup V \cup \{+, -, \times, /\} \cup \{\mathsf{true}, \mathsf{false}\} \cup \{=, \leq\} \cup \{\ \mathsf{or}\ ,\ \mathsf{and}\ , \mathsf{not}\ \})^\star$

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

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

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

Interprétation des expressions booléennes

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

... à partir de l'interprétation des expressions arithmétiques, il reste a interpréter les symboles de $\{\mathsf{true}, \mathsf{false}\} \cup \{=, \leq\} \cup \{\ \mathsf{or}\ ,\ \mathsf{and}\ , \mathsf{not}\ \}$.

$t \in \mathbb{B}$ est interprété par lui-même : $[\![\mathsf{true}]\!] = \mathsf{true}$ et $[\![\mathsf{false}]\!] = \mathsf{false}$

$v_1 \begin{pmatrix} [\![=]\!] \\ [\![\leq]\!] \end{pmatrix} v_2$	$v_2 \in \mathbb{Z}$	Err
$v_1 \in \mathbb{Z}$	$v_1 \begin{pmatrix} = \\ \leq \end{pmatrix} v_2$	Err
Err	Err	Err

$v_1 \begin{pmatrix} [\![\mathsf{and}]\!] \\ [\![\mathsf{or}]\!] \end{pmatrix} v_2$	$v_2 \in \mathbb{B}$	Err
$v_1 \in \mathbb{B}$	$v_1 \begin{pmatrix} \wedge \\ \vee \end{pmatrix} v_2$	Err
Err	Err	Err

$[\![\mathsf{not}\ ]\!]v$	$v \in \mathbb{B}$	Err
	$\neg t$	Err

Schéma d'interprétation des expressions booléennes

$$\mathcal{B}[\![\_]\!]\_ : E_B \to \mathcal{V}[\mathbb{Z}] \to \mathbb{B} \cup \{\mathsf{Err}\}$$

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

Expressions booléennes : 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 $v \in \mathbb{IB}$ étant donnée une valuation σ.

$$(B_1)\ \frac{}{\langle t, \sigma\rangle \rightsquigarrow t} \quad t \in \mathbb{IB}$$

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

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

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

$$(B_{14})\ \frac{\langle b, \sigma\rangle \rightsquigarrow t}{\langle \mathsf{not}\ b, \sigma\rangle \rightsquigarrow (\neg t)}$$

Choix des règles

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

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

La sémantique opérationnelle a donné une sémantique paresseuse aux opérateurs and et or .

Sémantique opérationnelle : Propriétés

Expressions booléennes équivalentes

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

Proposition $\sim$ est une congruence.

Preuve. Exercice ...

Proposition Unicité d'un résultat

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

Preuve. Exercice ...

Constructions impératives

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

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

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

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

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

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

Constructions impératives : Exemple (PGCD)

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

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

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 $\langle c, \sigma_1 \rangle \to \sigma_2$ à l'aide d'un système d'inférence.

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

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

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

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

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

Sémantique opérationnelle à grands pas : Exemple

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

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 $\langle x \leq 0, \sigma_1\rangle \rightsquigarrow \textsf{true}$ et $\langle x := x+1, \sigma_1\rangle \to \sigma_2$ et où D_3 est l'arbre :

$$(C_6)\dfrac{(B_3)\dfrac{(A_2)\dfrac{}{\langle x,\sigma_2\rangle \rightsquigarrow 1} \quad (A_1)\dfrac{}{\langle 0,\sigma_2\rangle \rightsquigarrow 0}}{\langle x \leq 0,\sigma_2\rangle \rightsquigarrow \textsf{false}}}{\langle \textsf{while } x \leq 0 \textsf{ do } x := x+1,\sigma_2\rangle \to \sigma_2}$$

Effets de bords lors de l'évaluation des expressions

Les règles :

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

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

$$(C_7)\dfrac{(B_1)\dfrac{}{\langle \mathsf{true}, \sigma\rangle \rightsquigarrow \mathsf{true}}\ (C_1)\dfrac{}{\langle \mathsf{skip}, \sigma\rangle \rightarrow \sigma}\ (C_7)\dfrac{\vdots}{\langle \mathsf{while\ true\ do\ skip}, \sigma\rangle \rightarrow \sigma'}}{\langle \mathsf{while\ true\ do\ skip}, \sigma\rangle \rightarrow \sigma'}$$

