UPMC/master/info/4I503 APS
Notes de cours

P. Manoury

Janvier 2018

Statiquement, un *programme* est un fichier: code source ou exécutable. Un programme a également une *dynamique* qui est son comportement lors de son exécution. Le lien entre la donnée statique d'un programme et sa dynamique est l'objet de la *sémantique*. On peut anticiper certaines propriétés du comportement dynamique des programmes par *analyse* de la donnée statique des programmes; typiquement, *l'analyse de type*.

Programme La donnée statique d'un programme correspond à son code source ou a son code compilé (exécutable ou code-octet). On traitera dans ce cours du code source des programmes.

Un code source est simplement une suite de caractères, en général stockée dans un ensemble de fichiers textes. Toutefois, toute suite de caractères n'est pas un code source. En effet, les codes sources des programmes doivent respecter les règles d'un langage. Ces règles définissent la la *syntaxe* des langages de programmation.

La définition de la syntaxe d'un langage comprend deux éléments: la définition d'un *lexique* et la définition d'une *grammaire*. Le lexique est l'ensemble des suites de caractère unitaires des langages, l'ensemble des *mots* et *symboles* utilisables dans le langage. Ces unités de langages sont des *lexèmes*. La grammaire énonce les règles d'agencements des lexémes. Elle définit l'ensemble des suites de lexèmes qui appartiennent au langage.

Lorsqu'elle sont formalisées, les définitions du lexique et de la grammaire d'un langage permettent la génération de fonctions d'analyse de suites de caractères et de suites de lexèmes capables de décider si une suite de caractères appartient ou non au langage défini. Ce genre d'analyse des suites de caractères est appelée *analyse syntaxique*. D'un point de vue théorique, on établit une relation entre les définitions formelles et des automates (automates à états finis, automates à pile, etc.) qui sont des objets facilement implémentables en machine.

Cette génération est opérationnalisée par des outils logiciels comme `lex` et `yacc`, pour ne parler que des ancêtres.

Sémantique Le comportement dynamique d'un programme est induit par le traitement de la donnée statique d'un programme par un micro-processeur, un interprète de code-octet, voire un interprète de code source.

En pratique, les programmes sont réalisés dans l'*intention* d'obtenir un certain comportement. Celui-ci est manifesté par la production d'un *résultat* ou d'un *effet* pour le dispositif sur lequel le programme est exécuté: affichage, production ou modification de fichier, etc. Dans tous les cas, on peut dire que le but d'un programme est de produire la modification d'un *état mémoire*; la mémoire de vive l'ordinateur, ou la mémoire de masse d'un système de fichiers.

La sémantique a pour objet d'exprimer la relation entre les constructions syntaxiques d'un programme et la production de résultats ou d'effets. Cette relation est définie en liant la construction syntaxique des langages de programmation avec leur effet sur le *contexte d'exécution* des programmes. Ce contexte est constitué par une certaine organisation en mémoire des *valeurs* manipulées par les programmes. La définition de la sémantique est *dirigée par la syntaxe*, en ce sens que, *grosso modo*, à chaque règle de contruction syntaxique des langages est associée une règle d'exécution, ou règle d'*évaluation*.

Donner la sémantique d'un langage de programmation, c'est donner la spécification formelle d'un interpréteur de code source.

Analyse L'analyse opère sur la donnée statique des programmes, souvent son code source. Elle a pour but d'anticiper leur comportement dynamique. Ceci permet en particulier de savoir diagnostiquer par avance qu'un programme est suceptible de provoquer une erreur d'exécution.

Le plus répandu de ce genre d'analyse est réalisé par les mécanismes de *vérification de type* qui accompagnent les compilateurs des langages de programmation. Les programmes manipulent et produisent des valeurs. Ces valeurs peuvent être classifiées par *types*. Manipuler ou produire une valeur n'appartenant pas au type attendu par une opération du programme peut provoquer une erreur d'exécution ou une incohérence du contexte d'exécution. La vérification de type, ou *typage*, appliqué à un programme peut éliminer ce risque d'erreur ou d'incohérence.

Le principe définition de l'analyse de type est similaire à celui mis en œuvre pour la sémantiue: l'analyse est guidée par la syntaxe. *Grosso modo*, à chaque règle syntaxique des langages est associée une règle de vérification de cohérence de type des valeurs manipulées et produites.

Mise en œuvre On mettra en œuvre les trois éléments de traitement des programmes que sont l'analyse syntaxique, l'analyse de type, et l'évaluation par la réalisation des composants logiciels correspondant.

Schéma de principe

```
           +--------------------+        +--------+        +------------+
[source]->| analyse syntaxique |->[AST]->| typage |->[AST]->| évaluateur |->[résultat]
           +----------+---------+        +----+---+        +------+-----+
                      |                       |                   |
                      v                       v                   v
                  *erreur*                *erreur*        *erreur* ou *rien*
```

L'analyse syntaxique produit une `*erreur*` lorsque la suite de caractères donnée en entrée n'appartient pas au langage. Lorsque ce n'est pas le cas, le processus d'analyse syntaxique produira un *arbre de syntaxte abstrait* du langage (*Abstract Syntax Tree*). Celui-ci sera donnée sous la forme d'un *terme prolog* (une chaîne de caractères) ou sous la forme d'une structure en mémoire (structure arborescente, instance de classe).

L'analyse de typage sera réalisée en PROLOG, dans ce cas, votre analyse syntaxique devra produire un terme PROLOG.

Le langage d'implantation de l'évaluateur est laissé libre. Vous adapterez la sortie de votre analyse syntaxique en conéquence.

Ce document présente une synthèse de la syntaxe, du typage et des sémantiques des différentes évolutions du langage étudié.

APS0 noyau fonctionnel: booléens, entiers et leurs primitives, alternative fonctionnelle, expressions fonctionnelles, et leur application, définitions récursives
+ instruction ECHO.

APS1 extension au noyau impératif: séquence, blocs, affectation, alternative, boucle, procédures et procédures récursives, définition et appel, variables (modifiables)

APS2 ajout des structures séquentielles homogènes (vecteurs) avec leurs primitives, dont allocation, *left-values* et *right-values*

APS3 fonctions procédurales, instruction RETURN

Grammaire incrémentale de APS0 à APS3

Typage incrémental de APS0 à APS1; revisité pour APS2; le typage des instructions et suites de commandes est refondu pour APS3

Sémantique revisitée plus ou moins profondément à chaque extension
La sémantique est opérationnelle avec quelques éléments de style dénotationnel (domaines).

1 APS0: noyau fonctionnel

1.1 Syntaxe

1.1.1 Lexique

Symboles réservés

`[ ] ( ) ; , : * ->`

Opérateurs primitifs l'ensemble de symboles oprim contient
`not and or eq lt add sub mul div`

Types primitifs l'ensemble de symboles tprim contient
`bool int`

Mots clef

```
if
CONST FUN REC
ECHO
```

Constantes booléennes bool qui contient `true false`

Constantes numériques num défini par *('-'?)['0'-'9']+*

Identificateurs ident défini par *(['a'-'z''A'-'Z'])(['a'-'z''A'-'Z''0'-'9'])** sauf bool, oprim, tprim et `if`

Les *séparateurs* sont l'espace, la tabulation et le retour-chariot.

1.1.2 Grammaire

Notation préfixe parenthésée des expressions

$$
\begin{array}{lcl}
\textsc{Prog} & ::= & \texttt{[}\ \textsc{Cmds}\ \texttt{]} \\
\textsc{Cmds} & ::= & \textsc{Stat} \\
 & | & \textsc{Dec}\ \texttt{;}\ \textsc{Cmds} \\
 & | & \textsc{Stat}\ \texttt{;}\ \textsc{Cmds} \\
\textsc{Dec} & ::= & \texttt{CONST}\ \mathsf{ident}\ \textsc{Type}\ \textsc{Expr} \\
 & | & \texttt{FUN}\ \mathsf{ident}\ \textsc{Type}\ \texttt{[}\ \textsc{Args}\ \texttt{]}\ \textsc{Expr} \\
 & | & \texttt{FUN REC}\ \mathsf{ident}\ \textsc{Type}\ \texttt{[}\ \textsc{Args}\ \texttt{]}\ \textsc{Expr} \\
\textsc{Stat} & ::= & \texttt{ECHO}\ \textsc{Expr} \\
\textsc{Expr} & ::= & \texttt{true} \mid \texttt{false} \mid \mathsf{num} \mid \mathsf{ident} \\
 & | & \texttt{(}\ \texttt{if}\ \textsc{Expr}\ \textsc{Expr}\ \textsc{Expr}\ \texttt{)} \\
 & | & \texttt{(}\ \mathsf{oprim}\ \textsc{Exprs}\ \texttt{)} \\
 & | & \texttt{[}\ \textsc{Args}\ \texttt{]}\ \textsc{Expr} \\
 & | & \texttt{(}\ \textsc{Expr}\ \textsc{Exprs}\ \texttt{)} \\
\textsc{Exprs} & ::= & \textsc{Expr} \\
 & | & \textsc{Expr}\ \textsc{Exprs} \\
\textsc{Args} & ::= & \textsc{Arg} \\
 & | & \textsc{Arg}\ \texttt{,}\ \textsc{Args} \\
\textsc{Arg} & ::= & \mathsf{ident}\ \texttt{:}\ \textsc{Type} \\
\textsc{Type} & ::= & \mathsf{tprim} \\
 & | & \texttt{(}\ \textsc{Types}\ \texttt{->}\ \textsc{Type}\ \texttt{)} \\
\textsc{Types} & ::= & \textsc{Type} \\
 & | & \textsc{Type}\ \texttt{*}\ \textsc{Types}
\end{array}
$$

1.2 Typage

On pose $\mathsf{sym} = \mathsf{bool} \cup \mathsf{oprim} \cup \mathsf{ident}$.

Contexte de typage fonction partielle Γ de type $\mathsf{sym} \to \textsc{Type}$. On pose: $G = \mathsf{sym} \to \textsc{Type}$

Extention d'un contexte de typage Γ: fonction notée $\Gamma[x : t]$ avec x élément de sym et t élément de $\textsc{Type}$ telle que $\Gamma[x : t](x) = t$ et $\Gamma[x : t](y) = \Gamma(y)$ lorsque x et y sont des symboles différents.

Remarque si x et y sont des symboles différents alors $\Gamma[x : t][y : t'] = \Gamma[y : t'][x : t]$.

