

# Sémantique opérationnelle de SCOL, Sémantique dynamique

Anne-Gwenn Bosser

Mars 2002

## Table des matières

|          |                                   |          |
|----------|-----------------------------------|----------|
| <b>1</b> | <b>Introduction</b>               | <b>1</b> |
| <b>2</b> | <b>Syntaxe abstraite de SCOL4</b> | <b>1</b> |
| 2.1      | Définitions . . . . .             | 2        |
| 2.2      | Expressions . . . . .             | 2        |
| <b>3</b> | <b>Sémantique</b>                 | <b>4</b> |
| 3.1      | Formalisme . . . . .              | 4        |
| 3.2      | Définitions . . . . .             | 6        |
| 3.3      | Expressions . . . . .             | 7        |

## 1 Introduction

Ce document présente la sémantique dynamique de SCOL4, utilisant comme formalisme SOS (Structural Operational Semantic), pour le noyau du langage.

Il complète le document relatif à la sémantique statique de SCOL4 et la grammaire du langage.

La section **Syntaxe abstraite de SCOL4** présente l'abstraction du noyau du langage SCOL4, à partir de laquelle on décrira les règles sémantiques dans la section **Sémantique**

## 2 Syntaxe abstraite de SCOL4

Les mots-clés du langage apparaissent en souligné.

Ne font pas partie de cette grammaire abstraite:

- les expressions de type: elles sont résolues au typage.
- les valeurs du langage: on ne décrira pas le système de règles correspondant à leur évaluation.
- les définitions et primitives de communication.

## 2.1 Définitions

**definition::=** variable-definition | function-definition

**variable-definition::=** var identifier  $\equiv$  val ;; | typeof identifier  $\equiv$  type-expression ;;

**function-definition::=** fun identifier  $\overrightarrow{(\text{identifier}_{i=0}^n)}$   $\equiv$  block ;;  
| proto identifier  $\equiv$  function-type-expression ;;

## 2.2 Expressions

**expression::=** logical-expression

**logical-expression::=** logical-expression && unary-logical-expression  
| logical-expression || unary-logical-expression | unary-logical-expression

**unary-logical-expression::=** ! unary-logical-expression | relational-expression

**relational-expression::=** relational-expression  $\square$  arithmetic-expression  
| relational-expression  $\equiv\equiv$  arithmetic-expression | relational-expression  $\underline{\equiv}$  arithmetic-expression  
| arithmetic-expression

$\square \in \{<, <=, >, >=, =, !=, ., <., <=., >., >=.\}$

**arithmetic-expression::=** arithmetic-expression  $\square$  multiplicative-expression | multiplicative-expression

$\square \in \{+, -, +., -.\}$

**multiplicative-expression::=** multiplicative-expression  $\square$  bitwise-expression | bitwise-expression

$\square \in \{*, /, *., /.\}$

**bitwise-expression::=** bitwise-expression  $\square$  unary-bitwise-expression  
| unary-bitwise-expression

$\square \in \{\&, |, ^, <<, >>\}$

**unary-bitwise-expression::=**  $\square$  unary-bitwise-expression | simple-expression

$\square \in \{-, \sim\}$

**simple-expression**::= literal | left-value | function-reference | function-application | tuple | { block ; }  
| ( block ; ) | { block } | ( block ) | assignment | conditional | let-expression | match-expression | iteration  
| exec-expression | mutate-expression

**literal**::= nil | integer-literal | floating-point-literal | string-literal

**left-value**::= identifier | left-value\_identifier | left-value\_expression

**tuple**::= [ expression-list ]

**function-application**::= identifier expression-list

**expression-list**::= expression-list expression | expression

**function-reference**::= @ identifier

**block**::= block ; list | list

**list**::= expression :: list | expression

**assignment**::= set identifier ≡ expression | set left-value\_identifier ≡ expression  
| set left-value\_expression ≡ expression

**conditional**::= if expression then expression else expression

**let-expression**::= let let-def in expression

**let-def**::= expression ⇒ let-pattern

**let-pattern**::= \_ | identifier | [ basic-pattern-list ] | nil

**match-expression**::= match expression with match-case-list

**match-case-list**::= match-case-list | match-case  
| match-case

**match-case**::= ( match-pattern ⇒ expression )

**match-pattern**::= \_ | uppercase-identifier | uppercase-identifier identifier  
| uppercase-identifier [ basic-pattern-list ]

**basic-pattern-list::=**  $\overrightarrow{\text{basic - pattern}}_{i=0}^n$

**basic-pattern::=**  $\_ \mid \text{identif ier} \mid \_ [ \text{basic-pattern-list} ]$

**iteration::=** while do-expression

**do-expression::=** expression do expression

**exec-expression::=** exec expression with expression

**mutate-expression::=** mutate expression  $\leftarrow$   $\_ [ \text{mutate-replacement-list} ]$

**mutate-replacement-list::=**  $\overrightarrow{\text{mutate - replacement - expression}}_{i=0}^n$

**mutate-replacement-expression::=**  $\_ \mid$  expression

**identif ier::=** un identif icateur de variable

**uppercase-identif ier::=** un identif icateur de variable commençant forcément par une majuscule.

### 3 Sémantique

#### 3.1 Formalisme

##### Environnements d'exécution, prédicat sémantique:

Afin que la description de la sémantique de SCOL4 soit la plus proche possible de l'exécution, on représente l'état de la machine SCOL via deux fonctions partielles  $H$  et  $S$ , représentant respectivement l'état mémoire et l'environnement d'exécution.

