MI030 — APS
Analyse des programmes et sémantique

© Jacques Malenfant, 2010–2014

avec la participation initiale d'**Olena Rogovchenko**

Université Pierre et Marie Curie
UFR 919 Ingénierie
Jacques.Malenfant@lip6.fr

Cours 8 et 9

Sémantique dénotationnelle de BOPL

Principes généraux

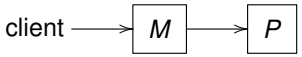
En tant que langage à objets, BOPL introduit deux concepts importants par rapport au mini-langage impératif précédent :

- l'héritage, par la clause `extends` et l'utilisation de `super`, et
- l'auto-référence des objets, par l'utilisation de `self`.

L'auto-référence est une forme de récurrence qui va nécessiter l'utilisation d'un point fixe dans la définition de sa dénotation.

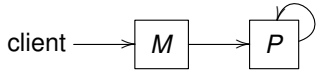
Vers l'auto-référence I

- Considérons une méthode M qui doit redéfinir une méthode P héritée.
- Par exemple, M fait un prétraitement, appelle P et enfin fait des post-traitements après P avant de retourner à son appelant.
- Du point de vue du client, la situation est la suivante :

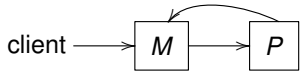


Vers l'auto-référence II

- Si la méthode P est récursive, elle doit possèder une référence à elle-même, que nous serions tentés de résoudre immédiatement de la manière suivante :

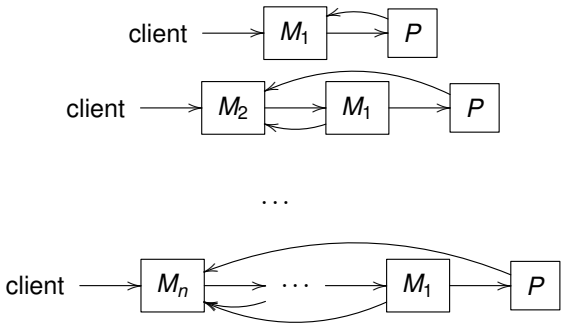


- Mais ce n'est pas le comportement attendu dans les langages à objets, puisque l'auto-référence de P devrait en fait voir la redéfinition opérée par M. Donc, le schéma souhaité serait plutôt :



Vers l'auto-référence III

- En généralisant, on voudrait obtenir, après une succession de redéfinitions M_1, M_2, ..., M_n, le schéma suivant :



Vers l'auto-référence IV

où le schéma pour la redéfinition M_i ne conserve que les auto-références à M_i pour tous les éléments $M_{i-1,\ldots,1}$ et P.

Comment appliquer ce patron aux objets ? I

- Pour les objets, supposons qu'on adopte une représentation sous la forme d'une fonction de l'ensemble des identifiants vers des λ-termes notée :

$$\{i_1 \mapsto \Lambda_1, \ldots, i_1 \mapsto \Lambda_n\}$$

- Soit la classe `Point` :

```
class Point(a, b)
  method x = a
  method y = b
  method distFromOrig() = sqrt(self.x^2 + self.y^2)
  method closerToOrig(p) =
            self.distFromOrig() < p.distFromOrig()
```


Comment appliquer ce patron aux objets ? II

- Elle sera représentée par une fonction engendrant ce que Cook et Palsberg ont appelé un *générateur* de points :

$$\begin{aligned} \mathit{MakeGenPoint}(a,b) = \lambda \mathit{self}.\ & \{x \mapsto a, \\ & y \mapsto b, \\ & \mathit{distFromOrig} \mapsto \sqrt{\mathit{self}.x^2 + \mathit{self}.y^2} \\ & \mathit{closerToOrig} \mapsto \\ & \quad \lambda p.(\mathit{self}.\mathit{distFromOrig} < p.\mathit{distFromOrig})\} \end{aligned}$$

- Le générateur est une fonction prenant *self* en paramètre et retournant l'instance de la classe `Point` selon la représentation choisie.

