

# Sémantique opérationnelle

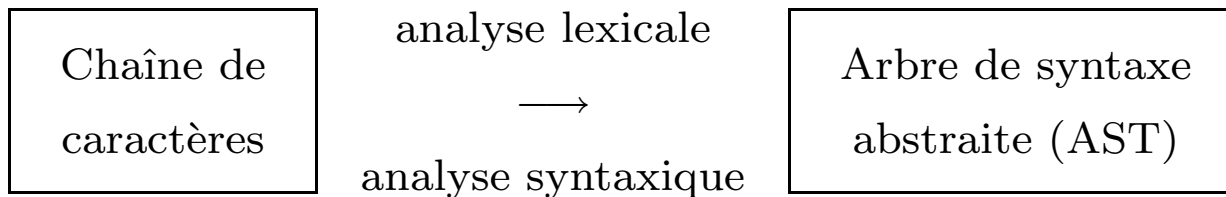
UPMC / CNAM  
Master M2  
Parcours Logiciels Sûrs  
2006/2007

Mathieu Jaume ([Mathieu.Jaume@lip6.fr](mailto:Mathieu.Jaume@lip6.fr))

## Sémantique : Quoi ?

*“Sémantique : Etude du sens des unités linguistiques et de leur composition.”*

[Petit Larousse, 1994]



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
  - ...

## 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 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

## Sémantique : Comment ? (2)

- *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.

cf. cours de V. Donzeau-Gouge

- *sémantique axiomatique* (vision déclarative)

programme = “transformateur” de propriétés (sur les états)

Logique de Hoare cf. cours de M. Simonot

*Sémantique statique* : le *typage* par exemple ... sans avoir recours à l'exécution ... cf. cours d'analyse statique et d'interprétation abstraite

## Systeme Coq (V8)

Assistant à la preuve développé par l'INRIA : <http://coq.inria.fr/>

- “*logical framework*” : calcul des constructions inductives ( $\lambda$ -calcul étendu avec types dépendants, types polymorphes et constructeurs de types)
- permet l'**extraction** de programmes à partir de preuves :

$$\forall x P(x) \Rightarrow \exists y Q(x, y)$$

Mécanisme basé sur l'**isomorphisme de Curry-Howard** : la preuve d'une proposition est un objet fonctionnel.

*Exemple* : une preuve de  $A \Rightarrow B$  est une fonction qui associe une preuve de  $B$  à partir d'une preuve de  $A$ .

- *Développement interactif de preuves formelles*

$$\frac{\text{context } 0}{\text{Goal}} \xrightarrow{\text{apply a tactic}} \frac{\text{context } 1}{\text{Subgoal } 1}, \dots, \frac{\text{context } n}{\text{Subgoal } n}$$

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

*Exemples* : règles du calcul des séquents, règles de typage du  $\lambda$ -calcul

$$\begin{array}{lll}
 (Ax) \frac{}{\Gamma, A \vdash A} & (MP) \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} & (I\Rightarrow) \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \\
 (Vr) \frac{}{\Gamma, x : A \vdash x : A} & (Ap) \frac{\Gamma \vdash t_1 : A \rightarrow B \quad \Gamma \vdash t_2 : A}{\Gamma \vdash (t_1 \ t_2) : B} & (Ab) \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \rightarrow B}
 \end{array}$$

## 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  $j_{k_1}, j_{k_2}, \dots, j_{k_n}$ , 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**  $\text{Th}(\Phi[\mathcal{J}]) \subseteq \mathcal{J}$  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 :  $(R) \frac{}{j}$  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.

## Systemes d'inférence et Définitions inductives

Etant donné un **systeme 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 :

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

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

PREUVE : exercice ...

## Systèmes d'inférence : exemples

$\mathcal{J}_1 = \{\text{suite finie de symboles} \in \{0, S, (, )\}\} = \{0, S(0), S(S(0)), (S, 000S, (S)), \dots\}$

...  $S(S(0)) \in \text{Th}(\Phi_1[\mathcal{J}_1]) = \mathbb{N}$  mais  $0S() \notin \text{Th}(\Phi_1[\mathcal{J}_1])$

$$(N_2) \frac{(N_1) \frac{}{0}}{S(0)}{S(S(0))}$$

$\mathcal{J}_2 = \{n_1 \leq n_2 \mid n_1, n_2 \in \mathbb{N}\} = \{S(S(0)) \leq S(S(S(S(S(0))))), S(S(0)) \leq 0, \dots\}$

...  $S(0) \leq S(S(S(0))) \in \text{Th}(\Phi_2[\mathcal{J}_2])$  mais  $S(S(0)) \leq 0 \notin \text{Th}(\Phi_2[\mathcal{J}_2])$

$$(L_2) \frac{(L_1) \frac{}{S(0) \leq S(0)}}{S(0) \leq S(S(0))}{S(0) \leq S(S(S(0)))}$$

## Définition inductive des expressions arithmétiques

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

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

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

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

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

$n$  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  $\mathbb{Z}$ .

## Expressions Arithmétiques : Syntaxe abstraite – Implantation

$$37 + v \in E_A ? \quad (\mathbb{A}_3) \frac{(\mathbb{A}_1) \overline{37} \quad (\mathbb{A}_2) \overline{v}}{37 + v} \quad \text{Arbre d'inférence} \\ = \text{arbre de syntaxe abstraite}$$

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 :=

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

PREUVE : exercice ...

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

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

## 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)

EA\_rect

```
: forall P : EA -> Type,  
  (forall z : Z, P (itg 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 :*

$$\begin{aligned} & (A_1 \wedge A_2 \wedge \cdots \wedge A_{n-1} \wedge A_n) \Rightarrow B \\ \equiv & (A_1 \Rightarrow (A_2 \Rightarrow \cdots \Rightarrow (A_{n-1} \Rightarrow A_n))) \Rightarrow B \end{aligned}$$

## 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

En COQ : Définition d’un type pour  $\mathbb{V}$

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

## Interprétation des symboles

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

$v_1 \begin{pmatrix} \llbracket + \rrbracket \\ \llbracket - \rrbracket \\ \llbracket \times \rrbracket \end{pmatrix} v_2$	$v_2 \in \mathbb{Z}$	Err
$v_1 \in \mathbb{Z}$	$v_1 \begin{pmatrix} + \\ - \\ \times \end{pmatrix} v_2$	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

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

$$\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) -> let v1 = (eval_arith s e1)  
in v1+(eval_arith s e2)  
| Moins(e1,e2) -> let v1 = (eval_arith s e1)  
in v1 -(eval_arith s e2)  
| Fois(e1,e2) -> let v1 = (eval_arith s e1)  
in v1 * (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
  | itg n => Vitg n
  | var x => Vitg (s x)
  | ...
  | dv a1 a2 => match (eval_EA_fct s a1, eval_EA_fct s a2) with
    | (Vitg n1, Vitg n2) => match n2 with
      | Z0 => Erreur
      | _ => Vitg (Zdiv n1 n2) end
    | _ => 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  $v \in \mathbb{Z} \cup \{\text{Err}\}$  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

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

$$(A_3) \frac{\langle a_1, \sigma \rangle \rightsquigarrow \text{Err}}{\langle a_1 \text{ op}_2 a_2, \sigma \rangle \rightsquigarrow \text{Err}} \quad (\text{op}_2 \in \{+, -, \times, /\})$$

$$(A_4) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow \text{Err}}{\langle a_1 \text{ op}_2 a_2, \sigma \rangle \rightsquigarrow \text{Err}} \quad (n_1, n_2 \in \mathbb{Z})$$

$$(A_5) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 + a_2, \sigma \rangle \rightsquigarrow n_1 + n_2} \quad (A_6) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 \times a_2, \sigma \rangle \rightsquigarrow n_1 \times n_2}$$

$$(A_7) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 - a_2, \sigma \rangle \rightsquigarrow n_1 - n_2} \quad (A_8) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow 0}{\langle a_1 / a_2, \sigma \rangle \rightsquigarrow \text{Err}}$$

$$(A_9) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 / a_2, \sigma \rangle \rightsquigarrow n_1 / n_2} \quad (n_2 \neq 0)$$

## Evaluation : exemple

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

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

$$\begin{array}{c} (A_1) \frac{}{\langle 2, \sigma \rangle \rightsquigarrow 2} \quad (A_1) \frac{}{\langle 3, \sigma \rangle \rightsquigarrow 3} \\ (A_6) \frac{}{\langle 2 \times 3, \sigma \rangle \rightsquigarrow 6} \quad (A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow 2} \\ (A_5) \frac{}{\langle (2 \times 3) + x, \sigma \rangle \rightsquigarrow 8} \end{array}$$

## Evaluation à grands pas : Implantation en Coq

Définition **inductive** d'une **relation**.

```
Inductive eval_EA : Sigma -> EA -> values -> Prop :=
| eval_itg : forall (s : Sigma) (n : Z), eval_EA s (itg n) (Vitg n)
| eval_var : forall (s : Sigma) (v : V), eval_EA s (var v) (Vitg (s v))
| ...
| eval_dv_1 : forall (s : Sigma) (a1 a2 : EA),
    eval_EA s a1 Erreur -> eval_EA s (dv a1 a2) Erreur
| eval_dv_2 : forall (s : Sigma) (a1 a2 : EA) (n : Z),
    eval_EA s a1 (Vitg n) -> eval_EA s a2 Erreur
    -> eval_EA s (dv a1 a2) Erreur
| eval_dv_3 : forall (s : Sigma) (a1 a2 : EA) (n : Z),
    eval_EA s a1 (Vitg n) -> eval_EA s a2 (Vitg Z0)
    -> eval_EA s (dv a1 a2) Erreur
| eval_dv_4 : forall (s : Sigma) (a1 a2 : EA) (n1 n2 : Z),
    eval_EA s a1 (Vitg n1) -> eval_EA s a2 (Vitg n2) -> ~ n2=Z0
    -> eval_EA s (dv a1 a2) (Vitg (Zdiv n1 n2)).
```

`eval_EA : Sigma->EA->values->Prop`

## Existence d'un résultat pour l'évaluation des expressions (1)

**Proposition :**  $\forall a \in E_A \quad \forall \sigma \in \mathcal{V}[\mathbb{Z}] \quad \exists v \in \mathbb{Z} \cup \{\text{Err}\} \quad \langle a, \sigma \rangle \rightsquigarrow v$

Lemma ex\_eval\_EA :

forall (a : EA) (s : Sigma), exists v : values, eval\_EA s a v.

Coq < 1 subgoal

=====

forall (a : EA) (s : Sigma), exists v : values, eval\_EA s a v

PREUVE : Par **induction** sur  $a$ .

induction a.

6 subgoal

...

## Existence d'un résultat pour l'évaluation des expressions (2)

Si  $a = z \in \mathbb{Z}$ , alors  $v$  existe, c'est  $z$  :  $(A_1) \frac{}{\langle z, \sigma \rangle \rightsquigarrow z}$

6 subgoals

$z : Z$

=====

forall  $s : \text{Sigma}$ , exists  $v : \text{values}$ , eval\_EA  $s$  (itg  $z$ )  $v$

intros.

6 subgoals

$z : Z$

$s : \text{Sigma}$

=====

exists  $v : \text{values}$ , eval\_EA  $s$  (itg  $z$ )  $v$

exists (V itg  $z$ ).

## Existence d'un résultat pour l'évaluation des expressions (3)

6 subgoals

`z : Z`

`s : Sigma`

=====

`eval_EA s (itg z) (Vitg z)`

`apply eval_itg.`

`eval_itg : forall (s : Sigma) (n : Z), eval_EA s (itg n) (Vitg n)`

5 subgoals

`v : V`

=====

`forall s : Sigma, exists v0 : values, eval_EA s (var v) v0`

## Existence d'un résultat pour l'évaluation des expressions (4)

Si  $a = x$ , alors  $v$  existe, c'est  $\sigma(x) \in \mathbb{Z} : (A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow \sigma(x)}$

```
intros;exists (Vitg (s v)).
```

5 subgoals

```
v : V
```

```
s : Sigma
```

```
=====
```

```
eval_EA s (var v) (Vitg (s v))
```

```
apply eval_var.
```

```
eval_var:forall (s : Sigma) (v : V), eval_EA s (var v) (Vitg (s v))
```

## Existence d'un résultat pour l'évaluation des expressions (5)

Si  $a = a_1 + a_2$ , alors, par **hypothèse d'induction**, il existe deux valeurs  $v'_1$  et  $v'_2$  telles que  $\langle a_1, \sigma \rangle \rightsquigarrow v'_1$  et  $\langle a_2, \sigma \rangle \rightsquigarrow v'_2$ .

1. Si  $v'_1 = \text{Err}$ , alors  $v$  existe, c'est  $\text{Err}$  :  $(A_3) \frac{\langle a_1, \sigma \rangle \rightsquigarrow \text{Err}}{\langle a_1 + a_2, \sigma \rangle \rightsquigarrow \text{Err}}$

2. Si  $v'_1 \in \mathbb{Z}$  et  $v'_2 = \text{Err}$ , alors  $v$  existe, c'est  $\text{Err}$  :

$$(A_4) \frac{\langle a_1, \sigma \rangle \rightsquigarrow v'_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow \text{Err}}{\langle a_1 + a_2, \sigma \rangle \rightsquigarrow \text{Err}}$$

3. Si  $v'_1, v'_2 \in \mathbb{Z}$ , alors  $v$  existe, c'est  $v'_1 + v'_2$  :

$$(A_5) \frac{\begin{array}{c} \vdots \\ (A_i) \frac{\langle a_1, \sigma \rangle \rightsquigarrow v'_1}{\langle a_1, \sigma \rangle \rightsquigarrow v'_1} \end{array} \quad \begin{array}{c} \vdots \\ (A_j) \frac{\langle a_2, \sigma \rangle \rightsquigarrow v'_2}{\langle a_2, \sigma \rangle \rightsquigarrow v'_2} \end{array}}{\langle a_1 + a_2, \sigma \rangle \rightsquigarrow v'_1 + v'_2}$$

Raisonnement similaire pour  $a = a_1 - a_2$ ,  $a = a_1 \times a_2$  et  $a = a_1/a_2$  (un cas supplémentaire est à considérer lorsque  $a = a_1/a_2$ ).

## Existence d'un résultat pour l'évaluation des expressions (6)

1 subgoal

...

IHa1 : forall s : Sigma, exists v : values, eval\_EA s a1 v

IHa2 : forall s : Sigma, exists v : values, eval\_EA s a2 v

=====

forall s : Sigma, exists v : values, eval\_EA s (dv a1 a2) v

intros.

...

IHa1 : forall s : Sigma, exists v : values, eval\_EA s a1 v

IHa2 : forall s : Sigma, exists v : values, eval\_EA s a2 v

s : Sigma

=====

exists v : values, eval\_EA s (dv a1 a2) v

IHa1 et IHa2 : Hypothèses d'induction sur a1 et a2

elim (IHa1 s).

## Quantificateur existentiel en Coq