On utilisera le prédicat  $\mathbf{H}, \mathbf{S} \models \mathbf{e} \Downarrow (\mathbf{H}', \mathbf{S}', \mathbf{v})$  pour décrire l'évaluation de  $e$ , qui appartient à une des catégories syntaxiques. Pour les définitions, on utilisera le prédicat  $\mathbf{H}, \mathbf{S} \models \mathbf{e} \Downarrow (\mathbf{H}', \mathbf{S}')$

- $H$  représente le tas de la machine SCOL. C'est une liste (avec répétitions) d'associations  $\{ref = v\}$ , où  $ref$  est une valeur sémantique de type référence, et  $v$  la valeur sémantique référencée.
- $S$  représente l'environnement d'exécution de la machine SCOL. C'est une liste (avec répétitions) d'associations  $\{identif ier = v\}$  ou  $identif ier$  est un identif iant du langage SCOL et  $v$  la valeur sémantique associée.

- $v$  est la valeur sémantique de l'expression  $e$  lorsque la machine SCOL est dans l'état  $\{H,S\}$ . Cette valeur peut être une référence (vers une liste, un tuple, une structure, un tableau...) ou une valeur simple (entier, flottant, chaîne de caractères, ...).
- $H',S'$  est l'état résultant de l'évaluation de l'expression  $e$ , qui peut avoir des effets de bord sur l'état initial.

Le prédicat sémantique peut donc se lire: *Lorsque la machine SCOL se trouve dans l'état  $\{H,S\}$ , l'expression  $e$  s'évalue en un nombre d'étapes fini en la valeur  $v$ . L'état résultant de cette évaluation est le couple  $H',S'$ .*

### Fonction sémantique:

La sémantique de SCOL4 est décrite pour toutes les catégories syntaxiques définies dans la section **Syntaxe abstraite de SCOL4**. On considère qu'on dispose d'une fonction sémantique, notée  $\mathcal{V}$  au niveau atomique des valeurs du langage.

Cette fonction sémantique dispose également de tous les opérateurs arithmétiques élémentaires, et des opérateurs de comparaison, sur les flottants et les entiers.

### Valeurs sémantiques:

On distingue deux types de valeurs sémantiques: les valeurs références et celles qui n'en sont pas.

Les entiers, réels et chaînes de caractères sont utilisés pour dénoter les entiers, flottants et chaînes de caractères de SCOL. On dispose également de la valeur  $NIL$  dénotant le NIL du langage SCOL.

Les valeurs références sont des valeurs de l'environnement d'exécution qui ont une image dans le tas.

A ces deux types de valeurs, on ajoute les valeurs  $\top$  et  $\perp$  qui n'ont aucune représentation en SCOL mais qui servent à la définition des règles d'évaluation pour certaines catégories syntaxiques.

### Notation des valeurs références:

- **Tuples:**  $t_{ref}$ ,  $H(t_{ref}) = [\vec{v}_{i=0}^n]$  où les  $v_i$  sont les valeurs du tuple.
- **Structures:**  $s_{ref}$ ,  $H(s_{ref}) = [c_i = \vec{v}_{i=0}^n]$ , où les  $v_i$  sont les valeurs des champs  $c_i$  de la structure.
- **Tableaux:**  $tab_{ref}$ ,  $H(tab_{ref}) = T[\vec{v}_{i=0}^n]$ , où les  $v_i$  sont les valeurs d'indices  $i$  du tableau.
- **Fonctions:**  $f_{ref}$ ,  $H(f_{ref}) = \langle e, \vec{x}_{i=0}^n \rangle$ , où  $e$  est le corps de la fonction (son code), et  $x_i$  ses variables libres (ses paramètres).
- **Types sommes:**  $u_{ref}$ ,  $H(u_{ref}) = U_i v_i$  où les  $U_i$  pour  $0 \neq i \neq n$  sont les constructeurs du type.

Les valeurs  $v_i$  peuvent elles-même être des références et avoir une image dans le tas.

### Opérateurs, notations et abréviations:

- **abréviation:** chaque fois que le détail ne sera pas nécessaire, l'état  $H, S$  sera abrégé en  $E$ .  $E$  vue comme une application partielle sera donc  $H \circ S$ . Par abus, on notera parfois  $E :: \{identifier = value\}$ , comme si  $E$  était  $S$ , lorsque les règles s'appliquent aussi bien à des valeurs références qu'à des valeurs simples. Pour une valeur référence cette notation abrège  $H :: \{value = v\}, S :: \{identifier = value\}$ .
- $::$  est le constructeur associé l'environnement d'exécution. Par exemple, lorsqu'on veut ajouter à  $S$  un identifiant pourvu d'une valeur sémantique on écrira:  $S :: \{identifier = v\}$ .
- $[\leftarrow]$  est l'opérateur d'affectation associé à l'environnement d'exécution. Par exemple, lorsqu'on veut remplacer la valeur d'un identifiant de  $S$  par une autre on écrira  $S[identifier \leftarrow v]$ .
- Si  $H, S \models e \Downarrow (H', S', v)$ ,  $H', S'$  est la représentation des effets de bords sur l'état initial  $H, S$ . Cela signifie que  $S'$  est le résultat de toutes les opérations d'affectation ayant eu lieu sur  $S$  lors de l'évaluation de  $e$ , mais que  $S$  contient exactement les mêmes identifiants que  $S'$ . De même pour  $H'$ , pour qui de plus l'ensemble de définition peut avoir changé (il peut y avoir des références en plus): on ne traite pas dans cette sémantique les libérations de la mémoire qui ne s'effectuent qu'au GC. Tout comme entre l'exécution de deux GC, donc, les valeurs du tas qui ne sont plus référencées dans la pile d'exécution resteront dans le domaine de définition de  $H$ . On a donc toujours  $dom(S) = dom(S')$  et  $dom(H) \subseteq dom(H')$  ( $dom$  étant l'application qui renvoie le domaine de définition d'une application).