Comment appliquer ce patron aux objets ? III

- *self* représentant la référence à l'objet lui-même, de la même manière qu'une fonction récursive fait référence à elle-même, on peut « résoudre » cette auto-référence en prenant le point fixe de ce générateur. Le point *p*, de coordonnées 3 et 4, sera alors représenté par :

$$\begin{aligned} p &= \mathbf{fix}(\mathit{MakeGenPoint}(3,4)) \\ &= \mathbf{fix}(\lambda \mathit{self}.\{x \mapsto 3, \\ &\qquad y \mapsto 4, \\ &\qquad \mathit{distFromOrig} \mapsto \sqrt{\mathit{self}.x^2 + \mathit{self}.y^2} \\ &\qquad \mathit{closerToOrig} \mapsto \\ &\qquad\quad \lambda p.(\mathit{self}.\mathit{distFromOrig} < p.\mathit{distFromOrig})\}) \end{aligned}$$

Comment tenir compte de l'héritage ? I

- Que se passe-t'il lorsqu'on veut étendre la classe `Point` ?
- Considérons l'extension suivante dans le pseudo-code précédent :

```
class Circle(a, b, r) inherits Point(a, b)
  method radius = r
  method distFromOrig() =
            max(super.distFromOrig - self.radius, 0)
```

- En plus des références à *self*, il faut être en mesure de résoudre les références à *super*, mais aussi les références directes aux éléments de `Point` qui ne sont pas masqués par `Circle` (`x`, `y` et `closerToOrig`).

Comment tenir compte de l'héritage ? II

- À cette fin, cette extension va être représentée par ce que Cook et Palsberg ont appelé une *enveloppe* (ou « *wrapper* ») :

$$\begin{aligned} \mathit{CircleWrapper}(a,b,r) = \lambda \mathit{self}.\lambda \mathit{super} \\ \{\mathit{radius} \mapsto r, \\ \mathit{distFromOrig} \mapsto \\ \max(\mathit{super}.\mathit{distFromOrig} - \mathit{self}.\mathit{radius}, 0)\} \end{aligned}$$

- L'idée est de « passer en paramètre » la « partie » de l'objet cercle qui est héritée de la classe `Point` de manière à y référer par *super*.

Comment tenir compte de l'héritage ? III

- La construction du générateur pour la classe `Circle` va être un peu plus ardue que dans le cas de `Point`. Cela demande la composition entre l'enveloppe de `Circle` et le générateur de `Point` avant de prendre le point fixe.
- Le point fixe devra permettre de résoudre correctement les références à *self* de `Point`, c'est-à-dire faire en sorte que ces références regardent d'abord les définitions de `Circle` avant celles de `Point`.

Comment tenir compte de l'héritage ? IV

- Cook et Palsberg ont défini un opérateur de composition $\triangleright$ prenant l'enveloppe de la sous-classe et le générateur de la superclasse pour retourner le générateur de la sous-classe :

$$W \triangleright G = \lambda \mathit{self}.(W(\mathit{self})(G(\mathit{self})) \boxplus G(\mathit{self}))$$

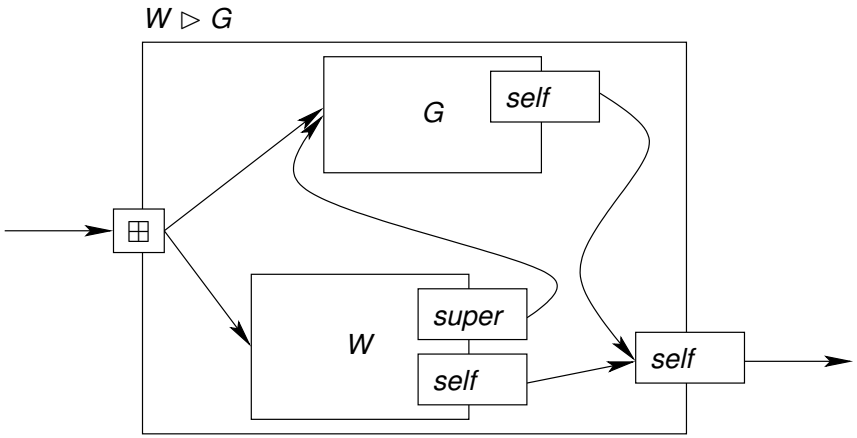
où $\boxplus$ est l'opérateur de combinaison masquante des fonctions représentant les objets qui, conceptuellement, remplace toutes les définitions communes de son opérande de droite par celles de son opérande de gauche.