Abréviation on écrit $\Gamma[x_1 : t_1; \ldots; x_n : t_n]$ pour $\Gamma[x_1 : t_1] \ldots [x_n : t_n]$.

1.2.1 Expressions

Relation de typage: $\vdash_{\textsc{Expr}}$ dans $G \times \textsc{Expr} \times \textsc{Type}$
Notation: on écrit $\Gamma \vdash_{\textsc{Expr}} e : t$ pour $\vdash_{\textsc{Expr}} (\Gamma, e, t)$.

(NUM) si $n \in \mathsf{num}$ alors $\Gamma \vdash_{\textsc{Expr}} n : \texttt{int}$

(SYM) si $x \in \mathsf{sym}$ et si $\Gamma(x) = t$ alors $\Gamma \vdash_{\textsc{Expr}} x : t$

(ABS) si $\Gamma[x_1 : t_1; \ldots; x_n : t_n] \vdash_{\textsc{Expr}} e : t$ alors $\Gamma \vdash_{\textsc{Expr}} \texttt{[}x_1 : t_1, \ldots, x_n : t_n\texttt{]}e : t_1\ \texttt{*}\ \ldots\ \texttt{*}\ t_n\ \texttt{->}\ t$

(APP) si $\Gamma \vdash_{\textsc{Expr}} e_1 : t_1$, ... si $\Gamma \vdash_{\textsc{Expr}} e_n : t_n$
et si $\Gamma \vdash_{\textsc{Expr}} e : t_1\ \texttt{*}\ \ldots\ \texttt{*}\ t_n\ \texttt{->}\ t$
alors $\Gamma \vdash_{\textsc{Expr}} \texttt{(}e\ e_1 \ldots e_n\texttt{)} : t$

(IF) si $\Gamma \vdash_{\textsc{Expr}} e_1 : \texttt{bool}$, si $\Gamma \vdash_{\textsc{Expr}} e_2 : t$ et si $\Gamma \vdash_{\textsc{Expr}} e_3 : t$
alors $\Gamma \vdash_{\textsc{Expr}} \texttt{(if}\ e_1\ e_2\ e_3\texttt{)} : t$

1.2.2 Définitions

Relation $\vdash_{\text{DEC}}$ dans $G \times \text{DEC} \times G$.

On écrit $\Gamma \vdash_{\text{DEC}} d : \Gamma'$.

(CONST) si $\Gamma \vdash_{\text{EXPR}} e : t$ alors $\Gamma \vdash_{\text{DEC}} (\texttt{CONST}\ x\ t\ e) : \Gamma[x : t]$

(FUN) si $\Gamma[x_1 : t_1; \ldots; x_n : t_n] \vdash_{\text{EXPR}} e : t$
alors $\Gamma \vdash_{\text{DEC}} (\texttt{FUN}\ x\ t\ \texttt{[}x_1\texttt{:}t_1\texttt{,}\ldots\texttt{,}x_n\texttt{:}t_n\texttt{]}\ e) : \Gamma[x : t_1\ \texttt{*}\ \ldots\ \texttt{*}\ t_n\ \texttt{->}\ t]$

(FUNREC) si $\Gamma[x_1 : t_1; \ldots; x_n : t_n; x : t_1\ \texttt{*}\ \ldots\ \texttt{*}\ t_n\ \texttt{->}\ t] \vdash_{\text{EXPR}} e : t$
alors $\Gamma \vdash_{\text{DEC}} (\texttt{FUN REC}\ x\ t\ \texttt{[}x_1\texttt{:}t_1\texttt{,}\ldots\texttt{,}x_n\texttt{:}t_n\texttt{]}\ e) : \Gamma[x : t_1\ \texttt{*}\ \ldots\ \texttt{*}\ t_n\ \texttt{->}\ t]$

1.2.3 Instruction

Un type vide: *void*.

Relation $\vdash_{\text{STAT}}$ dans $G \times \text{STAT} \times \{\textit{void}\}$

On écrit $\Gamma \vdash_{\text{STAT}} s : \mathsf{void}$.

(ECHO) si $\Gamma \vdash_{\text{EXPR}} e : \texttt{int}$ alors $\Gamma \vdash_{\text{STAT}} (\texttt{ECHO}\ e) : \textit{void}$

1.2.4 Suite de commandes

Une *commande vide*: ε. On pose $\text{CMDS}_\varepsilon = \{cs\texttt{;}\varepsilon \mid cs \in \text{CMDS}\}$

Relation $\vdash_{\text{CMDS}}$ dans $G \times \text{CMDS}_\varepsilon \times \{\textit{void}\}$

On écrit $\Gamma \vdash_{\text{CMDS}} cs : \textit{void}$

(DECS) si $d \in \text{DEC}$, si $\Gamma \vdash_{\text{DEC}} d : \Gamma'$ et si $\Gamma' \vdash_{\text{CMDS}} cs : \textit{void}$ alors $\Gamma \vdash_{\text{CMDS}} (d\texttt{;}\ cs) : \textit{void}$.

(STATS) si $s \in \text{STAT}$, si $\Gamma \vdash_{\text{STAT}} s : \textit{void}$ et si $\Gamma \vdash_{\text{CMDS}} cs : \textit{void}$ alors $\Gamma \vdash_{\text{CMDS}} (s\texttt{;}\ cs) : \textit{void}$.

(END) $\Gamma \vdash_{\text{CMDS}} \varepsilon : \textit{void}$.

1.2.5 Programme

Relation $\vdash$ dans $\text{PROG} \times \{\textit{void}\}$.

On se donne $\Gamma_0 \in G$ tel que

$$
\begin{array}{lcl}
\Gamma_0(\texttt{true}) & = & \texttt{bool} \\
\Gamma_0(\texttt{false}) & = & \texttt{bool} \\
\Gamma_0(\texttt{not}) & = & \texttt{bool -> bool} \\
\Gamma_0(\texttt{and}) & = & \texttt{bool * bool -> bool} \\
\Gamma_0(\texttt{or}) & = & \texttt{bool * bool -> bool} \\
\Gamma_0(\texttt{eq}) & = & \texttt{int * int -> bool} \\
\Gamma_0(\texttt{lt}) & = & \texttt{int * int -> bool} \\
\Gamma_0(\texttt{add}) & = & \texttt{int * int -> int} \\
\Gamma_0(\texttt{sub}) & = & \texttt{int * int -> int} \\
\Gamma_0(\texttt{mul}) & = & \texttt{int * int -> int} \\
\Gamma_0(\texttt{div}) & = & \texttt{int * int -> int}
\end{array}
$$

et $\Gamma_0(x)$ n'est pas défini pour tout x de ident.

(PROG) si $\Gamma_0 \vdash_{\text{CMDS}} (cs\texttt{;}\varepsilon) : \textit{void}$ alors $\vdash \texttt{[}cs\texttt{]} : \textit{void}$

1.3 Sémantique opérationnelle

1.3.1 Domaines et fonctions sémantiques

Les domaines

Valeurs immédiates N

Fermetures $F = \textsc{Expr} \times (V^* \to E)$

Fermetures récursives $FR = F \to F$

Valeurs $V = N \oplus F \oplus FR$

Environnement $E = \mathsf{ident} \to V$ (fonction partielle)

Flux de sortie $O = N^*$

Les fonctions

Injections canoniques si $x \in A$ alors $inA(x) \in A \oplus B$ et si $x \in B$ alors $inB(x) \in A \oplus B$.

Projections canoniques si $x \in A$ alors $outA(inA(x)) = x$

Conversion on se donne ν de type $\mathsf{num} \to N$ ($\nu(\mathtt{42}) = 42$).

Primitives on se donne π de type $\mathsf{oprim} \to (N^* \to N)$ telle que

$$\begin{array}{rcll}
\pi(\mathtt{not})(0) & = & 1 & \\
\pi(\mathtt{not})(1) & = & 0 & \\
\pi(\mathtt{and})(0,n) & = & 0 & \\
\pi(\mathtt{and})(1,n) & = & n & \\
\pi(\mathtt{or})(1,n) & = & 1 & \\
\pi(\mathtt{or})(0,n) & = & n & \\
\pi(\mathtt{eq})(n_1,n_2) & = & 1 & \text{si } n_1 = n_2 \\
 & = & 0 & \text{sinon} \\
\pi(\mathtt{lt})(n_1,n_2) & = & 1 & \text{si } n_1 < n_2 \\
 & = & 0 & \text{sinon} \\
\pi(\mathtt{add})(n_1,n_2) & = & n_1 + n_2 & \\
\pi(\mathtt{sub})(n_1,n_2) & = & n_1 - n_2 & \\
\pi(\mathtt{mul})(n_1,n_2) & = & n_1 \cdot n_2 & \\
\pi(\mathtt{div})(n_1,n_2) & = & n_1 \div n_2 &
\end{array}$$

Contexte (*alias* environnement) d'évaluation: fonction ρ de type E.

Extension du contexte ρ: fonction notée $\rho[x = v]$ où x est un élément de ident et v un élément de V telle que $\rho[x = v](x) = v$ et $\rho[x = v](y) = \rho(y)$ lorsque y est un identificateur différent de x.

Abréviation on écrit $\rho[x_1 = v_1; \ldots; x_n = v_n]$ pour $\rho[x_1 = v_1] \ldots [x_n = v_n]$

1.3.2 Expressions

Relation $\vdash_{\textsc{Expr}}$ dans $E \times \textsc{Expr} \times V$.
Notation: on écrit $\rho \vdash_{\textsc{Expr}} e \rightsquigarrow v$ pour $\vdash_{\textsc{Expr}} (\rho, e, v)$.