## 3.2 Définitions

### Variables

$$\frac{\mathcal{V}(value) = v}{E \models var\ ident = value; ; \Downarrow E :: \{ident = v\}} \quad (d_1)$$

Déclaration d'une variable par sa valeur. La fonction  $\mathcal{V}$  est la fonction sémantique donnant aux *valeurs* du langage SCOL une valeur sémantique. Cette règle s'applique aussi bien aux valeurs simples qu'aux valeurs références. Si  $v$  est une référence,  $v \in dom(H)$ .

$$\frac{}{E \models typeof\ ident = type-expression; ; \Downarrow E :: \{ident = NIL\}} \quad (d_2)$$

Déclaration d'une variable par son type.

### Fonctions

$$\frac{H(f_{ref}) = \langle \vec{x}_{i=0}^n, block \rangle \quad S(ident) = \perp}{H, S \models fun\ ident(\vec{x}_{i=0}^n) = block; ; \Downarrow (H[f_{ref} \leftarrow \langle \vec{x}_{i=0}^n, block \rangle], S)} \quad (d_3)$$

Définition d'une fonction précédemment prototypée.

$$\frac{S(ident) \neq \perp}{H, S \models fun\ ident(\vec{x}_{i=0}^n) = block; ; \Downarrow (H :: \{f_{ref} = \langle \vec{x}_{i=0}^n, block \rangle\} S :: \{ident = f_{ref}\})} \quad (d_4)$$

Cette règle s'applique dans deux cas:

- si l'identifiant appartient déjà à l'environnement, il a une image dans S. Cette définition lui en donne une nouvelle, qui sera utilisée dans les définitions suivantes mais l'ancienne ne disparaît pas pour autant.
- si l'identifiant n'appartient pas à l'environnement.

$$\overline{\overline{E \models \text{proto ident} = \text{function} - \text{type} - \text{expression}; ; \Downarrow E :: \{\text{ident} = \perp\}}} \quad (d_5)$$

Prototypage d'une fonction par son type.

### 3.3 Expressions

#### logical-expression

$$\frac{E \models e_1 \Downarrow (E', n_1) \quad E' \models e_2 \Downarrow (E'', n_2)}{E \models e_1 \&\& e_2 \Downarrow (E'', 1)} \quad (e_1) \quad n_1, n_2 \neq 0, NIL$$

$$\frac{E \models e_1 \Downarrow (E', 0)}{E \models e_1 \&\& e_2 \Downarrow (E', 0)} \quad (e_2)$$

$$\frac{E \models e_1 \Downarrow (E', n_1) \quad E' \models e_2 \Downarrow (E'', 0)}{E \models e_1 \&\& e_2 \Downarrow (E'', 0)} \quad (e_3) \quad n_1 \neq 0, NIL$$

$$\frac{E \models e_1 \Downarrow (E', n_1) \quad E' \models e_2 \Downarrow (E'', NIL)}{E \models e_1 \&\& e_2 \Downarrow (E'', NIL)} \quad (e_4) \quad n_1 \neq 0$$

$$\frac{E \models e_1 \Downarrow (E', NIL) \quad E' \models e_2 \Downarrow (E'', n_2)}{E \models e_1 \&\& e_2 \Downarrow (E'', NIL)} \quad (e_5)$$

Attention à cette dernière règle: même si  $n_2$  vaut 0, *if (NIL&& n2) then e1 else e3* exécutera le code de l'expression  $e_2$ .

$$\frac{E \models e_1 \Downarrow (E', n_1)}{E \models e_1 \parallel e_2 \Downarrow (E', n_1)} \quad (e_6) \quad n_1 \neq 0, NIL$$

Attention à cette dernière règle: elle renvoie la valeur  $n_1$  alors que les autres règles uniformisent un résultat *vrai* à 1.

$$\frac{E \models e_1 \Downarrow (E', 0) \quad E' \models e_2 \Downarrow (E'', n_2)}{E \models e_1 \parallel e_2 \Downarrow (E'', 1)} \quad (e_7) \quad n_2 \neq 0, NIL$$

$$\frac{E \models e_1 \Downarrow (E', 0) \quad E' \models e_2 \Downarrow (E'', 0)}{E \models e_1 \parallel e_2 \Downarrow (E'', 0)} (e_8)$$

$$\frac{E \models e_1 \Downarrow (E', NIL) \quad E' \models e_2 \Downarrow (E'', n_2)}{E \models e_1 \parallel e_2 \Downarrow (E'', NIL)} (e_9)$$

$$\frac{E \models e_1 \Downarrow (E', 0) \quad E' \models e_2 \Downarrow (E'', NIL)}{E \models e_1 \parallel e_2 \Downarrow (E'', NIL)} (e_{10})$$

