

# APS - Notes de Cours

Jordi Bertran de Balanda

## Cours 1

### Introduction

#### Programme

Description statique (fichier) d'un comportement dynamique (exécution). Statique: exécutable/bytecode/assembleur/code source Code source: suite de caractères obéissant à des règles de syntaxe.

Syntaxe:

- Composée de:
  - Lexique: ensemble d'unités lexicales (mots/symboles)
  - Grammaire: règles d'agencement des mots
- définit les suites de caractères qui sont des programmes
- donne un procédé de construction d'un mécanisme de reconnaissance des programmes (automates, langages formels)

#### Sémantique

- opérationnelle (grand pas/petit pas)
- dénotationnelle (lambda-calcul)
- axiomatique (cf. logique de Hoare en CPS)

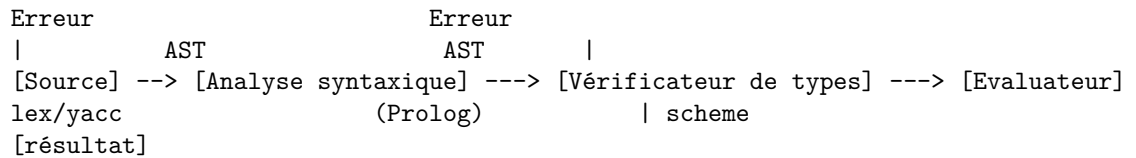
Comportement dynamique: résulte du traitement de données statiques (via CPU/interp bytecode/interp code source)

Règles/mécanismes d'interprétation = fonction d'interprétation

## Analyse

Dirigée par la syntaxe.

- Analyse syntaxique: éliminer les sources non évaluables
- Analyse de type: éliminer les erreurs prévisibles à l'exécution.



## Noyau impératif

- Expressions: bool, int, void
- Instructions: affectation, alternative, boucle, séquence, déclaration = Blocs

## Syntaxe

**Lexique** Mots clés (réservés): Key | Bool | Int ———|——-|—— VAR | true |  
add CONST | false | sub SET | and | mul IF | or | div WHILE | |

Symboles réservés:

[ ] ( ) ;

Description du lexique des noms de variables et des entiers: expressions régulières.  
En lex:

num : [0-9][0-9]\*  
ident : [a-z][a-z]\*

**Grammaire** Formalisation avec la Backus-Naur Form (BNF)

```
Prog ::= '[' Ccmds ']'
Ccmds ::= Cmd
        | Cmd ';' Ccmds
Cmd ::= Dec #(declaration)
        | Stat #(statement)
Dec ::= 'CONST' ident Exp
        | 'VAR' ident TypeExp
```

```

TypeExp ::= 'bool' | 'int'
Stat ::= 'SET' ident Exp
      | 'IF' Exp Prog Prog
      | 'WHILE' Exp Prog
UnOp ::= '(' 'not' Exp ')'
BinOp ::= '(' 'and' Exp Exp ')'
      | '(' 'or' Exp Exp ')'
      | '(' 'add' Exp Exp ')'
      | '(' 'sub' Exp Exp ')'
      | '(' 'mul' Exp Exp ')'
      | '(' 'div' Exp Exp ')'
Exp ::= 'true'
     | 'false'
     | num
     | ident
     | UnOp
     | BinOp

```

## Cours 2

### Typage

Définition: assigner un type (de données)

- aux expressions
- aux instructions
- aux programmes

### Contexte de typage

Un ensemble d'assignations de types. *Noté*  $\Gamma$ .

$\Gamma \vdash x : t$  si  $(x : t) \in \Gamma \forall$  type  $t$

Dans le cadre du langage réalisé en APS:

### Expressions

```

true  : bool
false : bool
n      : int  si n $\in$ num
x      : ?    si x $\in$ ident
(not e) : bool si e : bool

```

$(\text{and } e1 \ e2) : \text{bool}$  si  $e1 : \text{bool}$  et  $e2 : \text{bool}$   
 $(\text{or } e1 \ e2) : \text{bool}$  si  $e1 : \text{bool}$  et  $e2 : \text{bool}$

$\text{si } e1 : \text{int} \text{ et } e2 : \text{int}$   
 $(\text{add } e1 \ e2) : \text{int}$   
 $(\text{sub } e1 \ e2) : \text{int}$   
 $(\text{mul } e1 \ e2) : \text{int}$   
 $(\text{div } e1 \ e2) : \text{int}$