```
Inductive ex (A : Type) (P : A -> Prop) : Prop :=  
  ex_intro : forall x : A, P x -> ex P  
  
ex_ind  
  : forall (A : Type) (P : A -> Prop) (P0 : Prop),  
    (forall x : A, P x -> P0) -> ex P -> P0
```

## Existence d'un résultat pour l'évaluation des expressions (7)

...

IHa1 : forall s : Sigma, exists v : values, eval\_EA s a1 v

IHa2 : forall s : Sigma, exists v : values, eval\_EA s a2 v

s : Sigma

=====

forall x : values,

eval\_EA s a1 x -> exists v : values, eval\_EA s (dv a1 a2) v

intros.

...

IHa1 : forall s : Sigma, exists v : values, eval\_EA s a1 v

IHa2 : forall s : Sigma, exists v : values, eval\_EA s a2 v

s : Sigma      x : values      H : eval\_EA s a1 x

=====

exists v : values, eval\_EA s (dv a1 a2) v

elim (IHa2 s);intros.

## Existence d'un résultat pour l'évaluation des expressions (8)

...

x : values            H : eval\_EA s a1 x

x0 : values          H0 : eval\_EA s a2 x0

=====

exists v : values, eval\_EA s (dv a1 a2) v

generalize H.

x : values            H : eval\_EA s a1 x

x0 : values          H0 : eval\_EA s a2 x0

=====

eval\_EA s a1 x -> exists v : values, eval\_EA s (dv a1 a2) v

elim x.

## Existence d'un résultat pour l'évaluation des expressions (9)

2 subgoals

...

x : values            H : eval\_EA s a1 x

x0 : values          H0 : eval\_EA s a2 x0

=====

forall z : Z,

eval\_EA s a1 (Vitg z) -> exists v : values, eval\_EA s (dv a1 a2) v

generalize H0;elim x0.

3 subgoals

...

=====

forall z : Z,

eval\_EA s a2 (Vitg z) ->

forall z0 : Z,

eval\_EA s a1 (Vitg z0) -> exists v : values, eval\_EA s (dv a1 a2) v

intro;case z.

## Existence d'un résultat pour l'évaluation des expressions (10)

```
Inductive Z:Set:= Z0:Z | Zpos:positive -> Z | Zneg:positive -> Z
```

5 subgoals

...

=====

```
eval_EA s a2 (Vitg 0) ->
```

```
forall z0 : Z,
```

```
eval_EA s a1 (Vitg z0) -> exists v : values, eval_EA s (dv a1 a2) v
```

intros.

5 subgoals

...

```
H1 : eval_EA s a2 (Vitg 0)
```

```
z0 : Z
```

```
H2 : eval_EA s a1 (Vitg z0)
```

=====

```
exists v : values, eval_EA s (dv a1 a2) v
```

exists Erreur.

## Existence d'un résultat pour l'évaluation des expressions (11)

...

H1 : eval\_EA s a2 (Vitag 0)

H2 : eval\_EA s a1 (Vitag z0)

=====

eval\_EA s (dv a1 a2) Erreur

apply (eval\_dv\_3 s a1 a2 z0).

eval\_dv\_3: forall (s : Sigma) (a1 a2 : EA) (n : Z)

eval\_EA s a1 (Vitag n)->eval\_EA s a2 (Vitag Z0)->eval\_EA s (dv a1 a2) Erreur

6 subgoals

...

=====

eval\_EA s a1 (Vitag z0)

assumption.

## Existence d'un résultat pour l'évaluation des expressions (12)

5 subgoals

...

H1 : eval\_EA s a2 (Vitg 0)

=====

eval\_EA s a2 (Vitg 0)

assumption.

intros.

4 subgoals

p : positive      H1 : eval\_EA s a2 (Vitg (Zpos p))

z0 : Z            H2 : eval\_EA s a1 (Vitg z0)

=====

exists v : values, eval\_EA s (dv a1 a2) v

exists (Vitg (z0 / Zpos p)).

## Existence d'un résultat pour l'évaluation des expressions (13)

4 subgoals

...

p : positive            H1 : eval\_EA s a2 (Vitg (Zpos p))

z0 : Z                 H2 : eval\_EA s a1 (Vitg z0)

=====

eval\_EA s (dv a1 a2) (Vitg (z0 / Zpos p))

apply eval\_dv\_4.

```
eval_dv_4 : forall (s : Sigma) (a1 a2 : EA) (n1 n2 : Z),
  eval_EA s a1 (Vitg n1) -> eval_EA s a2 (Vitg n2) -> ~ n2 = Z0
  -> eval_EA s (dv a1 a2) (Vitg (Zdiv n1 n2)).
```

assumption. assumption.

4 subgoals

...

=====

Zpos p <> 0%Z

discriminate.

## Existence d'un résultat pour l'évaluation des expressions (14)

3 subgoals

...

=====

```
forall p : positive,  
eval_EA s a2 (Vitg (Zneg p)) ->  
forall z0 : Z,  
eval_EA s a1 (Vitg z0) -> exists v : values, eval_EA s (dv a1 a2) v
```

```
intros. exists (Vitg (z0 / Zneg p)). apply eval_dv_4. assumption.  
assumption. discriminate.
```

2 subgoals

...

=====

```
eval_EA s a2 Erreur ->  
forall z : Z,  
eval_EA s a1 (Vitg z) -> exists v : values, eval_EA s (dv a1 a2) v
```

```
intros.
```

## Existence d'un résultat pour l'évaluation des expressions (15)

2 subgoals

...

H1 : eval\_EA s a2 Erreur

H2 : eval\_EA s a1 (Vitg z)

=====

exists v : values, eval\_EA s (dv a1 a2) v

exists Erreur.

2 subgoals

...

=====

eval\_EA s (dv a1 a2) Erreur

apply (eval\_dv\_2 s a1 a2 z).

```
eval_dv_2 : forall (s : Sigma) (a1 a2 : EA) (n : Z),  
  eval_EA s a1 (Vitg n) -> eval_EA s a2 Erreur ->  
  eval_EA s (dv a1 a2) Erreur
```

assumption. assumption.

## Existence d'un résultat pour l'évaluation des expressions (16)

intros.

1 subgoal

...

x0 : values

H0 : eval\_EA s a2 x0

H1 : eval\_EA s a1 Erreur

=====

exists v : values, eval\_EA s (dv a1 a2) v

exists Erreur. apply eval\_dv\_1.

```
eval_dv_1 : forall (s : Sigma) (a1 a2 : EA),
            eval_EA s a1 Erreur -> eval_EA s (dv a1 a2) Erreur
```

assumption.

Proof completed.

Defined.

ex\_eval\_EA is defined

## Déterminisme de l'évaluation des expressions arithmétiques (1)

**Proposition :**

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

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 (itg z) v1      H0 : eval\_EA s (itg z) v2

=====

v1 = v2

`inversion H.`

6 subgoals

...

H : eval\_EA s (itg z) v1      H0 : eval\_EA s (itg z) v2

s0: Sigma    n: Z    H1: s0 = s    H3: n = z    H2: Vitg z = v1

=====

Vitg z = v2

`inversion H0.`

## Déterminisme de l'évaluation des expressions arithmétiques (3)

6 subgoals

...

```
H : eval_EA s (itg z) v1      H0 : eval_EA s (itg z) v2
s0: Sigma  n: Z  H1: s0 = s   H3: n = z   H2: Vitg z = v1
s1: Sigma  n0: Z H4: s1 = s   H6: n0 = z  H5: Vitg z = v2
```

=====

```
Vitg z = Vitg z
```

`apply refl_equal.`

`Inductive eq (A : Type) (x : A) : A -> Prop := refl_equal : x = x.`

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. apply refl_equal.`

## Déterminisme de l'évaluation des expressions arithmétiques (4)

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

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

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

Puisque, par hypothèse,  $\langle a_1 + a_2, \sigma \rangle \rightsquigarrow v_1$ , trois cas se présentent :

1. Règle  $A_3$  :  $\langle a_1, \sigma \rangle \rightsquigarrow \text{Err}$  et  $v_1 = \text{Err}$
2. Règle  $A_4$  :  $\langle a_1, \sigma \rangle \rightsquigarrow n_1 \in \mathbb{Z}$ ,  $\langle a_2, \sigma \rangle \rightsquigarrow \text{Err}$  et  $v_1 = \text{Err}$
3. Règle  $A_5$  :  $\langle a_1, \sigma \rangle \rightsquigarrow n_1 \in \mathbb{Z}$ ,  $\langle a_2, \sigma \rangle \rightsquigarrow n_2 \in \mathbb{Z}$  et  $v_1 = n_1 + n_2$

## Déterminisme de l'évaluation des expressions arithmétiques (5)

(1). Règle  $A_3$  :  $\langle a_1, \sigma \rangle \rightsquigarrow \text{Err}$  et  $v_1 = \text{Err}$

Puisque, par hypothèse,  $\langle a_1 + a_2, \sigma \rangle \rightsquigarrow v_2$ , trois cas se présentent :

1. Règle  $A_3$  :  $\langle a_1, \sigma \rangle \rightsquigarrow \text{Err}$  et  $v_2 = \text{Err}$  ce qui permet d'obtenir  $v_1 = v_2$
2. Règle  $A_4$  :  $\langle a_1, \sigma \rangle \rightsquigarrow n'_1 \in \mathbb{Z}$ ,  $\langle a_2, \sigma \rangle \rightsquigarrow \text{Err}$  et  $v_2 = \text{Err}$  ce qui permet d'obtenir  $v_1 = v_2$  (même si, par **hypothèse d'induction**, on aurait pu obtenir une contradiction à partir de  $\langle a_1, \sigma \rangle \rightsquigarrow \text{Err}$  et  $\langle a_1, \sigma \rangle \rightsquigarrow n'_1 \in \mathbb{Z}$ )
3. Règle  $A_5$  :  $\langle a_1, \sigma \rangle \rightsquigarrow n'_1 \in \mathbb{Z}$ ,  $\langle a_2, \sigma \rangle \rightsquigarrow n'_2 \in \mathbb{Z}$  et  $v_2 = n'_1 + n'_2$   
Par **hypothèse d'induction** sur  $a_1$ , on a  $\text{Err} = n'_1 \in \mathbb{Z}$  ce qui est contradictoire.

## Déterminisme de l'évaluation des expressions arithmétiques (6)

(2). Règle  $A_4$  :  $\langle a_1, \sigma \rangle \rightsquigarrow n_1 \in \mathbb{Z}$ ,  $\langle a_2, \sigma \rangle \rightsquigarrow \text{Err}$  et  $v_1 = \text{Err}$

Puisque, par hypothèse,  $\langle a_1 + a_2, \sigma \rangle \rightsquigarrow v_2$ , trois cas se présentent :

1. Règle  $A_3$  :  $\langle a_1, \sigma \rangle \rightsquigarrow \text{Err}$  et  $v_2 = \text{Err}$  ce qui permet d'obtenir  $v_1 = v_2$  (même si, par **hypothèse d'induction**, on aurait pu obtenir une contradiction à partir de  $\langle a_1, \sigma \rangle \rightsquigarrow n_1 \in \mathbb{Z}$  et  $\langle a_1, \sigma \rangle \rightsquigarrow \text{Err}$ )
2. Règle  $A_4$  :  $\langle a_1, \sigma \rangle \rightsquigarrow n'_1 \in \mathbb{Z}$ ,  $\langle a_2, \sigma \rangle \rightsquigarrow \text{Err}$  et  $v_2 = \text{Err}$  ce qui permet d'obtenir  $v_1 = v_2$
3. Règle  $A_5$  :  $\langle a_1, \sigma \rangle \rightsquigarrow n'_1 \in \mathbb{Z}$ ,  $\langle a_2, \sigma \rangle \rightsquigarrow n'_2 \in \mathbb{Z}$  et  $v_2 = n'_1 + n'_2$   
Par **hypothèse d'induction** sur  $a_2$ , on a  $\text{Err} = n'_2 \in \mathbb{Z}$  ce qui est contradictoire.

## Déterminisme de l'évaluation des expressions arithmétiques (7)

(3). Règle  $A_5$  :  $\langle a_1, \sigma \rangle \rightsquigarrow n_1 \in \mathbb{Z}$ ,  $\langle a_2, \sigma \rangle \rightsquigarrow n_2 \in \mathbb{Z}$  et  $v_1 = n_1 + n_2$

Puisque, par hypothèse,  $\langle a_1 + a_2, \sigma \rangle \rightsquigarrow v_2$ , trois cas se présentent :

1. Règle  $A_3$  :  $\langle a_1, \sigma \rangle \rightsquigarrow \text{Err}$  et  $v_2 = \text{Err}$

Par **hypothèse d'induction** sur  $a_1$ , on a  $\text{Err} = n_1 \in \mathbb{Z}$  ce qui est contradictoire.

2. Règle  $A_4$  :  $\langle a_1, \sigma \rangle \rightsquigarrow n'_1 \in \mathbb{Z}$ ,  $\langle a_2, \sigma \rangle \rightsquigarrow \text{Err}$  et  $v_2 = \text{Err}$

Par **hypothèse d'induction** sur  $a_2$ , on a  $\text{Err} = n_2 \in \mathbb{Z}$  ce qui est contradictoire.

3. Règle  $A_5$  :  $\langle a_1, \sigma \rangle \rightsquigarrow n'_1 \in \mathbb{Z}$ ,  $\langle a_2, \sigma \rangle \rightsquigarrow n'_2 \in \mathbb{Z}$  et  $v_2 = n'_1 + n'_2$

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

Raisonnement similaire pour  $a = a_1 - a_2$ ,  $a = a_1 \times a_2$  et  $a = a_1/a_2$  (un cas supplémentaire est à considérer lorsque  $a = a_1/a_2$ ).

## Déterminisme de l'évaluation des expressions arithmétiques (8)

intros.

4 subgoals