### unary-logical-expression

$$\frac{E \models e \Downarrow (E', n)}{E \models !e \Downarrow (E', 0)} (e_{11}) \quad n \neq 0, NIL$$

$$\frac{E \models e \Downarrow (E', 0)}{E \models !e \Downarrow (E', 1)} (e_{12})$$

$$\frac{E \models e \Downarrow (E', NIL)}{E \models !e \Downarrow (E', NIL)} (e_{13})$$

Attention à cette dernière règle: utilisés dans une condition, *NIL* et *!NIL* sont tous les deux considérés comme *vrai*.

### relational-expression

$$\frac{E \models e_1 \Downarrow (E', v_1) \quad E' \models e_2 \Downarrow (E'', v_2)}{E \models e_1 == e_2 \Downarrow (E'', v_1 == v_2)} (e_{14})$$

$$\frac{E \models e_1 \Downarrow (E', v_1) \quad E' \models e_2 \Downarrow (E'', v_2)}{E \models e_1 ! = e_2 \Downarrow (E'', v_1 ! = v_2)} (e_{15})$$

$$\frac{E \models e_1 \Downarrow (E', v_1) \quad E' \models e_2 \Downarrow (E'', v_2)}{E \models e_1 \square e_2 \Downarrow (E'', v_1 \square v_2)} (e_{16}) \quad v_1, v_2 \neq NIL$$

$$\frac{E \models e_1 \Downarrow (E', NIL) \quad E' \models e_2 \Downarrow (E'', v_2)}{E \models e_1 \square e_2 \Downarrow (E'', NIL)} (e_{17})$$

$$\frac{E \models e_1 \Downarrow (E', v_1) \quad E' \models e_2 \Downarrow (E'', NIL)}{E \models e_1 \square e_2 \Downarrow (E'', NIL)} (e_{18})$$

On considère que la fonction sémantique contient tous les opérateurs relationnels nécessaires.

$\square \in \{<, <=, >, >=, =, !=, =., <., <=., >., >=.\}$

Attention : remarquer la différence de comportement entre  $==$ ,  $!=$ , et les autres opérateurs du point de vue de  $NIL$ . Les opérateurs  $\square$  sont des fonctions pour un certain type de données (entiers, flottants).  $==$  et  $!=$  sont les identités sur toutes les valeurs (simples ou références), y compris pour la valeur  $NIL$ .

### arithmetic-expression

Idem règles 16 17 18 avec  $\square \in \{*, /, *., /.\}$

### multiplicative-expression

Idem règles 16 17 18 avec  $\square \in \{+, -, +., -.\}$

### bitwise-expression

Idem règles 16 17 18 avec  $\square \in \{\&, |, \ll, \gg\}$

### unary-bitwise-expression

$$\frac{E \models e \Downarrow (E', n)}{E \models \square e \Downarrow (E', \square n)} \quad (e_{19}) \quad n \neq NIL$$

$$\frac{E \models e \Downarrow (E', NIL)}{E \models \square e \Downarrow (E', NIL)} \quad (e_{20})$$

$\square \in \{-, \sim\}$

### simple-expression

$$\frac{E \models block \Downarrow (E', v)}{E \models \{block\} \Downarrow (E', v)} \quad (e_{21})$$

$$\frac{E \models block \Downarrow (E', v)}{E \models (block;) \Downarrow (E', v)} \quad (e_{22})$$

$$\frac{E \models block \Downarrow (E', v)}{E \models \{block\} \Downarrow (E', v)} \quad (e_{23})$$

$$\frac{E \models \text{block} \Downarrow (E', v)}{E \models (\text{block}) \Downarrow (E', v)} \quad (e_{24})$$

### literal

$$\frac{}{E \models \text{nil} \Downarrow (E, \text{NIL})} \quad (e_{25})$$

$$\frac{\mathcal{V}(\text{value}) = v}{E \models \text{value} \Downarrow (E, v)} \quad (e_{26})$$

Pour tous les littéraux de SCOL, on considère que la fonction sémantique les interprète par la valeur sémantique correspondante.

### left-value

$$\frac{S(\text{identifïer}) = v}{H, S \models \text{identifïer} \Downarrow (H, S, v)} \quad (e_{27})$$

$$\frac{H, S \models \text{lval} \Downarrow (H', S', s_{ref}) \quad H'(s_{ref}) = [\overline{c_i} = \overrightarrow{v_{i=0}^n}]}{H, S \models \text{lval}.c_k \Downarrow (H', S', v_k)} \quad (e_{28}) \quad 0 \leq k \leq n$$

$$\frac{H, S \models \text{lval} \Downarrow (H', S', t_{ref}) \quad H', S' \models e \Downarrow (H'', S'', k) \quad H''(t_{ref}) = T[\overrightarrow{v'_{i=0}^n}]}{H, S \models \text{lval}.e \Downarrow (H'', S'', v'_k)} \quad (e_{29}) \quad 0 \leq k \leq n, H'(t_{ref}) = T[\overrightarrow{v_{i=0}^n}]$$

