

Analyse de programmes et Sémantique

Master Sciences et Technologies – Mention Informatique – 2006/2007
Spécialité Science et Technologie du logiciel

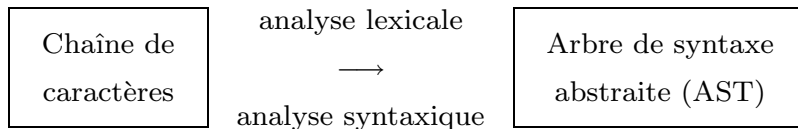
Sémantique opérationnelle

Mathieu Jaume (Mathieu.Jaume@lip6.fr)

1

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**

2

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
 - ...

3

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.

4

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.

- *sémantique axiomatique* (vision déclarative)

programme = "transformateur" de propriétés (sur les états)

Logique de Hoare

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

5

Rappel : Systèmes d'inférence

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

jugements de \mathcal{J} : $(R) \frac{j_1 \quad j_2 \quad \cdots \quad j_n}{j}$

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 $\Phi[\mathcal{J}]$:

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

$\Phi[\mathcal{J}]$ 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.

6

Systèmes d'inférence et Définitions inductives

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

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

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

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

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

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

PREUVE : exercice ...

7

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 :

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

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

8

Expressions Arithmétiques : Syntaxe abstraite – Implantation

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

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;;
```

9

Induction

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

Théorème

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

PREUVE : exercice ...

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

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

Induction sur les expressions arithmétiques

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

11

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

```
exception err;;
```

12

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 : $\llbracket n \rrbracket = n$

$v_1 \left(\begin{array}{c} \llbracket + \rrbracket \\ \llbracket - \rrbracket \\ \llbracket \times \rrbracket \end{array} \right) v_2$	$v_2 \in \mathbb{Z}$	Err	$v_1 \llbracket / \rrbracket v_2$	$v_2 \in \mathbb{Z} \setminus \{0\}$	0	Err
$v_1 \in \mathbb{Z}$	$v_1 \left(\begin{array}{c} + \\ - \\ \times \end{array} \right) v_2$	Err	$v_1 \in \mathbb{Z}$	v_1/v_2	Err	Err
Err	Err	Err	Err	Err	Err	Err

13

Schéma d'interprétation des expressions arithmétiques

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

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

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

14

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
```

15

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

16

Règles d'évaluation à grands pas

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

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

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

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

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

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

17

Evaluation : exemple

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

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

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

18

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

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

PREUVE : Par **induction** sur a .

19

Déterminisme de l'évaluation des expressions arithmétiques

Proposition :

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

PREUVE : Par **induction** sur a .

20

Equivalence sémantique

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

PREUVE. ...

21

Evaluation des expressions arithmétiques : variables (1)

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

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

Proposition :

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

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

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

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

22

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

23

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 :

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

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

24

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).

25

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 :

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

26

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

27

Expressions booléennes

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

Définition inductive de l'ensemble E_B des expressions booléennes.

Jugements :

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

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

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

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

28

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

29

Schéma d'interprétation des expressions booléennes

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

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

30

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 $v \in \mathbb{IB} \cup \{\text{Err}\}$ étant donnée une valuation σ .

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

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

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

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

31

Expressions booléennes : Sémantique opérationnelle à grands pas (2)

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

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

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

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

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

32

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

33

Choix des règles (2)

Exemple

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

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

34

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 ...

35

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 :

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

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

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

36

Constructions impératives : Exemple (PGCD)

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

$$\begin{array}{c}
 \frac{\frac{(C_2) \frac{}{y := y - x} \quad (C_2) \frac{}{x := x - y}}{(C_4) \frac{}{\text{if } x \leq y \text{ then } y := y - x \text{ else } x := x - y}}{(C_5) \frac{}{\text{while not } (x = y) \text{ do if } x \leq y \text{ then } y := y - x \text{ else } x := x - y}}
 \end{array}$$

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

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

37

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 .

38

Transitions

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

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

L'exécution de l'instruction c dans l'état σ_1 conduit à l'état σ_2 .

Description de l'exécution d'une instruction par le changement d'état qu'elle provoque.

Changement d'état

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

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

39

Sémantique opérationnelle à grands pas

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

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

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

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

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

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

40

Sémantique opérationnelle à grands pas : Exemple

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

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 :

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

41

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.

42

Boucle et Terminaison

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

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

Exemple `while true do skip`

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

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

43

Programmes équivalents

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

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

Exemples :

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

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

44

Terminaison de programmes (1)

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

Exemple Terminaison du programme Euclid :

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

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

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

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

45

Ordres (Rappel)

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)

46

Ordres bien fondés (Rappel)

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).