Exemple :

Comment tenir compte de l'héritage ? V

$$\begin{aligned} P &= (\mathit{MakeGenPoint}(3,4)\ \mathit{self}) \\ &= \{x \mapsto 3,\ y \mapsto 4, \\ &\quad \mathit{distFromOrig} \mapsto \sqrt{\mathit{self}.x^2 + \mathit{self}.y^2} \\ &\quad \mathit{closerToOrig} \mapsto \lambda p.(\mathit{self}.\mathit{distFromOrig} < p.\mathit{distFromOrig})\}) \\ C &= ((\mathit{CircleWrapper}(3,4,2)\ \mathit{self})\ P) \\ &= \{\mathit{radius} \mapsto 2, \\ &\quad \mathit{distFromOrig} \mapsto \max(\mathit{super}.\mathit{distFromOrig} - \mathit{self}.\mathit{radius}, 0)\} \\ C \boxplus P &= \{x \mapsto 3,\ y \mapsto 4, \\ &\quad \mathit{closerToOrig} \mapsto \lambda p.(\mathit{self}.\mathit{distFromOrig} < p.\mathit{distFromOrig})\}) \\ &\quad \mathit{radius} \mapsto 2, \\ &\quad \mathit{distFromOrig} \mapsto \max(\mathit{super}.\mathit{distFromOrig} - \mathit{self}.\mathit{radius}, 0)\} \end{aligned}$$

Visualisation de l'opérateur de combinaison



Principes de base

- Inspirée du modèle des générateurs et enveloppes de Cook et Palsberg.
- Deux entités plutôt qu'une :
  - les classes, et leurs enveloppes pour gérer le *super*,
  - les objets et leurs générateurs pour gérer le *self*.
- Les méthodes : entités qui prennent *super* et *self* en paramètres :
  - le *super* est lié statiquement par la classe, alors que
  - le *self* est lié à la création des instances

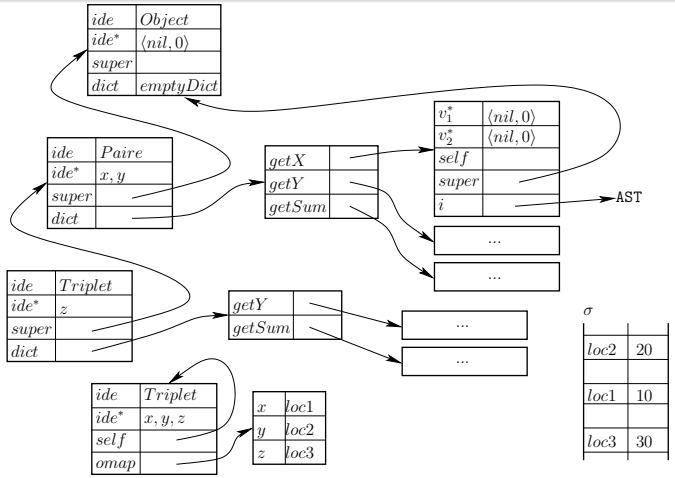
Un exemple de programme I

```
program
  Class Paire is
  vars
    Int x, y ;
  methods
    Int getX() begin return self.x end
    Int getY() begin return self.y end
    Int getSum() begin return self.getX() + self.getY() end
  end
  Class Triplet entends Pair is
  vars
    Int z ;
  methods
```


Un exemple de programme II

```
    Int getZ() begin return self.z end
    Int getSum() begin return super.getSum() + self.getZ() end
  end
let
  Paire p ;
  Triplet t ;
in
begin
  ...
end
```


Représentation des entités dans la SD



Les catégories syntaxiques et la grammaire abstraite

p	$\in$	*Program*	p	::= `program` c^* v^* i
c	$\in$	*Class*	c	::= `class` *id* *ce* v^* m^*
ce	$\in$	*CExp*	*ce*	::= `cexp` *id*
v	$\in$	*Var*	v	::= `var` *ce* *id*
m	$\in$	*Method*	m	::= `method` *id* v_1^* *ce* v_2^* i
i	$\in$	*Instructions*	i	::= `seq` i_1 i_2 | `assign` *id* e | `writefield` e_1 *id* e_2 |
e	$\in$	*Expressions*		`if` e i_1 i_2 | `while` e i | `return` e | `writeln` e
id	$\in$	*Identifiers*	e	::= n | `true` | `false` | `not` e | `nil` | `self` | `super` |
n	$\in$	*Numbers*		`new` *ce* | `instanceof` e *ce* | `methodcall` e *id* e^* |
				`readfield` e *id* | `plus` e_1 e_2 | `minus` e_1 e_2 |
				`times` e_1 e_2 | `equal` e_1 e_2 | `and` e_1 e_2 | `or` e_1 e_2 |
				`less` e_1 e_2