Attention: dans cette règle, il peut y avoir des effets de bord sur l'environnement lors de l'évaluation de  $e$ .

$$\frac{E \models \text{lval} \Downarrow (E', \text{NIL}) \quad E' \models e \Downarrow (E'', v)}{E \models \text{lval}.e \Downarrow (E'', \text{NIL})} \quad (e_{30})$$

$$\frac{E \models \text{lval} \Downarrow (E', v) \quad E' \models e \Downarrow (E'', \text{NIL})}{E \models \text{lval}.e \Downarrow (E'', \text{NIL})} \quad (e_{31})$$

Attention: ces deux dernières règles mettent en valeur le fait que si  $E \models \text{lval}.e \Downarrow (E', \text{NIL})$ , il n'y a aucun moyen de savoir laquelle des trois expressions  $\text{lval}$ ,  $e$  ou  $\text{lval}.e$  vaut  $\text{NIL}$

### tuple

$$\frac{}{(H, S) \models [] \Downarrow (H :: \{t_{ref} = [\emptyset]\}, S, t_{ref})} \quad (e_{32})$$

Création d'une nouvelle référence vers la valeur  $[]$ .

$$\frac{H, S \models \text{elist} \Downarrow (H', S', \vec{v}_{i=0}^n)}{H, S \models [\text{elist}] \Downarrow (H' :: \{t_{ref} = [\vec{v}_{i=0}^n]\}, S', t_{ref})} \quad (e_{33})$$

Création d'une nouvelle référence vers la valeur  $[\vec{v}_{i=0}^n]$ .

$$\frac{H, S \models e_1 \Downarrow (H', S', v_1) \quad H', S' \models e_2 \Downarrow (H'', S'', NIL)}{H, S \models [e_1 e_2] \Downarrow (H'' :: \{t_{ref} = [v_1 NIL]\}, S'', t_{ref})} \quad (e_{34})$$

### function-application

$$\frac{E(f) = \langle \emptyset, e \rangle \quad E \models e \Downarrow (E', v)}{E \models f \Downarrow (E', v)} \quad (e_{35})$$

$$\frac{E(f) = \langle \vec{x}_{i=0}^n, e \rangle \quad E \models \text{elist} \Downarrow (E', \vec{v}_{i=0}^n) \quad E' :: \overline{x_i = v_{i=0}^n} \models e \Downarrow (E'', v)}{E \models f \text{elist} \Downarrow (E', v)} \quad (e_{36})$$

### expression-list

$$\frac{E \models \vec{e}_{i=0}^{n-1} \Downarrow (E', \vec{v}_{i=0}^{n-1}) \quad E' \models e_n \Downarrow (E'', v_n)}{E \models \vec{e}_{i=0}^n \Downarrow (E'', \vec{v}_{i=0}^n)} \quad (e_{37})$$

L'ordre d'évaluation des paramètres se fait de gauche à droite.

### function-reference

$$\frac{S(f) = f_{ref}}{H, S \models @f \Downarrow (H, S, f_{ref})} \quad (e_{38})$$

### block

$$\frac{E \models \text{block} \Downarrow (E', v_1) \quad E' \models l \Downarrow (E'', v_2)}{E \models \text{block}; l \Downarrow (E'', v_2)} \quad (e_{39})$$

La valeur d'un block est celle de la dernière de ses expressions.

### list

$$\frac{H, S \models e_1 \Downarrow (H', S', v_1) \quad H', S' \models e_2 \Downarrow (H'', S'', NIL)}{H, S \models e_1 :: e_2 \Downarrow (H'' :: \{l_{ref} = [v_1 NIL]\}, S'', l_{ref})} \quad (e_{41})$$

Cas terminal.

$$\frac{H, S \models e_1 \Downarrow (H', S', v_1) \quad H', S' \models e_2 \Downarrow (H'', S'', v_{ref})}{H, S \models e_1 :: e_2 \Downarrow (H'' :: \{l_{ref} = [v_1 \ v_{ref}]\}, S'', l_{ref})} \quad (e_{42})$$

Cas récursif.

### assignment

$$\frac{H, S \models e \Downarrow (H', S', v)}{H, S \models \text{set identifier} = e \Downarrow (H', S'[\text{identifier} \leftarrow v], v)} \quad (e_{43})$$

$$\frac{H, S \models \text{lval} \Downarrow (H', S', s_{ref}) \quad H', S' \models e \Downarrow (H'', S'', v) \quad H''(s_{ref}) = S[\overline{c_i = v_{i=0}^n}]}{H, S \models \text{set lval}.c_k = e \Downarrow (H''[v_{ref} \leftarrow S[\overline{c_i = v_{i=0}^{k-1}}, c_k = v, \overline{c_i = v_{i=k+1}^n}], S'', v)} \quad (e_{44}), 0 \leq k \leq n$$

Attention: il peut y avoir eu des effets de bord sur les valeurs initiales (dans  $H$ ) des champs de la structure lors de l'évaluation de  $e$ .

