/SU/FSI/MASTER/INFO/MU4IN503 (APS)

Analyse des Programmes et Sémantique

Janvier 2021

Pascal Manoury – Romain Demangeon
pascal.manoury@lip6.fr

5 : *APS1*

APS1

$APS0$ + noyau impératif

Instructions

- affectation
- alternative
- boucle
- séquence

- déclarations de *variables*
- définitions de *procédures*

- appel de *procédure*

Expressivité

$$APS0 \equiv APS1$$

Mêmes «*résultats*» mais avec plus de moyens

$APS0$ est *Turing-complet*
$APS1$ aussi

Syntaxe

Lexique nouveaux mots clef

`VAR PROC`

`SET IF WHILE CALL`

Grammaire

- extension de DEC et STAT
- nouveau non terminal : BLOCK
- modification de CMDS (séquence d'instructions)

Remarque :
Le non terminal BLOCK peut être le pint d'entrée de la grammaire (BLOCK $\equiv$ PROG)

Grammaire

Dec	: :=	...
	|	`VAR ident` Type
	|	`PROC ident [` Args `]` Block
	|	`PROC REC ident [` Args `]` Block
Stat	: :=	...
	|	`SET ident` Expr
	|	`IF` Expr Block Block
	|	`WHILE` Expr Block
	|	`CALL ident` Exprs
Block	: :=	`[` Cmds `]`
Cmds	: :=	Stat
	|	Dec ; Cmds
	|	Stat ; Cmds

Typage

Pour *APS1*

1. une relation de typage pour $\textsc{Block}$:
   $\Gamma \vdash_{\textsc{Block}} bk : \texttt{void}$
2. extensions de $\vdash_{\textsc{Dec}}$ et $\vdash_{\textsc{Stat}}$

Remarque : inutile de modifier $\vdash_{\textsc{Cmds}}$ malgrè la modification de la règle de grammaire pour $\textsc{Cmds}$.

Block

$\vdash_{\text{Block}}$

Les blocs en $APS1$ sont comme les programmes $APS0$

(BLOC) si $\Gamma \vdash_{\text{Cmds}} (cs; \varepsilon) : \texttt{void}$
alors $\Gamma \vdash_{\text{Block}} [cs] : \texttt{void}$

Déclaration de variable

$\vdash_{\text{DEC}}$

Une déclaration de variable ajoute à l'environnement

(VAR) si $t \in \{\texttt{int}, \texttt{bool}\}$
alors $\Gamma \vdash_{\text{DEC}} (\texttt{VAR}\ x\ t) : \Gamma[x : t]$

La restriction sur t sera justifiée par la sémantique.

Définitions de procédures

$\vdash_{\text{DEC}}$

Procédures $\approx$ fonctions avec «type de retour» `void`

(PROC) si $\Gamma[x_1 : t_1; \ldots; x_n : t_n] \vdash_{\text{BLOCK}} bk$: `void`
alors $\Gamma \vdash_{\text{DEC}}$ (`PROC` x $[x_1 : t_1, \ldots, x_n : t_n]\, bk$)
: $\Gamma[x$: t_1 `*` `...` `*` t_n `->` `void`$]$

(PROCREC) si $\Gamma[x_1 : t_1; \ldots; x_n : t_n;$
x : t_1 `*` `...` `*` t_n `->` `void`$]$
$\vdash_{\text{BLOCK}} bk$: `void`
alors $\Gamma \vdash_{\text{DEC}}$ (`PROC REC` x $[x_1 : t_1, \ldots, x_n : t_n]\, bk$)
: $\Gamma[x$: t_1 `*` `...` `*` t_n `->` `void`$]$

Instructions

⊢STAT

`SET` x e : vérifier que l'identificateur x et l'expression e ont le même type.

`IF` e bk_1 bk_2 : vérifier que l'expression e est de type `bool` et que les blocs bk_1 et bk_2 sont correctement typés (de type `void`).

`WHILE` e bk : vérifier que l'expression e est de type `bool` et que le bloc bk est correctement typé (de type `void`).

`CALL` x $e_1 \dots e_n$: vérifier la cohérence entre le type (fonctionnel) de l'identificateur x et les types des expressions $e_1 \dots e_n$.