Les domaines sémantiques I

T	=	$\{true, false, \bot_{\mathbf{T}}\}$	
Z	=	$\{\ldots,-2,-1,0,1,2,\ldots\} \cup \{\bot_{\mathbf{Z}}\}$	
V	=	$\mathbf{T} \oplus \mathbf{Z}$	
Ide	=	*non-spécifié*	
Address	=	*non-spécifié*	
Loc	=	**Address**	
Oid	=	$\mathbf{Address} \oplus \{nil, \bot_{\mathbf{Nil}}\}$	
LV	=	**Loc**	// Left Values
RV	=	$\mathbf{V} \oplus \mathbf{Oid}$	// Right Values

Les domaines sémantiques II

U	=	$\{unbound, \bot_{\mathbf{U}}\}$	
DV	=	$\mathbf{LV} \oplus \mathbf{Class} \oplus \mathbf{Object} \oplus \mathbf{U}$	// Denotable Values
EV	=	**RV**	// Expression Values
PV	=	**RV**	// Parameter Values
UU	=	$\{undefined, unused, \bot_{UU}\}$	
SV	=	$\mathbf{RV} \oplus \mathbf{Object} \oplus \mathbf{UU}$	// Storable Values

Les fonctions sémantiques : environnement

$$
\begin{array}{rcl}
\textbf{Env} & = & \textbf{Ide} \to \textbf{DV} \\
\mathit{emptyEnv} & : & \textbf{Env} \\
& = & \lambda \mathit{ide}.\mathit{in}\textbf{DV}_4(\mathit{unbound}) \\
\mathit{extendEnv} & : & \textbf{Env} \to \textbf{Ide} \to \textbf{DV} \to \textbf{Env} \\
& = & \lambda\rho.\lambda \mathit{ide}.\lambda \mathit{dv}.\lambda \mathit{ide}_1.\textbf{if } \mathit{ide}_1 = \mathit{ide} \textbf{ then } \mathit{dv} \textbf{ else } (\rho\ \mathit{ide}_1) \\
\mathit{extendEnv}* & : & \textbf{Env} \to \textbf{Ide}^* \to \textbf{DV}^* \to \textbf{Env} \\
& = & \lambda\rho.\lambda \mathit{ide}^*.\lambda \mathit{dv}^*. \\
& & \quad ((((\textbf{fix } \lambda f.\lambda \mathit{ide}^*.\lambda \mathit{dv}^*.\lambda\rho. \\
& & \qquad \textbf{if } \neg \mathit{null}(\mathit{ide}^*) \\
& & \qquad \textbf{then } (((f\ (\mathit{tail}\ \mathit{ide}^*))\ (\mathit{tail}\ \mathit{dv}^*)) \\
& & \qquad\quad (((\mathit{extendEnv}\ \rho)\ (\mathit{head}\ \mathit{ide}^*))\ (\mathit{head}\ \mathit{dv}^*))) \\
& & \qquad \textbf{else } \rho) \\
& & \quad \mathit{ide}^*)\ \mathit{dv}^*)\ \rho)
\end{array}
$$

Les fonctions sémantiques : la mémoire I

$$
\begin{array}{rcl}
\Sigma & = & \textbf{Loc} \to \textbf{SV} \\
\mathit{emptyStore} & : & \Sigma \\
& = & \lambda l.\mathit{in}\textbf{SV}_3(\mathit{unused}) \\
\mathit{updateStore} & : & \Sigma \to \textbf{Loc} \to \textbf{SV} \to \Sigma \\
& = & \lambda\sigma.\lambda l.\lambda \mathit{sv}.\lambda l_1.\textbf{if } l = l_1 \textbf{ then } \mathit{sv} \textbf{ else } (\sigma\ l_1) \\
\mathit{updateStore}* & : & \Sigma \to \textbf{Loc}^* \to \textbf{SV}^* \to \Sigma \\
& = & \lambda\sigma.\lambda l^*.\lambda \mathit{sv}^*. \\
& & \quad ((((\textbf{fix } \lambda f.\lambda l^*.\lambda \mathit{sv}^*.\lambda\sigma. \\
& & \qquad \textbf{if } \neg \mathit{null}(l^*) \\
& & \qquad \textbf{then } (((f\ (\mathit{tail}\ l^*))\ (\mathit{tail}\ \mathit{sv}^*)) \\
& & \qquad\quad (((\mathit{updateStore}\ \sigma)\ (\mathit{head}\ l^*))\ (\mathit{head}\ \mathit{sv}^*))) \\
& & \qquad \textbf{else } \sigma)\ \ l^*)\ \mathit{sv}^*)\ \sigma)
\end{array}
$$