```
IHa1:forall (s:Sigma)(v1 v2:values),eval_EA s a1 v1->eval_EA s a1 v2->v1=v2
IHa2:forall (s:Sigma)(v1 v2:values),eval_EA s a2 v1->eval_EA s a2 v2->v1=v2
H:eval_EA s (pls a1 a2) v1      H0:eval_EA s (pls a1 a2) v2
```

=====

v1 = v2

inversion H.

6 subgoals

```
H:eval_EA s (pls a1 a2) v1      H0:eval_EA s (pls a1 a2) v2
s0:Sigma   a0:EA   a3:EA      H5:eval_EA s a1 Erreur
H2:s0=s    H1:a0=a1 H4:a3=a2  H3:Erreur=v1
```

=====

Erreur = v2

inversion H0.

## Déterminisme de l'évaluation des expressions arithmétiques (9)

8 subgoals

...

=====

Erreur = Erreur

`apply refl_equal.`

7 subgoals

...

H5 : eval\_EA s a1 Erreur

H9 : eval\_EA s a1 (Vitg n)

H11 : eval\_EA s a2 Erreur

=====

Erreur = Erreur

`apply refl_equal.`

## Déterminisme de l'évaluation des expressions arithmétiques (10)

6 subgoals

...

IHa1:forall (s:Sigma)(v1 v2:values),eval\_EA s a1 v1->eval\_EA s a1 v2->v1=v2

IHa2:forall (s:Sigma)(v1 v2:values),eval\_EA s a2 v1->eval\_EA s a2 v2->v1=v2

H5 : eval\_EA s a1 Erreur                      H9 : eval\_EA s a1 (Vitg n1)

H11 : eval\_EA s a2 (Vitg n2)                H10 : Vitg (n1 + n2) = v2

=====

Erreur = Vitg (n1 + n2)

absurd ((Vitg n1)=Erreur).

$$\frac{\text{Context0} \text{ absurd } P}{\text{Goal0}} \longrightarrow \frac{\text{Context0}}{\neg P}, \frac{\text{Context0}}{P}$$

Dans COQ (et plus généralement en logique intuitionniste),

$\neg A \equiv A \Rightarrow \text{false}$

## Déterminisme de l'évaluation des expressions arithmétiques (11)

7 subgoals ...

=====

Vitg n1 <> Erreur

unfold not.

7 subgoals ...

=====

Vitg n1 = Erreur -> False

intro faux.

7 subgoals ...

faux : (Vitg n1)=Erreur

=====

false

discriminate faux.

## Déterminisme de l'évaluation des expressions arithmétiques (12)

6 subgoals

...

IHa1:forall (s:Sigma)(v1 v2:values),eval\_EA s a1 v1->eval\_EA s a1 v2->v1=v2

IHa2:forall (s:Sigma)(v1 v2:values),eval\_EA s a2 v1->eval\_EA s a2 v2->v1=v2

H5:eval\_EA s a1 Erreur      H9:eval\_EA s a1 (Vitg n1)

H11:eval\_EA s a2 (Vitg n2)    H10:Vitg (n1 + n2) = v2

=====

Vitg n1 = Erreur

apply (IHa1 s).

assumption.

assumption.

## Déterminisme de l'évaluation des expressions arithmétiques (13)

5 subgoals

...

IHa1:forall (s:Sigma)(v1 v2:values),eval\_EA s a1 v1->eval\_EA s a1 v2->v1=v2

IHa2:forall (s:Sigma)(v1 v2:values),eval\_EA s a2 v1->eval\_EA s a2 v2->v1=v2

H:eval\_EA s (pls a1 a2) v1 H0:eval\_EA s (pls a1 a2) v2

H4:eval\_EA s a1 (Vitg n) H6:eval\_EA s a2 Erreur

=====

Erreur = v2

`inversion H0.`

7 subgoals ...

=====

Erreur=Erreur

subgoal 2 is:

Erreur=Erreur

`apply refl_equal. apply refl_equal.`

## Déterminisme de l'évaluation des expressions arithmétiques (14)

5 subgoals

```
...
IHa1:forall (s:Sigma)(v1 v2:values),eval_EA s a1 v1->eval_EA s a1 v2->v1=v2
IHa2:forall (s:Sigma)(v1 v2:values),eval_EA s a2 v1->eval_EA s a2 v2->v1=v2
H4:eval_EA s a1 (Vitg n)      H6:eval_EA s a2 Erreur
H10:eval_EA s a1 (Vitg n1)   H12:eval_EA s a2 (Vitg n2)
=====
Erreur = Vitg (n1 + n2)

absurd ((Vitg n2)=Erreur).
unfold not;intro faux;discriminate faux.
apply (IHa2 s).
assumption.
assumption.
```

## Déterminisme de l'évaluation des expressions arithmétiques (15)

4 subgoals

...

IHa1:forall (s:Sigma)(v1 v2:values),eval\_EA s a1 v1->eval\_EA s a1 v2->v1=v2

IHa2:forall (s:Sigma)(v1 v2:values),eval\_EA s a2 v1->eval\_EA s a2 v2->v1=v2

H:eval\_EA s (pls a1 a2) v1      H0:eval\_EA s (pls a1 a2) v2

H4:eval\_EA s a1 (Vitg n1)      H6:eval\_EA s a2 (Vitg n2)

H5 : Vitg (n1 + n2) = v1

=====

Vitg (n1 + n2) = v2

*inversion H0.*

## Déterminisme de l'évaluation des expressions arithmétiques (16)

6 subgoals

```
...
IHa1:forall (s:Sigma)(v1 v2:values),eval_EA s a1 v1->eval_EA s a1 v2->v1=v2
IHa2:forall (s:Sigma)(v1 v2:values),eval_EA s a2 v1->eval_EA s a2 v2->v1=v2
H4:eval_EA s a1 (Vitg n1)      H6:eval_EA s a2 (Vitg n2)
H11:eval_EA s a1 Erreur
=====
  Vitg (n1 + n2) = Erreur

absurd ((Vitg n1)=Erreur).
unfold not;intro faux;discriminate faux.
apply (IHa1 s).
assumption.
assumption.
```

## Déterminisme de l'évaluation des expressions arithmétiques (17)

5 subgoals

```
...
IHa1:forall (s:Sigma)(v1 v2:values),eval_EA s a1 v1->eval_EA s a1 v2->v1=v2
IHa2:forall (s:Sigma)(v1 v2:values),eval_EA s a2 v1->eval_EA s a2 v2->v1=v2
H4:eval_EA s a1 (Vitg n1)      H6:eval_EA s a2 (Vitg n2)
H10:eval_EA s a1 (Vitg n)     H12:eval_EA s a2 Erreur
=====
Vitg (n1 + n2) = Erreur

absurd ((Vitg n2)=Erreur).
unfold not;intro faux;discriminate faux.
apply (IHa2 s).
assumption.
assumption.
```

## Déterminisme de l'évaluation des expressions arithmétiques (18)

4 subgoals

...

IHa1:forall (s:Sigma)(v1 v2:values),eval\_EA s a1 v1->eval\_EA s a1 v2->v1=v2

IHa2:forall (s:Sigma)(v1 v2:values),eval\_EA s a2 v1->eval\_EA s a2 v2->v1=v2

H4:eval\_EA s a1 (Vitg n1)      H6:eval\_EA s a2 (Vitg n2)

H10:eval\_EA s a1 (Vitg n0)    H12:eval\_EA s a2 (Vitg n3)

=====

Vitg (n1 + n2) = Vitg (n0 + n3)

generalize (IHa1 s (Vitg n1) (Vitg n0) H4 H10).

4 subgoals

...

=====

Vitg n1 = Vitg n0 -> Vitg (n1 + n2) = Vitg (n0 + n3)

intro.

generalize (IHa2 s (Vitg n2) (Vitg n3) H6 H12);intro.

## Déterminisme de l'évaluation des expressions arithmétiques (19)

4 subgoals ...

H13:Vitg n1 = Vitg n0            H14:Vitg n2 = Vitg n3

=====

Vitg (n1 + n2) = Vitg (n0 + n3)

*inversion H13.*

4 subgoals ...

=====

Vitg (n0 + n2) = Vitg (n0 + n3)

*inversion H14.*

4 subgoals ...

=====

Vitg (n0 + n3) = Vitg (n0 + n3)

*apply refl\_equal.*

## Equivalence sémantique (1)

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

**Lemma eval\_EA\_fct\_rel** : forall (a:EA) (s:Sigma) (v:values),  
(eval\_EA s a v) <-> (eval\_EA\_fct s a) = v.

1 subgoal

=====

forall (a : EA) (s : Sigma) (v : values),  
eval\_EA s a v <-> eval\_EA\_fct s a = v

induction a.

6 subgoals

z : Z

=====

forall (s : Sigma) (v : values),  
eval\_EA s (itg z) v <-> eval\_EA\_fct s (itg z) = v

simpl.

## Equivalence sémantique (2)

6 subgoals

z : Z

=====

forall (s : Sigma) (v : values),

eval\_EA s (itg z) v <-> eval\_EA\_fct s (itg z) = v

intros.

6 subgoals

z : Z

s : Sigma

v : values

=====

eval\_EA s (itg z) v <-> Vitg z = v

split.

## Equivalence sémantique (3)

7 subgoals

...

=====

$\text{eval\_EA } s \text{ (itg } z) v \rightarrow \text{Vitg } z = v$

subgoal 2 is:

$\text{Vitg } z = v \rightarrow \text{eval\_EA } s \text{ (itg } z) v$

*intro.*

7 subgoals

$z : Z \quad s : \text{Sigma} \quad v : \text{values} \quad H : \text{eval\_EA } s \text{ (itg } z) v$

=====

$\text{Vitg } z = v$

*inversion H.*

## Equivalence sémantique (4)

7 subgoals

...

H : eval\_EA s (itg z) v

H1 : Vitg z = v

=====

Vitg z = Vitg z

apply refl\_equal.

## Equivalence sémantique (5)

intro.

6 subgoals

...

H : Vitg z = v

=====

eval\_EA s (itg z) v

rewrite <- H.

6 subgoals

...

H : Vitg z = v

=====

eval\_EA s (itg z) (Vitg z)

apply eval\_itg.

eval\_itg : forall (s : Sigma) (n : Z), eval\_EA s (itg n) (Vitg n)

## Equivalence sémantique (6)

```
forall (s : Sigma) (v0 : values),  
eval_EA s (var v) v0 <-> eval_EA_fct s (var v) = v0
```

4 subgoals

...

```
IHa1:forall (s:Sigma)(v:values),eval_EA s a1 v<->eval_EA_fct s a1 = v
```

```
IHa2:forall (s:Sigma)(v:values),eval_EA s a2 v<->eval_EA_fct s a2 = v
```

```
=====
```

```
forall (s : Sigma) (v : values),
```

```
eval_EA s (pls a1 a2) v <-> eval_EA_fct s (pls a1 a2) = v
```

```
intros;split;intros.
```

## Equivalence sémantique (7)

5 subgoals

...

IHa1:forall (s:Sigma)(v:values),eval\_EA s a1 v<->eval\_EA\_fct s a1 = v

IHa2:forall (s:Sigma)(v:values),eval\_EA s a2 v<->eval\_EA\_fct s a2 = v

H : eval\_EA s (pls a1 a2) v

=====

eval\_EA\_fct s (pls a1 a2) = v

IHa1 (resp. IHa2) : **Hypothèse d'induction** sur a1 (resp. a2).

**inversion H.**

## Equivalence sémantique (8)

7 subgoals

...

IHa1:forall (s:Sigma)(v:values),eval\_EA s a1 v<->eval\_EA\_fct s a1 = v

IHa2:forall (s:Sigma)(v:values),eval\_EA s a2 v<->eval\_EA\_fct s a2 = v

H:eval\_EA s (pls a1 a2) v    H4:eval\_EA s a1 Erreur    H2:Erreur=v

=====

eval\_EA\_fct s (pls a1 a2) = Erreur

elim (IHa1 s Erreur);intros.

## Equivalence sémantique (9)

7 subgoals

...

H5 : `eval_EA s a1 Erreur -> eval_EA_fct s a1 = Erreur`

H6 : `eval_EA_fct s a1 = Erreur -> eval_EA s a1 Erreur`

=====

`eval_EA_fct s (pls a1 a2) = Erreur`

`simpl.`

## Equivalence sémantique (10)

7 subgoals

...

H5 : eval\_EA s a1 Erreur -> eval\_EA\_fct s a1 = Erreur

=====

match eval\_EA\_fct s a1 with

| Vitg n1 =>

    match eval\_EA\_fct s a2 with

    | Vitg n2 => Vitg (n1 + n2)

    | Erreur => Erreur

    end

| Erreur => Erreur

end = Erreur

rewrite H5.

## Equivalence sémantique (11)

8 subgoals

...

=====

Erreur = Erreur

subgoal 2 is:

eval\_EA s a1 Erreur

`apply refl_equal.`

7 subgoals

...

H4 : eval\_EA s a1 Erreur

=====

eval\_EA s a1 Erreur

`assumption.`

## Equivalence sémantique (12)

6 subgoals

...

IHa1:forall (s:Sigma)(v:values),eval\_EA s a1 v<->eval\_EA\_fct s a1 = v

IHa2:forall (s:Sigma)(v:values),eval\_EA s a2 v<->eval\_EA\_fct s a2 = v

H3 : eval\_EA s a1 (Vitg n)                    H5 : eval\_EA s a2 Erreur

=====

eval\_EA\_fct s (pls a1 a2) = Erreur

simpl.

6 subgoals

...

=====

```
match eval_EA_fct s a1 with | Vitg n1 => match eval_EA_fct s a2 with |
      Vitg n2 => Vitg (n1 + n2)|Erreur => Erreur end
| Erreur => Erreur end = Erreur
```

elim (IHa1 s (Vitg n));intros;elim (IHa2 s Erreur);intros.

## Equivalence sémantique (13)

6 subgoals

...

H6 : eval\_EA s a1 (Vitg n) -> eval\_EA\_fct s a1 = Vitg n

H7 : eval\_EA\_fct s a1 = Vitg n -> eval\_EA s a1 (Vitg n)

H8 : eval\_EA s a2 Erreur -> eval\_EA\_fct s a2 = Erreur

H9 : eval\_EA\_fct s a2 = Erreur -> eval\_EA s a2 Erreur

=====