(TRUE) $\rho \vdash_{\textsc{Expr}} \mathtt{true} \rightsquigarrow inN(1)$

(FALSE) $\rho \vdash_{\text{Expr}} \texttt{false} \rightsquigarrow inN(0)$

(NUM) si $n \in \mathsf{num}$ alors $\rho \vdash_{\text{Expr}} n \rightsquigarrow inN(\nu(n))$

(ID) si $x \in \mathsf{ident}$ et $\rho(x) = v$ alors $\rho \vdash_{\text{Expr}} x \rightsquigarrow v$

(PRIM) si $x \in \mathsf{oprim}$, si $\rho \vdash_{\text{Expr}} e_1 \rightsquigarrow inN(n_1)$, ..., si $\rho \vdash_{\text{Expr}} e_k \rightsquigarrow inN(n_k)$ et si $\pi(x)(n_1, \ldots, n_k) = n$
alors $\rho \vdash_{\text{Expr}} \texttt{(}x\ e_1 \ldots e_n\texttt{)} \rightsquigarrow inN(n)$

(IF1) si $\rho \vdash_{\text{Expr}} e_1 \rightsquigarrow inN(1)$ et si $\rho \vdash_{\text{Expr}} e_2 \rightsquigarrow v$ alors $\rho \vdash_{\text{Expr}} \texttt{(if}\ e_1\ e_2\ e_3\texttt{)} \rightsquigarrow v$

(IF0) si $\rho \vdash_{\text{Expr}} e_1 \rightsquigarrow inN(0)$ et si $\rho \vdash_{\text{Expr}} e_3 \rightsquigarrow v$ alors $\rho \vdash_{\text{Expr}} \texttt{(if}\ e_1\ e_2\ e_3\texttt{)} \rightsquigarrow v$

(ABS) $\rho \vdash_{\text{Expr}} \texttt{[}x_1\texttt{:}t_1\texttt{,} \ldots \texttt{,}x_n\texttt{:}t_n\texttt{]}e \rightsquigarrow inF(e, \lambda v_1 \ldots v_n.\rho[x_1 = v_1; \ldots; x_n = v_n])$

(APP) si $\rho \vdash_{\text{Expr}} e \rightsquigarrow inF(e', r)$, si $\rho \vdash_{\text{Expr}} e_1 \rightsquigarrow v_1$, ..., si $\rho \vdash_{\text{Expr}} e_n \rightsquigarrow v_n$
et si $r(v_1, \ldots, v_n) \vdash_{\text{Expr}} e' \rightsquigarrow v$
alors $\rho \vdash \texttt{(}e\ e_1 \ldots e_n\texttt{)} \rightsquigarrow v$

(APPR) si $\rho \vdash_{\text{Expr}} e \rightsquigarrow inFR(\varphi)$, si $\varphi(inFR(\varphi)) = inF(e', r)$,
si $\rho \vdash_{\text{Expr}} e_1 \rightsquigarrow v_1$, ..., si $\rho \vdash_{\text{Expr}} e_n \rightsquigarrow v_n$
et si $r(v_1, \ldots, v_n) \vdash_{\text{Expr}} e' \rightsquigarrow v$
alors $\rho \vdash \texttt{(}e\ e_1 \ldots e_n\texttt{)} \rightsquigarrow v$

1.3.3 Déclarations

Relation $\vdash_{\text{Dec}}$ dans $E \times \text{Dec} \times E$.
On écrit $\rho \vdash_{\text{Dec}} d : \rho'$

(CONST) si $\rho \vdash_{\text{Expr}} e \rightsquigarrow v$ alors $\rho \vdash_{\text{Dec}} \texttt{(CONST}\ x\ t\ e\texttt{)} \rightsquigarrow \rho[x = v]$

(FUN) $\rho \vdash_{\text{Dec}} \texttt{(FUN}\ x\ t\ \texttt{[}x_1\texttt{:}t_1\texttt{,} \ldots \texttt{,}x_n\texttt{:}t_n\texttt{]}\ e\texttt{)} \rightsquigarrow \rho[x = inF(e, \lambda v_1 \ldots v_n.\rho[x_1 = v_1; \ldots; x_n = v_n])$

(FUNREC) $\rho \vdash_{\text{Dec}} \texttt{(FUN REC}\ x\ t\ \texttt{[}x_1\texttt{:}t_1\texttt{,} \ldots \texttt{,}x_n\texttt{:}t_n\texttt{]}\ e\texttt{)}$
$\rightsquigarrow \rho[x = inFR(\lambda f.inF(e, \lambda v_1 \ldots v_n.\rho[x_1 = v_1; \ldots; x_n = v_n][x = f])$

1.3.4 Instruction

Relation $\vdash_{\text{Stat}}$ dans $E \times O \times \text{Stat} \times O$
On écrit $\rho, \omega \vdash_{\text{Stat}} s \rightsquigarrow \omega$

(ECHO) si $\rho, \omega \vdash_{\text{Expr}} e \rightsquigarrow inN(n)$ alors $\rho, \omega \vdash_{\text{Stat}} \texttt{ECHO}\ e \rightsquigarrow (n; \omega)$

1.3.5 Suite de commandes

Relation $\vdash_{\text{Cmds}}$ dans $E \times O \times \text{Cmds}_\varepsilon \times O$.
Notation: on écrit $\rho, \omega \vdash_{\text{Cmds}} cs \rightsquigarrow \omega'$

(DECS) si $\rho, \omega \vdash_{\text{Dec}} d \rightsquigarrow \rho'$ et si $\rho', \omega \vdash_{\text{Cmds}} cs \rightsquigarrow \omega'$ alors $\rho, \omega \vdash_{\text{Cmds}} (d\texttt{;}\ cs) \rightsquigarrow \omega'$

(STATS) si $\rho, \omega \vdash_{\text{Stat}} s \rightsquigarrow \omega'$ et si $\rho, \omega' \vdash_{\text{Cmds}} cs \rightsquigarrow \omega''$ alors $\rho, \omega \vdash_{\text{Cmds}} (s\texttt{;}\ cs) \rightsquigarrow \omega''$

(END) $\rho, \omega \vdash_{\text{Cmds}} \varepsilon \rightsquigarrow \omega$

1.3.6 Programme

On note $\emptyset$ le contexte vide et le flux de sortie vide.
Relation $\vdash$ dans $\text{Prog} \times O$

(PROG) si $\emptyset, \emptyset \vdash_{\text{Cmds}} cs\texttt{;}\varepsilon \rightsquigarrow \omega$ alors $\vdash \texttt{[}cs\texttt{]} \rightsquigarrow \omega$

2 APS1: noyau impératif

2.1 Syntaxe

2.1.1 Lexique

Mots clef

```
VAR PROC
SET IF WHILE CALL
```

2.1.2 Grammaire

DEC	::=	...
	|	`VAR` ident TYPE
	|	`PROC` ident `[` ARGS `]` PROG
	|	`PROC REC` ident `[` ARGS `]` PROG
STAT	::=	...
	|	`SET` ident EXPR
	|	`IF` EXPR PROG PROG
	|	`WHILE` EXPR PROG
	|	`CALL` ident EXPRS
TYPE	::=	...
	|	`void`

2.2 Typage

2.2.1 Déclarations

Relation $\vdash_{\text{DEC}}$ dans $G \times \text{DEC} \times G$

(VAR) $\Gamma \vdash_{\text{DEC}} (\texttt{VAR}\ x\ t) : \Gamma[x : t]$

(PROC) si $\Gamma[x_1 : t_1; \ldots; x_n : t_n] \vdash_{\text{CMDS}} (cs;\varepsilon) : \texttt{void}$
alors $\Gamma \vdash_{\text{DEC}} (\texttt{PROC}\ x\ \texttt{[}x_1{:}t_1, \ldots, x_n{:}t_n\texttt{]}\ \texttt{[}cs\texttt{]}) : \Gamma[x : t_1\ \texttt{*}\ \ldots\ \texttt{*}\ t_n\ \texttt{->}\ \texttt{void}]$

(PROCREC) si $\Gamma[x_1 : t_1; \ldots; x_n : t_n; x : t_1\ \texttt{*}\ \ldots\ \texttt{*}\ t_n\ \texttt{->}\ \texttt{void}] \vdash_{\text{CMDS}} (cs;\varepsilon) : \texttt{void}$
alors $\Gamma \vdash_{\text{DEC}} (\texttt{PROC REC}\ x\ \texttt{[}x_1{:}t_1, \ldots, x_n{:}t_n\texttt{]}\ \texttt{[}cs\texttt{]}) : \Gamma[x : t_1\ \texttt{*}\ \ldots\ \texttt{*}\ t_n\ \texttt{->}\ \texttt{void}]$

2.2.2 Instructions

(SET) si $\Gamma(x) = t$ et si $\Gamma \vdash_{\text{EXPR}} e : t$ alors $\Gamma \vdash_{\text{STAT}} (\texttt{SET}\ x\ e) : \texttt{void}$

(IF) si $\Gamma \vdash_{\text{EXPR}} e : \texttt{bool}$, si $\Gamma \vdash_{\text{CMDS}} cs_1 : \texttt{void}$ et si $\Gamma \vdash_{\text{CMDS}} cs_2 : \texttt{void}$ alors $\Gamma \vdash_{\text{STAT}} (\texttt{IF}\ e\ \texttt{[}cs_1\texttt{]}\ \texttt{[}cs_2\texttt{]}) : \texttt{void}$

(WHILE) si $\Gamma \vdash_{\text{EXPR}} e : \texttt{bool}$ et si $\Gamma \vdash_{\text{CMDS}} cs : \texttt{void}$ alors $\Gamma \vdash_{\text{STAT}} (\texttt{WHILE}\ e\ \texttt{[}cs\texttt{]}) : \texttt{void}$

(CALL) si $\Gamma(x) = t_1\ \texttt{*}\ \ldots\ \texttt{*}\ t_n\ \texttt{->}\ \texttt{void}$, si $\Gamma \vdash_{\text{EXPR}} e_1 : t_1, \ldots$ et si $\Gamma \vdash_{\text{EXPR}} e_n : t_n$
alors $\Gamma \vdash_{\text{STAT}} (\texttt{CALL}\ x\ e_1 \ldots e_n) : \texttt{void}$

2.3 Sémantique

2.3.1 Domaines et opérations sémantiques

Domaines

Adresse A

Fermetures procédurales $P = \textsc{Cmds} \times (V^* \to E)$

Fermetures procédurales récursives $PR = P \to P$

Valeurs $V\oplus = A \oplus P \oplus PR$

Mémoire $S = A \to N$ (fonction partielle)

Valeurs et opérations

Mémoire vide $\emptyset$, fonction jamais définie

Valeur indéterminée *any*

Extension mémoire fonction notée $\sigma[a = any]$ avec σ élément de S et a, élément de A

Allocation *alloc* de type $S \to (A \times S)$ telle que $alloc(\sigma) = (a, \sigma')$ si et seulement si a n'est pas dans le domaine de σ (nouvelle adresse) et $\sigma' = \sigma[a = any]$

Modification mémoire fonction de type $S \to A \to N \to S$, notée $\sigma[a := v]$ telle que $\sigma[a = v'][a := v] = \sigma[a = v]$ et $\sigma[a' = v'][a := v] = \sigma[a := v][a' = v']$ lorsque a est différent de a'. Notez que $\emptyset[a := v]$ n'est pas défini.

