Université Bordeaux
Licence, module Compilation, 2017/2018

SÉMANTIQUE

1 Sémantique à petits pas

1.1 Langage IMP : syntaxe

Le langage source, IMP [1], est un langage de programmation impératif très simple, où toutes les variables sont de type entier. Voici une grammaire de ce langage (au format de Bison) :

```
 E: E Pl T
  | E Mo T
  | T


 T: T Mu F
  | F


 F: '(' E ')'
  | I
  | V


C : V Af E
  | Sk
  | '(' C ')'
  | If E Th C El C
  | Wh E Do C
  |  C Se C
```

où les symboles `I V Af Sk Se If Th El Wh Do Pl Mo Mu` représentent les unités lexicales suivantes :
`I` : une suite de chiffres, non-vide, commençant par un chiffre non-nul
`V` : un identificateur de variable

symbole	Af	Sk	Se	If	Th	El	Wh	Do	Pl	Mo	Mu
lexeme	:=	Skip	;	if	then	else	while	do	+	-	*

(i.e. ces unités lexicales ne comportent qu'un lexème, qui est donné dans le tableau). On appelle *commande* un mot engendré par le non-terminal C ; une *commande atomique* est une commande qui n'est pas décomposable sous la forme c_1 `Se` c_2 pour des commandes c_1, c_2 ; une *expression* est un mot e engendré par le non-terminal E.

1. qui est dû à J. Goubault-Larrecq, cours de sémantique et compilation, licence 1, ENS Cachan, 2013.

1.2 Langage IMP : sémantique à petits pas

La sémantique de IMP est définie ci-dessous. Le procédé de définition employé s'appelle une sémantique *opérationnelle à petits pas*. Notons V un ensemble de variables [2]. On appelle *environnement* sur V toute application $\rho : V \to \mathbb{Z}$. Pour tout mot w sur l'alphabet $\{0, 1, \dots, 9\}$, on note $\nu(w) \in \mathbb{N}$ l'entier dénoté par w en base 10. La *valeur* d'une expression e dans un environnement ρ est définie par :

$$[\![x]\!]\rho = \rho(x) \quad \text{pour toute variable } x \in V,$$

$$[\![w]\!]\rho = \nu(w) \text{ pour tout mot } w \in \{1, \dots, 9\}\{0, 1, \dots, 9\}^* \cup \{0\},$$

$$[\![e_1 \texttt{ Pl } e_2]\!]\rho = [\![e_1]\!]\rho + [\![e_2]\!]\rho, \quad [\![e_1 \texttt{ Mo } e_2]\!]\rho = [\![e_1]\!]\rho - [\![e_2]\!]\rho, \quad [\![e_1 \texttt{ Mu } e_2]\!]\rho = ([\![e_1]\!]\rho) * ([\![e_2]\!]\rho), \quad [\![(e)]\!]\rho = [\![e]\!],$$

pour toutes expressions e, e_1, e_2.
Une suite de commandes est un mot sur l'alphabet des commandes. On note $\cdot$ le produit de concaténation des suites de commandes. Pour toute variable $x \in V$, expression e, commande atomique c_0, commande c, suite de commandes C, on pose :

$$\begin{array}{rcl}
((c) \cdot C, \rho) \to & (c \cdot C, \rho) & \\
(c \texttt{ Se } c_0 \cdot C, \rho) \to & (c \cdot c_0 \cdot C, \rho) & \\
(\texttt{Sk } \cdot C, \rho) \to & (C, \rho) & \\
(x \texttt{ Af } e \cdot C, \rho) \to & (C, \rho[x \mapsto [\![e]\!]\rho]) & \\
(\texttt{If } e \texttt{ Th } c \texttt{ El } c_0 \cdot C, \rho) \to & (c \cdot C, \rho) & \text{si } [\![e]\!]\rho \neq 0 \\
(\texttt{If } e \texttt{ Th } c \texttt{ El } c_0 \cdot C, \rho) \to & (c_0 \cdot C, \rho) & \text{si } [\![e]\!]\rho = 0 \\
(\texttt{Wh } e \texttt{ Do } c_0 \cdot C, \rho) \to & (c_0 \cdot \texttt{Wh } e \texttt{ Do } c_0 \cdot C, \rho) & \text{si } [\![e]\!]\rho \neq 0 \\
(\texttt{Wh } e \texttt{ Do } c_0 \cdot C, \rho) \to & (C, \rho) & \text{si } [\![e]\!]\rho = 0.
\end{array}$$

La sémantique d'une commande c est définie par : $[\![c]\!]\rho = \rho'$ si et seulement si

$$(c, \rho) \to^* (\varepsilon, \rho').$$

2 Plus petits points fixes

Définition 2.1 *Soit $(E, \leq)$ un ensemble ordonné.*
$(E, \leq)$ est un <u>treillis</u> ssi :
- *toute paire d'éléments admet une borne supérieure*
- *et toute paire d'éléments admet une borne inférieure.*

Définition 2.2 *Un ensemble ordonné $(E, \leq)$ est un* treillis complet *(ou sup-complet), si toute partie de E admet une borne supérieure.*

<u>Exemple :</u> $(\mathcal{P}(F), \subseteq)$ est un treillis complet.