$$\frac{H, S \models \text{lval} \Downarrow (H', S', t_{ref}) \quad H', S' \models e_1 \Downarrow (H'', S'', k) \quad H'', S'' \models e_2 \Downarrow (H''', S''', v) \quad H'''(t_{ref}) = T[\overline{v_{i=0}^n}]}{H, S \models \text{set lval}.e_1 = e_2 \Downarrow (H'''[v_{ref} \leftarrow T[\overline{v_{i=0}^{k-1}}, v, \overline{v_{i=k+1}^n}], S''', v)} \quad (e_{45})$$

Attention: il peut y avoir eu des effets de bords sur les valeurs initiales (dans  $H$ ) des éléments du tableau lors de l'évaluation de  $e_1$  et  $e_2$ .

$$\frac{H, S \models \text{lval} \Downarrow (H', S', NIL) \quad H', S' \models e \Downarrow (H'', S'', v_2)}{H, S \models \text{set lval} = e \Downarrow (H'', S'', NIL)} \quad (e_{46})$$

Attention: même si l'expression  $\text{lval}$  vaut  $NIL$ , on calcule l'expression droite de l'*assignment*: les effets de bords sont à prendre en compte. Cette dernière règle a priorité sur les trois précédentes.

### conditional

$$\frac{E \models e_1 \Downarrow (E', v_1) \quad E' \models e_2 \Downarrow (E'', v_2)}{E \models \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \Downarrow (E'', v_2)} \quad (e_{47}) \quad v_1 \neq 0$$

$$\frac{E \models e_1 \Downarrow (E', 0) \quad E' \models e_3 \Downarrow (E'', v_3)}{E \models \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \Downarrow (E'', v_3)} \quad (e_{48})$$

$$\frac{E \models e_1 \Downarrow (E', NIL) \quad E' \models e_2 \Downarrow (E'', v_2)}{E \models \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \Downarrow (E'', v_2)} \quad (e_{49})$$

Attention: la valeur  $NIL$ , utilisée dans une condition, est considérée comme *vrai*.

### let-expression

$$\frac{E \models ldef \Downarrow (E' :: E_p, \top) \quad E' :: E_p \models e \Downarrow (E'', v)}{E \models let\ ldef\ in\ e \Downarrow (E'', v)} \quad (e_{50})$$

$E_p$  est l'ensemble des variables libre de  $e$ .

### let-def

$$\frac{E \models e \Downarrow (E', v) \quad E' :: \{against = v\} \models lpattern \Downarrow (E' :: E_p, \top)}{E \models e \rightarrow lpattern \Downarrow (E' :: E_p, \top)} \quad (e_{51}) \quad against \notin dom(E)$$

### let-pattern

$$\overline{\overline{E :: \{against = v\} \models identifier \Downarrow (E :: \{identifier = v\}, \top)}} \quad (e_{52})$$

$$\overline{\overline{E :: \{against = v\} \models \_ \Downarrow (E :: \{\_ = v\}, \top)}} \quad (e_{53})$$

Attention: le caractère  $\_$ , utilisé dans le *let-pattern* est considéré comme un identifiant, pas comme le *wildcard* du *basic-pattern*.

$$\frac{H(t_{ref}) = [\emptyset]}{H, S :: \{against = t_{ref}\} \models [] \Downarrow (H, S, \top)} \quad (e_{54})$$

$$\frac{\mathcal{I}(lpattern) = \vec{x}_{i=0}^n}{E :: \{against = NIL\} \models lpattern \Downarrow (E :: \{x_i = NIL_{i=0}^n\})} \quad (e_{55})$$

$\mathcal{I}$  est la fonction qui renvoie l'ensemble des identifiants (éventuellement vide ou le singleton  $\{\_ \}$ ) intervenant dans le *let-pattern*.

$$\frac{H(t_{ref}) = [\vec{v}_{i=0}^n] \quad H, S :: \{\overrightarrow{against_i = v_{i=0}^n}\} \models bplist \Downarrow (H', S' :: S_p, \top)}{H, S :: \{against = t_{ref}\} \models [bplist] \Downarrow (H', S' :: S_p, \top)} \quad (e_{56}) \quad against_i \notin dom(S)$$

### match-expression

$$\frac{E \models e \Downarrow (E', v_1) \quad E' :: \{against = v_1\} \models mclist \Downarrow (E'', v_2)}{E \models match\ e\ with\ mclist \Downarrow (E'', v_2)} \quad (e_{57}) \quad v_2 \neq \perp \quad against \notin dom(S)$$

$$\frac{E \models e \Downarrow (E', v_1) \quad E' :: \{ \textit{against} = v_1 \} \models \textit{mclist} \Downarrow (E', \perp)}{E \models \textit{match } e \textit{ with mclist} \Downarrow (E', \textit{NIL})} \quad (e_{58}) \quad \textit{against} \notin \textit{dom}(S)$$

Cas où le *pattern-matching* échoue.

### match-case-list

$$\frac{E :: \{ \textit{against} = v_1 \} \models \textit{mcase} \Downarrow (E', v_2)}{E :: \{ \textit{against} = v_1 \} \models \textit{mcase} \mid \textit{mclist} \Downarrow (E', v_2)} \quad (e_{59}) \quad v_2 \neq \perp$$

*mcase* *matche* la valeur  $v_1$ : on ne cherche pas à savoir si d'autres cas de la liste *matchent* cette valeur.

$$\frac{E :: \{ \textit{against} = v_1 \} \models \textit{mcase} \Downarrow (E, \perp) \quad E :: \{ \textit{against} = v_1 \} \models \textit{mclist} \Downarrow (E', v_2)}{E :: \{ \textit{against} = v_1 \} \models \textit{mcase} \mid \textit{mclist} \Downarrow (E', v_2)} \quad (e_{60})$$

*mcase* ne *matche* pas la valeur  $v_1$ : on évalue le cas suivant.