Restriction mémoire: restriction de σ aux valeurs allouées dans ρ: fonction de type $S \to E \to S$, notée (σ/ρ) telles que $(\sigma/\rho)(a) = \sigma(a)$ si et seulement il existe x tel que $\rho(x) = inA(a)$.

Un contexte d'évaluation est formé d'un environnement et d'une mémoire.

2.3.2 Expressions

Relation $\vdash_{\text{Expr}}$ dans $E \times S \times \textsc{Expr} \times V$
On écrit $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow v$

(ID1) si $x \in \mathsf{ident}$ et si $\rho(x) = inA(a)$ alors $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow inN(\sigma(a))$

(ID2) si $x \in \mathsf{ident}$, si $\rho(x) = v$ et si $v \neq inA(a)$ alors $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow v$

(TRUE) $\rho, \sigma \vdash_{\text{Expr}} \texttt{true} \rightsquigarrow inN(1)$

(FALSE) $\rho, \sigma \vdash_{\text{Expr}} \texttt{false} \rightsquigarrow inN(0)$

(NUM) si $n \in \mathsf{num}$ alors $\rho, \sigma \vdash_{\text{Expr}} n \rightsquigarrow inN(\nu(n))$

(PRIM) si $x \in \mathsf{oprim}$, si $\rho, \sigma \vdash_{\text{Expr}} e_1 \rightsquigarrow inN(n_1)$, ..., si $\rho, \sigma \vdash_{\text{Expr}} e_k \rightsquigarrow inN(n_k)$ et si $\pi(x)(n_1, \ldots, n_k) = n$
alors $\rho, \sigma \vdash_{\text{Expr}} \texttt{(}x\ e_1 \ldots e_n\texttt{)} \rightsquigarrow inN(n)$

(IF1) si $\rho, \sigma \vdash_{\text{Expr}} e_1 \rightsquigarrow inN(1)$ et si $\rho, \sigma \vdash_{\text{Expr}} e_2 \rightsquigarrow v$ alors $\rho, \sigma \vdash_{\text{Expr}} \texttt{(if}\ e_1\ e_2\ e_3\texttt{)} \rightsquigarrow v$

(IF0) si $\rho, \sigma \vdash_{\text{Expr}} e_1 \rightsquigarrow inN(0)$ et si $\rho, \sigma \vdash_{\text{Expr}} e_3 \rightsquigarrow v$ alors $\rho, \sigma \vdash_{\text{Expr}} \texttt{(if}\ e_1\ e_2\ e_3\texttt{)} \rightsquigarrow v$

(ABS) $\rho, \sigma \vdash_{\text{Expr}} \texttt{[}x_1\texttt{:}t_1\texttt{,} \ldots \texttt{,}x_n\texttt{:}t_n\texttt{]}e \rightsquigarrow inF(e, \lambda v_1 \ldots v_n.\rho[x_1 = v_1; \ldots; x_n = v_n])$

(APP) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow inF(e', r)$,
si $\rho, \sigma \vdash_{\text{Expr}} e_1 \rightsquigarrow v_1$, ..., si $\rho, \sigma \vdash_{\text{Expr}} e_n \rightsquigarrow v_n$ et si $r(v_1, \ldots, v_n), \sigma \vdash_{\text{Expr}} e' \rightsquigarrow v$
alors $\rho, \sigma \vdash \texttt{(}e\ e_1 \ldots e_n\texttt{)} \rightsquigarrow v$

(APPR) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow inFR(\varphi)$, si $\varphi(inFR(\varphi)) = inF(e', r)$,
si $\rho, \sigma \vdash_{\text{Expr}} e_1 \rightsquigarrow v_1$, ..., si $\rho, \sigma \vdash_{\text{Expr}} e_n \rightsquigarrow v_n$
et si $r(v_1, \ldots, v_n), \sigma \vdash_{\text{Expr}} e' \rightsquigarrow v$
alors $\rho, \sigma \vdash \texttt{(}e\ e_1 \ldots e_n\texttt{)} \rightsquigarrow v$

2.3.3 Déclarations

Relation $\vdash_{\text{Dec}}$ dans $E \times S \times \text{Dec} \times E \times S$

Notation: on écrit $\rho, \sigma \vdash_{\text{Dec}} d \rightsquigarrow (\rho', \sigma')$

(VAR) si $alloc(\sigma) = (a, \sigma')$ alors $\rho, \sigma \vdash_{\text{Dec}} (\texttt{VAR}\ x\ t) \rightsquigarrow (\rho[x = inA(a)], \sigma')$

(PROC) $\rho, \sigma \vdash_{\text{Dec}} (\texttt{PROC}\ x\ t\ \texttt{[}x_1\texttt{:}t_1\texttt{,} \ldots \texttt{,}x_n\texttt{:}t_n\texttt{]}\ bk) \rightsquigarrow (\rho[x = inP(bk, \lambda v_1 \ldots v_n.\rho[x_1 = v_1; \ldots ; x_n = v_n]), \sigma)$

(PROCREC) $\rho, \sigma \vdash_{\text{Dec}} (\texttt{PROC REC}\ x\ t\ \texttt{[}x_1\texttt{:}t_1\texttt{,} \ldots \texttt{,}x_n\texttt{:}t_n\texttt{]}\ bk)$
$\rightsquigarrow (\rho[x = inPR(\lambda f.inP(bk, \lambda v_1 \ldots v_n.\rho[x_1 = v_1; \ldots ; x_n = v_n][x = f]), \sigma)$

(CONST) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow v$ alors $\rho, \sigma \vdash_{\text{Dec}} (\texttt{CONST}\ x\ t\ e) \rightsquigarrow (\rho[x = v], \sigma)$

(FUN) $\rho, \sigma \vdash_{\text{Dec}} (\texttt{FUN}\ x\ t\ \texttt{[}x_1\texttt{:}t_1\texttt{,} \ldots \texttt{,}x_n\texttt{:}t_n\texttt{]}\ e) \rightsquigarrow (\rho[x = inF(e, \lambda v_1 \ldots v_n.\rho[x_1 = v_1; \ldots ; x_n = v_n]), \sigma)$

(FUNREC) $\rho, \sigma \vdash_{\text{Dec}} (\texttt{FUN REC}\ x\ t\ \texttt{[}x_1\texttt{:}t_1\texttt{,} \ldots \texttt{,}x_n\texttt{:}t_n\texttt{]}\ e)$
$\rightsquigarrow (\rho[x = inFR(\lambda f.inF(e, \lambda v_1 \ldots v_n.\rho[x_1 = v_1; \ldots ; x_n = v_n][x = f]), \sigma)$

2.3.4 Instructions

Relation $\vdash_{\text{Stat}}$ dans $E \times S \times O \times \text{Stat} \times S \times O$.

On écrit $\rho, \sigma, \omega \vdash_{\text{Stat}} s \rightsquigarrow (\sigma', \omega')$

(SET) si $\rho(x) = inA(a)$ et si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow v$ alors $\rho, \sigma, \omega \vdash_{\text{Stat}} (\texttt{SET}\ x\ e) \rightsquigarrow (\sigma[a := v], \omega)$

(IF1) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow inN(1)$ et si $\rho, \sigma, \omega \vdash_{\text{Block}} bk_1 \rightsquigarrow (\sigma', \omega')$ alors $\rho, \sigma, \omega \vdash_{\text{Stat}} (\texttt{IF}\ e\ bk_1\ bk_2) \rightsquigarrow (\sigma', \omega')$

(IF0) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow inN(0)$ et si $\rho, \sigma, \omega \vdash_{\text{Block}} bk_2 \rightsquigarrow (\sigma', \omega')$ alors $\rho, \sigma, \omega \vdash_{\text{Stat}} (\texttt{IF}\ e\ bk_1\ bk_2) \rightsquigarrow (\sigma', \omega')$

(LOOP0) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow inN(0)$ alors $\rho, \sigma, \omega \vdash_{\text{Stat}} (\texttt{WHILE}\ e\ bk) \rightsquigarrow (\sigma, \omega)$

(LOOP1) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow inN(1)$, si $\rho, \sigma, \omega \vdash_{\text{Block}} bk \rightsquigarrow (\sigma', \omega')$
et si $\rho, \sigma', \omega' \vdash_{\text{Stat}} (\texttt{WHILE}\ e\ bk) \rightsquigarrow (\sigma'', \omega'')$
alors $\rho, \sigma, \omega \vdash_{\text{Stat}} (\texttt{WHILE}\ e\ bk) \rightsquigarrow (\sigma'', \omega'')$

(CALL) si $\rho(x) = inP(bk, r)$,
si $\rho, \sigma \vdash_{\text{Expr}} e_1 \rightsquigarrow v_1$, ..., si $\rho, \sigma \vdash_{\text{Expr}} e_n \rightsquigarrow v_n$
et si $r(v_1, \ldots, v_n), \sigma, \omega \vdash_{\text{Expr}} bk \rightsquigarrow (\sigma', \omega')$
alors $\rho, \sigma, \omega \vdash (\texttt{CALL}\ x\ e_1 \ldots e_n) \rightsquigarrow (\sigma', \omega')$

(CALLR) si $\rho(x) = inFR(\varphi)$, si $\varphi(inFR(\varphi)) = inP(bk, r)$,
si $\rho, \sigma \vdash_{\text{Expr}} e_1 \rightsquigarrow v_1$, ..., si $\rho, \sigma \vdash_{\text{Expr}} e_n \rightsquigarrow v_n$
et si $r(v_1, \ldots, v_n), \sigma, \omega \vdash_{\text{Expr}} bk \rightsquigarrow (\sigma', \omega')$
alors $\rho, \sigma, \omega \vdash (\texttt{CALL}\ x\ e_1 \ldots e_n) \rightsquigarrow (\sigma', \omega')$

(ECHO) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow inN(n)$ alors $\rho, \sigma \vdash_{\text{Stat}} (\texttt{RETURN}\ e) \rightsquigarrow (\sigma, (n; \omega))$

2.3.5 Suite de commandes

Relation $\vdash_{\text{Cmds}}$ dans $E \times S \times O \times (\text{Cmds}_\varepsilon) \times S \times O$.