```
match eval_EA_fct s a1 with | Vitg n1 => match eval_EA_fct s a2 with |
      Vitg n2 => Vitg (n1 + n2)|Erreur => Erreur end
| Erreur => Erreur end = Erreur
```

rewrite H6.

## Equivalence sémantique (14)

7 subgoals

...

H6 : eval\_EA s a1 (Vitg n) -> eval\_EA\_fct s a1 = Vitg n

H8 : eval\_EA s a2 Erreur -> eval\_EA\_fct s a2 = Erreur

=====

match eval\_EA\_fct s a2 with

| Vitg n2 => Vitg (n + n2) | Erreur => Erreur

end = Erreur

rewrite H8.

## Equivalence sémantique (15)

8 subgoals

...

H3 : eval\_EA s a1 (Vitg n)

H5 : eval\_EA s a2 Erreur

H6 : eval\_EA s a1 (Vitg n) -> eval\_EA\_fct s a1 = Vitg n

H8 : eval\_EA s a2 Erreur -> eval\_EA\_fct s a2 = Erreur

=====

Erreur = Erreur

subgoal 2 is:

eval\_EA s a2 Erreur

subgoal 3 is:

eval\_EA s a1 (Vitg n)

apply refl\_equal.

assumption.

assumption.

## Equivalence sémantique (16)

5 subgoals

...

IHa1:forall (s:Sigma)(v:values),eval\_EA s a1 v<->eval\_EA\_fct s a1 = v

IHa2:forall (s:Sigma)(v:values),eval\_EA s a2 v<->eval\_EA\_fct s a2 = v

H3 : eval\_EA s a1 (Vitg n1)      H5 : eval\_EA s a2 (Vitg n2)

=====

eval\_EA\_fct s (pls a1 a2) = Vitg (n1 + n2)

`elim (IHa1 s (Vitg n1));intros;elim (IHa2 s (Vitg n2));intros.`

5 subgoals

...

H6 : eval\_EA s a1 (Vitg n1) -> eval\_EA\_fct s a1 = Vitg n1

H8 : eval\_EA s a2 (Vitg n2) -> eval\_EA\_fct s a2 = Vitg n2

=====

eval\_EA\_fct s (pls a1 a2) = Vitg (n1 + n2)

`simpl.`

## Equivalence sémantique (17)

5 subgoals

...

H3 : eval\_EA s a1 (Vitg n1)

H5 : eval\_EA s a2 (Vitg n2)

H6 : eval\_EA s a1 (Vitg n1) -> eval\_EA\_fct s a1 = Vitg n1

H8 : eval\_EA s a2 (Vitg n2) -> eval\_EA\_fct s a2 = Vitg n2

=====

match eval\_EA\_fct s a1 with

| Vitg n0 =>

    match eval\_EA\_fct s a2 with

    | Vitg n3 => Vitg (n0 + n3)

    | Erreur => Erreur

    end

| Erreur => Erreur

end = Vitg (n1 + n2)

rewrite H6. rewrite H8. apply refl\_equal. assumption. assumption.

## Equivalence sémantique (18)

4 subgoals

...

H : eval\_EA\_fct s (pls a1 a2) = v

=====

eval\_EA s (pls a1 a2) v

rewrite <- H;simpl.

4 subgoals

...

=====

eval\_EA s (pls a1 a2)

match eval\_EA\_fct s a1 with

| Vitg n1 =>

match eval\_EA\_fct s a2 with

| Vitg n2 => Vitg (n1 + n2)

| Erreur => Erreur end

| Erreur => Erreur end

## Equivalence sémantique (19)

```
elim (ex_eval_EA a1 s).
```

```
4 subgoals
```

```
...
```

```
=====
```

```
forall x : values,  
eval_EA s a1 x ->  
eval_EA s (pls a1 a2)    match eval_EA_fct s a1 with  
| Vitg n1 => match eval_EA_fct s a2 with  
| Vitg n2 => Vitg (n1 + n2)  
| Erreur => Erreur end  
| Erreur => Erreur  
end
```

```
intro ve.
```

```
4 subgoals ...
```

```
ve : values
```

```
=====
```

```
(eval_EA s a1 ve) -> ...
```

## Equivalence sémantique (20)

```
elim (ex_eval_EA a2 s);intro ve0.
```

4 subgoals

...

```
IHa1:forall (s:Sigma)(v:values),eval_EA s a1 v<->eval_EA_fct s a1 = v
```

```
IHa2:forall (s:Sigma)(v:values),eval_EA s a2 v<->eval_EA_fct s a2 = v
```

```
ve : values    ve0 : values
```

```
=====
```

```
eval_EA s a2 ve0 -> eval_EA s a1 ve ->
```

```
eval_EA s (pls a1 a2)
```

```
  match eval_EA_fct s a1 with
```

```
  | Vitg n1 => match eval_EA_fct s a2 with
```

```
    | Vitg n2 => Vitg (n1 + n2)
```

```
    | Erreur => Erreur  end
```

```
  | Erreur => Erreur  end
```

```
elim (IHa1 s ve).  elim (IHa2 s ve0).
```

## Equivalence sémantique (21)

4 subgoals

...

ve : values            ve0 : values

=====

(eval\_EA s a2 ve0 -> eval\_EA\_fct s a2 = ve0) ->

(eval\_EA\_fct s a2 = ve0 -> eval\_EA s a2 ve0) ->

(eval\_EA s a1 ve -> eval\_EA\_fct s a1 = ve) ->

(eval\_EA\_fct s a1 = ve -> eval\_EA s a1 ve) ->

eval\_EA s a2 ve0 ->

eval\_EA s a1 ve ->

eval\_EA s (pls a1 a2)

  match eval\_EA\_fct s a1 with

  | Vitg n1 => match eval\_EA\_fct s a2 with

    | Vitg n2 => Vitg (n1 + n2)

    | Erreur => Erreur end

  | Erreur => Erreur end

elim ve.

## Equivalence sémantique (22)

5 subgoals

...

ve : values    ve0 : values

=====

```
forall z : Z, (eval_EA s a2 ve0 -> eval_EA_fct s a2 = ve0) ->
(eval_EA_fct s a2 = ve0 -> eval_EA s a2 ve0) ->
(eval_EA s a1 (Vitg z) -> eval_EA_fct s a1 = Vitg z) ->
(eval_EA_fct s a1 = Vitg z -> eval_EA s a1 (Vitg z)) ->
eval_EA s a2 ve0 ->
eval_EA s a1 (Vitg z) ->
eval_EA s (pls a1 a2)
  match eval_EA_fct s a1 with
  | Vitg n1 => match eval_EA_fct s a2 with
    | Vitg n2 => Vitg (n1 + n2)
    | Erreur => Erreur end
  | Erreur => Erreur end
```

elim ve0.

## Equivalence sémantique (23)

6 subgoals

...

=====

```
forall z z0:Z, (eval_EA s a2 (Vitg z) -> eval_EA_fct s a2 = Vitg z) ->
(eval_EA_fct s a2 = Vitg z -> eval_EA s a2 (Vitg z)) ->
(eval_EA s a1 (Vitg z0) -> eval_EA_fct s a1 = Vitg z0) ->
(eval_EA_fct s a1 = Vitg z0 -> eval_EA s a1 (Vitg z0)) ->
eval_EA s a2 (Vitg z) ->
eval_EA s a1 (Vitg z0) ->
eval_EA s (pls a1 a2)
  match eval_EA_fct s a1 with
  | Vitg n1 => match eval_EA_fct s a2 with
    | Vitg n2 => Vitg (n1 + n2)
    | Erreur => Erreur end
  | Erreur => Erreur end
```

intros.

## Equivalence sémantique (24)

6 subgoals

...

H : eval\_EA\_fct s (pls a1 a2) = v

H0 : eval\_EA s a2 (Vitg z) -> eval\_EA\_fct s a2 = Vitg z

H2 : eval\_EA s a1 (Vitg z0) -> eval\_EA\_fct s a1 = Vitg z0

H4 : eval\_EA s a2 (Vitg z)      H5 : eval\_EA s a1 (Vitg z0)

=====

eval\_EA s (pls a1 a2)

  match eval\_EA\_fct s a1 with

  | Vitg n1 => match eval\_EA\_fct s a2 with

    | Vitg n2 => Vitg (n1 + n2)

    | Erreur => Erreur end

  | Erreur => Erreur end

rewrite (H0 H4). rewrite (H2 H5).

## Equivalence sémantique (25)

6 subgoals

...

H4 : eval\_EA s a2 (Vitg z)    H5 : eval\_EA s a1 (Vitg z0)

=====

eval\_EA s (pls a1 a2) (Vitg (z0 + z))

`apply eval_pls_3.`