### match-case

$$\frac{E :: \{ \textit{against} = v \} \models \textit{pattern} \Downarrow (E, \perp)}{E :: \{ \textit{against} = v \} \models (\textit{pattern} \rightarrow e) \Downarrow (E, \perp)} \quad (e_{61})$$

$$\frac{E :: \{ \textit{against} = v \} \models \textit{pattern} \Downarrow (E :: E_p, \top) \quad E :: E_p \models e \Downarrow (E' :: E'_p, v)}{E :: \{ \textit{against} = v \} \models (\textit{pattern} \rightarrow e) \Downarrow (E', v)} \quad (e_{62})$$

$E_p$  est l'ensemble des identifiants valués par le *pattern*. Cet ensemble d'identifiants contient les variables libres de  $e$ .

### pattern

$$\overline{\overline{\overline{E :: \{ \textit{against} = v \} \models \_ \Downarrow (E, \top)}}}} \quad (e_{63})$$

$\_$  est le *wildcard-pattern*.

$$\frac{H(u_{ref}) = U_i \emptyset}{H, S :: \{ \textit{against} = u_{ref} \} \models U_i \Downarrow (H, S, \top)} \quad (e_{64})$$

$$\frac{H(u_{ref}) = U_i v}{H, S :: \{ \textit{against} = u_{ref} \} \models U_i \textit{identifier} \Downarrow (H, S :: \{ \textit{identifier} = v \}, \top)} \quad (e_{65})$$

$$\frac{H(u_{ref}) = U_i t_{ref} \quad H, S :: \{against = t_{ref}\} \models [bplist] \Downarrow (H :: H_p, S :: S_p, \top)}{H, S :: \{against = u_{ref}\} \models U_i [bplist] \Downarrow (H :: H_p, S :: S_p, \top)} \quad (e_{66}) \quad against \notin dom(S)$$

$S_p$  est l'ensemble des variables libres de  $bplist$ .  $H_p$  les références éventuellement ajoutées vers le tas.

$$\frac{H(u_{ref}) = U_i v}{H, S :: \{against = u_{ref}\} \models U_j basicpattern \Downarrow (E, \perp)} \quad (e_{67}) \quad i \neq j$$

Cette règle vaut pour tout  $v$ , que ce soit une valeur simple, une valeur référence ou  $\emptyset$ .

### basic-pattern

$$\overline{\overline{E :: \{against = v\} \models identifier \Downarrow (E :: \{identifier = v\}, \top)}} \quad (e_{68})$$

$$\overline{\overline{E :: \{against = v\} \models - \Downarrow (E, \top)}} \quad (e_{69})$$

$-$  est le *wildcard-pattern*.

$$\frac{H(t_{ref}) = [\vec{v}_{i=0}^n] \quad H, S :: \{\overrightarrow{against}_i = \vec{v}_{i=0}^n\} \models bplist \Downarrow (H :: H_p, S :: S_p, \top)}{H, S :: \{against = t_{ref}\} \models [bplist] \Downarrow (H :: H_p, S :: S_p, \top)} \quad (e_{70}) \quad against_i \notin dom(S)$$

$$\frac{\mathcal{I}(bpattern) = \vec{x}_{i=0}^n}{E :: \{against = NIL\} \models bpattern \Downarrow (E :: \{x_i = NIL_{i=0}^n\}, \top)} \quad (e_{71})$$

### basic-pattern-list

$$\frac{E :: \{\overrightarrow{against}_i = \vec{v}_{i=0}^{n-1}\} \models \overrightarrow{bpattern}_{i=0}^{n-1} \Downarrow (E :: E_1, \top) \quad E :: \{against_n = v_n\} \models bpattern_n \Downarrow (E :: E_2, \top)}{E :: \{\overrightarrow{against}_i = \vec{v}_{i=0}^n\} \models \overrightarrow{bpattern}_{i=0}^n \Downarrow (E :: E_1 :: E_2, \top)} \quad (e_{72})$$

Attention: l'ordre d'évaluation d'un *basic-pattern-list* implique que si deux identifiants identiques sont utilisés dans le pattern, c'est la valeur du dernier évalué qui sera retenu. C'est cette valeur qui sera utilisée comme étant celle de la variable libre correspondante dans l'expression du *match-case*.