Notation: on écrit $\rho, \sigma, \omega \vdash_{\text{Cmds}} cs \rightsquigarrow (\sigma', \omega')$

(DECS) si $\rho, \sigma \vdash_{\text{Dec}} d \rightsquigarrow (\rho', \sigma')$ et si $\rho', \sigma', \omega \vdash_{\text{Cmds}} cs \rightsquigarrow (\sigma'', \omega')$ alors $\rho, \omega \vdash_{\text{Cmds}} (d\texttt{;}\ cs) \rightsquigarrow (\sigma'', \omega')$

(STATS) si $\rho, \sigma, \omega \vdash_{\text{Stat}} s \rightsquigarrow (\sigma', \omega')$ et si $\rho, \sigma', \omega' \vdash_{\text{Cmds}} cs \rightsquigarrow (\sigma'', \omega'')$ alors $\rho, \sigma, \omega \vdash_{\text{Cmds}} (s\texttt{;}\ cs) \rightsquigarrow (\sigma'', \omega'')$

(END) $\rho, \sigma, \omega \vdash_{\text{Cmds}} \varepsilon \rightsquigarrow (\sigma, \omega)$

2.3.6 Blocs de commandes

Relation $\vdash_{\text{Block}}$ dans $E \times S \times 0 \times \text{Prog} \times S \times O$
On écrit $\rho, \sigma, \omega \vdash bk \rightsquigarrow (\sigma', \omega')$

(BLOCK) si $\rho, \sigma, \omega \vdash_{\text{Cmds}} cs\texttt{;}\varepsilon \rightsquigarrow (\sigma', \omega')$ alors $\rho, \sigma, \omega \vdash_{\text{Block}} \texttt{[}cs\texttt{]} \rightsquigarrow ((\sigma'/\rho), \omega')$

2.3.7 Programme

Relation $\vdash$ dans $\text{Prog} \times S \times O$.
On écrit $\vdash \texttt{[}cs\texttt{]} \rightsquigarrow (\sigma, \omega)$.

(PROG) si $\emptyset, \emptyset, \emptyset \vdash_{\text{Cmds}} cs\texttt{;}\varepsilon \rightsquigarrow (\sigma, \omega)$ alors $\vdash \texttt{[}cs\texttt{]} \rightsquigarrow (\sigma, \omega)$.

3 APS2: tableaux

3.1 Syntaxe

3.1.1 Lexique

Type
`vec`

Opérateurs primitifs (oprim)
`len nth alloc`

Mots clef
`alloc`

3.1.2 Grammaire

Dec	::=	...
	|	`CONST` ident Type Expr
Stat	::=	...
	|	`SET` Lval Expr
Lval	::=	ident
	|	`(nth` Lval Expr `)`
Type	::=	...
	|	`(vec` Type `)`

3.2 Typage

3.2.1 Expression

Relation de typage: $\vdash_{\text{Expr}}$ dans $G \times \text{Expr} \times \text{Type}$
On écrit $\Gamma \vdash_{\text{Expr}} e : t$.
Les opérateurs primitifs sont *polymorphes*. On traite ce polymorphisme au niveau des règles.

(ALLOC) pour tout $t \in \text{Type}$, si $\Gamma \vdash_{\text{Expr}} e : \texttt{int}$ alors $\Gamma \vdash_{\text{Expr}} \texttt{(alloc } e\texttt{)} : \texttt{(vec } t\texttt{)}$

(NTH) pour tout $t \in \text{Type}$, si $\Gamma \vdash_{\text{Expr}} e_1 : \texttt{(vec } t\texttt{)}$ et si $\Gamma \vdash_{\text{Expr}} e_2 : \texttt{int}$ alors $\Gamma \vdash_{\text{Expr}} \texttt{(nth } e_1\ e_2\texttt{)} : t$

(LEN) pour tout $t \in \text{Type}$, si $\Gamma \vdash_{\text{Expr}} e : \texttt{(vec } t\texttt{)}$ alors $\Gamma \vdash_{\text{Expr}} \texttt{(len } e\texttt{)} : \texttt{int}$

3.2.2 *Left value*

Relation $\vdash_{\text{LVAL}}$ dans $G \times \text{LVAL} \times \text{TYPE}$
On écrit $G \vdash_{\text{LVAL}} e : t$

(LVAL) si $e \in \text{LVAL}$ et $G \vdash_{\text{EXPR}} e : t$ alors $G \vdash_{\text{LVAL}} e : t$

3.2.3 Instructions

(SET) si $\Gamma \vdash_{\text{EXPR}} lv : t$ et si $\Gamma \vdash_{\text{EXPR}} rv : t$ alors $\Gamma \vdash_{\text{STAT}} (\texttt{SET}\ lv\ rv) : \texttt{void}$

3.2.4 Déclarations

(CONST) si $\Gamma \vdash_{\text{EXPR}} rv : t$ alors $\Gamma \vdash_{\text{DEC}} (\texttt{CONST}\ x\ t\ rv) : \Gamma[x : t]$

3.3 Sémantique

3.3.1 Domaines et opérations

Domaines

Adresses $A = N$ (ordonnées avec incrément)

Blocs mémoires $B = A \times N$

Valeurs $V \oplus = B$

Mémoire $M = A \to N \oplus B$

Opérations

Allocation *allocn*, de type $S \times N \to (A \times S)$ telle que $allocn(\sigma, n) = (a, \sigma')$ si et seulement si, pour tout $i \in [0, n[$, $a + i$ n'est pas dans le domaines de σ et $\sigma' = \sigma[a = any; \ldots ; a + n - 1 = any]$.

Restriction mémoire: ensemble des adresses accessibles depuis l'environemment. On pose

$$\begin{array}{rcl} A_0 & = & \bigcup_{x \in \mathsf{ident}} \{a \mid \rho(x) = inA(a)\} \\ A_{n+1} & = & \bigcup_{a \in A_n} \bigcup_{i=0}^{n-1} \{a' + i \mid \sigma(a) = inB(a', n)\} \end{array}$$

On définit (σ/ρ) comme la restriction de σ au domaine $\bigcup_{i \in I\!N} A_i$

3.3.2 Expressions

Relation $\vdash_{\text{EXPR}}$ dans $E \times S \times \text{EXPR} \times V \times S$
On écrit $\rho, \sigma \vdash_{\text{EXPR}} e \rightsquigarrow (v, \sigma')$

(ALLOC) si $\rho, \sigma \vdash_{\text{EXPR}} e \rightsquigarrow (inN(n), \sigma')$ et si $allocn(\sigma', n) = (a, \sigma'')$
alors $\rho, \sigma \vdash_{\text{EXPR}} (\texttt{alloc}\ e) \rightsquigarrow (inB(a, n), \sigma'')$

(NTH) si $\rho, \sigma \vdash_{\text{EXPR}} e_1 \rightsquigarrow (inB(a, n), \sigma')$ et si $\rho, \sigma' \vdash_{\text{EXPR}} (inN(i), \sigma'')$
alors $\rho, \sigma \vdash_{\text{EXPR}} (\texttt{nth}\ e_1\ e_2) \rightsquigarrow (\sigma''(a + i), \sigma'')$

(LEN) si $\rho, \sigma \vdash_{\text{EXPR}} e \rightsquigarrow (inB(a, n), \sigma')$ alors $\rho, \sigma \vdash_{\text{EXPR}} (\texttt{len}\ e) \rightsquigarrow (inN(n), \sigma')$

(ID1) si $x \in \mathsf{ident}$ et si $\rho(x) = inA(a)$ alors $\rho, \sigma \vdash_{\text{EXPR}} e \rightsquigarrow (inN(\sigma(a)), \sigma)$

(ID2) si $x \in \mathsf{ident}$, si $\rho(x) = v$ et si $v \neq inA(a)$ alors $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (v, \sigma)$

(TRUE) $\rho, \sigma \vdash_{\text{Expr}} \texttt{true} \rightsquigarrow (inN(1), \sigma)$

(FALSE) $\rho, \sigma \vdash_{\text{Expr}} \texttt{false} \rightsquigarrow (inN(0), \sigma)$

(NUM) si $n \in \mathsf{num}$ alors $\rho, \sigma \vdash_{\text{Expr}} n \rightsquigarrow (inN(\nu(n)), \sigma)$

(PRIM) si $x \in \mathsf{oprim}$, si $\rho, \sigma \vdash_{\text{Expr}} e_1 \rightsquigarrow (inN(n_1), \sigma_1)$, $\ldots$, si $\rho, \sigma_{k-1} \vdash_{\text{Expr}} e_k \rightsquigarrow (inN(n_k), \sigma_k)$ et si $\pi(x)(n_1, \ldots, n_k) = n$
alors $\rho, \sigma \vdash_{\text{Expr}} \texttt{(}x\ e_1 \ldots e_n\texttt{)} \rightsquigarrow (inN(n), \sigma_k)$

(IF1) si $\rho, \sigma \vdash_{\text{Expr}} e_1 \rightsquigarrow (inN(1), \sigma')$ et si $\rho, \sigma' \vdash_{\text{Expr}} e_2 \rightsquigarrow (v, \sigma'')$ alors $\rho, \sigma \vdash_{\text{Expr}} \texttt{(if}\ e_1\ e_2\ e_3\texttt{)} \rightsquigarrow (v, \sigma'')$

(IF0) si $\rho, \sigma \vdash_{\text{Expr}} e_1 \rightsquigarrow (inN(0), \sigma')$ et si $\rho, \sigma' \vdash_{\text{Expr}} e_3 \rightsquigarrow (v, \sigma'')$ alors $\rho, \sigma \vdash_{\text{Expr}} \texttt{(if}\ e_1\ e_2\ e_3\texttt{)} \rightsquigarrow (v, \sigma'')$

(ABS) $\rho, \sigma \vdash_{\text{Expr}} \texttt{[}x_1 \texttt{:} t_1 \texttt{,} \ldots \texttt{,} x_n \texttt{:} t_n\texttt{]}e \rightsquigarrow (inF(e, \lambda v_1 \ldots v_n.\rho[x_1 = v_1; \ldots; x_n = v_n]), \sigma)$

(APP) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (inF(e', r), \sigma')$,
si $\rho, \sigma' \vdash_{\text{Expr}} e_1 \rightsquigarrow (v_1, \sigma_1)$, $\ldots$, si $\rho, \sigma_{n-1} \vdash_{\text{Expr}} e_n \rightsquigarrow (v_n, \sigma_n)$ et si $r(v_1, \ldots, v_n), \sigma_n \vdash_{\text{Expr}} e' \rightsquigarrow (v, \sigma'')$
alors $\rho, \sigma \vdash \texttt{(}e\ e_1 \ldots e_n\texttt{)} \rightsquigarrow (v, \sigma'')$