```
eval_pls_3 :forall (s : Sigma) (a1 a2 : EA) (n1 n2 : Z),  
  eval_EA s a1 (Vitg n1) -> eval_EA s a2 (Vitg n2) ->  
  eval_EA s (pls a1 a2) (Vitg (Zplus n1 n2))
```

`assumption. assumption.`

## Equivalence sémantique (26)

intros.

5 subgoals

H0 : eval\_EA s a2 Erreur -> eval\_EA\_fct s a2 = Erreur

H2 : eval\_EA s a1 (Vitg z) -> eval\_EA\_fct s a1 = Vitg z

H4 : eval\_EA s a2 Erreur      H5 : eval\_EA s a1 (Vitg z)

=====

eval\_EA s (pls a1 a2)

  match eval\_EA\_fct s a1 with

  | Vitg n1 => match eval\_EA\_fct s a2 with

    | Vitg n2 => Vitg (n1 + n2)

    | Erreur => Erreur end

  | Erreur => Erreur end

rewrite (H0 H4). rewrite (H2 H5). apply (eval\_pls\_2 s a1 a2 z).

eval\_pls\_2:forall (s:Sigma)(a1 a2:EA)(n:Z) eval\_EA s a1 (Vitg n)

-> eval\_EA s a2 Erreur -> eval\_EA s (pls a1 a2) Erreur

assumption. assumption.

## Equivalence sémantique (27)

intros.

4 subgoals ...

H2 : eval\_EA s a1 Erreur -> eval\_EA\_fct s a1 = Erreur

H5 : eval\_EA s a1 Erreur

=====

eval\_EA s (pls a1 a2)

match eval\_EA\_fct s a1 with

| Vitg n1 => match eval\_EA\_fct s a2 with

| Vitg n2 => Vitg (n1 + n2)

| Erreur => Erreur end

| Erreur => Erreur end

rewrite (H2 H5). apply eval\_pls\_1.

eval\_pls\_1 : forall (s : Sigma) (a1 a2 : EA),

eval\_EA s a1 Erreur -> eval\_EA s (pls a1 a2) Erreur

assumption.

## Evaluation des expressions arithmétiques : variables (1)

Le résultat de l'évaluation d'une expression arithmétique  $a$ , étant donnée une valuation  $\sigma$ , ne dépend que de la valeur associée par  $\sigma$  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 :**

$$\forall a \in E_A, \forall \sigma_1, \sigma_2 \in \mathcal{V}[\mathbb{Z}],$$

$$\sigma_1 =_{\vartheta(a)} \sigma_2 \Rightarrow (\forall v \in \mathbb{V} \cup \{\text{Err}\} \langle a, \sigma_1 \rangle \rightsquigarrow v \Leftrightarrow \langle a, \sigma_2 \rangle \rightsquigarrow v)$$

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

*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 :

1. à partir de la règle  $A_5$  :

$$(A_5) \frac{(A_i) \frac{\vdots}{\langle a_1, \sigma_1 \rangle \rightsquigarrow n_1} \quad (A_j) \frac{\vdots}{\langle a_2, \sigma_1 \rangle \rightsquigarrow n_2}}{\langle a_1 + a_2, \sigma_1 \rangle \rightsquigarrow v}$$

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  $\langle a_1 + a_2, \sigma_2 \rangle \rightsquigarrow v$ .

## Evaluation des expressions arithmétiques : variables (4)

2. à partir de la règle  $A_3$  :

$$(A_3) \frac{(A_i) \frac{\vdots}{\langle a_1, \sigma_1 \rangle \rightsquigarrow \text{Err}}}{\langle a_1 + a_2, \sigma_1 \rangle \rightsquigarrow \text{Err}}$$

Par **hypothèse d'induction**, puisque  $\vartheta(a_1) \subseteq \vartheta(a)$  (et donc  $\sigma_1 =_{\vartheta(a_1)} \sigma_2$ ) on a  $\langle a_1, \sigma_2 \rangle \rightsquigarrow \text{Err}$ , et en appliquant la règle  $A_3$  on a  $\langle a_1 + a_2, \sigma_2 \rangle \rightsquigarrow \text{Err}$ .

3. à partir de la règle  $A_4$  ... raisonnement similaire au cas précédent.

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 (un cas supplémentaire est à envisager pour la division).

## Expressions arithmétiques équivalentes (1)

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

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

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

Une **relation d'équivalence**  $\mathcal{R}$  sur  $E \times E$  est une relation :

- **réflexive**  $\forall x \in E \quad x \mathcal{R} x$
- **symétrique**  $\forall x, y \in E, x \mathcal{R} y$  implique  $y \mathcal{R} x$
- **transitive**  $\forall x, y, z \in E$ , si  $x \mathcal{R} y$  et  $y \mathcal{R} z$ , alors  $x \mathcal{R} z$

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 :

$$(A_5) \frac{(A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow \sigma(x)} \quad (A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow \sigma(x)}}{\langle x + x, \sigma \rangle \rightsquigarrow \sigma(x) + \sigma(x)}$$

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

$$(A_6) \frac{(A_1) \frac{}{\langle 2, \sigma \rangle \rightsquigarrow 2} \quad (A_2) \frac{}{\langle x, \sigma \rangle \rightsquigarrow \sigma(x)}}{\langle 2 \times x, \sigma \rangle \rightsquigarrow 2 \times \sigma(x)}$$

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  $\langle x + x, \sigma \rangle \rightsquigarrow n$ .

Si  $\langle x + x, \sigma \rangle \rightsquigarrow \text{Err}$ , alors ...

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

De plus, puisque  $\sim$  est une congruence, on a

$$(x + x) + (x + x) \sim (2 \times x) + (2 \times x).$$

## Sémantique opérationnelle d'évaluation à petits pas (1)

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  $\sigma$  conduit à évaluer une expression  $e'$  “plus simple” à partir de la même valuation.

*Exemples*

$$\langle 3 + (2 \times 8), \sigma \rangle \hookrightarrow \langle 3 + 16, \sigma \rangle$$
$$\langle 2/0, \sigma \rangle \hookrightarrow \langle \text{Err}, \sigma \rangle$$

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

$E_A^+$  : ensemble défini inductivement à partir des règles définissant  $E_A$  et de la règle :

$$(R_{\text{Err}}) \overline{\text{Err}}$$

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

## 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  $\hookrightarrow$ .

$\langle e, \sigma \rangle \xrightarrow{*} \langle e', \sigma \rangle$  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  $e_k = v \in \mathbb{V}$  (configuration terminale), alors  $\langle e, \sigma \rangle \xrightarrow{*} v$ .

## Sémantique opérationnelle d'évaluation à petits pas (3)

Obtention d'une configuration terminale

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

$$(AS_2) \frac{}{\langle \text{Err op}_2 e, \sigma \rangle \hookrightarrow \langle \text{Err}, \sigma \rangle} \text{ (si } \text{op}_2 \in \{+, -, \times, /\})$$

$$(AS_3) \frac{}{\langle n \text{ op}_2 \text{Err}, \sigma \rangle \hookrightarrow \langle \text{Err}, \sigma \rangle} \text{ (si } \text{op}_2 \in \{+, -, \times, /\} \text{ et } n \in \mathbb{Z})$$

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

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

$$(AS_6) \frac{}{\langle n_1/n_2, \sigma \rangle \hookrightarrow \langle \text{Err}, \sigma \rangle} \text{ (si } n_1 \in \mathbb{Z} \text{ et } n_2 = 0)$$

## Sémantique opérationnelle d'évaluation à petits pas (4)

Evaluation “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

$\sigma$  : 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

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

## Terminaison (1)

**Proposition** Toutes les séquences de calcul décrivant l'évaluation d'une expression arithmétique sont finies.

IDÉE DE LA PREUVE : A chaque étape de calcul, on obtient une expression “plus simple” ... on obtient donc une configuration terminale en temps fini.

- définir formellement la relation d'ordre “est une expression plus simple”  
ordre lexicographique
- montrer qu'il n'existe pas de suites infinies strictement décroissantes pour la relation d'ordre “est une expression plus simple”  
ordre bien fondé
- montrer que chaque étape de calcul conduit à une expression “plus simple”  
induction sur un arbre d'inférence

## Terminaison (2)

Définition de la relation “est une expression plus simple”.

A chaque étape de calcul, soit le nombre de variables diminue strictement, soit le nombre d'opérateurs diminue strictement.

$e \in E_A^+$  : expression

- $\mathcal{N}_V(e) \in \mathbb{N}$  : nombre d'occurrences de variables dans  $e$
- $\mathcal{N}_O(e) \in \mathbb{N}$  : nombre d'opérateurs apparaissant dans  $e$

Comment définir une relation d'ordre pour laquelle il n'existe pas de suites infinies strictement décroissantes à partir de  $\mathcal{N}_V(e) \in \mathbb{N}$  et  $\mathcal{N}_O(e) \in \mathbb{N}$  ?

... ordre lexicographique sur  $\mathbb{N} \times \mathbb{N}$  obtenu à partir de l'ordre bien fondé  $\leq$  sur  $\mathbb{N}$

## Ordres

Une *relation de préordre*  $\mathcal{R}$  sur  $E \times E$  est une relation :

- *réflexive*  $\forall x \in E \quad x \mathcal{R} x$
- *transitive*  $\forall x, y, z \in E$ , si  $x \mathcal{R} y$  et  $y \mathcal{R} z$ , alors  $x \mathcal{R} z$

Une *relation d'ordre*  $\mathcal{R}$  sur  $E \times E$  est un *préordre* :

- *antisymétrique*  $\forall x, y \in E$ , si  $x \mathcal{R} y$  et  $y \mathcal{R} x$ , alors  $x = y$

*Exemple* :  $\leq$  est une relation d'ordre sur  $\mathbb{N}$ .

Une relation d'ordre  $\preceq$  sur  $E \times E$  est *totale* si deux éléments quelconques de  $E$  sont en relation par cet ordre :  $\forall e_1, e_2 \in E \quad e_1 \preceq e_2$  ou  $e_2 \preceq e_1$

Sinon l'ordre est dit *partiel*.

*Exemple* :  $\subseteq$  est une relation d'ordre partiel sur l'ensemble  $\wp(E)$  des parties d'un ensemble  $E$ .

*Ordre strict*  $\prec$  associé à  $\preceq$  :  $x \prec y$  ssi  $x \preceq y$  et  $x \neq y$ .

... un ordre strict n'est pas une relation d'ordre ! (réflexivité)

## Ordres bien fondés (1)

Une relation d'ordre  $\preceq$  sur un ensemble  $E$  est *bien fondée* s'il n'existe pas de suite infinie strictement décroissante  $e_1 \succ e_2 \succ \dots$  d'éléments de  $E$ .

*Exemple* :  $\leq$  est un ordre bien fondé sur  $\mathbb{N}$  tandis que ce n'est pas un ordre bien fondé sur  $\mathbb{Z}$

**Théorème** Un ordre, défini sur  $E$ , est bien fondé si et seulement si toute partie non vide de  $E$  admet un élément minimal (pour cet ordre).

Un *élément minimal* d'une partie  $X$  de  $E$  est un élément de  $X$  tel qu'il n'existe pas, dans  $X$ , un élément plus petit que lui.

*Remarque* : Tout ordre sur un ensemble fini est bien fondé.

## Ordres bien fondés (2)

PREUVE

( $\Rightarrow$ ). Soit  $X$  une partie non vide de  $E$ . Si  $X$  n'admet pas d'élément minimal, alors :

$$\forall x \in X \exists y \in X y \prec x$$

Puisque  $X$  n'est pas vide, on peut choisir un élément  $x_0 \in X$  et il existe donc un élément  $x_1 \in X$  tel que  $x_1 \prec x_0$ . On peut alors construire “de proche en proche” une suite infinie décroissante ce qui contredit l'hypothèse de bonne fondation de l'ordre.

( $\Leftarrow$ ). Si toute partie non vide de  $E$  admet un élément minimal, alors c'est en particulier le cas pour une suite strictement décroissante. Soit  $p$  l'indice de cet élément minimal. Tous les éléments d'indice supérieur à  $p$  lui sont supérieurs et, puisque la suite est strictement décroissante,  $p$  est le plus grand indice de la suite qui est donc finie.

## Récurrance bien fondée

**Théorème** Si  $E$  est muni d'un ordre bien fondé  $\preceq$  et  $P$  est une propriété sur  $E$  alors

si  $(\forall x \in E (\text{si } (\forall y \prec x P(y)) \text{ alors } P(x)))$  alors  $\forall x \in E P(x)$

PREUVE Soit  $X = \{x \in E \mid \neg P(x)\}$ . Si  $X$  est non vide alors, d'après le théorème précédent,  $X$  admet un élément minimal  $x_0$  qui vérifie donc  $\forall y \prec x_0, y \notin X$  et donc  $P(y)$  est vrai. En utilisant l'hypothèse on en déduit que  $P(x_0)$  est vrai ce qui contredit  $x_0 \in X$ . Donc  $X$  est vide ce qui signifie que  $\forall x \in E P(x)$ .

## Ordre lexicographique (1)

Définir un ordre lexicographique, c'est définir une relation d'ordre sur un produit cartésien d'ensembles à partir des relations d'ordre définies sur chacun des ensembles invoqués dans le produit cartésien.

*Exemple* : l'ordre alphabétique sur les mots est un ordre lexicographique obtenu à partir de l'ordre qui existe sur les lettres de l'alphabet.

Soit  $E_i$ , pour  $1 \leq i \leq n$ , un ensemble ordonné par  $\preceq_i$ . *Ordre lexicographique*  $\preceq$  sur le produit cartésien  $E_1 \times E_2 \times \cdots \times E_n$  défini par :

$$(e_1, \dots, e_n) \preceq (f_1, \dots, f_n) \Leftrightarrow \left( \begin{array}{l} (e_1, e_2, \dots, e_n) = (f_1, f_2, \dots, f_n) \\ \vee (\exists m > 0 \forall i < m e_i = f_i \wedge e_m \prec_m f_m) \end{array} \right)$$

*Exemple* : Ordre lexicographique sur  $\mathbb{N} \times \mathbb{N}$

$$(n_1, n_2) \preceq (n'_1, n'_2) \text{ ssi } (n_1 \leq n'_1) \text{ ou } (n_1 = n'_1 \text{ et } n_2 \leq n'_2)$$

**Théorème** Si  $\preceq_i$  est un ordre bien fondé sur  $E_i$ , pour  $1 \leq i \leq n$ , alors l'ordre lexicographique  $\preceq$  sur le produit cartésien  $E_1 \times E_2 \times \cdots \times E_n$  est bien fondé.

## Ordre lexicographique (2)

PREUVE Soit  $X$  une partie non vide de  $E_1 \times E_2 \times \cdots \times E_n$ . Montrons tout d'abord par récurrence sur  $k \in \mathbb{N}$ , que si  $1 \leq k \leq n$ , alors :

$$X_k = \left\{ \begin{array}{l} x_k \in E_k \mid \exists (m_1, m_2, \dots, m_{k-1}, x_k, \dots, x_n) \in X \\ \text{tel que } \forall (x'_1, x'_2, \dots, x'_n) \in X \ m_i \preceq_i x'_i \ (1 \leq i \leq k-1) \end{array} \right\}$$

n'est pas vide.

Pour  $n = 1$ , on a  $X_1 = \{x_1 \in E_1 \mid \exists (x_1, x_2, \dots, x_n) \in X\} \neq \emptyset$ . Puisque  $\preceq_1$  est un ordre bien fondé sur  $E_1$ ,  $X_1$  admet un élément minimal  $m_1 \in X_1$ . Supposons à présent que pour un entier  $i < n$ ,  $X_i$  soit non vide.  $X_i$  admet un élément minimal  $m_i \in X_i$  et donc l'ensemble  $X_{i+1}$  n'est pas vide.

Chacun des ensembles  $X_j$  (pour  $1 \leq j \leq n$ ) n'est donc pas vide et admet un élément minimal  $m_j$ . Par construction, on a alors