PREUVE. Exercice.

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.

Réurrence bien fondée (Rappel)

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

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

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 $E_1 \times E_2 \times \dots \times E_n$ 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 $E_1 \times E_2 \times \dots \times E_n$ est bien fondé.

49

Ordre lexicographique (2)

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^* !

50

Terminaison de programmes (1) Rappel

Terminaison du programme Euclid :

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

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

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

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 ?

51

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 :

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

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

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

Remarque

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

52

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.

53

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 ...

54

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** :

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

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

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** !

55

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

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

56

Déterminisme (3)

PREUVE :

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

57

Non déterminisme (1)

On ajoute la règle $(C_{ND}) \frac{c_1 \quad c_2}{c_1 \text{ or } c_2}$ au système d'inférence définissant E_C .

Sémantique informelle : exécuter c_1 or c_2 , c'est exécuter c_1 ou c_2

Sémantique opérationnelle à grands pas

$$(C_{ND}^1) \frac{\langle c_1, \sigma \rangle \rightarrow \sigma'}{\langle c_1 \text{ or } c_2, \sigma \rangle \rightarrow \sigma'} \quad (C_{ND}^2) \frac{\langle c_2, \sigma \rangle \rightarrow \sigma'}{\langle c_1 \text{ or } c_2, \sigma \rangle \rightarrow \sigma'}$$

Exemples :

$\langle x := 1 \text{ or } (x := 3; x := x \times x), \sigma \rangle \rightarrow ?$

$\langle x := 1 \text{ or while true do skip}, \sigma \rangle \rightarrow ?$

58

Non déterminisme (2)

- $\langle x := 1 \text{ or } (x := 3; x := x \times x), \sigma \rangle \rightarrow ?$

$$(C_{ND}^1) \frac{(C_2) \overline{\langle x := 1, \sigma \rangle \rightarrow \sigma[x \leftarrow 1]}}{\langle x := 1 \text{ or } (x := 3; x := x \times x), \sigma \rangle \rightarrow \sigma[x \leftarrow 1]}$$

$$(C_{ND}^2) \frac{(C_2) \overline{\langle x := 3; x := x \times x, \sigma \rangle \rightarrow \sigma[x \leftarrow 9]}}{\langle x := 1 \text{ or } (x := 3; x := x \times x), \sigma \rangle \rightarrow \sigma[x \leftarrow 9]}$$

Comme son nom l'indique, l'ajout de la construction non-déterministe **or** fait perdre la propriété du déterminisme de l'exécution des instructions : l'état que l'on peut obtenir après exécution d'un programme n'est pas unique.

59

Non déterminisme (3)

- $\langle x := 1 \text{ or while true do skip}, \sigma \rangle \rightarrow ?$

En utilisant la règle C_{ND}^1 , on obtient $\sigma[x \leftarrow 1]$.

En utilisant la règle C_{ND}^2 , le programme boucle et il n'existe pas d'état σ' tel que $\langle \text{while true do skip}, \sigma \rangle \rightarrow \sigma'$

Le seul état σ' tel que $\langle x := 1 \text{ or while true do skip}, \sigma \rangle \rightarrow \sigma'$ est donc $\sigma[x \leftarrow 1]$.

L'ajout de la construction non déterministe **or** permet, dans certains cas, de "supprimer les boucles infinies".

60

Parallélisme ?

On ajoute la règle $(C_P) \frac{c_1 \quad c_2}{c_1 \parallel c_2}$ au système d'inférence définissant E_C .

Sémantique informelle : exécuter $c_1 \parallel c_2$, c'est exécuter c_1 et c_2 en parallèle.

Sémantique opérationnelle à grands pas

$$\boxed{(C_P^1) \frac{\langle c_1, \sigma \rangle \rightarrow \sigma' \quad \langle c_2, \sigma' \rangle \rightarrow \sigma''}{\langle c_1 \parallel c_2, \sigma \rangle \rightarrow \sigma''} \quad (C_P^2) \frac{\langle c_2, \sigma \rangle \rightarrow \sigma' \quad \langle c_1, \sigma' \rangle \rightarrow \sigma''}{\langle c_1 \parallel c_2, \sigma \rangle \rightarrow \sigma''}}$$

Problème : Ces règles ne rendent pas compte de l'entrelacement des exécutions de c_1 et c_2 .

La sémantique opérationnelle à grands pas n'est pas assez fine pour prendre en compte le parallélisme.

61

Blocs (1)

On ajoute $(C_B) \frac{c}{\text{begin } D \text{ c end}}$ au syst. d'inférence définissant E_C .