$\text{si } e1 : t \text{ et } e2 : t \text{ pour tout type } t$   
 $(\text{eq } e1 \ e2) : \text{bool}$   
 $(\text{lt } e1 \ e2) : \text{bool}$

**Instructions** *Relation* entre:

- Un contexte de typage
- Une expression
- Un type

$\Gamma \vdash (\text{SET } x \ e) : \text{void}$  si  $\Gamma \vdash x : t$  et  $\Gamma \vdash e : t \ \forall \text{ type } t$   
 $\Gamma \vdash (\text{IF } e \ b1 \ b2) : \text{void}$  si  $\Gamma \vdash e : \text{bool}$  et  $\Gamma \vdash b1 : \text{void}$  et  $\Gamma \vdash b2 : \text{void}$   
 $\Gamma \vdash (\text{WHILE } e \ b) : \text{void}$  si  $\Gamma \vdash e : \text{bool}$  et  $b : \text{void}$

**Suite de commandes** *Relation* entre:

- Un contexte de typage
- Un bloc (programme)
- void

Deux cas à considérer:

- [declaration; command sequence]
- [statement; command sequence]

$\Gamma \vdash [] : \text{void}$   
 $\Gamma \vdash (s::cs) : \text{void}$  si  $s : \text{void}$  et  $\Gamma \vdash cs : \text{void} \ \forall s \in \text{statement}$   
 $\Gamma \vdash ((\text{CONST } x \ e)::cs) : \text{void}$  si  $\Gamma; x:t \vdash cs : \text{void}$  et  $\Gamma \vdash e : t$   
 $\Gamma \vdash ((\text{VAR } x \ t)::cs) : \text{void}$  si  $\Gamma; x:t \vdash cs : \text{void}$   
 Notation ci-dessus:  $\Gamma; x:t$  abrège  $\Gamma \cup \{x:t\}$

**Programmes/blocs** Bloc: suite de déclarations et d'instructions

$\vdash [cs] : \text{void}$  si  $\vdash cs : \text{void}$

Notation d'une règle d'inférence:

$$\frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash b : \text{void}}{\Gamma \vdash (\text{WHILE } e \text{ b}) : \text{void}}$$

$$\frac{\Gamma; x:t \quad \vdash cs : \text{void}}{\Gamma \vdash ((\text{VAR } x \text{ t}) :: cs) : \text{void}}$$

$$\frac{x:\text{bool} \vdash \text{true} : \text{bool} \quad \frac{x:\text{bool} \vdash x : \text{bool}}{x:\text{bool} \vdash (\text{not } x) : \text{bool}}}{\Gamma \vdash (\text{and true (not x)}) : \text{bool}}$$

## Simplification

On peut enlever des règles de typage en les ajoutant à la description du programme.

Par exemple:

$\text{true} : \text{bool}; \text{false} : \text{bool} \vdash [cs] : \text{void}$

On peut ainsi enlever toutes les opérations dont les paramètres sont identiques à la valeur de retour.

$$\frac{\Gamma \vdash op : t_1 * t_2 \rightarrow t \quad \Gamma \vdash e_1 : t_1 \quad \Gamma \vdash e_2 : t_2}{\Gamma \vdash (op \ e_1 \ e_2) : t}$$

En généralisant:

$$\frac{\Gamma \vdash op : t_1 * \dots * t_n \rightarrow t \quad \Gamma \vdash e_1 : t_1 \dots \Gamma \vdash e_n : t_n}{\Gamma \vdash (op \ e_1 \ e_2 \dots e_n) : t}$$

## Ajout des types dans la grammaire

```
type ::= 'bool'
      | 'int'
      | type '*' .. '*' type -> type
      | vartype
```

## Cours 2

### Sémantique opérationnelle

- *Expressions*: fonctionnel  $\sim \rightarrow$  valeurs
- *Instructions*: impératif  $\sim \rightarrow$  état mémoire
- *Déclaration*:  $\sim \rightarrow$  état mémoire, environnement

## Environnement

Association entre identificateurs et \* valeurs \* adresses mémoire

Notation:

#v - valeurs

@a - adresses

Soit r un environnement.