Remarque 2.3 *Si $(E, \leq)$ vérifie la définition 2.2 alors,*

2. On assimile chaque identificateur, qui est un mot concret, à une seule variable.

** il s'agit bien d'un treillis : il suffit de vérifier que toute paire $\{a, b\}$ possède bien ue borne inférieure. Or*

$$\sup(\{m \in E, m \leq a \textit{ et } m \leq b\})$$

est bien la borne inférieure de $\{a, b\}$.
** il admet un plus petit 'elément, $\perp = \sup(\emptyset)$*
** il admet un plus grand élément, $\top = \sup(E)$*

Définition 2.4 *Une application $f : (E_1, \leq_1) \to (E_2 \leq_2)$ est dite continue ssi, pour toute partie $A \subseteq E_1$, admettant une borne supérieure $a = \sup(A)$, $\{f(x) \mid x \in A\}$ admet aussi une borne supérieure et elle est égale à $f(a)$.*

Remarque 2.5 ** Lorsque les $(E_i, \leq_i)$ $(1 \leq i \leq 2)$ sont des treillis complets, f est continue ssi, pour toute partie $A \subseteq E_1$, $f(\sup A) = \sup(f(A))$*
** Si f est continue, f est croissante.*
On appelle "point fixe" de $f : E \to E$, tout $x \in E$ tel que $f(x) = x$.

Théorème 2.6 *Si f est une application continue d'un treillis complet dans lui-même, alors f a un plus petit point fixe. Ce plus petit point fixe est égal à*

$$\sup(\{f^n(\perp) \mid n \in \mathbb{N}\}).$$

Preuve: Posons $x = \sup(\{f^n(\perp) \mid n \in \mathbb{N}\})$
Comme f est continue, $f(x) = \sup(\{f^{n+1}(\perp) \mid n \in \mathbb{N}\})$.
Mais $\{f^n(\perp) \mid n \in \mathbb{N}\}$ et $\{f^{n+1}(\perp) \mid n \in \mathbb{N}\}$ ont les mêmes majorants, donc la même borne supérieure. Donc $f(x) = x$.
Soit $y = f(y)$
$\perp \leq y \;\Rightarrow\; \forall n \in \mathbb{N},\ f^n(\perp) \leq f^n(y) = y$
Donc y est un majorant de $\{f^n(\perp) \mid n \in \mathbb{N}\}$, donc $x \leq y$.
□

3 Sémantique à grands pas

3.1 Langage IMP : sémantique à grands pas

$$\text{Valeur}\ \frac{[\![e]\!]\rho = v}{\rho/e \vdash \rho/v} \qquad \text{Parenthèses}\ \frac{\rho/c \vdash \rho'}{\rho/(c) \vdash \rho'} \qquad \text{Séquence}\ \frac{\rho/c_1 \vdash \rho_1, \quad \rho_1/c_2 \vdash \rho_2}{\rho/c_1 \texttt{ Se } c_2 \vdash \rho_2} \qquad \text{Skip}\ \frac{}{\rho/\texttt{ Sk } \rho}$$

$$\text{Affectation}\ \frac{\rho/e \vdash \rho/v}{\rho/x \texttt{ Af } e \vdash \rho[x \mapsto v]} \qquad \text{If true}\ \frac{\rho/e \vdash \rho/v, \quad v \neq 0, \quad \rho/c_1 \vdash \rho_1}{\rho/\texttt{If } e \texttt{ Th } c_1 \texttt{ El } c_2 \vdash \rho_1} \qquad \text{If false}\ \frac{\rho/e \vdash \rho/0, \quad \rho/c_2 \vdash \rho_2}{\rho/\texttt{If } e \texttt{ Th } c_1 \texttt{ El } c_2 \vdash \rho_2}$$

$$\text{While true}\ \frac{\rho/e \vdash \rho/v, \quad v \neq 0, \quad \rho/c \vdash \rho_1, \quad \rho_1/\texttt{Wh } e \texttt{ Do } c \vdash \rho_2}{\rho/\texttt{Wh } e \texttt{ Do } c \vdash \rho_2} \qquad \text{While false}\ \frac{\rho/e \vdash \rho/0,}{\rho/\texttt{Wh } e \texttt{ Do } c \vdash \rho}$$

Fixons un ensemble fini de variables V. Soit F l'ensemble des triplets de la forme

$$(\rho, c, \rho')$$

où $\rho : V \to \mathbb{Z}$ est une application, c est une commande de `IMP` et $\rho' : V \to \mathbb{Z}$ est une application. Soit F' l'ensemble des quadruplets de la forme

$$(\rho, e, \rho', v)$$

où $\rho, \rho' : V \to \mathbb{Z}$ sont des applications, e est une expression de `IMP` et $v \in \mathbb{Z}$. Soient

$$E := \mathcal{P}(F \cup F') \text{ et } \Phi : E \to E$$

l'application définie par

$$\Phi(P) := \{f \in F \cup F' \mid \exists f_1, \ldots, f_k \in F \cup F', \frac{f_1, \ldots, f_k}{f} \text{ est une instance de règle}\}.$$

On vérifie que Φ est continue. Par le théorème 2.6 Φ a un *plus petit point fixe* de Φ. On note $\vdash$ ce plus petit point fixe de Φ. La sémantique d'une commande c est définie par : $[\![c]\!]\rho = \rho'$ si et seulement si $(\rho, c, \rho') \in \vdash$, ce que l'on note aussi

$$\rho / c \vdash \rho'.$$

3.2 Langage *Léa* : syntaxe

Définition formelle Le langage *Léa* (dans une version légérement simplifiée) est défini (syntaxiquement) par la grammaire suivante (au format de Bison) :

Programme

```
program:
variable_declaration_part
procedure_and_function_definition_part
TOKEN_BEGIN
statement_list
TOKEN_END
```

Définition des types

```
type:   simple_type
        | subrange_type
        | array_type
        | pointer_type

simple_type: 'integer'
        | 'boolean'

subrange_type: ZERO '..' INTEGER

array_type: 'array' '[' subrange_type ']' 'of' type

pointer_type: '^' type
```