### iteration

$$\frac{E \models e_1 \Downarrow (E', 0)}{E \models while e_1 do e_2 \Downarrow (E', NIL)} \quad (e_{73})$$

Cas où on ne rentre jamais dans la boucle.

$$\frac{E \models e_1 \Downarrow (E', v_1) \quad E' \models e_2 \Downarrow (E'', v_2) \quad E'' :: \{loc = v_2\} \models e_1 \text{ do } e_2 \Downarrow (E''', v)}{E \models \text{while } e_1 \text{ do } e_2 \Downarrow (E''', v)} \quad (e74) \quad v_1 \neq 0$$

### do-expression

$$\frac{E \models e_1 \Downarrow (E', 0)}{E :: \{loc = v\} \models e_1 \text{ do } e_2 \Downarrow (E', v)} \quad (e75)$$

Fin de la boucle.

$$\frac{E \models e_1 \Downarrow (E', v_1) \quad E' \models e_2 \Downarrow (E'', v_2) \quad E'' :: \{loc = v_2\} \models e_1 \text{ do } e_2 \Downarrow (E''', v)}{E \models \text{while } e_1 \text{ do } e_2 \Downarrow (E''', v)} \quad (e76) \quad v_1 \neq 0$$

Itération.

### exec-expression

$$\frac{H, S \models e_f \Downarrow (H', S', f_{ref}) \quad H'(f_{ref}) = \langle \vec{x}_{i=0}^n, e \rangle \quad H', S' \models e_{tup} \Downarrow (H'', S'', NIL)}{H, E \models \text{exec } e_f \text{ with } e_{tup} \Downarrow (H'', E'', NIL)} \quad (e77)$$

$$\frac{E \models e_f \Downarrow (E', NIL) \quad E' \models e_{tup} \Downarrow (E'', v)}{E \models \text{exec } @f \text{ with } e_{tup} \Downarrow (E'', NIL)} \quad (e78)$$

Attention: même si l'expression  $e_f$  vaut  $NIL$ , on calcule l'expression droite de l'*exec-expression*: les effets de bords sont à prendre en compte. De plus, dans ce cas précis, l'expression droite n'est pas nécessairement un tuple.

$$\frac{H, S \models e_f \Downarrow (H', S', f_{ref}) \quad H'(f_{ref}) = \langle \vec{x}_{i=0}^n, e \rangle \quad H', S' \models e_{tup} \Downarrow (H'', S'', t_{ref}) \quad H''(t_{ref}) = [\vec{v}_{i=0}^n] \quad H'', S'' :: \{\vec{x}_i = \vec{v}_{i=0}^n\} \models e \Downarrow (H''', S''', v)}{H, S \models \text{exec } e_f \text{ with } e_{tup} \Downarrow (H''', S''', v)} \quad (e79)$$

### mutate-expression

$$\frac{H, S \models e \Downarrow (H', S', t_{ref}) \quad H(t_{ref}) = [\emptyset]}{H, S \models \text{mutate } e \leftarrow [] \Downarrow (H', S', t_{ref})} \quad (e80)$$

$$\frac{H, S \models e \Downarrow (H', S', t_{ref}) \quad H', S' :: \{loc = t_{ref}\} \models mrl \Downarrow (H'', S'' :: \{loc = t_{ref}\}, \top)}{H, S \models \text{mutate } e \leftarrow mrl \Downarrow (H'', S'', t_{ref})} \quad (e81)$$

L'identifiant  $loc$  correspond à une variable locale dont la valeur de pile ne peut pas être modifiée par effet de bord.

### mutate-replacement-list

$$\frac{H, S :: \{loc = t_{ref}\} \models \overrightarrow{mre}_{i=0}^{k-1} \Downarrow (H', S' :: \{loc = t_{ref}\}, \top) \quad H', S' :: \{loc = t_{ref}\} \models mre_k \Downarrow (H'', S'' :: \{loc = t_{ref}\}, v_k) \quad H''(t_{ref}) = [\overrightarrow{v''}_{i=0}^n]}{H, S \models \overrightarrow{mre}_{i=0}^k \Downarrow (H''[t_{ref} \leftarrow [\overrightarrow{v''}_{i=0}^{k-1}, v_k, \overrightarrow{v''}_{i=k+1}^n]], S'' :: \{loc = t_{ref}\}, \top)} \quad (e_{82})$$

pour  $2 < k \leq n$

$$\frac{H, S' :: \{loc = t_{ref}\} \models mre_0 \Downarrow (H', S' :: \{loc = t_{ref}\}, v_0) \quad H'(t_{ref}) = [\overrightarrow{v'}_{i=0}^n] \quad H'[t_{ref} \leftarrow [v_0, \overrightarrow{v'}_{i=1}^n]], S' :: \{loc = t_{ref}\} \models mre_1 \Downarrow (H'', E'' :: \{loc = t_{ref}\}, v_1) \quad H''(t_{ref}) = [v'_0, \overrightarrow{v''}_{i=1}^n]}{H, S \models mre_1 \, mre_2 \Downarrow (H''[t_{ref} \leftarrow [v'_0, v_1, \overrightarrow{v''}_{i=2}^n]], E'' :: \{loc = t_{ref}\}, \top)} \quad (e_{83})$$

Les substitutions s'effectuent de gauche à droite dans l'ordre ou les *mutate-replacement-expression* sont évaluées. Attention aux manifestations des effets de bords dus à l'évaluation de ces expressions.

### mutate-replacement-expression

$$\frac{H(t_{ref}) = [\overrightarrow{v'}_{i=0}^n]}{H, E :: \{loc = t_{ref}\} \models \_ \Downarrow (H, E, v_k)} \quad (e_{84}) \quad 0 \leq k \leq n$$

Le caractère  $\_$  signifie que la valeur correspondante dans le tuple muté ne sera pas modifiée.