$(m_1, m_2, \dots, m_n) \in X$  qui est un élément minimal de  $X$  pour l'ordre lexicographique sur  $E_1 \times E_2 \times \cdots \times E_n$ . On peut alors conclure que cet ordre lexicographique est bien fondé.

### Ordre lexicographique (3)

*Remarque* : Ce théorème **ne s'étend pas** à  $(E_i)_{i \in \mathbb{N}}$ . En effet, le fait que le produit cartésien porte sur un nombre fini d'ensembles est important.

*Exemple.*

On considère un alphabet  $A = \{a, b\}$  et une relation d'ordre  $\preceq$  sur  $A$  définie par  $a \preceq b$ .

$A$  étant fini,  $\preceq$  est un ordre bien fondé sur  $A$ .

... mais l'ordre lexicographique sur  $A^*$  (ensemble des suites de longueurs finies d'éléments de  $A$ ) n'est pas un ordre bien fondé :  $(a^n b)_{n \in \mathbb{N}}$  est une suite infinie strictement décroissante d'éléments de  $A^*$  !

### Terminaison (3)

$e \in E_A^+$  : expression

- $\mathcal{N}_V(e) \in \mathbb{N}$  : nombre d'occurrences de variables dans  $e$
- $\mathcal{N}_O(e) \in \mathbb{N}$  : nombre d'opérateurs apparaissant dans  $e$

Ordre lexicographique  $\preceq$  sur  $\mathbb{N} \times \mathbb{N}$  :

$$(n_1, n_2) \preceq (n'_1, n'_2) \text{ ssi } (n_1 \leq n'_1) \text{ ou } (n_1 = n'_1 \text{ et } n_2 \leq n'_2)$$

Puisque  $\leq$  est un ordre bien fondé sur  $\mathbb{N}$ , d'après le théorème précédent  $\preceq$  est aussi un ordre bien fondé sur  $\mathbb{N} \times \mathbb{N}$ .

En utilisant le schéma d'induction associé au système d'inférence définissant la relation  $\hookrightarrow$ , nous allons montrer une propriété  $P$  exprimant que si  $\langle e, \sigma \rangle \hookrightarrow \langle e', \sigma \rangle$ , alors  $(\mathcal{N}_V(e'), \mathcal{N}_O(e')) \prec (\mathcal{N}_V(e), \mathcal{N}_O(e))$  où  $\prec$  est l'ordre strict associé à  $\preceq$ .

$\preceq$  étant bien fondé, toute séquence de calcul correspondant à l'évaluation d'une expression arithmétique est donc finie.

## Terminaison (4)

**Propriété  $P$**  :  $\langle e, \sigma \rangle \hookrightarrow \langle e', \sigma \rangle \Rightarrow (\mathcal{N}_V(e'), \mathcal{N}_O(e')) \prec (\mathcal{N}_V(e), \mathcal{N}_O(e))$

PREUVE Induction sur l'arbre d'inférence du jugement  $\langle e, \sigma \rangle \hookrightarrow \langle e', \sigma \rangle$ .

$\forall x \in V$   $P(\langle x, \sigma \rangle \hookrightarrow \langle \sigma(x), \sigma \rangle)$  (AS<sub>1</sub>)  $\frac{}{\langle x, \sigma \rangle \hookrightarrow \langle \sigma(x), \sigma \rangle}$

$\sigma(x) \in \mathbb{Z}$  et on a donc  $\mathcal{N}_V(x) = 1$  et  $\mathcal{N}_V(\sigma(x)) = 0$  ce qui permet de conclure.

$\forall e \in E_A$   $P(\langle \text{Err op}_2 e, \sigma \rangle \hookrightarrow \langle \text{Err}, \sigma \rangle)$  (AS<sub>2</sub>)  $\frac{}{\langle \text{Err op}_2 e, \sigma \rangle \hookrightarrow \langle \text{Err}, \sigma \rangle}$

On a  $\mathcal{N}_V(\text{Err op}_2 e) \geq \mathcal{N}_V(\text{Err})$  et  $\mathcal{N}_O(\text{Err op}_2 e) > \mathcal{N}_O(\text{Err})$  ce qui permet de conclure.

$\forall n \in \mathbb{Z}$   $P(\langle n \text{ op}_2 \text{Err}, \sigma \rangle \hookrightarrow \langle \text{Err}, \sigma \rangle)$  (AS<sub>3</sub>)  $\frac{}{\langle n \text{ op}_2 \text{Err}, \sigma \rangle \hookrightarrow \langle \text{Err}, \sigma \rangle}$

On a  $\mathcal{N}_V(n \text{ op}_2 \text{Err}) \geq \mathcal{N}_V(\text{Err})$  et  $\mathcal{N}_O(n \text{ op}_2 \text{Err}) > \mathcal{N}_O(\text{Err})$  ce qui permet de conclure.

## Terminaison (5)

$$(AS_4) \frac{}{\langle n_1 \text{ op}_2 n_2, \sigma \rangle \hookrightarrow \langle n, \sigma \rangle}$$

$$\forall n, n_1, n_2 \in \mathbb{Z} (n = n_1 \text{ op}_2 n_2 \Rightarrow P(\langle n_1 \text{ op}_2 n_2, \sigma \rangle \hookrightarrow \langle n, \sigma \rangle))$$

On a  $\mathcal{N}_V(n) = \mathcal{N}_V(n_1 + n_2)$  et  $\mathcal{N}_O(n) = \mathcal{N}_O(n_1 + n_2) - 1$  ce qui permet de conclure.

$$(AS_5) \frac{}{\langle n_1/n_2, \sigma \rangle \hookrightarrow \langle n, \sigma \rangle}$$

$$\forall n, n_1 \in \mathbb{Z} \quad \forall n \in \mathbb{Z} \setminus \{0\} (n = n_1/n_2 \Rightarrow P(\langle n_1/n_2, \sigma \rangle \hookrightarrow \langle n, \sigma \rangle))$$

On a  $\mathcal{N}_V(n) = \mathcal{N}_V(n_1/n_2)$  et  $\mathcal{N}_O(n) = \mathcal{N}_O(n_1/n_2) - 1$  ce qui permet de conclure.

$$(AS_6) \frac{}{\langle n_1/n_2, \sigma \rangle \hookrightarrow \langle \text{Err}, \sigma \rangle}$$

$$\forall n_1, n_2 \in \mathbb{Z} (n_2 = 0 \Rightarrow P(\langle n_1/n_2, \sigma \rangle \hookrightarrow \langle \text{Err}, \sigma \rangle))$$

On a  $\mathcal{N}_V(n_1/n_2) = \mathcal{N}_V(\text{Err})$  et  $\mathcal{N}_O(n_1/n_2) > \mathcal{N}_O(\text{Err})$  ce qui permet de conclure.

## Terminaison (6)

$$(AS_7) \frac{\langle e_1, \sigma \rangle \hookrightarrow \langle e'_1, \sigma \rangle}{\langle e_1 \text{ op}_2 e_2, \sigma \rangle \hookrightarrow \langle e'_1 \text{ op}_2 e_2, \sigma \rangle}$$

$\forall e_1, e'_1, e_2 \in E_A$

$(P(\langle e_1, \sigma \rangle \hookrightarrow \langle e'_1, \sigma \rangle) \Rightarrow P(\langle e_1 \text{ op}_2 e_2, \sigma \rangle \hookrightarrow \langle e'_1 \text{ op}_2 e_2, \sigma \rangle))$

Par **hypothèse d'induction**, on a  $(\mathcal{N}_V(e'_1), \mathcal{N}_O(e'_1)) \prec (\mathcal{N}_V(e_1), \mathcal{N}_O(e_1))$ .

Aussi, soit  $\mathcal{N}_V(e'_1) < \mathcal{N}_V(e_1)$  et il vient :

$$\mathcal{N}_V(e'_1 \text{ op}_2 e_2) = \mathcal{N}_V(e'_1) + \mathcal{N}_V(e_2) < \mathcal{N}_V(e_1) + \mathcal{N}_V(e_2) = \mathcal{N}_V(e_1 \text{ op}_2 e_2)$$

et on peut conclure, soit  $\mathcal{N}_V(e'_1) = \mathcal{N}_V(e_1)$  et  $\mathcal{N}_O(e'_1) < \mathcal{N}_O(e_1)$  et il vient :

$$\mathcal{N}_V(e'_1 \text{ op}_2 e_2) = \mathcal{N}_V(e'_1) + \mathcal{N}_V(e_2) = \mathcal{N}_V(e_1) + \mathcal{N}_V(e_2) = \mathcal{N}_V(e_1 \text{ op}_2 e_2)$$

$$\mathcal{N}_O(e'_1 \text{ op}_2 e_2) = \mathcal{N}_O(e'_1) + \mathcal{N}_O(e_2) + 1 < \mathcal{N}_O(e_1) + \mathcal{N}_O(e_2) + 1 = \mathcal{N}_O(e_1 \text{ op}_2 e_2)$$

ce qui permet encore de conclure.

## Terminaison (7)

$$(AS_8) \frac{\langle e_2, \sigma \rangle \hookrightarrow \langle e'_2, \sigma \rangle}{\langle n_1 \text{ op}_2 e_2, \sigma \rangle \hookrightarrow \langle n_1 \text{ op}_2 e'_2, \sigma \rangle}$$

$\forall n_1 \in \mathbb{Z} \forall e_2, e'_2 \in E_A$

$(P(\langle e_2, \sigma \rangle \hookrightarrow \langle e'_2, \sigma \rangle) \Rightarrow P(\langle n_1 \text{ op}_2 e_2, \sigma \rangle \hookrightarrow \langle n_1 \text{ op}_2 e'_2, \sigma \rangle))$

Par **hypothèse d'induction**, on a  $(\mathcal{N}_V(e'_2), \mathcal{N}_O(e'_2)) \prec (\mathcal{N}_V(e_2), \mathcal{N}_O(e_2))$ .

Aussi, soit  $\mathcal{N}_V(e'_2) < \mathcal{N}_V(e_2)$  et il vient :

$$\mathcal{N}_V(n_1 \text{ op}_2 e'_2) = \mathcal{N}_V(e'_2) < \mathcal{N}_V(e_2) = \mathcal{N}_V(n_1 \text{ op}_2 e_2)$$

et on peut conclure, soit  $\mathcal{N}_V(e'_2) = \mathcal{N}_V(e_2)$  et  $\mathcal{N}_O(e'_2) < \mathcal{N}_O(e_2)$  et il vient :

$$\mathcal{N}_V(n_1 \text{ op}_2 e'_2) = \mathcal{N}_V(e'_2) = \mathcal{N}_V(e_2) = \mathcal{N}_V(n_1 \text{ op}_2 e_2)$$

$$\mathcal{N}_O(n_1 \text{ op}_2 e'_2) = \mathcal{N}_O(e'_2) + 1 < \mathcal{N}_O(e_2) + 1 = \mathcal{N}_O(n_1 \text{ op}_2 e_2)$$

ce qui permet encore de conclure.

## Equivalence sémantique (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  $\langle x, \sigma \rangle \rightsquigarrow \langle \sigma(x), \sigma \rangle$  et on a donc  $\langle x, \sigma \rangle \xrightarrow{*} v$ .

- si  $e = a_1 \text{ op}_2 a_2$  avec  $\text{op}_2 \in \{+, -, \times\}$ , alors trois cas se présentent.

## Equivalence sémantique (2)

**Cas 1.**  $\langle a_1, \sigma \rangle \rightsquigarrow \text{Err}$  L'arbre de dérivation de  $\langle e, \sigma \rangle \rightsquigarrow n$  est :

$$(A_3) \frac{(A_i) \frac{\vdots}{\langle a_1, \sigma \rangle \rightsquigarrow \text{Err}}}{\langle a_1 \text{ op}_2 a_2, \sigma \rangle \rightsquigarrow \text{Err}}$$

Par **hypothèse d'induction**, on a  $\langle a_1, \sigma \rangle \xrightarrow{*} \text{Err}$  et il existe donc une séquence de calcul :

$$\langle a_1, \sigma \rangle = \langle a_1^0, \sigma \rangle \hookrightarrow \langle a_1^1, \sigma \rangle \hookrightarrow \langle a_1^2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle a_1^{k_1}, \sigma \rangle = \langle \text{Err}, \sigma \rangle$$

à partir de laquelle, en appliquant la règle  $AS_7$ , on peut construire :

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

et la règle  $AS_2$  permet alors de conclure.

### Equivalence sémantique (3)

**Cas 2.**  $\langle a_2, \sigma \rangle \rightsquigarrow \text{Err}$  L'arbre de dérivation de  $\langle e, \sigma \rangle \rightsquigarrow v$  est :

$$(A_4) \frac{(A_j) \frac{\vdots}{\langle a_1, \sigma \rangle \rightsquigarrow v_1} \quad (A_i) \frac{\vdots}{\langle a_2, \sigma \rangle \rightsquigarrow \text{Err}}}{\langle a_1 \text{ op}_2 a_2, \sigma \rangle \rightsquigarrow \text{Err}}$$

Par **hypothèse d'induction** on a  $\langle a_2, \sigma \rangle \xrightarrow{*} \text{Err}$  et il existe donc une séquence de calcul :

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

D'autre part,  $v_1 \in \mathbb{Z}$  et, par **hypothèse d'induction**, on a  $\langle a_1, \sigma \rangle \xrightarrow{*} v_1$ . Il existe donc une séquence de calcul :

$$\langle a_1, \sigma \rangle = \langle a_1^0, \sigma \rangle \hookrightarrow \langle a_1^1, \sigma \rangle \hookrightarrow \langle a_1^2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle a_1^{k_1}, \sigma \rangle = \langle v_1, \sigma \rangle \quad (2)$$

### Equivalence sémantique (4)

A partir de (2), en appliquant la règle  $AS_7$ , on obtient :

$$\begin{aligned}
 & \langle a_1 \text{ op}_2 a_2, \sigma \rangle = \langle a_1^0 \text{ op}_2 a_2, \sigma \rangle \\
 \hookrightarrow & \langle a_1^1 \text{ op}_2 a_2, \sigma \rangle \hookrightarrow \langle a_1^2 \text{ op}_2 a_2, \sigma \rangle \hookrightarrow \dots \\
 \hookrightarrow & \langle a_1^{k_1} \text{ op}_2 a_2, \sigma \rangle = \langle v_1 \text{ op}_2 a_2, \sigma \rangle
 \end{aligned} \tag{3}$$

La règle  $AS_8$  permet de transformer chacune des transitions  $\langle a_2^i, \sigma \rangle \hookrightarrow \langle a_2^{i+1}, \sigma \rangle$  de la séquence (1) en une transition  $\langle v_1 \text{ op}_2 a_2^i, \sigma \rangle \hookrightarrow \langle v_1 \text{ op}_2 a_2^{i+1}, \sigma \rangle$ . On obtient alors la séquence :

$$\begin{aligned}
 & \langle v_1 \text{ op}_2 a_2, \sigma \rangle = \langle v_1 \text{ op}_2 a_2^0, \sigma \rangle \hookrightarrow \langle v_1 \text{ op}_2 a_2^1, \sigma \rangle \hookrightarrow \langle v_1 \text{ op}_2 a_2^2, \sigma \rangle \hookrightarrow \dots \\
 & \dots \hookrightarrow \langle v_1 \text{ op}_2 a_2^{k_2}, \sigma \rangle = \langle v_1 \text{ op}_2 \text{Err}, \sigma \rangle
 \end{aligned} \tag{4}$$

Enfin, la règle  $AS_3$  nous permet de considérer la transition  $\langle v_1 \text{ op}_2 \text{Err}, \sigma \rangle \hookrightarrow \langle \text{Err}, \sigma \rangle$ , et en concaténant les séquences (3) et (4) avec cette transition on obtient finalement  $\langle a_1 \text{ op}_2 a_2, \sigma \rangle \xrightarrow{*} \text{Err}$  ce qui permet de conclure.

## Equivalence sémantique (5)

**Cas 3.**  $\langle a_1, \sigma \rangle \rightsquigarrow n_1$  et  $\langle a_2, \sigma \rangle \rightsquigarrow n_2$  ( $n_1, n_2 \in \mathbb{Z}$ )

L'arbre de dérivation de  $\langle e, \sigma \rangle \rightsquigarrow v$  est :

$$(A_k) \frac{\begin{array}{c} \vdots \\ (A_i) \frac{\quad}{\langle a_1, \sigma \rangle \rightsquigarrow n_1} \end{array} \quad \begin{array}{c} \vdots \\ (A_j) \frac{\quad}{\langle a_2, \sigma \rangle \rightsquigarrow n_2} \end{array}}{\langle a_1 \text{ op}_2 a_2, \sigma \rangle \rightsquigarrow n_1 \text{ op}_2 n_2} \quad (k \in \{4, 5, 6\})$$

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 :

$$\langle a_1, \sigma \rangle = \langle a_1^0, \sigma \rangle \hookrightarrow \langle a_1^1, \sigma \rangle \hookrightarrow \langle a_1^2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle a_1^{k_1}, \sigma \rangle = \langle n_1, \sigma \rangle \quad (5)$$

$$\langle a_2, \sigma \rangle = \langle a_2^0, \sigma \rangle \hookrightarrow \langle a_2^1, \sigma \rangle \hookrightarrow \langle a_2^2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle a_2^{k_2}, \sigma \rangle = \langle n_2, \sigma \rangle \quad (6)$$

La règle  $AS_7$  permet de transformer chaque  $\langle a_1^i, \sigma \rangle \hookrightarrow \langle a_1^{i+1}, \sigma \rangle$  de (5) 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 :

$$\begin{aligned} \langle a_1 \text{ op}_2 a_2, \sigma \rangle &= \langle a_1^0 \text{ op}_2 a_2, \sigma \rangle \hookrightarrow \langle a_1^1 \text{ op}_2 a_2, \sigma \rangle \hookrightarrow \langle a_1^2 \text{ op}_2 a_2, \sigma \rangle \hookrightarrow \dots \\ &\dots \hookrightarrow \langle a_1^{k_1} \text{ op}_2 a_2, \sigma \rangle = \langle n_1 \text{ op}_2 a_2, \sigma \rangle \end{aligned} \quad (7)$$

## Equivalence sémantique (6)

La règle  $AS_8$  permet de transformer chaque  $\langle a_2^i, \sigma \rangle \hookrightarrow \langle a_2^{i+1}, \sigma \rangle$  de (6) 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 :

$$\begin{aligned} \langle n_1 \text{ op}_2 a_2, \sigma \rangle &= \langle n_1 \text{ op}_2 a_2^0, \sigma \rangle \hookrightarrow \langle n_1 \text{ op}_2 a_2^1, \sigma \rangle \hookrightarrow \langle n_1 \text{ op}_2 a_2^2, \sigma \rangle \hookrightarrow \dots \\ \dots \hookrightarrow \langle n_1 \text{ op}_2 a_2^{k_2}, \sigma \rangle &= \langle n_1 \text{ op}_2 n_2, \sigma \rangle \end{aligned} \tag{8}$$

