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

Analyse des Programmes et Sémantique

Janvier 2021

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

6 : *APS1*, analyse critique

$APS1a$

Erreur d'exécution

Un programme

- syntaxiquement correct ;
- correctement typé ;
- **non évaluable**

```
[
  CONST x int 0;
  SET x 42
]
```

Par $(\textsc{const})$ $\emptyset, \emptyset \vdash_{\textsc{Dec}} \texttt{CONST x int 0} \rightsquigarrow [\mathrm{x} = \mathit{inZ}(0)]$
Par $(\textsc{set})$

1. $[\mathrm{x} = \mathit{inZ}(0)](\mathrm{x}) = \mathit{inZ}(0)$
2. la règle $(\textsc{set})$ demande $\mathit{inA}(a)$

Constantes *vs* variables

Rappel

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

APS1 : les constantes sont immuables

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

$\vdash_{\text{EXPR}}$ définie sur $E \times S \times \text{EXPR} \times V_0$ avec $V_0 = Z \oplus F \oplus FR$

Théorème : quelque soit ρ, σ, e et v, si
$\rho, \sigma \vdash_{\text{EXPR}} e \rightsquigarrow v$ *alors, quelque soit a, $v \neq inA(a)$*

Analyse de programme

Repérer les affectation de constantes.

Analyse statique[1] de cohérence du code

$\approx$

Analyse de type

1. Ajouter l'information «*modifiable*» au type des identificateurs (*à la OCAML* : `(ref int)`, `(ref bool)`)
2. L'intégrer aux règles de typage

1. avant évaluation

Variable et affectation

Langage de types (pour le typage uniquement)

$$\tau ::= \texttt{int} \mid \texttt{bool} \mid (\texttt{ref}\ \tau) \mid (\tau s)\ \texttt{->}\ \tau$$

$$\tau s ::= \tau \mid t * \tau s$$

Nouvelle règle de typage pour les déclarations de variable

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

Nouvelle règle de typage pour l'affectation

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

Expressions : identificateurs

On confond τ et $(\texttt{ref}\ \ \tau)$.

$(\textsc{idr})$ si $x \in \texttt{ident}$,
si $\Gamma(x) = (\texttt{ref}\ \ \tau)$
alors $\Gamma \vdash_{\textsc{Expr}} x : \tau$

$(\textsc{idv})$ si $x \in \texttt{ident}$,
si $\Gamma(x) = \tau$
alors $\Gamma \vdash_{\textsc{Expr}} x : \tau$

Effet de bord

(pas de)

Avec la nouvelle règle pour l'affectation

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

La déclaration `PROC inc [x:int] [ SET x (add x 1) ]`
n'est pas typable

En effet, $[\texttt{x} : \mathit{int}] \vdash_{\text{STAT}} (\texttt{SET x (add x 1)}) : \mathit{void}$ échoue
car $[\texttt{x} : \mathit{int}](\texttt{x}) = \mathit{int} \neq (\texttt{ref}\ \ \mathit{int})$

Passage par valeur

Passage par référence

Des adresses comme paramètres :

```
PROC inc [var x:int] [ SET x (add x 1) ]
```
[2]

Des adresses comme arguments

```
CALL inc (adr n)
```
[3]

Impact sur

1. la syntaxe
2. le typage
3. la sémantique

---

2. Inspiré du langage Pascal
3. Inspiré du langage C

Syntaxe (déclaration)

Lexique : nouveau mot clé : `var`

Grammaire : déclaration des procédures

Une nouvelle catégorie de paramètres : ARGP

DEC	: :=	...
	|	`PROC ident [` ARGSP `]` BLOCK
	|	`PROC REC ident [` ARGSP `]` BLOCK
ARGSP : :=	: :=	ARGP
	|	ARGP , ARGSP
ARGP	: :=	`ident :` TYPE
	|	`var ident :` TYPE

Typage (déclaration)

Posons : $p_i = x_i$ ou $p_i = \texttt{var}\ \ x_i$ avec $x_i \in \texttt{ident}$