Les fonctions sémantiques : la mémoire II

$$
\begin{array}{rcl}
\mathit{allocate} & : & \Sigma \to \Sigma \otimes \textbf{Loc} \\
& = & \lambda\sigma.\langle(((\mathit{updateStore}\ \sigma)\ l)\ \mathit{in}\textbf{SV}_3(\mathit{undefined})), l\rangle \\
& & \quad \mathit{where}\ l \in \textbf{Loc} \mid \mathit{is}\textbf{UU}(\sigma\ l) \wedge \mathit{out}\textbf{UU}(\sigma\ l) = \mathit{unused} \\
\mathit{allocate}* & : & \Sigma \to \mathbb{N} \to \Sigma \otimes \textbf{Loc}^* \\
& = & \lambda\sigma.\lambda n. \\
& & \quad (((\textbf{fix } \lambda f.\lambda\sigma.\lambda n. \\
& & \qquad \textbf{if } n = 0 \textbf{ then } \langle\sigma, \langle \mathit{nil}, 0\rangle\rangle \\
& & \qquad \textbf{else} \\
& & \qquad\quad \textbf{let } p_1 = ((f\ \sigma)\ (-\ n\ 1)) \\
& & \qquad\quad \textbf{and } p_2 = (\mathit{allocate}\ (\mathit{first}\ p_1)) \\
& & \qquad\quad \textbf{in } \langle(\mathit{first}\ p_2), (\mathit{prefix}\ (\mathit{second}\ p_2)\ (\mathit{second}\ p_1))\rangle) \\
& & \quad \sigma)\ n)
\end{array}
$$

Les fonctions sémantiques : la mémoire III

$$
\begin{array}{rcl}
\mathit{deallocate} & : & \Sigma \to \textbf{Loc} \to \Sigma \\
& = & \lambda\sigma.\lambda l.\lambda l_1.\textbf{if } l = l_1 \textbf{ then } \mathit{in}\textbf{SV}_3(\mathit{unused}) \textbf{ else } (\sigma\ l_1) \\
\mathit{deallocate}* & : & \Sigma \to \textbf{Loc}^* \to \Sigma \\
& = & \lambda\sigma.\lambda l^*.(((\textbf{fix } \lambda f.\lambda\sigma.\lambda l^*. \\
& & \qquad \textbf{if } \mathit{null}(l^*) \textbf{ then } \sigma \\
& & \qquad \textbf{else } ((f\ ((\mathit{deallocate}\ \sigma)\ (\mathit{head}\ l^*)))\ (\mathit{tail}\ l^*))) \\
& & \quad \sigma)\ l^*)
\end{array}
$$

Les fonctions sémantiques : dictionnaire de méthodes I

$$
\begin{array}{rcl}
\textbf{Method} & = & \textbf{Object} \to \textbf{Env} \to \Sigma \to \textbf{Out} \to \textbf{PV}^* \to (\textbf{EV} \otimes \Sigma \otimes \textbf{Out}) \\
\textbf{Dict} & : & \textbf{Ide} \to (\textbf{Method} \oplus \textbf{U}) \\
\mathit{emptyDict} & : & \textbf{Dict} \\
 & = & \lambda \mathit{ide}.\mathit{in}(\textbf{Method} \oplus \textbf{U})_2(\mathit{unbound}) \\
\mathit{extendDict} & : & \textbf{Dict} \to \textbf{Ide} \to (\textbf{Method} \oplus \textbf{U}) \to \textbf{Dict} \\
 & = & \lambda d.\lambda \mathit{ide}.\lambda m. \\
 & & \quad \lambda \mathit{ide}_1.\textbf{if } \mathit{ide} = \mathit{ide}_1 \textbf{ then } \mathit{in}(\textbf{Method} \oplus \textbf{U})_1(m) \textbf{ else } (d\ \mathit{ide}_1)
\end{array}
$$