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

Programmes équivalents

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

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

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	$\sigma(x) = 2$

$\langle c;(\text{if } b \text{ then } c' \text{ else } c''), \sigma \rangle \to \sigma'$	$\sigma'(x) = 5$
$\langle \text{if } b \text{ then } (c;c') \text{ else } (c;c''), \sigma \rangle \to \sigma''$	$\sigma''(x) = 1$

Programmes équivalents : Exemple (1)

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

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

Si on dispose d'un arbre d'inférence de $\langle$(if b then c else c'); $c'', \sigma\rangle \to \sigma'$, il a forcément été obtenu à partir de la règle C_3 :

$$(C_3)\dfrac{(C_i)\dfrac{\vdots}{\langle \text{if } b \text{ then } c \text{ else } c', \sigma\rangle \to \sigma_1} \quad (C_j)\dfrac{\vdots}{\langle c'', \sigma_1\rangle \to \sigma'}}{\langle(\text{if } b \text{ then } c \text{ else } c'); c'', \sigma\rangle \to \sigma'}$$

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 :

$$(C_3)\dfrac{(C_4)\dfrac{(B_i)\dfrac{\vdots}{\langle b,\sigma\rangle \rightsquigarrow \mathsf{true}} \quad (C_j)\dfrac{\vdots}{\langle c,\sigma\rangle \to \sigma_1}}{\langle \mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else}\ c',\sigma\rangle \to \sigma_1} \quad (C_k)\dfrac{\vdots}{\langle c'',\sigma_1\rangle \to \sigma'}}{\langle (\mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else}\ c');c'',\sigma\rangle \to \sigma'}$$

on peut obtenir l'arbre :

$$(C_4)\dfrac{(B_i)\dfrac{\vdots}{\langle b,\sigma\rangle \rightsquigarrow \mathsf{true}} \quad (C_3)\dfrac{(C_j)\dfrac{\vdots}{\langle c,\sigma\rangle \to \sigma_1} \quad (C_k)\dfrac{\vdots}{\langle c'',\sigma_1\rangle \to \sigma'}}{\langle c;c'',\sigma\rangle \to \sigma'}}{\langle \mathsf{if}\ b\ \mathsf{then}\ (c;c'')\ \mathsf{else}\ (c';c''),\sigma\rangle \to \sigma'}$$

et *vice versa*.

Programmes équivalents : Exemple (3)

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

$$(C_3)\dfrac{(C_5)\dfrac{(B_i)\dfrac{\vdots}{\langle b,\sigma\rangle \rightsquigarrow \mathsf{false}}\ (C_j)\dfrac{\vdots}{\langle c',\sigma\rangle \to \sigma_1}}{\langle \mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else}\ c',\sigma\rangle \to \sigma_1}\ (C_k)\dfrac{\vdots}{\langle c'',\sigma_1\rangle \to \sigma'}}{\langle (\mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else}\ c');c'',\sigma\rangle \to \sigma'}$$

on peut obtenir l'arbre :

$$(C_5)\dfrac{(B_i)\dfrac{\vdots}{\langle b,\sigma\rangle \rightsquigarrow \mathsf{false}}\ (C_3)\dfrac{(C_j)\dfrac{\vdots}{\langle c',\sigma\rangle \to \sigma_1}\ (C_k)\dfrac{\vdots}{\langle c'',\sigma_1\rangle \to \sigma'}}{\langle c';c'',\sigma\rangle \to \sigma'}}{\langle \mathsf{if}\ b\ \mathsf{then}\ (c;c'')\ \mathsf{else}\ (c';c''),\sigma\rangle \to \sigma'}$$

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 :

$$\mathsf{while\ not}\ (x = y)\ \mathsf{do\ if}\ x \leq y\ \mathsf{then}\ y := y - x\ \mathsf{else}\ x := x - y$$

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

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

Utilisation de la récurrence bien fondée pour prouver la propriété $P(\sigma)$:
$(\exists \sigma' \in \mathcal{V}[\mathbb{Z}]\ \langle \mathsf{Euclid}, \sigma \rangle \rightarrow \sigma')$ 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 (2)

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 :

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

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

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

Remarque

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

Terminaison de programmes (3)

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

S'il existait une suite infinie strictement décroissante $\sigma_1 \succ \sigma_2 \succ \cdots$, 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 (4)