Déclaration et définition des procédures et des fonctions

```
procedure_and_function_definition_part: procedure_and_function_definition_list
        | %empty

procedure_and_function_definition_list:
        procedure_and_function_definition_list procedure_and_function_definition
        | procedure_and_function_definition

procedure_and_function_definition:
        procedure_and_function_definition_head variable_declaration_part block
        | procedure_and_function_definition_head ';'

procedure_and_function_definition_head:
'procedure' IDENTIFIER '(' argt_part ')'
        | 'function' IDENTIFIER '(' argt_part ')' ':' type

argt_part: argt_list
        | %empty

argt_list: argt_list ',' argt
        | argt

argt: IDENTIFIER ':' type
```

Déclaration des variables typées

```
variable_declaration_part: 'var' variable_declaration_list
        | %empty

variable_declaration_list: variable_declaration_list variable_declaration
        | variable_declaration

variable_declaration: identifier_list ':' type ';'

identifier_list: identifier_list ',' IDENTIFIER
        | IDENTIFIER
```

Déclaration et définition des procédures et des fonctions

```
procedure_and_function_definition_part: procedure_and_function_definition_list
        | %empty

procedure_and_function_definition_list:
procedure_and_function_definition_list procedure_and_function_definition
        | procedure_and_function_definition

procedure_and_function_definition:
procedure_and_function_definition_head variable_declaration_part block
        | procedure_and_function_definition_head ';'

procedure_and_function_definition_head: 'procedure' IDENTIFIER '(' argt_part ')'
        | 'function' IDENTIFIER '(' argt_part ')' ':' type

argt_part: argt_list
| %empty

argt_list: argt_list ',' argt
        | argt

argt: IDENTIFIER ':' type
```

Blocs

```
block: 'begin' statement_list 'end'
```

Instructions

```
statement_list: statement_list statement
        | statement

statement: simple_statement
        | structured_statement

simple_statement:assignment_statement
        | procedure_statement
        | new_statement
        | dispose_statement
        | return_statement
        | read_statement
        | write_statement

assignment_statement: variable_access ':=' expression ';'
```

```
procedure_statement: procedure_expression ';'

procedure_expression: IDENTIFIER '(' expression_part ')'

expression_part: expression_list
| %empty

expression_list: expression_list ',' expression
        | expression

new_statement: 'new '(' variable_access ')' ';'

dispose_statement: 'dispose' '(' variable_access ')' ';'

return_statement: 'return' '(' expression ')' ';'

read_statement: 'read' '(' variable_access ')' ';'

write_statement: 'write' '(' expression ')' ';'

structured_statement: block
        | 'if' expression 'then' statement 'else' statement
        |'while' expression 'do' statement
```

Expressions

```
variable_access : IDENTIFIER
        | indexed_variable
        | variable_access '^'

indexed_variable: variable_access '[' expression ']'

expression: variable_access
        | expression bop expression
        | uop expression
        | '(' expression ')'
        | procedure_expression
        | literal
```

Un programme *Léa* est un texte engendré par cette grammaire qui vérifie les contraintes supplémentaires suivantes :
- toute expression est typable
- toute affectation a des membres gauche et droit de *types compatibles* (ici nous entendons par *type* un mot engendré par le non-terminal `type` dans la grammaire de *Léa* ; la notion de compatibilité entre deux types est définie plus loin, au § *typage*)
- aucune affectation, ni test d'égalité, ne se fait entre expressions de type tableau
- tout appel de fonction (resp. procédure) a des paramètres d'appel qui ont la *même suite de types* que la suite des paramètres formels de la fonction (resp. procédure).
- pour toute expression e, si l'instruction `return(e)` se trouve dans le corps d'une fonction qui renvoie un type τ, alors e est de type τ.
- dans chaque déclaration de fonction (ou de procédure), l'ensemble des paramètres est disjoint de l'ensemble des variables locales ;
- un nom de fonction/procédure ne fait l'objet que d'une seule définition ; il peut faire l'objet d'une déclaration et d'une définition ; dans ce cas la déclaration précède la définition ;
- tous les identificateurs (variables, fonctions et procédures) sont définis avant d'être utilisés.

3.3 Langage *Léa* : sémantique à grands pas

Les objets atomiques manipulés par un programme *Léa* sont les entiers (en fait des éléments de $\mathbb{Z}/\mathrm{N}\mathbb{Z}$, avec $\mathrm{N} := 2^{32}$) les booléen $\top, \bot$ et des " adresses" (que nous précisons ci-dessous). Les objets généraux sont les objets atomiques ainsi que les tableaux d'objets et les pointeurs sur objet. La sémantique de *Léa* est définie ci-dessous. Le procédé de définition employé s'appelle une sémantique *opérationnelle à grands pas.*

Idée générale
La sémantique définit l'évolution de l'état d'une machine abstraite sous l'effet d'une commande (ou d'une expression) de *Léa*. La machine possède un ensemble dénombrable infini $\mathcal{V}$ (resp. $\mathcal{D}$) de variables (resp. de noeuds). Un *état* est un 5-uplet (u, G, H, E, v) où G est un environnement global (un vecteur de valeurs pour les variables globale), H ("heap", en Anglais) est un tas, E est un environnement local (un vecteur de valeurs pour les variables locales d'une fonction/procédure) et u (l'entrée) v (la sortie) sont des mots sur l'alphabet $\mathbb{Z}/\mathrm{N}\mathbb{Z}$.