Les fonctions sémantiques : dictionnaire de méthodes II

$$
\begin{array}{rcl}
\mathit{make\text{-}method} & = & \lambda \mathit{ide}_1^*.\lambda \mathit{ide}_2^*.\lambda i.\lambda \mathit{super}. \\
 & & \lambda \mathit{self}.\lambda \rho.\lambda \sigma.\lambda o.\lambda \mathit{pv}^*. \\
 & & \quad \textbf{let}* \langle \sigma_1, \mathit{loc}^* \rangle = (\mathit{allocate}* \ \sigma \sharp \mathit{pv}^*) \\
 & & \quad \textbf{and } \rho_1 = (((\mathit{extendEnv}* \ \rho)\ \mathit{ide}_1^*)\ \mathit{in}\textbf{DV}_1^*(\mathit{in}\textbf{LV}_1^*(\mathit{loc}^*))) \\
 & & \quad \textbf{and } \sigma_2 = (((\mathit{updateStore}* \ \sigma_1)\ \mathit{loc}^*)\ \textbf{PV}^*\mathit{to}\textbf{SV}^*(\mathit{pv}^*)) \\
 & & \quad \textbf{and } \langle \sigma_3, \mathit{loc}_3^* \rangle = (\mathit{allocate}* \ \sigma_2 \sharp \mathit{ide}_2^*) \\
 & & \quad \textbf{and } \rho_3 = (((\mathit{extendEnv}* \ \rho_1)\ \mathit{ide}_2^*)\ \mathit{in}\textbf{DV}_1^*(\mathit{in}\textbf{LV}_1^*(\mathit{loc}_3^*))) \\
 & & \quad \textbf{and } \langle \sigma_4, \mathit{loc}_4 \rangle = (\mathit{allocate}\ \sigma_3) \\
 & & \quad \textbf{and } \rho_4 = (((\mathit{extendEnv}* \ \rho_3)\ \mathscr{I}[\![\texttt{return}]\!])\ \mathit{in}\textbf{DV}_1(\mathit{in}\textbf{LV}_1(\mathit{loc}_4))) \\
 & & \quad \textbf{and } \rho_5 = (((\mathit{extendEnv}\ (((\mathit{extendEnv}\ \rho_4)\ \mathscr{I}[\![\texttt{self}]\!])\ \mathit{in}\textbf{DV}_3(\mathit{self})) \\
 & & \qquad\qquad \mathscr{I}[\![\texttt{super}]\!])\ \mathit{in}\textbf{DV}_3(\mathit{super})) \\
 & & \quad \textbf{and } \langle \sigma_5, o_1 \rangle = \mathscr{C}[\![i]\!]\rho_5\sigma_4 o \\
 & & \quad \textbf{in } \langle \textbf{SV}\mathit{to}\textbf{EV}(\sigma_5\ \mathit{out}\textbf{Loc}(\mathit{out}\textbf{LV}(\rho_5\ \mathscr{I}[\![\texttt{return}]\!]))), \\
 & & \qquad (\mathit{deallocate}* \ (\mathit{deallocate}* \ (\mathit{deallocate}\ \sigma_5\ \mathit{loc}_4)\ \mathit{loc}_3^*)\ \mathit{loc}^*), \\
 & & \qquad o_1 \rangle
\end{array}
$$

Les fonctions sémantiques : classes et objets I

$$
\begin{array}{rcll}
\textbf{Class} & : & \mathbb{N} \to (\textbf{Dict} \oplus & \text{// recherche de méthodes} \\
 & & \quad (\Sigma \to \textbf{Oid} \otimes \Sigma) \oplus & \text{// instantiation} \\
 & & \quad (\cdot \to \textbf{Ide}^*)) & \text{// variables d'instance} \\
\textbf{ClassWrapper} & : & \textbf{Class} \to \textbf{Class} & \\
\textbf{ObjectMap} & : & \textbf{Ide} \to \textbf{Loc} & \\
\textbf{Object} & : & \mathbb{N} \to ((\textbf{Ide} \to \Sigma \to \textbf{EV}) \oplus & \text{// readfield} \\
 & & \quad (\textbf{Ide} \to \textbf{SV} \to \Sigma \to \Sigma) \oplus & \text{// writefield} \\
 & & \quad (\textbf{Ide} \to \textbf{Env} \to \textbf{Method}) \oplus & \text{// recherche de méthodes} \\
 & & \quad (\textbf{Ide} \to \textbf{T})) & \text{// test instanceof}
\end{array}
$$