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 (7) et (8) 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  $e = a_1/a_2$ , alors le raisonnement est similaire au cas précédent. Il suffit de considérer le cas supplémentaire où  $\langle a_2, \sigma \rangle \rightsquigarrow 0$ .

## Equivalence sémantique (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  $\langle e_1, \sigma \rangle \hookrightarrow \langle e_2, \sigma \rangle$  et  $\langle e_2, \sigma \rangle \rightsquigarrow v$ , alors  $\langle e_1, \sigma \rangle \rightsquigarrow v$ .

PREUVE Induction sur  $e_1$ .

- $e_1 = n \in \mathbb{Z}$ . Impossible car  $\langle e_1, \sigma \rangle$  serait une configuration terminale.
- $e_1 = x \in V$ . On a forcément  $e_2 = \sigma(x) \in \mathbb{Z}$  et  $v = \sigma(x)$ . On peut conclure en construisant l'arbre :

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

- $e_1 = a_1 \text{ op}_2 a_2$ . On distingue 3 cas.

## Equivalence sémantique (8)

(1).  $a_1, a_2 \in \mathbb{Z}$  (avec  $a_2 \neq 0 \vee \text{op}_2 \neq /$  puisque sinon  $e_2$  vaudrait **Err** ce qui est impossible puisque  $e_2 \in E_A$ ). On a alors  $e_2 = \langle v, \sigma \rangle$  avec  $v = a_1 \text{ op}_2 a_2 \in \mathbb{Z}$  et on peut conclure en construisant l'arbre :

$$(A_j) \frac{(A_1) \frac{}{\langle a_1, \sigma \rangle \rightsquigarrow a_1} \quad (A_1) \frac{}{\langle a_2, \sigma \rangle \rightsquigarrow a_2}}{\langle a_1 \text{ op}_2 a_2, \sigma \rangle \rightsquigarrow v} \quad j \in \{5, 6, 7, 9\}$$

(2).  $a_1 \in E_A \setminus \mathbb{Z}$  et il vient  $e_2 = a'_1 \text{ op}_2 a_2$  avec  $\langle a_1, \sigma \rangle \hookrightarrow \langle a'_1, \sigma \rangle$ . L'arbre d'inférence de  $\langle a'_1 \text{ op}_2 a_2, \sigma \rangle \rightsquigarrow v$  contient forcément le sous-arbre :

$$(A_j) \frac{}{\langle a'_1, \sigma \rangle \rightsquigarrow v'_1}$$

et donc par hypothèse d'induction, il vient  $\langle a_1, \sigma \rangle \rightsquigarrow v'_1$  et à partir de l'arbre d'inférence de  $\langle a'_1 \text{ op}_2 a_2, \sigma \rangle \rightsquigarrow v$  on peut obtenir facilement  $\langle a_1 \text{ op}_2 a_2, \sigma \rangle \rightsquigarrow v$

(3).  $a_1 \in \mathbb{Z}$  et  $a_2 \in E_A \setminus \mathbb{Z}$  Raisonement similaire au cas précédent.

## Equivalence sémantique (9)

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

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

Si  $k = 0$ , alors  $e = v \in E_A \cap \mathbb{V} = \mathbb{Z}$  et la règle  $A_1$  permet de conclure.

Si  $k = k_0 + 1$ , alors on procède par cas sur la première transition de la séquence.

## Equivalence sémantique (10)

- $\langle x, \sigma \rangle \hookrightarrow \langle \sigma(x), \sigma \rangle$  (règle  $AS_1$ ) avec  $e = x$ ,  $v = \sigma(x)$  et  $k_0 = 0$ . La règle  $A_2$  permet de conclure.
- $\langle \text{Err op}_2 e', \sigma \rangle \hookrightarrow \langle \text{Err}, \sigma \rangle$  (règle  $AS_2$ ) avec  $k_0 = 0$ . Impossible car  $\text{Err op}_2 e' \notin E_A$ .
- $\langle n \text{ op}_2 \text{Err}, \sigma \rangle \hookrightarrow \langle \text{Err}, \sigma \rangle$  (règle  $AS_3$ ) avec  $k_0 = 0$ . Impossible car  $n \text{ op}_2 \text{Err} \notin E_A$ .
- $\langle n_1 \text{ op}_2 n_2, \sigma \rangle \hookrightarrow \langle n, \sigma \rangle$  (règle  $AS_4$ ) avec  $\text{op}_2 \in \{+, -, \times\}$ ,  $n = n_1 \text{ op}_2 n_2$  et  $k_0 = 0$ . Les règles  $A_1$ ,  $A_5$ ,  $A_6$  et  $A_7$  permettent de conclure :

$$(A_j) \frac{(A_1) \frac{}{\langle n_1, \sigma \rangle \rightsquigarrow n_1} \quad (A_1) \frac{}{\langle n_2, \sigma \rangle \rightsquigarrow n_2}}{\langle n_1 \text{ op}_2 n_2, \sigma \rangle \rightsquigarrow n} \quad j \in \{5, 6, 7\}$$

## Equivalence sémantique (11)

- $\langle n_1/n_2, \sigma \rangle \hookrightarrow \langle n, \sigma \rangle$  (règle  $AS_5$ ) avec  $n_1 \in \mathbb{Z}$ ,  $n_2 \in \mathbb{Z} \setminus \{0\}$ ,  $n = n_1/n_2$  et  $k_0 = 0$ . La règle  $A_9$  permet de conclure :

$$(A_9) \frac{(A_1) \frac{}{\langle n_1, \sigma \rangle \rightsquigarrow n_1} \quad (A_1) \frac{}{\langle n_2, \sigma \rangle \rightsquigarrow n_2}}{\langle n_1/n_2, \sigma \rangle \rightsquigarrow n}$$

- $\langle n_1/n_2, \sigma \rangle \hookrightarrow \langle \text{Err}, \sigma \rangle$  (règle  $AS_6$ ) avec  $n_1 \in \mathbb{Z}$ ,  $n_2 = 0$ , et  $k_0 = 0$ . La règle  $A_8$  permet de conclure :

$$(A_8) \frac{(A_1) \frac{}{\langle n_1, \sigma \rangle \rightsquigarrow n_1} \quad (A_1) \frac{}{\langle n_2, \sigma \rangle \rightsquigarrow 0}}{\langle n_1/n_2, \sigma \rangle \rightsquigarrow \text{Err}}$$

## Equivalence sémantique (12)

- $\langle e_1 \text{ op}_2 e_2, \sigma \rangle \hookrightarrow \langle e'_1 \text{ op}_2 e_2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle e^k, \sigma \rangle$  avec  $\langle e_1, \sigma \rangle \hookrightarrow \langle e'_1, \sigma \rangle$ .

On distingue deux sous-cas :

(1). Si  $e'_1 \in E_A$ , alors, par hypothèse d'induction, on a  $\langle e'_1 \text{ op}_2 e_2, \sigma \rangle \rightsquigarrow v$  et le lemme précédent permet d'obtenir  $\langle e_1 \text{ op}_2 e_2, \sigma \rangle \rightsquigarrow v$ .

(2). Si  $e'_1 \in E_A^+ \setminus E_A$ , alors, puisque  $e_1 \in E_A$ , on a forcément  $e_1 = n/0$  puisque la seule règle permettant d'introduire **Err** dans une expression ne contenant pas **Err** est la règle  $AS_6$  et on a donc  $v = e^k = \mathbf{Err}$ . On peut alors conclure en construisant l'arbre :

$$\begin{array}{c}
 (A_1) \frac{}{\langle n, \sigma \rangle \rightsquigarrow n} \quad (A_1) \frac{}{\langle 0, \sigma \rangle \rightsquigarrow 0} \\
 (A_8) \frac{}{\langle n/0, \sigma \rangle \rightsquigarrow \mathbf{Err}} \\
 (A_3) \frac{}{\langle n/0 \text{ op}_2 e_2, \sigma \rangle \rightsquigarrow \mathbf{Err}}
 \end{array}$$

- $\langle e_1 \text{ op}_2 e_2, \sigma \rangle \hookrightarrow \langle e_1 \text{ op}_2 e'_2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle e^k, \sigma \rangle$  avec  $\langle e_2, \sigma \rangle \hookrightarrow \langle e'_2, \sigma \rangle$  et  $e_1 \in \mathbb{Z}$ . Raisonnement similaire au cas 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 :

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

---


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

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


---

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

## Interprétation des expressions booléennes

Interpréter une expression booléenne c'est lui donner une **valeur** appartenant à  $\mathbb{B} \cup \{\text{Err}\}$ .

... 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

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

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

## Expressions booléennes : Sémantique opérationnelle à grands pas (1)

**Système d'inférence** définissant un sous-ensemble de jugements de la forme  $\langle b, \sigma \rangle \rightsquigarrow v$ , exprimant le fait qu'une expression booléenne  $b \in E_B$  s'évalue en une valeur  $v \in \mathbb{B} \cup \{\text{Err}\}$  étant donnée une valuation  $\sigma$ .

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

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

$$(B_4) \frac{\langle a_1, \sigma \rangle \rightsquigarrow \text{Err}}{\langle a_1 \leq a_2, \sigma \rangle \rightsquigarrow \bar{\text{Err}}} \quad (B_5) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow \text{Err}}{\langle a_1 \leq a_2, \sigma \rangle \rightsquigarrow \bar{\text{Err}}}$$

$$(B_6) \frac{\langle a_1, \sigma \rangle \rightsquigarrow \text{Err}}{\langle a_1 = a_2, \sigma \rangle \rightsquigarrow \bar{\text{Err}}} \quad (B_7) \frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow \text{Err}}{\langle a_1 = a_2, \sigma \rangle \rightsquigarrow \bar{\text{Err}}}$$

## Expressions booléennes : Sémantique opérationnelle à grands pas (2)

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

$$(B_{10}) \frac{\langle b_1, \sigma \rangle \rightsquigarrow \text{Err}}{\langle b_1 \text{ and } b_2, \sigma \rangle \rightsquigarrow \text{Err}}$$

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

$$(B_{13}) \frac{\langle b_1, \sigma \rangle \rightsquigarrow \text{Err}}{\langle b_1 \text{ or } b_2, \sigma \rangle \rightsquigarrow \text{Err}}$$

$$(B_{14}) \frac{\langle b, \sigma \rangle \rightsquigarrow t}{\langle \text{not } b, \sigma \rangle \rightsquigarrow (\neg t)} \quad (B_{15}) \frac{\langle b, \sigma \rangle \rightsquigarrow \text{Err}}{\langle \text{not } b, \sigma \rangle \rightsquigarrow \text{Err}} \quad t \in \text{IB}$$

## Choix des règles (1)

- 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  $\langle (2 = 2) \text{ or } (2/0 \leq x), \sigma \rangle \rightsquigarrow \text{true}$

- Remplacer les règles  $B_8, B_9, B_{10}, B_{11}, B_{12}$  et  $B_{13}$  par

$$(B'_8) \frac{\langle b_2, \sigma \rangle \rightsquigarrow \text{true} \quad \langle b_1, \sigma \rangle \rightsquigarrow v}{\langle b_1 \text{ and } b_2, \sigma \rangle \rightsquigarrow v} \quad (B'_9) \frac{\langle b_2, \sigma \rangle \rightsquigarrow \text{false}}{\langle b_1 \text{ and } b_2, \sigma \rangle \rightsquigarrow \text{false}} \quad v \in \text{IB} \cup \{\text{Err}\}$$

$$(B'_{10}) \frac{\langle b_2, \sigma \rangle \rightsquigarrow \text{Err}}{\langle b_1 \text{ and } b_2, \sigma \rangle \rightsquigarrow \text{Err}}$$

$$(B'_{11}) \frac{\langle b_2, \sigma \rangle \rightsquigarrow \text{false} \quad \langle b_1, \sigma \rangle \rightsquigarrow v}{\langle b_1 \text{ or } b_2, \sigma \rangle \rightsquigarrow v} \quad (B'_{12}) \frac{\langle b_2, \sigma \rangle \rightsquigarrow \text{true}}{\langle b_1 \text{ or } b_2, \sigma \rangle \rightsquigarrow \text{true}} \quad v \in \text{IB} \cup \{\text{Err}\}$$

$$(B'_{13}) \frac{\langle b_2, \sigma \rangle \rightsquigarrow \text{Err}}{\langle b_1 \text{ or } b_2, \sigma \rangle \rightsquigarrow \text{Err}}$$

Sémantiques équivalentes ? Non

## Choix des règles (2)

*Exemple*

Avec un valuation  $\sigma$  telle que  $\sigma(x) = 0$

$e$	$B_8, B_9, B_{10},$ $B_{11}, B_{12}, B_{13}$	$B'_8, B'_9, B'_{10},$ $B'_{11}, B'_{12}, B'_{13}$
$x = 0$ or $(10/x \leq 5)$	$\langle e, \sigma \rangle \rightsquigarrow \text{true}$	$\langle e, \sigma \rangle \rightsquigarrow \text{Err}$
(not $x = 0$ ) and $(10/x \leq 5)$	$\langle e, \sigma \rangle \rightsquigarrow \text{false}$	$\langle e, \sigma \rangle \rightsquigarrow \text{Err}$

## Sémantique opérationnelle : Propriétés

### Expressions booléennes équivalentes

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

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

PREUVE. Exercice ...

**Proposition** Existence et unicité d'un résultat  $\forall b \in E_B \quad \forall \sigma \in \mathcal{V}[\mathbb{Z}]$

$$\exists v \in \mathbb{B} \cup \{\text{Err}\} \quad \langle b, \sigma \rangle \rightsquigarrow v$$

$$\forall v_1, v_2 \in \mathbb{B} \cup \{\text{Err}\} \quad (\langle b, \sigma \rangle \rightsquigarrow v_1 \text{ et } \langle b, \sigma \rangle \rightsquigarrow v_2) \Rightarrow v_1 = v_2$$

PREUVE. Exercice ...

## Constructions impératives

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

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

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

---

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

$(C_1) \frac{}{\text{skip}} \quad (C_2) \frac{}{x := a}$

$(C_3) \frac{c_1 \quad c_2}{c_1; c_2} \quad (C_4) \frac{c_1 \quad c_2}{\text{if } b \text{ then } c_1 \text{ else } c_2} \quad (C_5) \frac{c}{\text{while } b \text{ do } c}$

---

## Constructions impératives : Exemple (PGCD)

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

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

si  $x, y \in V$ ,  $\text{not } (x = y), x \leq y \in E_B$  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  $x$ .

## Transitions

Le rôle d'un programme  $c$  est de modifier l'état courant  $\sigma_1$  de la mémoire en un état  $\sigma_2$  :

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

L'exécution de l'instruction  $c$  dans l'état  $\sigma_1$  conduit à l'état  $\sigma_2$ .

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`

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

Il n'existe pas d'état  $\sigma'$  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.

$c$

$x := 0$

$c'$

$x := x + 1$

$c''$

$x := x + 5$

$b$

$1 \leq x$

$\sigma$

$\sigma(x) = 2$

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

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

## Programmes équivalents : Exemple (1)

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

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  $C_3$  :

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

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 :

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

on peut obtenir l'arbre :

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

et *vice versa*.

### Programmes équivalents : Exemple (3)

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

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

on peut obtenir l'arbre :

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

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  $\sigma$  revient à montrer qu'il existe un état  $\sigma'$  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  $\sigma$  tels que  $\sigma(x) \geq 1$  et  $\sigma(y) \geq 1$ .