Donnons une définition plus précise

constantes C, l'ensemble des constantes, est l'ensemble $\mathbb{Z}/\mathrm{N}\mathbb{Z} \cup \{\top, \bot\} \cup \{\varepsilon\}$.

types l'ensemble des types, $\mathcal{T}$, est l'ensemble des mots engendré par le non-terminal `type`. Les ensembles $C, \mathcal{V} \times \mathcal{T}, \mathcal{D}$ sont disjoints.

tas H est une fonction, de domaine fini $\mathrm{Dom}(H) = D_H \subseteq \mathcal{D}$ et d'image $\mathrm{Im}(H) \subseteq (C \cup D_H) \bigcup_{d \geq 0} (C \cup D_H)^{R_1 \times \ldots \times R_d}$ où les ensembles R_j sont de la forme $[0, k_j]$ avec $k_j \leq 2^{31} - 1$;

environnement (typé) un environnement typé E est une fonction, de domaine fini $\mathrm{Dom}(E) = D_E \subseteq \mathcal{V} \times \mathcal{T}$ et d'image $\mathrm{Im}(E) \subseteq (C \cup D_H) \cup \bigcup_{d \geq 0} (C \cup D_H)^{R_1 \times \ldots \times R_d}$, où les ensembles R_j sont de la forme $[0, k_j]$ avec $k_j \leq 2^{31} - 1$; on note V_E la première projection de D_E ; D_E (resp. V_E) est l'ensemble des variables typées (resp. l'ensemble des variables) de l'environnement E.

Typage
On définit une relation d'équivalence entre types $\equiv$ par :

$$\frac{\tau \in \mathcal{T}}{\tau \equiv \tau} \quad \frac{k, k' \in [0, 2^{31} - 1]}{[0..k] \equiv [0..k']} \quad \frac{k \in [0, 2^{31} - 1]}{[0..k] \equiv \text{integer}} \quad \frac{k \in [0, 2^{31} - 1]}{\text{integer} \equiv [0..k]}$$

$$\frac{\tau \equiv \sigma, \quad \tau' \equiv \sigma'}{\text{array } \tau \text{ of } \tau' \equiv \text{array } \sigma \text{ of } \sigma'} \quad \frac{\tau \equiv \sigma}{\uparrow \tau \equiv \uparrow \sigma}$$

Deux types τ, τ' sont dits *compatibles* ssi $\tau \equiv \tau'$.

Pour tout état $S = (u, G, H, E, v)$ et expression e, on définit $\mathrm{ty}(S, e)$ le type défini par les règles suivantes :
- le type d'une expression formée d'opérandes qui ont des types simples (`boolean` ou `integer`) et des opérateurs binaires ou unaires est défini comme il est d'usage
- si (x, τ) est une variable typée de $\mathrm{Dom}(E)$ alors $\mathrm{ty}(S, x) = \tau$
- si (x, τ) est une variable typée de $\mathrm{Dom}(G) \setminus \mathrm{Dom}(E)$ alors $\mathrm{ty}(S, x) = \tau$

$$\frac{\mathrm{ty}(S, e_1) \equiv \mathrm{ty}(S, e_2)}{\mathrm{ty}(S, e_1 = e_2) = \text{boolean}} \quad \frac{\mathrm{ty}(S, e_1) = \text{array } [0..k] \text{ of } \tau, \quad \mathrm{ty}(S, e_2) \equiv [0..k]}{\mathrm{ty}(S, e_1[e_2]) = \tau} \quad \frac{\mathrm{ty}(S, e) = \uparrow \tau}{\mathrm{ty}(S, e \uparrow) = \tau}$$

Pour toute fonction f déclarée par

$$f(x_1 : \tau_1, \ldots, x_n : \tau_n) : \tau \text{ var } x'_1 : \tau'_1, \ldots, \text{var } x'_m : \tau'_m c$$

$$\frac{\mathrm{ty}(S, e_1) \equiv \tau_1, \quad \ldots, \mathrm{ty}(S, e_n) \equiv \tau_n}{\mathrm{ty}(S, f(e_1, \ldots, e_n)) = \tau}$$

On dit que l'expression e est bien-typée ssi $\mathrm{ty}(S, e)$ est défini.

Un type τ a une *dimension* qui est une suite finie d'entiers définie par :

$$\dim(\text{boolean}) = \dim(\text{integer}) = \dim(\uparrow \tau') = \varepsilon, \ \ \dim(\text{ array } [0..\text{k}] \text{ of } \tau') = (k+1) \cdot \dim(\tau'),$$

et un *type de base* $\mathrm{tyb}(\tau)$ défini par :

$$\mathrm{tyb}(\text{boolean}) = \text{boolean}, \ \ \mathrm{tyb}(\text{integer}) = \text{integer}, \ \ \mathrm{tyb}(\uparrow \tau') = \uparrow \tau', \ \ \mathrm{tyb}(\text{ array } [0..\text{k}] \text{ of } \tau') = \mathrm{tyb}(\tau').$$

Si le tableau est de dimension $(k_1, \ldots, k_p)$ alors il comportera $\prod_{i=1}^{p} k_i$ "cases" et chaque case contiendra un objet de type $\mathrm{tyb}(\tau)$.
Une *instruction* (ou commande) c fait passer la machine d'un état (G, H, E) [3] à un état (G', H', E') ce que l'on note

$$G, H, E/c \rightarrow G', H', E'$$

Une *expression* e fait passer la machine d'un état (G, H, E) à un état (G', H', E'), et produit la valeur v, ce que l'on note

$$G, H, E/e \rightarrow G', H', E'/v.$$

Les règles (voir le paragraphe suivant) décrivent comment, connaissant certaines transitions de la machine, on peut en déduire d'autres. L'ensemble des transitions de la machine est le *plus petit* ensemble de transitions satisfaisant les règles (c'est le plus petit point fixe d'une fonctionnelle que l' on peut définir à partir des règles).