Les fonctions sémantiques : classes et objets II

$$
\begin{array}{rcl}
\mathit{make\text{-}ClassWrapper} & = & \lambda \mathit{ide}.\lambda \mathit{ide}^*.\lambda \mathit{dict}.\lambda \mathit{super}. \\
 & & \quad \lambda n. \\
 & & \quad\quad \textbf{if } (=\ n\ 0) \textbf{ then} \\
 & & \quad\quad\quad \lambda \mathit{ide}_1.\textbf{let}* \ \mathit{found} = (\mathit{dict}\ \mathit{ide}_1) \\
 & & \quad\quad\quad\quad \textbf{in if } \mathit{is}\textbf{U}(\mathit{found}) \textbf{ then } ((\mathit{super}\ 0)\ \mathit{ide}_1) \\
 & & \quad\quad\quad\quad\quad \textbf{else } \mathit{out}\textbf{Method}(\mathit{found}) \\
 & & \quad\quad \textbf{else if } (=\ n\ 1) \textbf{ then} \\
 & & \quad\quad\quad \lambda.(\mathit{append}\ \mathit{ide}^*\ (\mathit{super}\ 1)) \\
 & & \quad\quad \textbf{else if } (=\ n\ 2) \textbf{ then} \\
 & & \quad\quad\quad \lambda\sigma.(\mathit{make\text{-}object}\ \mathit{ide}\ (\mathit{append}\ \mathit{ide}^*\ (\mathit{super}\ 1))\ \sigma) \\
 & & \quad\quad \textbf{else } \mathit{erreur}
\end{array}
$$

Les fonctions sémantiques : la classe Object

$$\begin{aligned} \mathit{theObjectClass} \quad = \quad & \lambda n.\textbf{if } (= \ n\,0) \textbf{ then } \mathit{emptyDict} \\ & \textbf{else if } (= \ n\,1) \textbf{ then } \lambda.\langle \mathit{nil}, 0\rangle \\ & \textbf{else } \mathit{erreur} \end{aligned}$$

Les fonctions sémantiques : générateurs d'objets I

$$\begin{aligned} \mathit{make\text{-}ObjectGen} \quad = \quad & \lambda \mathit{ide}.\lambda \mathit{ide}^*.\lambda\sigma. \\ & \textbf{let } (\mathit{omap}, \sigma_1) = (\mathit{make\text{-}ObjectMap}\ \mathit{ide}^*\ \sigma) \\ & \textbf{in } \langle \lambda \mathit{self}.\lambda n. \\ & \qquad \textbf{if } (= \ n\,0) \textbf{ then} \\ & \qquad\quad \lambda \mathit{ide}.\lambda\sigma.\textbf{let } \mathit{loc} = (\mathit{omap}\ \mathit{ide}) \\ & \qquad\qquad\quad \textbf{in if } \mathit{is}\textbf{Address}(\mathit{loc}) \textbf{ then } \textbf{SV}\mathit{to}\textbf{EV}(\sigma\ \mathit{loc}) \\ & \qquad\qquad\quad \textbf{else } \mathit{erreur} \\ & \qquad \textbf{else if } (= \ n\,1) \textbf{ then} \\ & \qquad\quad \lambda \mathit{ide}.\lambda \mathit{sv}.\lambda\sigma. \\ & \qquad\qquad \textbf{let } \mathit{loc} = (\mathit{omap}\ \mathit{ide}) \\ & \qquad\qquad \textbf{in if } \mathit{is}\textbf{Address}(\mathit{loc}) \textbf{ then } (((\mathit{updateStore}\ \sigma)\ \mathit{loc})\ \mathit{sv}) \\ & \qquad\qquad\quad \textbf{else } \mathit{erreur} \end{aligned}$$

Les fonctions sémantiques : générateurs d'objets II

$$\begin{aligned} & \textbf{else if } (= \ n\,2) \textbf{ then} \\ & \quad \lambda \mathit{ide}_1.\lambda\rho.(((\mathit{out}\textbf{Class}(\rho\ \mathit{ide})\ 0)\ \mathit{ide}_1)\ \mathit{self}) \\ & \textbf{else if } (= \ n\,3) \textbf{ then} \\ & \quad \lambda \mathit{ide}_1.\mathit{in}\textbf{EV}_1(\mathit{in}\textbf{RV}_1(\mathit{in}\textbf{V}_1(= \mathit{ide}\ \mathit{ide}_1))) \\ & \textbf{else } \mathit{erreur}, \\ & \sigma_1\rangle \end{aligned}$$