- Accès, avec  $x : \text{Id}$ 
  - $r(x)$
  - $[ ](x) = \text{error}$
- Ajout, avec w une valeur:
  - $r[x = w]$
  - $r[x = w](x) = w$
  - $r[x = w](y) = r(y)$  avec  $x \neq y$

## Mémoire

Associations entre adresses et

- valeurs
- autres adresses

Avec m une mémoire:

- Accès:
  - $m(a)$
  - $[ ](a) = \text{error}$
- Extension (allocation):
  - $m[a := \text{new}] - m(a)$  est non défini
- Modification:  $m[a := w]$ 
  - $m[a := w][a := w'] = m[a := w']$
  - $m[a' := w][a := w'] = m[a := w'][a' := w]$  avec  $a \neq a'$
  - $[ ][a := w] = \text{error}$
- $(m/r)(a) = m(a)$  si il existe x tq  $r(x) = @a$
- error sinon

## Jugement sémantique

- *Expression*:  $r, m \vdash e \rightsquigarrow v$
- *Instructions*:  $r, m \vdash s \rightsquigarrow m'$
- *Déclarations*:  $r, m \vdash d \rightsquigarrow r', m'$
- *Suite de commandes/Bloc*:  $r, m \vdash [cs] \rightsquigarrow m'$

## Constantes

- $r, m \vdash \text{true} \rightsquigarrow \#t$
- $r, m \vdash \text{false} \rightsquigarrow \#f$
- $r, m \vdash n \rightsquigarrow \#n$  ( $n : \text{num}$ )

## Opérateurs booléens

- $r, m \vdash (\text{not } e) \rightsquigarrow \#f$  si  $r, m \vdash e \rightsquigarrow \#t$
- $r, m \vdash (\text{not } e) \rightsquigarrow \#t$  si  $r, m \vdash e \rightsquigarrow \#f$
- $r, m \vdash (\text{or } e1 \ e2) \rightsquigarrow \#t$  si  $r, m \vdash e1 \rightsquigarrow \#t$
- $r, m \vdash (\text{or } e1 \ e2) \rightsquigarrow \#t$  si  $r, m \vdash e1 \rightsquigarrow \#f$  et  $r, m \vdash e2 \rightsquigarrow \#t$
- $r, m \vdash (\text{or } e1 \ e2) \rightsquigarrow \#f$  si  $r, m \vdash e1 \rightsquigarrow \#f$  et  $r, m \vdash e2 \rightsquigarrow \#f$
- $r, m \vdash (\text{and } e1 \ e2) \rightsquigarrow \#f$  si  $r, m \vdash e1 \rightsquigarrow \#f$
- $r, m \vdash (\text{or } e1 \ e2) \rightsquigarrow \#f$  si  $r, m \vdash e1 \rightsquigarrow \#t$  et  $r, m \vdash e2 \rightsquigarrow \#f$
- $r, m \vdash (\text{or } e1 \ e2) \rightsquigarrow \#t$  si  $r, m \vdash e1 \rightsquigarrow \#t$  et  $r, m \vdash e2 \rightsquigarrow \#t$

## Opérateurs arithmétiques

- $r, m \vdash (\text{add } e1 \ e2) \rightsquigarrow v1 + v2$  si  $r, m \vdash e1 \rightsquigarrow v1$  et  $r, m \vdash e2 \rightsquigarrow v2$
- $r, m \vdash (\text{sub } e1 \ e2) \rightsquigarrow v1 - v2$  si  $r, m \vdash e1 \rightsquigarrow v1$  et  $r, m \vdash e2 \rightsquigarrow v2$
- $r, m \vdash (\text{mul } e1 \ e2) \rightsquigarrow v1 * v2$  si  $r, m \vdash e1 \rightsquigarrow v1$  et  $r, m \vdash e2 \rightsquigarrow v2$
- $r, m \vdash (\text{div } e1 \ e2) \rightsquigarrow v1 / v2$  si  $r, m \vdash e1 \rightsquigarrow v1$  et  $r, m \vdash e2 \rightsquigarrow v2$  et  $v2 \neq 0$

## Instructions

- $r, m \vdash (\text{SET } x \ e) \rightsquigarrow m[a := v]$  si  $r, m \vdash e \rightsquigarrow v$  et  $r(x) = @a$
- $r, m \vdash (\text{IF } b \ e1 \ e2) \rightsquigarrow m'$  si  $r, m \vdash b \rightsquigarrow \#t$  et  $r, m \vdash e1 \rightsquigarrow m'$
- $r, m \vdash (\text{IF } b \ e1 \ e2) \rightsquigarrow m'$  si  $r, m \vdash b \rightsquigarrow \#f$  et  $r, m \vdash e2 \rightsquigarrow m'$
- $r, m \vdash (\text{WHILE } e \ b) \rightsquigarrow m$  si  $r, m \vdash e \rightsquigarrow \#f$
- $r, m \vdash (\text{WHILE } e \ b) \rightsquigarrow m' \text{'si } r, m \vdash e \rightsquigarrow \#t \text{ et } r, m \vdash b \rightsquigarrow m' \text{ et } r, m' \vdash (\text{WHILE } e \ b) \rightsquigarrow m'$