D : Déclaration des **variables locales** au bloc.

$$\boxed{D ::= \text{var } x := a; D \mid \varepsilon}$$

Description de l'exécution de **begin** D **c end** dans l'état σ .

(1). Construction d'un état σ' dans lequel on exécute c .

$$\langle D, \sigma \rangle \mapsto \sigma'$$

$$\boxed{(E_1) \frac{}{\langle \varepsilon, \sigma \rangle \mapsto \sigma} \quad (E_2) \frac{\langle D, \sigma[x \leftarrow \mathcal{A}[[a]]_\sigma] \rangle \mapsto \sigma'}{\langle \text{var } x := a; D, \sigma \rangle \mapsto \sigma'}}$$

(2). Exécution de c dans σ' :

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

62

Blocs (2)

(3). “reconstitution” à partir de σ'' de l'état initial σ pour les variables locales déclarées dans D .

$$\sigma''[\vartheta(D) \Leftarrow \sigma]$$

$$\vartheta(D) = \begin{cases} \emptyset & \text{si } D = \varepsilon \\ \{x\} \cup \vartheta(D') & \text{si } D = \text{var } x := a; D' \end{cases}$$

$$\sigma''[\vartheta(D) \Leftarrow \sigma](x) = \begin{cases} \sigma(x) & \text{si } x \in \vartheta(D) \\ \sigma''(x) & \text{sinon} \end{cases}$$

$$(C_B) \frac{\langle D, \sigma \rangle \mapsto \sigma' \quad \langle c, \sigma' \rangle \rightarrow \sigma''}{\langle \text{begin } D \ c \ \text{end}, \sigma \rangle \rightarrow \sigma''[\vartheta(D) \Leftarrow \sigma]}$$

63

Blocs (3)

Exemple : Programme c

begin var $y := 1$;

($x := 1$;

Programme $c' \rightarrow$ begin var $x := 2; y := x + 1$ end;

$x := y + x$)

end

64

Blocs (4)

$$(C_B) \frac{(E_2) \frac{\vdots}{\langle \text{var } y := 1, \sigma \rangle \mapsto \sigma[y \leftarrow 1]} (C_3) \frac{\nabla_1 \quad \nabla_2}{\langle x := 1; c'; x := y + x, \sigma[y \leftarrow 1] \rangle \rightarrow \sigma_1}}{\langle c, \sigma \rangle \rightarrow \sigma_2}$$

Arbre ∇_1 :

$$(C_2) \frac{}{\langle x := 1, \sigma[y \leftarrow 1] \rangle \rightarrow \sigma \begin{bmatrix} x \leftarrow 1 \\ y \leftarrow 1 \end{bmatrix}}$$

Arbre ∇_2 :

$$(C_3) \frac{\nabla_3 \quad \nabla_4}{\left\langle c'; x := y + x, \sigma \begin{bmatrix} x \leftarrow 1 \\ y \leftarrow 1 \end{bmatrix} \right\rangle \rightarrow \sigma_1}$$

65

Blocs (5)

Arbre ∇_3 :

$$(C_B) \frac{(E_2) \frac{\vdots}{\left\langle \text{var } x := 2, \sigma \begin{bmatrix} x \leftarrow 1 \\ y \leftarrow 1 \end{bmatrix} \right\rangle \mapsto \sigma \begin{bmatrix} x \leftarrow 2 \\ y \leftarrow 1 \end{bmatrix}} (C_2) \frac{}{\left\langle y := x + 1, \sigma \begin{bmatrix} x \leftarrow 2 \\ y \leftarrow 1 \end{bmatrix} \right\rangle \rightarrow \sigma \begin{bmatrix} x \leftarrow 2 \\ y \leftarrow 3 \end{bmatrix}}}{\left\langle c', \sigma \begin{bmatrix} x \leftarrow 1 \\ y \leftarrow 1 \end{bmatrix} \right\rangle \rightarrow \sigma \begin{bmatrix} x \leftarrow 1 \\ y \leftarrow 3 \end{bmatrix}}$$

Arbre ∇_4 :

$$(C_2) \frac{}{\left\langle x := y + x, \sigma \begin{bmatrix} x \leftarrow 1 \\ y \leftarrow 3 \end{bmatrix} \right\rangle \rightarrow \sigma \begin{bmatrix} x \leftarrow 4 \\ y \leftarrow 3 \end{bmatrix}}$$

Donc $\sigma_1 = \sigma \begin{bmatrix} x \leftarrow 4 \\ y \leftarrow 3 \end{bmatrix}$ et $\sigma_2 = \sigma[x \leftarrow 4]$.

66