Soit $\sigma \in S$, 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 :

$$(C_6)\dfrac{(B_{14})\dfrac{(B_2)\dfrac{(A_2)\dfrac{}{\langle x, \sigma\rangle \rightsquigarrow m} \quad (A_2)\dfrac{}{\langle y, \sigma\rangle \rightsquigarrow n}}{\langle x = y, \sigma\rangle \rightsquigarrow \mathsf{true}}}{\langle \mathsf{not}\ (x = y), \sigma\rangle \rightsquigarrow \mathsf{false}}}{\langle \mathsf{Euclid}, \sigma\rangle \rightarrow \sigma}$$

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

$$(B_{14})\dfrac{(B_2)\dfrac{(A_2)\dfrac{}{\langle x, \sigma\rangle \rightsquigarrow m} \quad (A_2)\dfrac{}{\langle y, \sigma\rangle \rightsquigarrow n}}{\langle x = y, \sigma\rangle \rightsquigarrow \mathsf{false}}}{\langle \mathsf{not}\ (x = y), \sigma\rangle \rightsquigarrow \mathsf{true}} \quad (D)$$

Deux sous-cas sont possibles.

Terminaison de programmes (5)

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

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

où D_2 est l'arbre :

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

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

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

Terminaison de programmes (6)

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

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

où D_2 est l'arbre :

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

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

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

Terminaison de programmes (7)

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

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

Déterminisme (1)

Proposition

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

Peut-on prouver cette proposition par induction sur c ?

Cas de la règle $(\mathbb{C}_5)\dfrac{c}{\textsf{while } b \textsf{ do } c}$. Si b s'évalue à $\textsf{true}$:

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

$$(C_7)\dfrac{(B_i)\dfrac{\vdots}{\langle b, \sigma\rangle \rightsquigarrow \textsf{true}} \quad (C_{j'})\dfrac{\vdots}{\langle c, \sigma\rangle \to \sigma'_1} \quad (C_{k'})\dfrac{\vdots}{\langle \textsf{while } b \textsf{ do } c, \sigma'_1\rangle \to \sigma'_2}}{\langle \textsf{while } b \textsf{ do } c, \sigma\rangle \to \sigma'_2}$$

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 $\textsf{while } b \textsf{ do } c$!

Déterminisme (2)

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

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

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 petits pas

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.

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

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

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

$$(S_4)\ \frac{\langle b, \sigma\rangle \rightsquigarrow \mathsf{true}}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, \sigma\rangle \hookrightarrow \langle c_1, \sigma\rangle} \quad (S_5)\ \frac{\langle b, \sigma\rangle \rightsquigarrow \mathsf{false}}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, \sigma\rangle \hookrightarrow \langle c_2, \sigma\rangle}$$

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

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

Séquence de calcul

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

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

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 : $\langle \mathsf{skip}, \sigma \rangle$

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

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

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

Sémantique opérationnelle à petits pas : Exemple

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

Composition

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

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 $n+1$:

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

Par hypothèse d'induction, il existe une séquence :

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

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

Equivalence sémantique (1)

Proposition $\forall c \in E_C \quad \forall \sigma_1, \sigma_2 \in \Sigma \quad \langle c, \sigma_1 \rangle \to \sigma_2 \Rightarrow \langle c, \sigma_1 \rangle \overset{\star}{\hookrightarrow} \sigma_2$

Preuve : Induction sur $\langle c, \sigma_1 \rangle \to \sigma_2$.

$$(C_1)\,\frac{}{\langle \mathsf{skip}, \sigma \rangle \to \sigma}$$

Puisque $\langle \mathsf{skip}, \sigma \rangle$ est une configuration terminale, on a bien $\langle \mathsf{skip}, \sigma \rangle \overset{\star}{\hookrightarrow} \sigma$

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

On peut construire l'arbre :

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

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

Equivalence sémantique (2)

$$(C_3)\ \frac{(C_i)\ \dfrac{\vdots}{\langle c_1, \sigma\rangle \to \sigma_1} \qquad (C_j)\ \dfrac{\vdots}{\langle c_2, \sigma_1\rangle \to \sigma_2}}{\langle c_1; c_2, \sigma\rangle \to \sigma_2}$$

Par hypothèse d'induction, on a $\langle c_1, \sigma\rangle \overset{\star}{\hookrightarrow} \sigma_1$ et $\langle c_2, \sigma_1\rangle \overset{\star}{\hookrightarrow} \sigma_2$ et donc :

$$\langle c_1, \sigma\rangle \hookrightarrow \cdots \hookrightarrow \langle \mathsf{skip}, \sigma_1\rangle \tag{5}$$

$$\langle c_2, \sigma_1\rangle \hookrightarrow \cdots \hookrightarrow \langle \mathsf{skip}, \sigma_2\rangle \tag{6}$$