### Suite de commandes

- Si  $s$  une instruction:  $r, m \vdash s; cs \sim> r, m' \vdash s \sim> m'$
- Si  $d \equiv \text{CONST } x \ e$ :  $r, m \vdash d; cs \sim> r[x := \#v], m$  si  $r, m \vdash e \sim> \#v$
- Si  $d \equiv \text{VAR } x$ :  $r, m \vdash d; cs \sim> r[x := @a], m[a = \text{new}]$

### Bloc

- $r, m \vdash [cs] \sim> r, (m'/r)$  si  $r, m \vdash cs \sim> r', m'$

### Typage

Distinguer les identificateurs modifiables des immuables.

Introduction du type “**pointeur vers**”: **@t** “pointeur vers  $t$ ”

- Déclaration/suite de commandes
  - $G \vdash (\text{VAR } c \ t; cs) : \text{void}$  si  $G, x:@t \vdash cs : \text{void}$
- Expression
  - $G, x:@t \vdash x:t \parallel G \vdash x:t$  si  $G(x) = t$  ou  $G(@x) = t$
- Affectation
  - $G \vdash (\text{SET } x \ e) : \text{void}$  si  $G \vdash x:@t$  et  $G \vdash e:t$

### Sémantique

- $r, m \vdash (\text{SET } x \ e) \sim> m[r(x) := v]$  si  $r, m \vdash e \sim> v$

## Cours 4

### Retour sur la portée lexicale

**Blocs** Sémantique: si  $r, m \vdash cs \sim> m'$  alors  $r, m \vdash [cs] \sim> m'/r$

```
[
  VAR x int;
  SET x 0;
  IF (true)
    [ VAR x int;
      SET x 12 ]
    [ SET x 1 ]
  SET x (x + 1)
]
```



À la fin de l'exécution du programme,  $x$  vaut 1. Si on retire  $/r$  de la règle de sémantique du bloc, on peut avoir le programme suivant où  $x$  vaut 4:

```
[
  VAR x int;
  IF (true)
    [ SET x 3 ]
    [ ... ]
  SET x (x + 1)
]
```

## Fonctions et procédures

- Fonctions: abstractions vis-à-vis d'une expression
- Procédures: abstractions vis-à-vis d'un bloc d'expressions

### Fonction

- Domaine de départ, codomaine d'arrivée.
- Abstraction d'une expression avec ses variables associées
- En APS:  $[x:\text{int}] (x + 1) : \text{expression fonctionnelle}$ .

### Syntaxe

```
Expr ::= ..
      | '[' TypeIds ']' Expr
      | '(' Exprs ')'

Exprs ::= Expr
       | Expr Exprs

TypeIds ::= ident ':' type
         | ident ':' type ';' TypeIds
```

### Exemple

```
CONST xor [b1:bool;b2:bool]
          (and (or b1 b2)
              (not (and b1 b2))); ..
(xor true false)
```

## Typage

- **Si**  $G; x_1 : t_1; x_2 : t_2; \dots; x_n : t_n \vdash e : t$  **alors**  $G \vdash [x_1 : t_1; \dots; x_n : t_n]e : (t_1 * \dots * t_n) \rightarrow t$
- **Si**  $G \vdash e : (t_1 * \dots * t_n) \rightarrow t$  et  $G \vdash e_1 : t_1 \dots G \vdash e_n : t_n$  **alors**  $G \vdash (e \ e_1 \dots e_n) : t$

Type ::= ..  
| Typeargs '->' Type  
| '(' Type ')'

Typeargs ::= Type  
| Type '\*' Typeargs

## Sémantique

Ajouter les définitions des expressions fonctionnelles, des valeurs fonctionnelles ( $\Rightarrow$  fermetures).