(APPR) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (inFR(\varphi), \sigma')$, si $\varphi(inFR(\varphi)) = inF(e', r)$,
si $\rho, \sigma' \vdash_{\text{Expr}} e_1 \rightsquigarrow (v_1, \sigma_1)$, $\ldots$, si $\rho, \sigma_{n-1} \vdash_{\text{Expr}} e_n \rightsquigarrow (v_n, \sigma_n)$
et si $r(v_1, \ldots, v_n), \sigma_n \vdash_{\text{Expr}} e' \rightsquigarrow (v, \sigma'')$
alors $\rho, \sigma \vdash \texttt{(}e\ e_1 \ldots e_n\texttt{)} \rightsquigarrow (v, \sigma'')$

3.3.3 Déclaration

Relation $\vdash_{\text{Dec}}$ dans $E \times S \times \text{Dec} \times E \times S$

(CONST) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (v, \sigma')$ alors $\rho, \sigma \vdash_{\text{Dec}} \texttt{(CONST}\ x\ t\ e\texttt{)} \rightsquigarrow (\rho[x = v], \sigma')$

3.3.4 *Left value*

Relation $\vdash_{\text{Lval}}$ dans $E \times S \times \text{Lval} \times A \times S$
On écrit $\rho, \sigma \vdash_{\text{Lval}} lv \rightsquigarrow (a, \sigma')$

(LID) si $x \in \mathsf{ident}$ et si $\rho(x) = inA(a)$ alors $\rho, \sigma \vdash_{\text{Lval}} x \rightsquigarrow (a, \sigma)$

(LNTH) si $\rho, \sigma \vdash_{\text{Expr}} lv \rightsquigarrow (inB(a, n), \sigma')$ et si $\rho, \sigma' \vdash_{\text{Expr}} \rightsquigarrow (inN(i), \sigma'')$
alors $\rho, \sigma \vdash_{\text{Lval}} \texttt{(nth}\ lv\ e\texttt{)} \rightsquigarrow (a + i, \sigma'')$

3.3.5 Instructions

Relation $\vdash_{\text{Stat}}$ dans $E \times S \times O \times \text{Stat} \times S \times O$
On écrit $\rho, \sigma, \omega \vdash_{\text{Stat}} s \rightsquigarrow (\sigma', \omega')$

(SET) $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (v, \sigma')$ et si $\rho, \sigma' \vdash_{\text{Lval}} lv \rightsquigarrow (a, \sigma'')$ alors $\rho, \sigma, \omega \vdash_{\text{Stat}} \texttt{(SET}\ lv\ rv\texttt{)} \rightsquigarrow (\sigma''[a := v], \omega)$

(IF1) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (inN(1), \sigma')$ et si $\rho, \sigma', \omega \vdash_{\text{Block}} bk_1 \rightsquigarrow (\sigma'', \omega')$ alors $\rho, \sigma, \omega \vdash_{\text{Stat}} \texttt{(IF}\ e\ bk_1\ bk_2\texttt{)} \rightsquigarrow (\sigma'', \omega')$

(IF0) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (inN(0), \sigma')$ et si $\rho, \sigma, \omega \vdash_{\text{Block}} bk_2 \rightsquigarrow (\sigma'', \omega')$ alors $\rho, \sigma, \omega \vdash_{\text{Stat}} \texttt{(IF}\ e\ bk_1\ bk_2\texttt{)} \rightsquigarrow (\sigma'', \omega')$

(LOOP0) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (inN(0), \sigma')$ alors $\rho, \sigma, \omega \vdash_{\text{Stat}}$ $(\texttt{WHILE}\ e\ bk) \rightsquigarrow (\sigma', \omega)$

(LOOP1) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (inN(1), \sigma')$, si $\rho, \sigma', \omega \vdash_{\text{Block}} bk \rightsquigarrow (\sigma'', \omega')$
et si $\rho, \sigma'', \omega' \vdash_{\text{Stat}} (\texttt{WHILE}\ e\ bk) \rightsquigarrow (\sigma''', \omega'')$
alors $\rho, \sigma, \omega \vdash_{\text{Stat}} (\texttt{WHILE}\ e\ bk) \rightsquigarrow (\sigma''', \omega'')$

(CALL) si $\rho(x) = inP(bk, r)$,
si $\rho, \sigma \vdash_{\text{Expr}} e_1 \rightsquigarrow (v_1, \sigma_1), \ldots,$ si $\rho, \sigma_{n-1} \vdash_{\text{Expr}} e_n \rightsquigarrow (v_n, \sigma_n)$
et si $r(v_1, \ldots, v_n), \sigma_n, \omega \vdash_{\text{Expr}} bk \rightsquigarrow (\sigma', \omega')$
alors $\rho, \sigma, \omega \vdash (\texttt{CALL}\ x\ e_1 \ldots e_n) \rightsquigarrow (\sigma', \omega')$

(CALLR) si $\rho(x) = inFR(\varphi)$, si $\varphi(inFR(\varphi)) = inP(bk, r)$,
si $\rho, \sigma \vdash_{\text{Expr}} e_1 \rightsquigarrow (v_1, \sigma_1), \ldots,$ si $\rho, \sigma_{n-1} \vdash_{\text{Expr}} e_n \rightsquigarrow (v_n, \sigma_n)$
et si $r(v_1, \ldots, v_n), \sigma_n, \omega \vdash_{\text{Expr}} bk \rightsquigarrow (\sigma', \omega')$
alors $\rho, \sigma, \omega \vdash (\texttt{CALL}\ x\ e_1 \ldots e_n) \rightsquigarrow (\sigma', \omega')$

(ECHO) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (inN(n), \sigma')$ alors $\rho, \sigma \vdash_{\text{Stat}} (\texttt{RETURN}\ e) \rightsquigarrow (\sigma', (n; \omega))$

4 APS3: fonctions procédurales

4.1 Syntaxe

4.1.1 Lexique

Mot clef
`RETURN`

4.1.2 Grammaire

CMDS	::=	...
	|	RET
RET	::=	`RETURN` EXPR
DEC	::=	...
	|	`FUN` ident TYPE `[` ARGS `]` PROG
	|	`FUN REC` ident TYPE `[` ARGS `]` PROG

4.2 Typage

4.2.1 `RETURN`

(RET) si $\Gamma \vdash_{\text{Expr}} e : t$ alors $\Gamma \vdash_{\text{Ret}} (\texttt{RETURN}\ e) : t$

4.2.2 Instructions

Type somme $t + \texttt{void}$ avec $\texttt{void} + \texttt{void} = \texttt{void}$

(SET) si $\Gamma \vdash_{\text{Lval}} lv : t$ et si $\Gamma \vdash_{\text{Expr}} rv : t$ alors $\Gamma \vdash_{\text{Stat}} (\texttt{SET}\ lv\ rv) : \texttt{void}$

(IF0) si $\Gamma \vdash_{\text{Expr}} e : bool$, si $\Gamma \vdash_{\text{Block}} blk_1 : t$ et si $\Gamma \vdash_{\text{Block}} blk_2 : t$ alors $\Gamma \vdash_{\text{Stat}} (\texttt{IF}\ e\ blk_1\ blk_2) : t$

(IF1) si $\Gamma \vdash_{\text{Expr}} e : bool$, si $\Gamma \vdash_{\text{Block}} blk_1 : \texttt{void}$ et si $\Gamma \vdash_{\text{Block}} blk_2 : t$ avec $t \neq \texttt{void}$
alors $\Gamma \vdash_{\text{Stat}} (\texttt{IF}\ e\ blk_1\ blk_2) : t + \texttt{void}$

(IF2) si $\Gamma \vdash_{\textsc{Expr}} e : bool$, si $\Gamma \vdash_{\textsc{Block}} blk_1 : t$ avec $t \neq \texttt{void}$ et si $\Gamma \vdash_{\textsc{Block}} blk_2 : \texttt{void}$
alors $\Gamma \vdash_{\textsc{Stat}} (\texttt{IF}\ e\ blk_1\ blk_2) : t + \texttt{void}$

(WHILE) si $\Gamma \vdash_{\textsc{Expr}} e : bool$ et si $\Gamma \vdash_{\textsc{Block}} blk : t$ alors $\Gamma \vdash_{\textsc{Stat}} (\texttt{WHILE}\ e\ blk) : t + \texttt{void}$

(CALL) si $\Gamma \vdash_{\textsc{Expr}} x : t_1$ `* ... *` t_n `-> void`, si $\Gamma \vdash_{\textsc{Expr}} e_1 : t_1, \ldots$ et si $\Gamma \vdash_{\textsc{Expr}} e_n : t_n$
alors $\Gamma \vdash_{\textsc{Stat}} (\texttt{CALL}\ x\ e_1 \ldots e_n) : \texttt{void}$

4.2.3 Déclarations

(CONST) si $G \vdash_{\textsc{Expr}} rv : t$ alors $G \vdash_{\textsc{Dec}} (\texttt{CONST}\ x\ t\ rv) : G[x : t]$

4.2.4 Suite de commandes

Polymorphisme de ε

(STAT0) si $\Gamma \vdash_{\textsc{Stat}} s : \texttt{void}$ et $\Gamma \vdash_{\textsc{Cmds}} cs : t$ alors $\Gamma \vdash_{\textsc{Cmds}} s\texttt{;}cs : t$

(STAT1) si $\Gamma \vdash_{\textsc{Stat}} s : t + \texttt{void}$ et $\Gamma \vdash_{\textsc{Cmds}} cs : t$ alors $\Gamma \vdash_{\textsc{Cmds}} s\texttt{;}cs : t$

(DEC) si $\Gamma \vdash_{\textsc{Dec}} d : \Gamma'$ et si $\Gamma' \vdash_{\textsc{Cmds}} cs : t$ alors $\Gamma \vdash_{\textsc{Cmds}} d\texttt{;}cs : t$

(END1) si $\Gamma \vdash_{\textsc{Ret}} r : t$ alors $\Gamma \vdash_{\textsc{Cmds}} (r\texttt{;}\varepsilon) : t$

(END0) si $\Gamma \vdash_{\textsc{Stat}} s$: void alors $\Gamma \vdash_{\textsc{Cmds}} (s\varepsilon) : \texttt{void}$

4.2.5 Blocs

(BLOCK) si $\Gamma \vdash_{\textsc{Cmds}} (cs\texttt{;}\varepsilon) : t$ alors $\Gamma \vdash_{\textsc{Block}} \texttt{[}cs\texttt{]} : t$