Soit $A([p_1 : t_1, \ldots, x_n : t_n]) = [x_1 : t'_1, \ldots, x_n : t'_n]$ avec

$$t'_i = \begin{cases} t_i & \text{si } p_i = x_i \\ (\texttt{ref}\ \ t_i) & \text{si } p_i = \texttt{var}\ \ x_i \end{cases}$$

(PROC) si $A([p_1 : t_1, \ldots, x_n : t_n]) = [x_1 : t'_1, \ldots, x_n : t'_n]$
si $\Gamma[x_1 : t'_1; \ldots; x_n : t'_n] \vdash_{\text{BLOCK}} bk : \texttt{void}$
alors $\Gamma \vdash_{\text{DEC}} (\texttt{PROC}\ x\ [p_1 : t_1, \ldots, p_n : t_n]\, bk)$
$: \Gamma[x : t'_1\ \texttt{*}\ \ldots\ \texttt{*}\ t'_n\ \texttt{->}\ \texttt{void}]$

(PROCREC) si $A([p_1 : t_1, \ldots, x_n : t_n]) = [x_1 : t'_1, \ldots, x_n : t'_n]$
si $\Gamma[x_1 : t'_1; \ldots; x_n : t'_n;$
$x : t'_1\ \texttt{*}\ \ldots\ \texttt{*}\ t'_n\ \texttt{->}\ \texttt{void}]$
$\vdash_{\text{BLOCK}} bk : \texttt{void}$
alors $\Gamma \vdash_{\text{DEC}} (\texttt{PROC REC}\ x\ [p_1 : t_1, \ldots, p_n : t_n]\, bk)$
$: \Gamma[x : t'_1\ \texttt{*}\ \ldots\ \texttt{*}\ t'_n\ \texttt{->}\ \texttt{void}]$

Sémantique (déclaration)

Peu de changement : récupérer les noms des paramètres.

Soit $X([p_1 : t_1 , \ldots p_n : t_n]) = x_1, \ldots, x_n$

$$(\text{PROC})\ \ \rho, \sigma \vdash_{\text{DEC}} (\texttt{PROC}\ x\ t\ [p_1 : t_1 , \ldots p_n : t_n]\ bk) \rightsquigarrow (\rho[x = inP(bk, \lambda v_1 \ldots v_n.\rho[x_1 = v_1; \ldots; x_n = v_n]), \sigma)$$

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

Syntaxe (appel de procédure)

Expliciter le *passage par référence* des variables

Lexique : nouveau mot clé : `adr`

Grammaire : nouvelle catégorie d'expressions EXPRP

STAT	: :=	...
	|	`CALL ident` EXPRSP
EXPRSP	: :=	EXPRP
	|	EXPRP EXPRSP
EXPRP	: :=	EXPR
	|	`(adr ident)`

Analogie avec le `&x` de C

Typage (appel de procédure)

Traitement particulier des paramètres d'appel ($\textsc{Exprp}$) :

- simple expression
- ou référence

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

Nouvelle relation de typage : $\vdash_{\textsc{Exprp}}$

($\textsc{ref}$) si $\Gamma(x) = (\texttt{ref}\ t)$
alors $\Gamma \vdash_{\textsc{Exprp}} (\texttt{adr}\ x) : (\texttt{ref}\ t)$

($\textsc{val}$) si $\Gamma \vdash_{\textsc{Expr}} e : t$
alors $\Gamma \vdash_{\textsc{Exprp}} e : t$

Sémantique (appel de procédure)

Tenir compte du *passage par référence*

Nouvelle relation sémantique pour l'évaluation des paramètres d'appel : $\vdash_{\text{EXPRP}}$

Domaine étendu aux adresses : $V_1 = V_0 \oplus A$

Domaine de $\vdash_{\text{EXPRP}}$: $E \times S \times \text{EXPRP} \times V_1$

(REF) si $\rho(x) = inA(a)$
alors $\rho, \sigma \vdash_{\text{EXPRP}} (\texttt{adr}\ \ x) \rightsquigarrow inA(a)$