$(([x_1 : t_1; \dots; x_n : t_n]e') \ e_1 \dots e_n)$

Fermeture: (notation)  $\langle e, r + [x_1 = \#] \dots [x_n = \#] \rangle$

**Sémantique de la fermeture:**  $r, m \vdash [x_1 : t_1; \dots; x_n : t_n]e \sim \langle e, r + [x_1 = \#] \dots [x_n = \#] \rangle$

## Application - Liaison statique

- **Si**  $r, m \vdash e \sim \langle e', r'' + [x_1 = \#] \dots [x_n = \#] \rangle$
- **et**  $r, m \vdash e_1 \sim v_1 \dots r, m \vdash e_n \sim v_n$
- **et**  $r'[x_1 = v_1] \dots [x_n = v_n], m \vdash e' \sim v$
- **alors**  $r, m \vdash (e \ e_1 \dots e_n) \sim v$

```
[ CONST f [x:int] (add x 1)
  CONST g [x:int] (add (f x) 1)
  CONST f [x:int] (add x 3)
  VAR r int
  SET r (g 42)
]
```

Liaison statique: la valeur de g est fabriquée au moment de sa déclaration, g capture la valeur de f à l'instant de la déclaration de g. La valeur de r est donc 44.

## Procédures

Avec  $d = \text{PROC } p [x_1 : t_1; \dots; x_n : t_n] \text{ blk}$

### Syntaxe

$\text{Dec} ::= \dots$   
| 'PROC' ident '[' TypeIds ']' Prog

$\text{Stat} ::= \dots$   
| 'CALL' ident Exprs

**Typage** Si  $G; p:t_1 * \dots * t_n \rightarrow \text{void} \vdash \text{cs} : \text{void}$  et  $G \vdash \text{blk} : \text{void}$  **alors**  $G \vdash d; \text{cs} : \text{void}$

**Appel** Si  $G \vdash p:t_1 * \dots * t_n \rightarrow \text{void}$  **et**  $G \vdash e_1 : t_1 \dots G \vdash e_n : t_n$  **alors**  $G \vdash \text{CALL } p \ e_1..e_n : \text{void}$

### Sémantique

Fermeture “procédurale”:  $\langle \text{blk}, r + [x_1=\#]..[x_n=\#] \rangle$

**Si**  $r [p = \langle \text{blk}, r + [x_1=\#]..[x_n=\#] \rangle, m \vdash \text{cs} \leadsto (r', m') \text{ alors } r, m \vdash d; \text{cs} \leadsto (r', m')$

### Application - Liaison statique

- **Si**  $r(p) = \langle \text{blk}, r + [x_1=\#]..[x_n=\#] \rangle$
- **et**  $r, m \vdash e_1 \leadsto v_1 \dots r, m \vdash e_n \leadsto v_n$
- **et**  $r'[x_1 = v_1..x_n = v_n], m \vdash \text{blk} \leadsto m'$
- **alors**  $r, m \vdash (\text{CALL } p e_1..e_n) \leadsto m'$

## Cours 5

### L’instruction RETURN

$\text{stat} ::= \dots$   
| RETURN Expr

## Typage

- Expression:
  - Si  $G \vdash e:t$  alors  $G \vdash (\text{RETURN } e):t$
- Séquences: soit  $s != (\text{RETURN } e)$ 
  - Si  $G \vdash \text{void}$  et  $G \vdash cs:t$  alors  $G \vdash (s;cs):t$
  - Si  $G \vdash s:t$  avec  $t != \text{void}$  et  $G \vdash cs:t$  alors  $G \vdash (s; cs):t$
- Si  $G \vdash e:t$  et  $G \vdash x:t$  alors  $G \vdash (\text{SET } x e):\text{void}$
- Si  $G \vdash e:\text{bool}$  et  $G \vdash cs:t$  alors  $G \vdash (\text{WHILE } e cs):t$
- Si  $G \vdash e:\text{bool}$  et  $G \vdash cs1:t$  et  $G \vdash cs2:t$  alors  $G \vdash (\text{IF } e cs1 cs2):t$

## Sémantique

Instruction:

- une non valeur
- une valeur  $v$

On note  $\omega$  pour  $v$  ou  $\emptyset$ .

Relation sémantique des instructions:

- Env, Mem, Stat  $\rightarrow$  (Val, Mem)
- $r, m \vdash s \sim> (\omega, m')$

**RETURN** Si  $r, m \vdash e \sim> v$  alors  $r, m \vdash (\text{RETURN } e) \sim> (v, m)$

**Affectation** Si  $r, m \vdash (\text{SET } x e) \sim> (\emptyset, m[r(x):=v])$

## Boucle

- Si  $r, m \vdash e \sim> \#f$  alors  $r, m \vdash (\text{WHILE } e \text{ blk}) \sim> (\emptyset, m)$
- Si  $r, m \vdash e \sim> \#t$  alors
  - Si  $r, m \vdash \text{blk} \sim> (\emptyset, m')$  et  $r, m' \vdash (\text{WHILE } e \text{ blk}) \sim> (\omega, m'')$  alors  $r, m \vdash (\text{WHILE } e \text{ blk}) \sim> (\omega, m'')$
  - Si  $r, m \vdash \text{blk} \sim> (v, m')$  alors  $r, m \vdash (\text{WHILE } e \text{ blk}) \sim> (v, m')$