4.2.6 Programme

(PROG) si $\Gamma_0 \vdash_{\textsc{Block}} p : \texttt{void}$ alors $\vdash p : \texttt{void}$

4.3 Sémantique

4.3.1 Expression et allocation (*Right value*)

Relation $\vdash_{\textsc{Expr}}$ dans $E \times S \times \times O \times \textsc{Expr} \times V \times S \times O$
On écrit $\rho, \sigma, \omega \vdash_{\textsc{Expr}} rv \rightsquigarrow (v, \sigma', \omega')$

(TRUE) $\rho, \sigma, \omega \vdash_{\textsc{Expr}} \texttt{true} \rightsquigarrow (inN(1), \sigma, \omega)$

(FALSE) $\rho, \sigma, \omega \vdash_{\textsc{Expr}} \texttt{false} \rightsquigarrow (inN(0), \sigma, \omega)$

(NUM) si $n \in \mathsf{num}$ alors $\rho, \sigma, \omega \vdash_{\textsc{Expr}} n \rightsquigarrow (inN(\nu(n)), \sigma, \omega)$

(ID1) si $x \in \mathsf{ident}$ et $\rho(x) = inA(a)$ alors $\rho, \sigma, \omega \vdash_{\textsc{Expr}} x \rightsquigarrow (inN(\sigma(a)), \sigma, \omega)$

(ID2) si $x \in \mathsf{ident}$ et $\rho(x) = v$, avec $v \neq inA(a)$ alors $\rho, \sigma, \omega \vdash_{\textsc{Expr}} x \rightsquigarrow (v, \sigma, \omega)$

(ALLOC) si $\rho, \sigma, \omega \vdash_{\textsc{Expr}} e \rightsquigarrow (inN(n), \sigma', \omega')$ et si $allocn(\sigma', n) = (a, \sigma'')$ alors $\rho, \sigma \vdash_{\textsc{Expr}} (\texttt{alloc}\ e) \rightsquigarrow (inB(a, n), \sigma'', \omega')$

(NTH) si $\rho, \sigma, \omega \vdash_{\textsc{Expr}} e_1 \rightsquigarrow (inB(a, n), \sigma', \omega')$ et si $\rho, \sigma', \omega' \vdash_{\textsc{Expr}} (inN(i), \sigma'', \omega'')$ alors $\rho, \sigma \vdash_{\textsc{Expr}} (\texttt{nth}\ e_1\ e_2) \rightsquigarrow (\sigma''(a + i), \sigma'', \omega'')$

(LEN) si $\rho, \sigma, \omega \vdash_{\textsc{Expr}} e \rightsquigarrow (inB(a, n), \sigma', \omega')$ alors $\rho, \sigma, \omega \vdash_{\textsc{Expr}} (\texttt{len}\ e) \rightsquigarrow (inN(n), \sigma', \omega')$

(PRIM) si $x \in$ oprim, si $\rho,\sigma,\omega \vdash_{\text{Expr}} e_1 \rightsquigarrow (inN(n_1),\sigma_1,\omega_1), \ldots,$ si $\rho,\sigma_{k-1},\omega_{k-1} \vdash_{\text{Expr}} e_k \rightsquigarrow (inN(n_k),\sigma_k,\omega_k)$ et si $\pi(x)(n_1,\ldots,n_k) = n$ alors $\rho,\sigma,\omega \vdash_{\text{Expr}} (x\ e_1 \ldots e_n) \rightsquigarrow (inN(n),\sigma_k,\omega_k)$

(IF1) si $\rho,\sigma,\omega \vdash_{\text{Expr}} e_1 \rightsquigarrow (inN(1),\sigma',\omega')$ et si $\rho,\sigma',\omega' \vdash_{\text{Expr}} e_2 \rightsquigarrow (v,\sigma'',\omega'')$ alors $\rho,\sigma,\omega \vdash_{\text{Expr}} (\texttt{if}\ e_1\ e_2\ e_3) \rightsquigarrow (v,\sigma'',\omega'')$

(IF2) si $\rho,\sigma,\omega \vdash_{\text{Expr}} e_1 \rightsquigarrow (inN(0),\sigma',\omega')$ et si $\rho,\sigma',\omega' \vdash_{\text{Expr}} e_3 \rightsquigarrow (v,\sigma'',\omega'')$ alors $\rho,\sigma,\omega \vdash_{\text{Expr}} (\texttt{if}\ e_1\ e_2\ e_3) \rightsquigarrow (v,\sigma'',\omega'')$

(ABS) $\rho,\sigma,\omega \vdash_{\text{Expr}} [x_1{:}t_1,\ldots,x_n{:}t_n]e \rightsquigarrow (inF(e,\lambda v_1\ldots v_n.\rho[x_1=v_1;\ldots;x_n=v_n]),\sigma,\omega)$

(APP) si $\rho,\sigma,\omega \vdash_{\text{Expr}} e \rightsquigarrow (inF(e',r),\sigma',\omega')$,
si $\rho,\sigma',\omega' \vdash_{\text{Expr}} e_1 \rightsquigarrow (v_1,\sigma_1,\omega_1), \ldots,$ si $\rho,\sigma_{n-1},\omega_{n-1} \vdash_{\text{Expr}} e_n \rightsquigarrow (v_n,\sigma_n,\omega_n)$
et si $r(v_1,\ldots,v_n),\sigma_n,\omega_n \vdash_{\text{Expr}} e' \rightsquigarrow (v,\sigma'',\omega'')$
alors $\rho,\sigma,\omega \vdash (e\ e_1 \ldots e_n) \rightsquigarrow (v,\sigma'',\omega'')$

(APPR) si $\rho,\sigma,\omega \vdash_{\text{Expr}} e \rightsquigarrow (inFR(\varphi),\sigma',\omega($, si $\varphi(inFR(\varphi)) = inF(e',r)$,
si $\rho,\sigma',\omega' \vdash_{\text{Expr}} e_1 \rightsquigarrow (v_1,\sigma_1,\omega_1), \ldots,$ si $\rho,\sigma_{n-1},\omega_{n-1} \vdash_{\text{Expr}} e_n \rightsquigarrow (v_n,\sigma_n,\omega_n)$
et si $r(v_1,\ldots,v_n),\sigma_n,\omega_n \vdash_{\text{Expr}} e' \rightsquigarrow (v,\sigma'',\omega'')$
alors $\rho,\sigma,\omega \vdash (e\ e_1 \ldots e_n) \rightsquigarrow (v,\sigma'',\omega'')$

(APP) si $\rho,\sigma,\omega \vdash_{\text{Expr}} e \rightsquigarrow (inP(bk,r),\sigma',\omega')$,
si $\rho,\sigma',\omega' \vdash_{\text{Expr}} e_1 \rightsquigarrow (v_1,\sigma_1,\omega_1), \ldots,$ si $\rho,\sigma_{n-1},\omega_{n-1} \vdash_{\text{Expr}} e_n \rightsquigarrow (v_n,\sigma_n,\omega_n)$
et si $r(v_1,\ldots,v_n),\sigma_n,\omega_n \vdash_{\text{Block}} bk \rightsquigarrow (v,\sigma'',\omega'')$
alors $\rho,\sigma,\omega \vdash (e\ e_1 \ldots e_n) \rightsquigarrow (v,\sigma'',\omega'')$

(APPR) si $\rho,\sigma,\omega \vdash_{\text{Expr}} e \rightsquigarrow (inPR(\varphi),\sigma',\omega')$, si $\varphi(inFR(\varphi)) = inF(bk,r)$,
si $\rho,\sigma',\omega' \vdash_{\text{Expr}} e_1 \rightsquigarrow (v_1,\sigma_1,\omega_1), \ldots,$ si $\rho,\sigma_{n-1},\omega_{n-1} \vdash_{\text{Expr}} e_n \rightsquigarrow (v_n,\sigma_n,\omega_n)$
et si $r(v_1,\ldots,v_n),\sigma_n,\omega_n \vdash_{\text{Block}} bk \rightsquigarrow (v,\sigma'',\omega'')$
alors $\rho,\sigma,\omega \vdash (e\ e_1 \ldots e_n) \rightsquigarrow (v,\sigma'',\omega'')$

(ALLOC) si $\rho,\sigma,\omega \vdash_{\text{Expr}} e \rightsquigarrow (inN(n),\sigma',\omega')$ et si $allocn(\sigma',n) = (a,\sigma'')$ alors $\rho,\sigma,\omega \vdash_{\text{Expr}} (\texttt{alloc}\ e) \rightsquigarrow (inB(a,n),\sigma'',\omega')$

4.3.2 Déclaration

Relation $\vdash_{\text{Dec}}$ dans $E \times S \times O \times \text{Dec} \times E \times S \times O$

On écrit $\rho,\sigma,\omega \vdash_{\text{Dec}} d \rightsquigarrow (\rho',\sigma',\omega')$

(CONST) si $\rho,\sigma,\omega \vdash_{\text{Expr}} rv \rightsquigarrow (v,\sigma',\omega')$ alors $\rho,\sigma,\omega \vdash_{\text{Dec}} (\texttt{CONST}\ x\ t\ rv) \rightsquigarrow (\rho[x=v],\sigma',\omega')$

(VAR) si $alloc(\sigma) = (a,\sigma')$ alors $\rho,\sigma,\omega \vdash_{\text{Dec}} (\texttt{VAR}\ x\ t) \rightsquigarrow (\rho[x=inA(a)],\sigma',\omega)$

(FUN) $\rho,\sigma,\omega \vdash_{\text{Dec}} (\texttt{FUN}\ x\ t\ [x_1{:}t_1,\ldots,x_n{:}t_n]\ e)$
$\rightsquigarrow (\rho[x=inF(e,\lambda v_1\ldots v_n.\rho[x_1=v_1;\ldots;x_n=v_n]),\sigma,\omega)$

(FUNREC) $\rho,\sigma,\omega \vdash_{\text{Dec}} (\texttt{FUN REC}\ x\ t\ [x_1{:}t_1,\ldots,x_n{:}t_n]\ e)$
$\rightsquigarrow (\rho[x=inFR(\lambda f.inF(e,\lambda v_1\ldots v_n.\rho[x_1=v_1;\ldots;x_n=v_n][x=f]),\sigma,\omega)$