(VAL) si $\rho, \sigma \vdash_{\text{EXPR}} e \rightsquigarrow v$
alors $\rho, \sigma \vdash_{\text{EXPRP}} e \rightsquigarrow v$

Exemple

Le programme

```
[
  PROC inc [var x:int] [ SET x (add x 1) ];
  VAR n int;
  SET n 41;
  CALL inc (adr n);
  ECHO n
]
```

- est typable
- s'évalue et produit la sortie (42)

Typage de `PROC inc [var x:int] [SET x (add x 1)]`

$\Gamma_0 \vdash_{\text{DEC}}$ `PROC inc [var x:int] [SET x (add x 1)]`
$: \Gamma_0[\texttt{inc} : (\texttt{ref}\ \textit{int})\texttt{->}\textit{void}]$

Par (PROC)

- $\Gamma_0[\texttt{x} : (\texttt{ref}\ \textit{int})] \vdash_{\text{STAT}}$ `(SET x (add x 1))` $: \textit{void}$

  Par (SET)

  - $\Gamma_0[\texttt{x} : (\texttt{ref}\ \textit{int})](\texttt{x}) = (\texttt{ref}\ \textit{int})\bullet$
  - $\Gamma_0[\texttt{x} : (\texttt{ref}\ \textit{int})] \vdash_{\text{EXPR}}$ `(add x 1)` $: \textit{int}$

    Par (APP)

    - $\Gamma_0[\texttt{x} : (\texttt{ref}\ \textit{int})](\texttt{add}) = \Gamma_0(\texttt{add}) = \textit{int}*\textit{int}\texttt{->}\textit{int}\bullet$
    - $\Gamma_0[\texttt{x} : (\texttt{ref}\ \textit{int})] \vdash_{\text{EXPR}}$ `x` $: \textit{int}$

      Par (IDR)

      - $\Gamma_0[\texttt{x} : (\texttt{ref}\ \textit{int})](\texttt{x}) = (\texttt{ref}\ \textit{int})\bullet$

    - $\Gamma_0[\texttt{x} : (\texttt{ref}\ \textit{int})] \vdash_{\text{EXPR}}$ `1` $: \textit{int}$ par $(\text{NUM})\bullet$

Posons $\Gamma_1 = \Gamma_0[\texttt{inc} : (\texttt{ref}\ \textit{int})\ \texttt{->}\ \textit{void}]$

Typage de `VAR n int; SET n 41; CALL inc (adr n); ECHO n`

$\Gamma_1 \vdash_{\text{DEC}}$ `VAR n int` : Γ_1[`n` : (`ref` *int*)] par (VAR)•

Soit $\Gamma_2 = \Gamma_1$[`n` : (`ref` *int*)]

$\Gamma_2 \vdash_{\text{STAT}}$ `SET n 41` : *void*

Par (SET)

- Γ_2(`n`) = (`ref` *int*)
- $\Gamma_2 \vdash_{\text{EXPR}}$ `41` : *int* par (NUM)•

$\Gamma_2 \vdash_{\text{STAT}}$ `CALL inc (adr n)` : *void*

Par (CALL)

- Γ_2(`inc`) = (`ref` *int*) `->` *void*•
- $\Gamma_2 \vdash_{\text{EXPRP}}$ `(adr n)` : (`ref` *int*)

  Par (REF) car Γ_2(`n`) = (`ref` *int*)•

$\Gamma_2 \vdash_{\text{STAT}}$ `ECHO n` : *void*

Par (ECHO)

- $\Gamma_2 \vdash_{\text{EXPR}}$ `n` : *int*

Par (IDR) car Γ_2(`x`) = (`ref` *int*)•

Évaluation de `PROC inc ...; VAR n int; SET n 41;`

$\emptyset, \emptyset \vdash_{\text{DEC}} (\texttt{PROC inc [var x:int][SET x (add x 1)]})$
$\rightsquigarrow (\rho_1, \emptyset)$
avec $\rho_1 = [\texttt{inc} = \mathit{inP}([\texttt{SET ...}], \lambda v.[\texttt{x} = v])$