Tableaux
Un tableau t de dimension $(k_1, \ldots, k_p)$ est une fonction [4]

$$\mathrm{Dom}(t) = [0, k_1 - 1] \times \ldots \times [0, k_p - 1].$$

Pour tout $v_1 \in [0, k_1 - 1]$, on note $t(v_1)$ le tableau t' de dimension $(k_2, \ldots, k_p)$ défini par

$$t'(v_2, \ldots, v_p) = t(v_1, v_2, \ldots, v_p).$$

Plus généralement, si $1 \leq q \leq p-1$, on note $t(v_1, v_2, \ldots, v_q)$ le tableau t' de dimension $(k_{q+1}, \ldots, k_p)$ défini par

$$t'(v_{q+1}, \ldots, v_p) = t(v_1, v_2, \ldots, v_q, v_{q+1} \ldots, v_p).$$

Valeurs par défaut

$$\mathrm{default}(\text{integer}) = \mathrm{default}([0..k] = 0, \ \ \mathrm{default}(\text{boolean}) = \bot, \ \ \mathrm{default}(\uparrow \tau) = \epsilon.$$

Si $\dim(\tau) = (k_1, \ldots, k_p)$ alors $\mathrm{default}(\tau) : [0, k_1 - 1] \times \ldots \times [0, k_p - 1] \rightarrow C \cup D_H$ est défini par

$$\forall (v_1, \ldots, v_p) \in [0, k_1 - 1] \times \ldots \times [0, k_p - 1], \ \ \mathrm{default}(\tau)((v_1, \ldots, v_p) = \mathrm{default}(\mathrm{tyb}(\tau)).$$

3. les entrées-sorties u, v sont invariantes pour la plupart des instructions/expressions ; nous ne les mentionnerons que lorsque cela sera nécessaire

4. au sens de la théorie des ensembles ; il ne s'agit pas d'une "fonction" déclarée dans un programme *Léa*

Sémantique des déclarations Les règles suivantes valent pour tout environnement (typé) E.

$$\frac{x \in \mathcal{V},\quad \tau \in \mathcal{T},}{E/\text{var } x : \tau; \to E \cup \{(x:\tau) \mapsto \text{default}(\tau)\}}\;\text{declaration}$$

$$\frac{E/\text{var } x_1 : \tau_1; \to E_1,\ \dots\ E_{k-1}/\text{var } x_k : \tau_k; \to E_k,\ \dots\ E_{n-1}/\text{var } x_n : \tau_n; \to E_n}{E/\text{var } x_1 : \tau_1; \dots; \text{var } x_k : \tau_k; \dots \text{var } x_n : \tau_n; \to E_n}\;\text{liste de déclarations}$$

Les règles suivantes valent pour tout triplet G, H, E.

$$\frac{s : \texttt{var\_declaration\_list},\quad E/s \to E'}{G,H,E/s \to G,H,E'}\;\text{liste de déclarations} \qquad \frac{s : \texttt{var\_declaration\_list},\quad G/s \to G'}{G,H,E/s \to G',H,E}\;\text{liste de déclarations}$$

Sémantique des opérateurs
On rappelle que $N := 2^{32}$. Les opérateurs d'arité k ($k \in \{1,2\}$) sur les entiers (resp. les booléens) ont la sémantique habituelle, qui est une application $(\mathbb{Z}/N\mathbb{Z})^k \to \mathbb{Z}/N\mathbb{Z}$ (dans le cas de `+,*,-,/`) ou bien $(\mathbb{Z}/N\mathbb{Z})^k \to \mathbb{B}$ (dans le cas de `<,<=,>,>=,=,!=`) ou bien $\mathbb{B}^k \to \mathbb{B}$ (dans le cas de `||,&&,!`).

Sémantique des expressions

$$\frac{}{G,H,E/k \to G,H,E/[\![k]\!]}\;\text{Littéral} \qquad \frac{(x,\tau) \in dom(E)}{G,H,E/x \to G,H,E/E(x)}\;\text{Variable locale} \qquad \frac{(x,\tau) \in dom(G) \setminus dom(E)}{G,H,E/x \to G,H,E/G(x)}\;\text{Variable globale}$$

$$\frac{G,H,E/e \to G',H',E'/v}{G,H,E/uop\ e \to G,H,E/[\![uop]\!](e)}\;\text{Opérateur unaire} \qquad \frac{S/e_1 \to S'/v_1 \quad S'/e_2 \to S''/v_2}{S/e_1\ bop\ e_2 \to S''/[\![bop]\!](v_1,v_2)}\;\text{Opérateur binaire}$$

$$\frac{S/e_1 \to S'/t_1 \quad S'/e_2 \to S''/n}{S/e_1[e_2] \to S''/t_1(n)}\;\text{Variable indexée} \qquad \frac{\begin{array}{c} S/e \to G',H',E'/\ell \quad \ell \in dom(H') \\ H'(\ell) = v \end{array}}{S/e \uparrow\to G',H',E'/v}\;\text{Variable pointée}$$