## Alternative

- Si  $r, m \vdash e \sim> \#t$  et  $r, m \vdash \text{blk1} \sim> (\omega, m')$  alors  $r, m \vdash (\text{IF } e \text{ blk1 blk2}) \sim> (\omega, m')$
- Si  $r, m \vdash e \sim> \#f$  et  $r, m \vdash \text{blk2} \sim> (\omega, m')$  alors  $r, m \vdash (\text{IF } e \text{ blk1 blk2}) \sim> (\omega, m')$

**Suites de commandes** Pour toute instruction  $s$  et toute continuation  $cs$ :  
\* Si  $r, m \vdash s \sim> (\emptyset, m')$  et  $r, m' \vdash cs \sim> (\omega, m'')$  alors  $r, m \vdash (s;cs) \sim> (\omega, m'')$   
\* Si  $r, m \vdash s \sim> (v, m')$  alors  $r, m \vdash (s;cs) \sim> (v, m')$

## Déclarations

- Si  $d \equiv \text{CONST } x \ e$ 
  - Si  $r, m \vdash e \sim> v$  et  $r[x=v], m \vdash cs \sim> (\omega, m')$  alors  $r, m \vdash (d;cs) \sim> (\omega, m')$
- Si  $d \equiv \text{VAR } x \ t$ 
  - Si  $r[x=a], m[x=\text{new}] \vdash cs \sim> (\omega, m')$  alors  $r, m \vdash (d;cs) \sim> (\omega, m')$

## Fonctions procédurales

### Syntaxe

Dec ::= ..  
| 'FUN' ident Type '[' TypeIds ']' Prog  
| 'FUN' ident Type '[' ']' Prog

### Typage

- Expression/application:
  - Si  $G \vdash f:\text{void} \rightarrow t$  alors  $G \vdash (f):t$
- Déclaration/suite de commandes:
  - $d \equiv \text{FUN } f \ t \ [] \ \text{blk}$ 
    - \* Si  $G[f:\text{void} \rightarrow t] \vdash cs:t'$  et  $G \vdash \text{blk}:t$  alors  $G \vdash (d;cs):t'$
  - $d \equiv \text{FUN } f \ t \ [x_1 : t_1; .. x_n : t_n] \ \text{blk}$ 
    - \* Si  $G[f:t_1 * .. * t_n] \vdash cs:t'$  et  $G[x_1 : t_1; ..; x_n : t_n] \vdash \text{blk}:t$  alors  $G \vdash (d;cs):t'$

## Sémantique

Relation sémantique des expressions:  $r, m \vdash e \sim> (v, m')$

Valeur d'une fonction procédurale:  $\langle \text{bd}, r+[x1=\#] \rangle$

## Application

- Si
  - $r, m \vdash e \sim> (\langle r'+[x1=\#..xn=\#] \rangle, m')$
  - $r, m' \vdash e1 \sim> (v1, m1)$  et .. et  $r, mn-1 \vdash en \sim> (vn, mn)$
  - $r'[x1=v1]..[xn=vn], mn \vdash \text{bd} \sim> m''$
- alors  $r, m \vdash (e \text{ e1 } .. \text{ en}) \sim> (v, m'')$

## Fonctions récursives

### Syntaxe

```
Dec ::= ..  
      | 'FUN' 'REC' ident Type '[' TypeIds ']' Prog  
      | 'FUN' 'REC' ident Type '[' ']' Prog
```

### Sémantique

Valeur d'une fonction récursive:  $\langle \text{bd}, r+[f=!][x1=\#]..[xn=\#] \rangle$

## Application

- Si
  - $r, m \vdash e \sim> (\langle r'+[f=!][x1=\#]..[xn=\#] \rangle, m')$
  - $r, m' \vdash e1 \sim> (v1, m1)$  et .. et  $r, mn-1 \vdash en \sim> (vn, mn)$
  - $r'[f=\langle \text{bd}, r+[f=!][x1=\#]..[xn=\#] \rangle][x1=v1]..[xn=vn], mn \vdash \text{bd} \sim> m''$
- alors  $r, m \vdash (e \text{ e1 } .. \text{ en}) \sim> (v, m'')$