Règles

$\vdash_{\text{STAT}}$

(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 $\Gamma \vdash_{\text{BLOCK}} bk_1 : \texttt{void}$ et si $\Gamma \vdash_{\text{BLOCK}} bk_2 : \texttt{void}$
alors $\Gamma \vdash_{\text{STAT}} (\texttt{IF}\ e\ bk_1\ bk_2) : \texttt{void}$

(WHILE) si $\Gamma \vdash_{\text{EXPR}} e : \texttt{bool}$
et si $\Gamma \vdash_{\text{BLOCK}} bk : \texttt{void}$
alors $\Gamma \vdash_{\text{STAT}} (\texttt{WHILE}\ e\ bk) : \texttt{void}$

(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 $\Gamma \vdash_{\text{STAT}} (\texttt{CALL}\ x\ e_1 \ldots e_n) : \texttt{void}$

Sémantique

APS1

Programmation impérative $\Rightarrow$ modification état *mémoire*

Modélisation de la mémoire :

association entre *adresses* et valeurs

APS1 : on se resteint aux valeurs *entières* (domaine Z)

- Domaine d'adresses (abstrait) : A
- Mémoire : $S = A \rightarrow Z$ (fonction partielle)

On note : $\sigma(a)$ la valeur de la mémoire σ à l'adresse a

$\emptyset$ la mémoire *vide*

Mémoire

Allocation

nouvelle adresse

Extension du domaine de la mémoire avec une valeur quelconque (notée *any*)

$$\sigma[a = any] \text{ (avec } a \notin \text{dom}(\sigma))$$

Spécification axiomatique de la fonction d'allocation
$alloc : S \to A \times S$

$$alloc(\sigma) = (a, \sigma')$$
si et seulement si
$$a \notin \text{dom}(\sigma) \text{ et } \sigma' = \sigma[a = any]$$

Mémoire

Modification

$\sigma[a := v]$ définie par

- $\sigma[a = v][a := v'] = \sigma[a = v']$
- $\sigma[a' = v][a := v'] = \sigma[a := v'][a' = v]$ lorsque $a \neq a'$

non définie si $a \notin \mathrm{dom}(\sigma)$

Valeurs sémantiques

Les *valeurs* des *variables* sont des *adresses*

domaine A

Les valeurs des procédures sont des *fermetures procédurales*

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

Procédures récursives : *fermetures procédurales récursives*

$$PR = V \to P$$

Ensembles des valeur spour $APS1$

$$V = Z \oplus F \oplus FR \oplus A \oplus P \oplus PR$$

On note $V_0 = Z \oplus F \oplus FR$ (valeurs $APS0$)

Relations sémantiques

Contexte d'évaluation :
environnement, mémoire, flux de sortie
$E \times S \times O$

Le résultat de l'exécution d'un programme est
un état mémoire et un flot de sortie

Programme : $\textsc{Prog} \times S \times O$

Suites de commandes et bloc :
$E \times S \times O \times \textsc{Cmds} \times S \times O$

Déclaration : $E \times S \times \textsc{Dec} \times E \times S$

Instruction : $E \times S \times O \times \textsc{Stat} \times S \times O$

Expression : $E \times S \times \textsc{Expr} \times V_0$

Programmes et blocs

$\vdash$ et $\vdash_{\text{Block}}$

Programmes : similaire à $APS0$ sauf pour le «résultat»

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

Blocs : le contexte d'évaluation est non nécessairement vide

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

Suites de commandes

$\vdash_{\text{CMDS}}$

Similaire à $APS0$ sauf pour les «résultats»

Une déclaration modifie l'environnement ou [1] la mémoire

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

Une instruction modifie la mémoire ou le flux de sortie

(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 $\rho, \sigma, \omega \vdash_{\text{CMDS}} (s\,;\ \ cs) \rightsquigarrow (\sigma'', \omega'')$

ε ne touche à rien

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

---

1. inclusif

Déclaration de variable

$\vdash_{\text{DEC}}$

1. La *valeur* d'une variable est une *adresse*
2. Une déclaration de variable affecte l'environnement **et** la mémoire
   - liaison nom/adresse dans l'environnement
   - extension mémoire (*alloc*)

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

avec $\sigma' = \sigma[a = \mathit{any}]$ et $a \notin \text{dom}(\sigma)$

Déclarations de procédures

$\vdash_{\text{DEC}}$

Similaires aux déclarations de fonctions, mais avec des fermeture *procédurales*.