Pour toute fonction f déclarée par

$$f(x_1 : \tau_1, \dots, x_n : \tau_n) : \tau \text{ var } x'_1 : \tau'_1; \dots, \text{var } x'_m : \tau'_m;\ c$$

$$\frac{\begin{array}{c} \forall j \in [1,n], S_{j-1}/e_j \to S_j/v_j,\quad S_n = G',H',E_n \\ E_n/\text{ var } x'_1 : \tau'_1; \dots; \text{var } x'_m : \tau'_m; \to \hat{E} \\ \hat{E}/\text{var } \#fn : \tau; \to E' \\ G',H',E'/c \to G'',H'',E'' \quad v = E''(\#fn) \end{array}}{S_0/f(e_1,\dots,e_n) \to G'',H'',E_n/v}\;\text{Appel de fonction}$$

Sémantique des instructions (ou commandes)

Bloc

$$\frac{S_0/c_1 \to S_1, \ldots S_{i-1}/c_i \to S_i, \ldots S_{p-1}/c_p \to S_p}{S_0/\ \texttt{begin}\, c_1 \ldots c_p\ \texttt{end}\ \to S_p}$$

Affectation : variable locale

$$\frac{S/e \to G', H', E'/v \quad (x, \tau) \in \mathrm{Dom}(E')}{S/x := e \to G', H', E'[(x, \tau) \mapsto v]}$$

Affectation : variable globale

$$\frac{S/e \to G', H', E'/v \quad x \in V_{G'} \setminus V_{E'}, \quad (x, \tau) \in \mathrm{Dom}(G')}{S/x := e \to G'[(x, \tau) \mapsto v], H', E'}$$

Affectation à la valeur de retour

$$\frac{\mathrm{ty}(S, e) = \tau, \quad S/e \to G', H', E'/v}{S/\mathrm{return}(e) \ \to \ G', H', E'[(\# fn, \tau) \mapsto v]}$$

While true

$$\frac{S/e \to S_1/\top \quad S_1/c \to S_2 \quad S_2/\mathrm{begin}\ c\ \mathrm{Wh}\ e\ \mathrm{Do}\ c\ \mathrm{end} \to S'}{S/\ \mathrm{Wh}\ e\ \mathrm{Do}\ c \to S'}$$

While false

$$\frac{S/e \to S'/\bot}{S/\ \mathrm{Wh}\ e\ \mathrm{Do}\ c \to S'}$$

IfThEl true

$$\frac{S/e \to S'/\top \quad S'/c_1 \to S''}{S/\mathrm{If} e \mathrm{Th} c_1 \mathrm{El} c_2 \to S''}$$

IfThEl false

$$\frac{S/e \to S'/\bot \quad S'/c_2 \to S''}{S/\mathrm{If} e \mathrm{Th} c_1 \mathrm{El} c_2 \to S''}$$

Affectation : variable dynamique indexée

$$\frac{\begin{array}{c} S_0/e \to S'/\ell \ \ \ell \in \mathrm{Dom}(H) \\ S'/e_1 \to S_1/n_1, \ldots S_{p-1}/e_p \to S_p/n_p, \quad S_p/e' \to S''/v' \\ S'' = G'', H'', E'', \quad H''(\ell) = t, \quad t' = t[(n_1, \ldots, n_p) \mapsto v'] \end{array}}{S_0/e[e_1] \ldots [e_q] := e' \to G'', H''[\ell \mapsto t'], E''}$$

Affectation : variable globale indexée

$$\frac{\begin{array}{c} (x, \tau) \in \mathrm{Dom}(G), \quad \dim(\tau) = (k_1, \ldots, k_p) \\ S/e_1 \to S_1/n_1, \ldots S_{p-1}/e_p \to S_p/n_p, \quad S_p/e' \to S'/v' \\ S' = G', H', E', \quad t' = t[(n_1, \ldots, n_p) \mapsto v'] \end{array}}{S_0/x[e_1] \ldots [e_q] := e' \to G'[(x, \tau) \mapsto t'], H', E'}$$

Affectation : variable locale indexée

$$\frac{\begin{array}{c} (x, \tau) \in \mathrm{Dom}(E), \quad \dim(\tau) = (k_1, \ldots, k_p) \\ \vdots \end{array}}{S_0/x[e_1] \ldots [e_q] := e' \to G', H', E'[(x, \tau) \mapsto t']}$$

Affectation : variable pointée

$$\frac{\begin{array}{c} S/e_1 \to S'/\ell_1 \quad S'/e_2 \to S''/v_2 \\ S'' = G'', H'', E'', \end{array}}{S/e_1 \uparrow := e_2 \to G'', H''[\ell_1 \mapsto v_2], E''}$$

Allocation mémoire, variable locale

$$\frac{S = G, H, E \ \ (x, \uparrow \tau) \in \mathrm{Dom}(E), \quad \ell \notin \mathrm{Dom}(H), \quad H' = H \cup \{\ell \mapsto \mathrm{default}(\tau)\}}{S/new(x) \to G, H', E[(x, \uparrow \tau) \mapsto \ell]}$$

Allocation mémoire, variable globale

$$\frac{S = G, H, E \ \ (x, \uparrow \tau) \in \mathrm{Dom}(G), \quad \ell \notin \mathrm{Dom}(H), \quad H' = H \cup \{\ell \mapsto \mathrm{default}(\tau)\}}{S/new(x) \to G[(x, \uparrow \tau) \mapsto \ell], H', E}$$