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

$$\langle c_1; c_2, \sigma\rangle \hookrightarrow \cdots \hookrightarrow \langle \mathsf{skip}; c_2, \sigma_1\rangle \tag{7}$$

La règle S_3 permet de construire la transition :

$$\langle \mathsf{skip}; c_2, \sigma_1\rangle \hookrightarrow \langle c_2, \sigma_1\rangle \tag{8}$$

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

$$\langle c_1; c_2, \sigma\rangle \hookrightarrow \cdots \hookrightarrow \langle \mathsf{skip}, \sigma_2\rangle$$

ce qui permet finalement d'établir $\langle c_1; c_2, \sigma\rangle \overset{\star}{\hookrightarrow} \sigma_2$.

Equivalence sémantique (3)

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

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

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

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

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

Raisonnement similaire au cas précédent.

Equivalence sémantique (4)

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

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

$$(S_6)\,\dfrac{\begin{array}{c}\vdots\\ (B_i)\,\dfrac{}{\langle b,\sigma\rangle \rightsquigarrow \mathsf{false}}\end{array}}{\langle \mathsf{while}\ b\ \mathsf{do}\ c,\sigma\rangle \hookrightarrow \langle \mathsf{skip},\sigma\rangle}$$

ce qui permet d'établir $\langle \mathsf{while}\ b\ \mathsf{do}\ c,\sigma\rangle \overset{\star}{\hookrightarrow} \sigma$.

Equivalence sémantique (5)

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

Par hypothèse d'induction on a :

$$\langle c,\sigma\rangle \hookrightarrow \cdots \hookrightarrow \langle \mathsf{skip},\sigma_1\rangle \tag{9}$$

$$\langle \mathsf{while}\ b\ \mathsf{do}\ c,\sigma_1\rangle \hookrightarrow \cdots \hookrightarrow \langle \mathsf{skip},\sigma_2\rangle \tag{10}$$

A partir de (9), on obtient :

$$\langle c; \mathsf{while}\ b\ \mathsf{do}\ c,\sigma\rangle \hookrightarrow \cdots \hookrightarrow \langle \mathsf{skip}; \mathsf{while}\ b\ \mathsf{do}\ c,\sigma_1\rangle \tag{11}$$

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

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

A partir de (13), (11), (12) et (10), on obtient :
$\langle \mathsf{while}\ b\ \mathsf{do}\ c,\sigma\rangle \hookrightarrow \cdots \hookrightarrow \langle \mathsf{skip},\sigma_2\rangle$ et donc $\langle \mathsf{while}\ b\ \mathsf{do}\ c,\sigma\rangle \overset{\star}{\hookrightarrow} \sigma_2$.

Equivalence sémantique (6)

Lemme Si $\langle c_1; c_2, \sigma\rangle \overset{k}{\hookrightarrow} \sigma''$, alors $\exists \sigma' \in \mathcal{V}[\mathbb{Z}]$, $\exists k_1, k_2 \in \mathbb{N}$, tels que $\langle c_1, \sigma\rangle \overset{k_1}{\hookrightarrow} \sigma'$, $\langle c_2, \sigma'\rangle \overset{k_2}{\hookrightarrow} \sigma''$ 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). $\langle c_1; c_2, \sigma\rangle \hookrightarrow \langle c_1'; c_2, \sigma_1\rangle \overset{k_0}{\hookrightarrow} \sigma''$ 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 $\langle c_1', \sigma_1\rangle \overset{k_1'}{\hookrightarrow} \sigma_1'$, $\langle c_2, \sigma_1'\rangle \overset{k_2'}{\hookrightarrow} \sigma''$ avec $k_0 = k_1' + k_2'$. et on peut conclure avec $k_1 = k_1' + 1$ et $k_2 = k_2'$.

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

Equivalence sémantique (7)

Proposition $\forall c \in E_C \quad \forall \sigma, \sigma' \in \Sigma \quad \langle c, \sigma \rangle \overset{\star}{\hookrightarrow} \sigma' \Rightarrow \langle c, \sigma \rangle \rightarrow \sigma'$

PREUVE : On a la séquence :

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

Induction sur la séquence de calcul.