$\rho_1, \emptyset \vdash_{\text{DEC}} (\texttt{VAR n int}) \rightsquigarrow (\rho_2, \sigma_1)$
avec $\rho_2 = \rho_1[\texttt{n} = \mathit{inA}(a_1)]$ et $\sigma_1 = [a_1 = \mathit{any}]$

$\rho_2, \sigma_1, \emptyset \vdash_{\text{STAT}} (\texttt{SET n 41}) \rightsquigarrow (\sigma_2, \emptyset)$
Par (SET)
- $\rho_2(\texttt{n}) = \mathit{inA}(a_1)\bullet$
- $\rho_2, \sigma_1 \vdash_{\text{EXPR}} 41 \rightsquigarrow \mathit{inZ}(41)$ par $(\text{NUM})\bullet$
- $\sigma_2 = \sigma_1[a_1 := \mathit{inZ}(41)] = [a_1 = \mathit{any}][a_1 := \mathit{inZ}(41)]$
  $= [a_1 = \mathit{inZ}(41)]\bullet$

Évaluation de `CALL inc (adr n)`

$\rho_2, \sigma_2, \emptyset \vdash_{\text{STAT}} (\texttt{CALL inc (adr n)}) \rightsquigarrow (\sigma_3, \emptyset)$

Par (CALL)

- $\rho_2(\texttt{inc}) = inP([\texttt{SET x (add x 1)}], \lambda v.[\texttt{x} = v])\bullet$
- $\rho_2, \sigma_2 \vdash_{\text{EXPRP}} \texttt{(adr n)} \rightsquigarrow inA(a_1)$ par $(\text{REF})\bullet$
- $\rho = \lambda v.[\texttt{x} = v](inA(a_1)) = [\texttt{x} = inA(a_1)]$

  $\rho, \sigma_2, \emptyset \vdash_{\text{STAT}} \texttt{SET x (add x 1)} \rightsquigarrow (\sigma_3, \emptyset)$

  Par (SET)

  - $\rho(\texttt{x}) = inA(a_1)\bullet$
  - $\rho, \sigma_2 \vdash_{\text{EXPR}} \texttt{(add x 1)} \rightsquigarrow inZ(42)$

    Par (PRIM)

    - $\rho, \sigma_2 \vdash_{\text{EXPR}} \texttt{x} \rightsquigarrow inZ(41)$

      Par (ID1) car $\rho(\texttt{x}) = inA(a_1)$ et $\sigma_2(a_1) = inZ(41)\bullet$
    - $\rho, \sigma_2 \vdash_{\text{EXPR}} 1 \rightsquigarrow inZ(1)$ par $(\text{NUM})\bullet$
    - $\pi(\text{add})(41, 1) = 41 + 1 = 42\bullet$
  - $\sigma_3 = \sigma_2[a_1 := inZ(42)] = [a_1 = inZ(41)][a_1 := inZ(42)]$

    $= [a_1 = inZ(42)]\bullet$

Évaluation de `ECHO n`

$\rho_2, \sigma_3, \emptyset \vdash_{\text{STAT}} \texttt{ECHO n} \rightsquigarrow (\sigma_3, (42 \cdot \emptyset))$

Par (ECHO)

- $\rho_2, \sigma_3 \vdash_{\text{EXPR}} \texttt{n} \rightsquigarrow \mathit{inZ}(42)$

  Par (ID1)

  - $\rho_2(\texttt{n}) = \mathit{inA}(a_1)$ et $\sigma_3(a_1) = \mathit{inZ}(42)$•

Typage + évaluation

Notez que le programme

```
[
  PROC inc [var x:int] [ SET x (add x 1) ];
  CONST n int 41;
  CALL inc n;
  ECHO n
]
```

n'est pas typable et provoque une erreur à l'évaluation.

De même pour

```
[
  PROC inc [var x:int] [ SET x (add x 1) ];
  CONST n int 41;
  CALL inc (adr n);
  ECHO n
]
```