Allocation mémoire : variable dynamique indexée

$$\frac{\begin{array}{c}\mathrm{ty}(S_0, e[e_1]\dots[e_p]) =\uparrow \tau \\ S_0/e \to S'/\ell \ \ \ell \in \mathrm{Dom}(H) \\ S'/e_1 \to S_1/n_1, \dots S_{p-1}/e_p \to S_p/n_p \\ S_p = G_p, H_p, E_p, \ \ H_p(\ell) = t, \ \ \ell' \notin \mathrm{Dom}(H_p) \\ t' = t[(n_1, \dots, n_p) \mapsto \ell'] \\ H_{p+1} = H_p[\ell \mapsto t'] \cup \{\ell' \mapsto \mathrm{default}(\tau)\}, \ \ S_{p+1} = G_p, H_{p+1}, E_p\end{array}}{S_0/\mathrm{new}\ (e[e_1]\dots[e_p]) \to S_{p+1}}$$

Allocation mémoire : variable globale indexée

$$\frac{\begin{array}{c}(x,\tau) \in \mathrm{Dom}(G), \ \ \mathrm{tyb}(\tau) =\uparrow \tau', \ \ \dim(\tau) = (k_1, \dots, k_p) \\ S/e_1 \to S_1/n_1, \dots S_{p-1}/e_p \to S_p/n_p \\ S_p = G_p, H_p, E_p, \ \ G_p(x) = t, \ \ \ell' \notin \mathrm{Dom}(H_p) \\ t' = t[(n_1, \dots, n_p) \mapsto \ell'] \\ G_{p+1} = G_p[(x,\tau) \mapsto t'], \ \ H_{p+1} = H_p \cup \{\ell' \mapsto \mathrm{default}(\tau')\}, \ \ S_{p+1} = G_{p+1}, H_{p+1}, E_p\end{array}}{S/\mathrm{new}\ (x[e_1]\dots[e_p]) \to S_{p+1}}$$

Allocation mémoire : variable locale indexée

$$\frac{\begin{array}{c}(x,\tau) \in \mathrm{Dom}(E), \ \ \mathrm{tyb}(\tau) =\uparrow \tau', \ \ \dim(\tau) = (k_1, \dots, k_p) \\ \vdots \\ E_{p+1} = E_p[(x,\tau) \mapsto t'], \ \ H_{p+1} = H_p \cup \{\ell' \mapsto \mathrm{default}(\tau')\}, \ \ S_{p+1} = G_{p+1}, H_{p+1}, E_p\end{array}}{S/\mathrm{new}\ (x[e_1]\dots[e_p]) \to S_{p+1}}$$

Allocation mémoire : variable pointée

$$\frac{\begin{array}{c}\mathrm{ty}(S, e) =\uparrow\uparrow \tau, \ \ S/e \to S'/\ell \ \ S' = G', H', E', \ \ \ell \in \mathrm{Dom}(H'), \ \ \ell' \notin \mathrm{Dom}(H') \\ H'' = H' \cup \{\ell' \mapsto \mathrm{default}(\tau)\}, \ \ H''' = H''[\ell \mapsto \ell']\end{array}}{S/\mathrm{new}\ (e\uparrow) \to G', H''', E'}$$

Désallocation mémoire : variable locale

$$\frac{\begin{array}{c}S = G, H, E \ \ E(x, \uparrow\tau) = \ell \\ \mathrm{Dom}(H') = \mathrm{Dom}(H) \setminus \{\ell\}, \ \ \forall \ell' \in \mathit{dom}(H'), H'(\ell') = H(\ell')\end{array}}{S/\mathit{dispose}(x) \to G, H', E[(x,\tau) \mapsto \varepsilon]}$$

Désallocation mémoire : variable globale

$$\frac{\begin{array}{c}\dim(\tau) = d, \ \ S = G, H, E \ \ (x, \uparrow\tau) \in \mathrm{Dom}(G) \setminus \mathrm{Dom}(E), \ \ G(x, \uparrow\tau) = \ell, \\ \mathrm{Dom}(H') = \mathrm{Dom}(H) \setminus \{\ell\}, \ \ \forall \ell' \in \mathit{dom}(H'), H'(\ell') = H(\ell')\end{array}}{S/\mathit{dispose}(x) \to G[(x, \uparrow\tau) \mapsto \varepsilon], H', E}$$

Désallocation mémoire : variable dynamique indexée

$$\frac{\begin{array}{c}\mathrm{ty}(S_0, e[e_1]\dots[e_p]) =\uparrow \tau \\ S_0/e \to S'/\ell \ \ \ell \in \mathrm{Dom}(H) \\ S/e_1 \to S_1/n_1, \ \ S'/e_p \to S_p/n_p \\ S_p = G_p, H_p, E_p, \ \ G_p(x) = t, \ \ t(n_1, \dots, n_p) = v \\ t' = t[(n_1, \dots, n_p) \mapsto \varepsilon] \\ \mathrm{Dom}(H_{p+1}) = \mathrm{Dom}(H_p) \setminus \{v\}, \ \forall \ell' \in \mathit{dom}(H_{p+1}), H_{p+1}(\ell') = H_p(\ell')\end{array}}{S/\mathrm{dispose}\ (x[e_1]\dots[e_p]) \to G_p[(x,\tau) \mapsto t'], H_{p+1}, E_p}$$

$$\frac{\begin{array}{c}\text{Désallocation mémoire : variable globale indexée}\\ (x,\tau)\in \mathrm{Dom}(G),\quad \mathrm{tyb}(\tau)=\uparrow\tau',\quad \dim(\tau)=(k_1,\ldots,k_p)\\ S/e_1\to S_1/n_1,\quad S'/e_p\to S_p/n_p\\ S_p=G_p,H_p,E_p,\quad H_p(\ell)=t,\quad t(n_1,\ldots,n_p)=v\\ t'=t[(n_1,\ldots,n_p)\mapsto\varepsilon]\\ \mathrm{Dom}(H_{p+1})=\mathrm{Dom}(H_p)\setminus\{v\},\ \forall\ell'\in \mathit{dom}(H_{p+1}),\ H_{p+1}(\ell')=H_p(\ell')\end{array}}{S/\mathrm{dispose}\ (x[e_1]\ldots[e_p])\to G_p[(x,\tau)\mapsto t'],H_{p+1}[\ell\mapsto t'],E_p}$$