(FUNP) $\rho,\sigma,\omega \vdash_{\text{Dec}} (\texttt{FUN}\ x\ t\ [x_1{:}t_1,\ldots,x_n{:}t_n]\ bk)$
$\rightsquigarrow (\rho[x=inP(bk,\lambda v_1\ldots v_n.\rho[x_1=v_1;\ldots;x_n=v_n]),\sigma,\omega)$

(FUNPREC) $\rho,\sigma,\omega \vdash_{\text{Dec}} (\texttt{FUN REC}\ x\ t\ [x_1{:}t_1,\ldots,x_n{:}t_n]\ bk)$
$\rightsquigarrow (\rho[x=inPR(\lambda f.inP(bk,\lambda v_1\ldots v_n.\rho[x_1=v_1;\ldots;x_n=v_n][x=f]),\sigma,\omega)$

(PROC) $\rho,\sigma,\omega \vdash_{\text{Dec}} (\texttt{PROC}\ x\ t\ [x_1{:}t_1,\ldots,x_n{:}t_n]\ bk)$
$\rightsquigarrow (\rho[x=inP(bk,\lambda v_1\ldots v_n.\rho[x_1=v_1;\ldots;x_n=v_n]),\sigma,\omega)$

(PROCREC) $\rho,\sigma,\omega \vdash_{\text{Dec}} (\texttt{PROC REC}\ x\ t\ [x_1{:}t_1,\ldots,x_n{:}t_n]\ bk)$
$\rightsquigarrow (\rho[x=inPR(\lambda f.inP(bk,\lambda v_1\ldots v_n.\rho[x_1=v_1;\ldots;x_n=v_n][x=f]),\sigma,\omega)$

4.3.3 RETURN

Relation $\vdash_{\textsc{Ret}}$ dans $E \times S \times O \times \textsc{Ret} \times V \times S \times O$

On écrit $\rho, \sigma, \omega \vdash_{\textsc{Ret}} r \rightsquigarrow (v, \sigma', \omega')$

(RET) si $\rho, \sigma, \omega \vdash_{\textsc{Expr}} rv \rightsquigarrow (v, \sigma', \omega')$ alors $\rho, \sigma, \omega \vdash_{\textsc{Ret}} (\texttt{RETURN}\ rv) \rightsquigarrow (v, \sigma', \omega')$

4.3.4 Instruction

Une *non valeur*: ε. On pose $V_\varepsilon = V \cup \{\varepsilon\}$

Relation $\vdash_{\textsc{Stat}}$ dans $E \times S \times 0 \times \textsc{Stat} \times V_\varepsilon \times S \times O$

On écrit $\rho, \sigma, \omega \vdash_{\textsc{Stat}} s \rightsquigarrow (v, \sigma', \omega')$

(ECHO) si $\rho, \sigma, \omega \vdash_{\textsc{Expr}} e \rightsquigarrow (inN(n), \sigma', \omega')$ alors $\rho, \sigma, \omega \vdash_{\textsc{Stat}} (\texttt{ECHO}\ e) \rightsquigarrow (\varepsilon, \sigma', (n, \omega))$

(SET) si $\rho, \sigma, \omega \vdash_{\textsc{Lval}} lv \rightsquigarrow a$ et si $\rho, \sigma, \omega \vdash_{\textsc{Expr}} rv \rightsquigarrow (v, \sigma', \omega')$ alors $\rho, \sigma, \omega \vdash_{\textsc{Stat}} (\texttt{SET}\ lv\ rv) \rightsquigarrow (\varepsilon, \sigma'[x = v], \omega')$

(IF1) si $\rho, \sigma, \omega \vdash_{\textsc{Expr}} e \rightsquigarrow (inN(1), \sigma', \omega')$ et si $\rho, \sigma', \omega' \vdash_{\textsc{Block}} bk_1 \rightsquigarrow (v, \sigma'', \omega'')$
alors $\rho, \sigma, \omega \vdash_{\textsc{Stat}} (\texttt{IF}\ bk_1\ bk_2) \rightsquigarrow (v, \sigma'', \omega'')$

(IF2) si $\rho, \sigma, \omega \vdash_{\textsc{Expr}} e \rightsquigarrow (inN(0), \sigma', \omega')$ et si $\rho, \sigma', \omega' \vdash_{\textsc{Block}} bk_2 \rightsquigarrow (v, \sigma'', \omega'')$
alors $\rho, \sigma, \omega \vdash_{\textsc{Stat}} (\texttt{IF}\ bk_1\ bk_2) \rightsquigarrow (v, \sigma'', \omega'')$

(LOOP0) si $\rho, \sigma, \omega \vdash_{\textsc{Expr}} e \rightsquigarrow (inN(0), \sigma', \omega')$ alors $\rho, \sigma, \omega \vdash_{\textsc{Stat}} (\texttt{WHILE}\ e\ blk) \rightsquigarrow (\varepsilon, \sigma', \omega')$

(LOOP1) si $\rho, \sigma, \omega \vdash_{\textsc{Expr}} e \rightsquigarrow (inN(1), \sigma', \omega')$,
si $\rho, \sigma', \omega' \vdash_{\textsc{Block}} blk \rightsquigarrow (\varepsilon, \sigma'', \omega'')$ et si $\rho, \sigma'', \omega'' \vdash_{\textsc{Stat}} (\texttt{WHILE}\ e\ blk) \rightsquigarrow (v, \sigma''', \omega''')$
alors $\rho, \sigma, \omega \vdash_{\textsc{Stat}} (\texttt{WHILE}\ e\ blk) \rightsquigarrow (v, \sigma''', \omega''')$

(LOOP2) si $\rho, \sigma, \omega \vdash_{\textsc{Expr}} e \rightsquigarrow (inN(1), \sigma', \omega')$ et si $\rho, \sigma', \omega' \vdash_{\textsc{Block}} blk \rightsquigarrow (v, \sigma'', \omega'')$, avec $v \neq \varepsilon$
alors $\rho, \sigma, \omega \vdash_{\textsc{Stat}} (\texttt{WHILE}\ e\ blk) \rightsquigarrow (v, \sigma'', \omega'')$

(CALL) si $\rho(x) = inP(bk, r)$,
si $\rho, \sigma, \omega \vdash_{\textsc{Expr}} (e_1, \sigma_1) \rightsquigarrow (v_1, \sigma_1, \omega_1)$, ..., si $\rho, \sigma_{n-1}, \omega_{n-1} \vdash_{\textsc{Expr}} e_n \rightsquigarrow (v_n, \sigma_n, \omega_n)$
et si $r(v_1, \ldots, v_n), \sigma_n, \omega_n \vdash_{\textsc{Block}} bk \rightsquigarrow (v, \sigma', \omega')$
alors $\rho, \sigma, \omega \vdash_{\textsc{Stat}} (\texttt{CALL}\ x\ e_1 \ldots e_n) \rightsquigarrow (v, \sigma', \omega')$

(CALLR) si $\rho(x) = inPR(\varphi)$, si $\varphi(inFR(\varphi)) = inP(bk, r)$,
si $\rho, \sigma, \omega \vdash_{\textsc{Expr}} e_1 \rightsquigarrow (v_1, \sigma_1, \omega_1)$, ..., si $\rho, \sigma_{n-1}, \omega_{n-1} \vdash_{\textsc{Expr}} e_n \rightsquigarrow (v_n, \sigma_n, \omega_n)$
et si $r(v_1, \ldots, v_n), \sigma_n, \omega_n \vdash_{\textsc{Block}} bk \rightsquigarrow (v, \sigma', \omega'))$
alors $\rho, \omega \vdash_{\textsc{Stat}} (\texttt{CALL}\ x\ e_1 \ldots e_n) \rightsquigarrow (v, \sigma', \omega')$

4.3.5 Suite de commandes

Relation $\vdash_{\textsc{Cmds}}$ dans $E \times S \times O \times \textsc{Cmds}_\varepsilon \times V_\varepsilon \times S \times O$

On écrit $\rho, \sigma, \omega \vdash_{\textsc{Cmds}} cs \rightsquigarrow (v, \sigma', \omega')$

(STAT0) si $\rho, \sigma, \omega \vdash_{\textsc{Stat}} s \rightsquigarrow (\varepsilon, \sigma', \omega')$ et si $\rho, \sigma', \omega' \vdash_{\textsc{Cmds}} cs \rightsquigarrow (v, \sigma'', \omega'')$ alors $\rho, \sigma, \omega \vdash_{\textsc{Cmds}} (s\,\texttt{;}cs) \rightsquigarrow (v, \sigma'', \omega'')$

(STAT1) si $\rho, \sigma, \omega \vdash_{\textsc{Stat}} s \rightsquigarrow (v, \sigma', \omega')$ avec $v \neq \varepsilon$ alors $\rho, \sigma, \omega \vdash_{\textsc{Cmds}} (s\,\texttt{;}cs) \rightsquigarrow (v, \sigma', \omega')$

(END0) $\rho, \sigma, \omega \vdash_{\textsc{Cmds}} \varepsilon \rightsquigarrow (\varepsilon, \sigma, \omega)$

(END1) si $\rho, \sigma, \omega \vdash_{\textsc{Ret}} r \rightsquigarrow (v, \sigma', \omega')$ alors $\rho, \sigma, \omega \vdash_{\textsc{Cmds}} (r\,\texttt{;}cs) \rightsquigarrow (v, \sigma', \omega')$

4.3.6 Blocs

Relation $\vdash_{\textsc{Block}}$ dans $E \times S \times O \times \textsc{Cmds} \times V_\varepsilon \times S \times O$
On écrit $\rho, \sigma, \omega \vdash_{\textsc{Block}} bk \rightsquigarrow (v, \sigma', \omega')$

(BLOCK) si $\rho, \sigma, \omega \vdash_{\textsc{Cmds}} (cs\,\texttt{;}\,\varepsilon) \rightsquigarrow (v, \sigma', \omega')$ alors $\rho, \sigma, \omega \vdash_{\textsc{Block}} \texttt{[}cs\texttt{]} \rightsquigarrow (v, (\sigma'$
$rho), \omega')$

4.3.7 Programme

Relation $\vdash$ dans $\textsc{Prog} \times S \times O$
On écrit $\vdash p \rightsquigarrow (\sigma, \omega)$

(PROG) si $\emptyset, \emptyset, \emptyset \vdash_{\textsc{Cmds}} (cs\,\texttt{;}\,\varepsilon) \rightsquigarrow (\varepsilon, \sigma, \omega)$ alors $\vdash \texttt{[}cs\texttt{]} \rightsquigarrow (\sigma, \omega)$