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

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

Equivalence sémantique (8)

• $\langle c_1; c_2, \sigma\rangle \hookrightarrow \langle c_1'; c_2, \sigma_1\rangle \hookrightarrow \cdots \hookrightarrow \langle \mathsf{skip}, \sigma'\rangle$ 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 $\langle c_1, \sigma\rangle \overset{k_1}{\hookrightarrow} \sigma_0$ et $\langle c_2, \sigma_0\rangle \overset{k_2}{\hookrightarrow} \sigma'$. Par hypothèse d'induction on a $\langle c_1, \sigma\rangle \rightsquigarrow \sigma_0$ et $\langle c_2, \sigma_0\rangle \rightsquigarrow \sigma'$ et on conclut en construisant l'arbre :

$$(C_3)\,\frac{(C_i)\,\dfrac{\vdots}{\langle c_1, \sigma\rangle \rightsquigarrow \sigma_0} \qquad (C_j)\,\dfrac{\vdots}{\langle c_2, \sigma_0\rangle \rightsquigarrow \sigma'}}{\langle c_1; c_2, \sigma\rangle \rightsquigarrow \sigma'}$$

• $\langle \mathsf{skip}; c_2, \sigma\rangle \hookrightarrow \langle c_2, \sigma\rangle \hookrightarrow \cdots \hookrightarrow \langle \mathsf{skip}, \sigma'\rangle$ Par hypothèse d'induction on a $\langle c_2, \sigma\rangle \rightsquigarrow \sigma'$ et on conclut en construisant l'arbre :

$$(C_3)\,\frac{(C_i)\,\dfrac{\vdots}{\langle \mathsf{skip}, \sigma\rangle \rightsquigarrow \sigma} \qquad (C_j)\,\dfrac{\vdots}{\langle c_2, \sigma\rangle \rightsquigarrow \sigma'}}{\langle c_1; c_2, \sigma\rangle \rightsquigarrow \sigma'}$$

Equivalence sémantique (9)

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

$$(C_4)\dfrac{(B_i)\dfrac{\vdots}{\langle b, \sigma\rangle \rightsquigarrow \textsf{true}} \quad (C_j)\dfrac{\vdots}{\langle c_1, \sigma\rangle \rightsquigarrow \sigma'}}{\langle \textsf{if } b \textsf{ then } c_1 \textsf{ else } c_2, \sigma\rangle \rightsquigarrow \sigma'}$$

• $\langle \textsf{if } b \textsf{ then } c_1 \textsf{ else } c_2, \sigma\rangle \hookrightarrow \langle c_2, \sigma\rangle \hookrightarrow \cdots \hookrightarrow \langle \textsf{skip}, \sigma'\rangle$ avec $\langle b, \sigma\rangle \rightsquigarrow \textsf{false}$. Raisonnement similaire au cas précédent.

• $\langle \textsf{while } b \textsf{ do } c, \sigma\rangle \hookrightarrow \langle \textsf{skip}, \sigma\rangle$ avec $\langle b, \sigma\rangle \rightsquigarrow \textsf{false}$. On conclut en construisant l'arbre :

$$(C_6)\dfrac{(B_i)\dfrac{\vdots}{\langle b, \sigma\rangle \rightsquigarrow \textsf{false}}}{\langle \textsf{while } b \textsf{ do } c, \sigma\rangle \rightsquigarrow \sigma}$$

Equivalence sémantique (10)

• $\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \hookrightarrow \langle c; \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \hookrightarrow \cdots \hookrightarrow \langle \mathsf{skip}, \sigma'\rangle$ avec $\langle b, \sigma\rangle \rightsquigarrow \mathsf{true}$. Par hypothèse d'induction on a $\langle c; \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \rightsquigarrow \sigma'$. D'autre part on montre que $\mathsf{while}\ b\ \mathsf{do}\ c \equiv \mathsf{if}\ b\ \mathsf{then}\ c; \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{else}\ \mathsf{skip}$ et on conclut en construisant l'arbre :

$$(C_4)\ \frac{(B_i)\ \dfrac{\vdots}{\langle b, \sigma\rangle \rightsquigarrow \mathsf{true}} \qquad (C_j)\ \dfrac{\vdots}{\langle c; \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \rightsquigarrow \sigma'}}{\langle \mathsf{if}\ b\ \mathsf{then}\ c; \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{else}\ \mathsf{skip}, \sigma\rangle \rightsquigarrow \sigma'}$$