$$\frac{\begin{array}{c}\text{Désallocation mémoire : variable locale indexée}\\ (x,\tau)\in\mathrm{Dom}(E),\\ \vdots\end{array}}{S_0/\mathrm{dispose}\ (x[e_1]\ldots[e_p])\to G_p,H_{p+1},E_p[(x,\tau)\mapsto t']}$$

$$\frac{\begin{array}{c}\text{Désallocation mémoire : variable pointée}\\ \mathrm{ty}(S,e)=\uparrow\uparrow\tau,\quad S/e\to S'/\ell,\quad S'=G',H',E\\ H'(\ell)=v\ \ \mathit{dom}(H'')=\mathrm{Dom}(H')\setminus\{v\},\quad \forall\ell'\in\mathit{dom}(H''),H''(\ell')=H'(\ell')\end{array}}{S/\mathrm{dispose}\ (e\uparrow)\to G',H''[\ell\mapsto\varepsilon],E}$$

Pour toute procédure f déclarée par

$$f(x_1:\tau_1,\ldots,x_n:\tau_n)\ \mathrm{var}\ x'_1:\tau'_1,\ldots,\mathrm{var}\ x'_m:\tau'_m c$$

$$\frac{\begin{array}{c}\text{Appel de procédure}\\ \forall j\in[1,n],S_{j-1}/e_j\to S_j/v_j,\quad S_n=G',H',E_n\\ E_n/\ \mathrm{var}\ x'_1:\tau'_1,\ldots,\mathrm{var}\ x'_m:\tau'_m\to E'\\ G',H',E'/c\to G'',H'',E''\end{array}}{S_0/f(e_1,\ldots,e_n)\to G'',H'',E_n}$$

Sémantique des entrées-sorties

$$\frac{\begin{array}{c}\text{Lecture d'un entier : variable globale}\\ u\in(\mathbb{Z}/\mathrm{N}\mathbb{Z})^*,\quad n\in\mathbb{Z}/\mathrm{N}\mathbb{Z},\quad v\in(\mathbb{Z}/\mathrm{N}\mathbb{Z})^*\\ S=G,H,E,\quad (x,\mathrm{integer})\in\mathrm{Dom}(G),\quad G'=G[(x,\mathrm{integer})\mapsto n]\end{array}}{(u\cdot n|G,H,E|v)/\mathrm{read}(x)\to(u|G',H,E|v)}$$

$$\frac{\begin{array}{c}\text{Lecture d'un entier : variable locale}\\ u\in(\mathbb{Z}/\mathrm{N}\mathbb{Z})^*,\quad n\in\mathbb{Z}/\mathrm{N}\mathbb{Z},\quad v\in(\mathbb{Z}/\mathrm{N}\mathbb{Z})^*\\ S=G,H,E,\quad (x,\mathrm{integer})\in\mathrm{Dom}(E),\quad E'=E[(x,\mathrm{integer})\mapsto n]\end{array}}{(u\cdot n|G,H,E|v)/\mathrm{read}(x)\to(u|G,H,E'|v)}$$

$$\frac{\begin{array}{c}\text{Écriture d'une expression entière}\\ u\in(\mathbb{Z}/\mathrm{N}\mathbb{Z})^*,\quad v\in(\mathbb{Z}/\mathrm{N}\mathbb{Z})^*\\ \mathrm{ty}(S,e)=\mathrm{integer},\quad S/e\to S'/n,\quad S'=G',H',E'\end{array}}{(u|S|v)/\mathrm{write}(e)\to(u|G',H',E'|n\cdot v)}$$

$$\frac{\begin{array}{c}\text{Entrées-sorties versus calculs}\\ G,H,E/c\to G',H',E'\end{array}}{(u|G,H,E|v)/c\to(u|G',H',E'|v)}$$

Sémantique d'un programme

$$\frac{\begin{array}{c}\text{Programme}\\ P = \text{ var } x_1 : \tau_1, \ldots, \text{var } x_n : \tau_n \;\; d_1 d_2 \ldots d_n c\\ \emptyset, \emptyset / \text{ var } x_1 : \tau_1, \ldots, \text{var } x_n : \tau_n \to H, G\\ (u|G, H, \emptyset|\varepsilon)/c \to (\varepsilon|G', H', E'|v)\end{array}}{(u|\emptyset, \emptyset, \emptyset|\varepsilon)/P \to (\varepsilon|G', H', E'|v)}$$

La fonction $[\![P]\!] : (\mathbb{Z}/\mathrm{N}\mathbb{Z})^* \to (\mathbb{Z}/\mathrm{N}\mathbb{Z})^*$ est alors définie par

$$[\![P]\!](u) = v$$

ssi, il existe un état G', H', E' tel que

$$(u|\emptyset, \emptyset, \emptyset|\varepsilon)/P \to (\varepsilon|G', H', E'|v).$$

Ici aussi on peut définir, pour un programme fixe des ensembles F, F' puis $\mathcal{E} = \mathcal{P}(F \cup F')$ et une application continue $\Phi : \mathcal{E} \to \mathcal{E}$ à partir des règles ci-dessus. On définit la relation $\to$ (sur les états de la machine abstraite) comme le *plus petit point fixe* de Φ.