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

Utilisation de la **réurrence 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 (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  $S$  :

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

**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 (3)

$\succ$  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 (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 :

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

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

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

Deux sous-cas sont possibles.

## Terminaison de programmes (5)

(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 (6)

(2.1) 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,  $\sigma[x \leftarrow m - n] \prec \sigma$ , 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 (7)

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

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

## Déterminisme (1)

### Proposition

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

*Exercice* : Le prouver par récurrence bien fondée en utilisant la notion de sous-arbre.

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** :

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

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

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  $\langle c, \sigma \rangle \rightarrow \sigma'$ .

$$\forall \sigma \in \mathcal{V}[\mathbb{Z}] \quad P(\langle \text{skip}, \sigma \rangle \rightarrow \sigma)$$

et  $\forall a \in E_A \quad \forall \sigma \in \mathcal{V}[\mathbb{Z}] \quad \forall n \in \mathbb{Z} \quad \forall x \in V \quad \langle a, \sigma \rangle \rightsquigarrow n \Rightarrow P(\langle x := a, \sigma \rangle \rightarrow \sigma[x \leftarrow n])$

et  $\forall c_1, c_2 \in E_C \quad \forall \sigma, \sigma', \sigma'' \in \mathcal{V}[\mathbb{Z}]$

$$(P(\langle c_1, \sigma \rangle \rightarrow \sigma') \text{ et } P(\langle c_2, \sigma' \rangle \rightarrow \sigma'')) \Rightarrow P(\langle c_1; c_2, \sigma \rangle \rightarrow \sigma'')$$

et  $\forall c_1, c_2 \in E_C \quad \forall \sigma, \sigma' \in \mathcal{V}[\mathbb{Z}] \quad \forall b \in E_B$

$$(\langle b, \sigma \rangle \rightsquigarrow \text{true} \text{ et } P(\langle c_1, \sigma \rangle \rightarrow \sigma')) \Rightarrow P(\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma')$$

et  $\forall c_1, c_2 \in E_C \quad \forall \sigma, \sigma' \in \mathcal{V}[\mathbb{Z}] \quad \forall b \in E_B$

$$(\langle b, \sigma \rangle \rightsquigarrow \text{false} \text{ et } P(\langle c_2, \sigma \rangle \rightarrow \sigma')) \Rightarrow P(\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma')$$

et  $\forall c \in E_C \quad \forall \sigma \in \mathcal{V}[\mathbb{Z}] \quad \forall b \in E_B \quad \langle b, \sigma \rangle \rightsquigarrow \text{false} \Rightarrow P(\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma)$

et  $\forall c \in E_C \quad \forall \sigma, \sigma', \sigma'' \in \mathcal{V}[\mathbb{Z}] \quad \forall b \in E_B$

$$(\langle b, \sigma \rangle \rightsquigarrow \text{true} \text{ et } \boxed{P(\langle c, \sigma \rangle \rightarrow \sigma')} \text{ et } \boxed{P(\langle \text{while } b \text{ do } c, \sigma' \rangle \rightarrow \sigma'')})$$

$$\Rightarrow P(\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma'')$$

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

### Déterminisme (3)

PREUVE :

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

- Si  $c$  est l'instruction **skip**, alors le seul arbre d'inférence possible pour  $\langle c, \sigma \rangle \rightarrow \sigma_1$  et  $\langle c, \sigma \rangle \rightarrow \sigma_2$  est :

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

et on a alors  $\sigma_1 = \sigma_2$ .

## Déterminisme (4)

- Si  $c$  est l'instruction  $x := a$ , alors on a :

$$(C_2) \frac{(A_i) \frac{\vdots}{\langle a, \sigma \rangle \rightsquigarrow n_1}}{\langle x := a, \sigma \rangle \rightarrow \sigma[x \leftarrow n_1]}$$

et donc  $\sigma_1 = \sigma[x \leftarrow n_1]$ . L'arbre de  $\langle c, \sigma \rangle \rightarrow \sigma_2$  a nécessairement une forme identique :

$$(C_2) \frac{(A_j) \frac{\vdots}{\langle a, \sigma \rangle \rightsquigarrow n_2}}{\langle x := a, \sigma \rangle \rightarrow \sigma[x \leftarrow n_2]}$$

et donc  $\sigma_2 = \sigma[x \leftarrow n_2]$ .

Or, puisque l'évaluation des expressions arithmétiques est déterministe, on a  $n_1 = n_2$  et donc  $\sigma_1 = \sigma_2$ .

## Déterminisme (5)

- Si  $c$  est l'instruction  $c_1; c_2$ , on a l'arbre :

$$(C_3) \frac{(C_i) \frac{\vdots}{\langle c_1, \sigma \rangle \rightarrow \sigma'_1} \quad (C_j) \frac{\vdots}{\langle c_2, \sigma'_1 \rangle \rightarrow \sigma_1}}{\langle c_1; c_2, \sigma \rangle \rightarrow \sigma_1}$$

De même, l'arbre de  $\langle c, \sigma \rangle \rightarrow \sigma_2$  a nécessairement une forme identique :

$$(C_3) \frac{(C_{i'}) \frac{\vdots}{\langle c_1, \sigma \rangle \rightarrow \sigma'_2} \quad (C_{j'}) \frac{\vdots}{\langle c_2, \sigma'_2 \rangle \rightarrow \sigma_2}}{\langle c_1; c_2, \sigma \rangle \rightarrow \sigma_2}$$

Or, par hypothèse d'induction, on a  $P(\langle c_1, \sigma \rangle \rightarrow \sigma'_1)$  et  $P(\langle c_2, \sigma'_1 \rangle \rightarrow \sigma_1)$ , et il vient donc  $\sigma'_1 = \sigma'_2$  et  $\sigma_1 = \sigma_2$ .

## Déterminisme (6)

- Si  $c$  est l'instruction **if  $b$  then  $c_1$  else  $c_2$** .

(1) Si  $b$  s'évalue à **true**, alors on a :

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

De même, l'arbre de  $\langle c, \sigma \rangle \rightarrow \sigma_2$  a nécessairement une forme identique :

$$(C_4) \frac{(B_i) \frac{\vdots}{\langle b, \sigma \rangle \rightsquigarrow \text{true}} \quad (C_{j'}) \frac{\vdots}{\langle c_1, \sigma \rangle \rightarrow \sigma_2}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma_2}$$

Or, par hypothèse d'induction, on a  $P(\langle c_1, \sigma \rangle \rightarrow \sigma_1)$  et il vient donc  $\sigma_1 = \sigma_2$ .

(2) Raisonnement similaire lorsque  $b$  s'évalue à **false**.

## Déterminisme (7)

- Si  $c$  est l'instruction **while**  $b$  **do**  $c$ .

(1) Si  $b$  s'évalue à **false**, alors on a  $(C_6) \frac{(B_i) \frac{\vdots}{\langle b, \sigma \rangle \rightsquigarrow \text{false}}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma}$  qui est aussi l'arbre de  $\langle c, \sigma \rangle \rightarrow \sigma_2$  ce qui permet de conclure.

(2) Si  $b$  s'évalue à **true**, alors on a :

$$(C_7) \frac{(B_i) \frac{\vdots}{\langle b, \sigma \rangle \rightsquigarrow \text{true}} \quad (C_j) \frac{\vdots}{\langle c, \sigma \rangle \rightarrow \sigma'_1} \quad (C_k) \frac{\vdots}{\langle \text{while } b \text{ do } c, \sigma'_1 \rangle \rightarrow \sigma_1}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma_1}$$

L'arbre de  $\langle c, \sigma \rangle \rightarrow \sigma_2$  a nécessairement une forme identique :

$$(C_7) \frac{(B_i) \frac{\vdots}{\langle b, \sigma \rangle \rightsquigarrow \text{true}} \quad (C_{j'}) \frac{\vdots}{\langle c, \sigma \rangle \rightarrow \sigma'_2} \quad (C_{k'}) \frac{\vdots}{\langle \text{while } b \text{ do } c, \sigma'_2 \rangle \rightarrow \sigma_2}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma_2}$$

Or, par hypothèse d'induction, on a  $P(\langle c, \sigma \rangle \rightarrow \sigma'_1)$  et  $P(\langle \text{while } b \text{ do } c, \sigma'_1 \rangle \rightarrow \sigma_1)$  et il vient donc  $\sigma'_1 = \sigma'_2$  et  $\sigma_1 = \sigma_2$ .

## 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  $\sigma$  conduit dans un état  $\sigma'$  où il restera à exécuter  $c'$ .

## Relation de transition

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

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

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

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

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

## 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** :  $\langle \text{skip}, \sigma \rangle$

$\langle c, \sigma \rangle \xrightarrow{*} \langle e', \sigma' \rangle$  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  $c = c_0$ ,  $\sigma = \sigma_0$ ,  $c' = c_k$  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  $c = c_0$ ,  $\sigma = \sigma_0$ ,  $c' = c_k$  et  $\sigma' = \sigma_k$ .

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

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

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

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

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

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  $\langle x := a, \sigma \rangle \xrightarrow{*} \sigma[x \leftarrow n]$ .

## Equivalence sémantique (2)

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

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

$$\langle c_1, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma_1 \rangle \quad (9)$$

$$\langle c_2, \sigma_1 \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma_2 \rangle \quad (10)$$

D'après le lemme précédent, à partir de la séquence (9), on a :

$$\langle c_1; c_2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}; c_2, \sigma_1 \rangle \quad (11)$$

La règle  $S_3$  permet de construire la transition :

$$\langle \text{skip}; c_2, \sigma_1 \rangle \hookrightarrow \langle c_2, \sigma_1 \rangle \quad (12)$$

et à partir de (11), (12) et (10), on peut obtenir la séquence :

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

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

### Equivalence sémantique (3)

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

Par hypothèse d'induction,  $\langle c_1, \sigma \rangle \xrightarrow{*} \sigma_1$  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  $\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \xrightarrow{*} \sigma_1$ .

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

Raisonnement similaire au cas précédent.

## Equivalence sémantique (4)

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

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 (5)

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

Par hypothèse d'induction on a :

$$\langle c, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma_1 \rangle \quad (13)$$

$$\langle \text{while } b \text{ do } c, \sigma_1 \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma_2 \rangle \quad (14)$$

A partir de (13), on obtient :

$$\langle c; \text{while } b \text{ do } c, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}; \text{while } b \text{ do } c, \sigma_1 \rangle \quad (15)$$

$$\text{règle } S_3 \quad \langle \text{skip}; \text{while } b \text{ do } c, \sigma_1 \rangle \hookrightarrow \langle \text{while } b \text{ do } c, \sigma_1 \rangle \quad (16)$$

$$\text{règle } S_7 \quad \langle \text{while } b \text{ do } c, \sigma \rangle \hookrightarrow \langle c; \text{while } b \text{ do } c, \sigma \rangle \quad (17)$$

A partir de (17), (15), (16) et (14), on obtient :

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

## Equivalence sémantique (6)

**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  $\sigma'$  tels que

$\langle c'_1, \sigma_1 \rangle \xrightarrow{k'_1} \sigma'$ ,  $\langle c_2, \sigma'_1 \rangle \xrightarrow{k'_2} \sigma''$  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  $k_2 = k_0$ .

## Equivalence sémantique (7)

**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 :

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

## Equivalence sémantique (8)

- $\langle c_1; c_2, \sigma \rangle \hookrightarrow \langle c'_1; c_2, \sigma_1 \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma' \rangle$  avec  $\langle c_1, \sigma \rangle \hookrightarrow \langle c'_1, \sigma_1 \rangle$ .

D'après le lemme précédent, il existe une valuation  $\sigma_0$  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  $\langle c_1, \sigma \rangle \rightsquigarrow \sigma_0$  et  $\langle c_2, \sigma_0 \rangle \rightsquigarrow \sigma'$  et on conclut en construisant l'arbre :

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

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

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

## Equivalence sémantique (9)

- $\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \hookrightarrow \langle c_1, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma' \rangle$  avec  $\langle b, \sigma \rangle \rightsquigarrow \text{true}$ .

Par hypothèse d'induction on a  $\langle c_1, \sigma \rangle \rightsquigarrow \sigma'$  et on conclut en construisant l'arbre :

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

- $\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \hookrightarrow \langle c_2, \sigma \rangle \hookrightarrow \dots \hookrightarrow \langle \text{skip}, \sigma' \rangle$  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 :

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

## Equivalence sémantique (10)

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

D'autre part on montre que

$\text{while } b \text{ do } c \equiv \text{if } b \text{ then } c; \text{while } b \text{ do } c \text{ else skip}$  et on conclut en construisant l'arbre :

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