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

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

Autres déclarations

$\vdash_{\text{DEC}}$

Pas de changement sauf contexte d'évaluation (ρ, σ)
Seul l'environnement est impacté

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

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

Affectation

$\vdash_{\text{STAT}}$

Modification de l'état mémoire

1. obtenir l'adresse associée à la variable $(\rho(x))$
2. calculer la nouvelle valeur $(e \rightsquigarrow v)$
3. ranger la valeur en mémoire $(\sigma[a := v])$

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

Alternative

$\vdash_{\text{STAT}}$

Similaire au `if` fonctionnel

Par cas sur la valeur de la condition

(IF1) si $\rho, \sigma \vdash_{\text{EXPR}} e \rightsquigarrow \mathit{inZ}(1)$
et si $\rho, \sigma, \omega \vdash_{\text{BLOCK}} \mathit{bk}_1 \rightsquigarrow (\sigma', \omega')$
alors $\rho, \sigma, \omega \vdash_{\text{STAT}} (\texttt{IF}\ e\ \mathit{bk}_1\ \mathit{bk}_2) \rightsquigarrow (\sigma', \omega')$

(IF0) si $\rho, \sigma \vdash_{\text{EXPR}} e \rightsquigarrow \mathit{inZ}(0)$
et si $\rho, \sigma, \omega \vdash_{\text{BLOCK}} \mathit{bk}_2 \rightsquigarrow (\sigma', \omega')$
alors $\rho, \sigma, \omega \vdash_{\text{STAT}} (\texttt{IF}\ e\ \mathit{bk}_1\ \mathit{bk}_2) \rightsquigarrow (\sigma', \omega')$

Boucle

$\vdash_{\text{STAT}}$

Définition *récursive* par cas sur la condition

(LOOP0) si $\rho, \sigma \vdash_{\text{EXPR}} e \rightsquigarrow \mathit{inZ}(0)$
alors $\rho, \sigma, \omega \vdash_{\text{STAT}} (\texttt{WHILE}\ e\ \mathit{bk}) \rightsquigarrow (\sigma, \omega)$

(LOOP1) si $\rho, \sigma \vdash_{\text{EXPR}} e \rightsquigarrow \mathit{inZ}(1)$,
si $\rho, \sigma, \omega \vdash_{\text{BLOCK}} \mathit{bk} \rightsquigarrow (\sigma', \omega')$
et si $\rho, \sigma', \omega' \vdash_{\text{STAT}} (\texttt{WHILE}\ e\ \mathit{bk}) \rightsquigarrow (\sigma'', \omega'')$
alors $\rho, \sigma, \omega \vdash_{\text{STAT}} (\texttt{WHILE}\ e\ \mathit{bk}) \rightsquigarrow (\sigma'', \omega'')$

Définition mal fondée
Possibilité de boucle infinie

Appel de procédure

$\vdash_{\text{STAT}}$

Similaire à l'appel de fonction

Plus restreint : nom de procédure uniquement

(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 $r(v_1, \ldots, v_n), \sigma, \omega \vdash_{\text{BLOCK}} bk \rightsquigarrow (\sigma', \omega')$
alors $\rho, \sigma, \omega \vdash (\texttt{CALL}\ x\ e_1 \ldots e_n) \rightsquigarrow (\sigma', \omega')$

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

Expression : identificateur

$\vdash_{\text{EXPR}}$

La valeur d'un identificateur est dans l'environnement : $\rho(x)$
On veut un résultat dans $V_0 = Z \oplus F \oplus FR$

Deux cas possibles :

1. c'est une variable, la valeur est une adresse
   $\Rightarrow$ *indirection* vers la mémoire
2. c'est une constante, un paramètre, etc., la valeur n'est pas une adresse
   $\Rightarrow$ la valeur est directement dans l'environnement

(ID1) si $\rho(x) = inA(a)$
alors $\rho, \sigma \vdash_{\text{EXPR}} x \rightsquigarrow inZ(\sigma(a))$

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

Autres expressions

$\vdash_{\text{EXPR}}$

Similaire à $APS0$ sauf contexte d'évaluation

$$\rho, \sigma \vdash_{\text{EXPR}} e \rightsquigarrow v$$

à la place de

$$\rho \vdash_{\text{EXPR}} e \rightsquigarrow v$$

- N'affecte pas le contexte d'évaluation
- seuls $(\text{ID}1)$ utilise explicitement la mémoire