Les fonctions sémantiques : création d'objets

$$\begin{aligned} \mathit{make\text{-}Object} \quad = \quad & \lambda \mathit{ide}.\lambda \mathit{ide}^*.\lambda\sigma. \\ & \textbf{let*}\ \langle \mathit{og}, \sigma_1\rangle = (\mathit{make\text{-}ObjectGen}\ \mathit{ide}\ \mathit{ide}^*\ \sigma) \\ & \textbf{and } o = (\textbf{fix } \mathit{og}) \\ & \textbf{and } \langle \sigma_2, \mathit{loc}\rangle = (\mathit{allocate}\ \sigma_1) \\ & \textbf{in } \langle \textbf{Loc}\mathit{to}\textbf{Oid}(\mathit{loc}), (((\mathit{updateStore}\ \sigma_1)\ \mathit{loc})\ \mathit{in}\textbf{SV}_2(o))\rangle \end{aligned}$$

Fonctions et types

$$
\begin{array}{rcl}
\mathscr{P} & : & \mathit{Program} \to (\Sigma \otimes \mathbf{Out}) \\
\mathscr{K} & : & \mathit{Class} \to \mathbf{Env} \to \mathbf{Env} \\
\mathscr{K}^* & : & \mathit{Class}^* \to \mathbf{Env} \to \mathbf{Env} \\
\mathscr{CE} & : & \mathit{CExp} \to \mathbf{Ide} \\
\mathscr{V} & : & \mathit{Var} \to \mathbf{Ide} \\
\mathscr{V}^* & : & \mathit{Var}^* \to \mathbf{Ide}^* \\
\mathscr{M} & : & \mathit{Method} \to \mathbf{Class} \to \mathbf{Dict} \to \mathbf{Dict} \\
\mathscr{M}^* & : & \mathit{Method}^* \to \mathbf{Class} \to \mathbf{Dict} \to \mathbf{Dict} \\
\mathscr{C} & : & \mathit{Instructions} \to \mathbf{Env} \to \Sigma \to \mathbf{Out} \to (\Sigma \otimes \mathbf{Out}) \\
\mathscr{E} & : & \mathit{Expressions} \to \mathbf{Env} \to \Sigma \to \mathbf{Out} \to (\mathbf{EV} \otimes \Sigma \otimes \mathbf{Out}) \\
\mathscr{E}^* & : & \mathit{Expressions}^* \to \mathbf{Env} \to \Sigma \to \mathbf{Out} \to (\mathbf{EV}^* \otimes \Sigma \otimes \mathbf{Out}) \\
\mathscr{I} & : & \mathit{Identifiers} \to \mathbf{Ide} \\
\mathscr{I}^* & : & \mathit{Identifiers}^* \to \mathbf{Ide}^* \\
\mathscr{Z} & : & \mathit{Numbers} \to \mathbf{Z}
\end{array}
$$

Équations sémantiques : programmes

$$
\begin{array}{l}
\mathscr{P}[\![\texttt{program}\ c^*\ v^*\ i]\!] = \\
\quad \textbf{let}* \ \mathit{ide}^* = \mathscr{V}^*[\![v^*]\!] \\
\quad \textbf{and}\ \rho = (((\mathit{extendEnv}\ \mathit{emptyEnv})\ \mathscr{I}[\![\texttt{Object}]\!])\ \mathit{in}\mathbf{DV}_2(\mathit{theObjectClass})) \\
\quad \textbf{and}\ (\sigma, \mathit{loc}^*) = ((\mathit{allocate}* \ \mathit{emptyStore})\ \sharp \mathit{ide}^*) \\
\quad \textbf{and}\ \rho_1 = (((\mathit{extendEnv}* \ \rho)\ \mathit{ide}^*)\ \mathit{in}\mathbf{DV}_1^*(\mathit{in}\mathbf{LV}_1^*(\mathit{loc}^*))) \\
\quad \textbf{in}\ (((\mathscr{C}[\![i]\!]\ \mathscr{K}^*[\![c^*]\!]\rho_1)\ \sigma)\ \mathit{emptyOut})
\end{array}
$$

Équations sémantiques : variables

$$
\begin{array}{l}
\mathscr{V}^*[\![v^*]\!] = \textbf{let}\ \mathit{loop} = \lambda f.\lambda v^*. \\
\qquad\qquad\qquad\qquad \textbf{if}\ \neg \mathit{null}(v^*) \\
\qquad\qquad\qquad\qquad \textbf{then}\ (\mathit{prefix}\ \mathscr{V}[\![\mathit{head}(v^*)]\!]\ (f\ (\mathit{tail}\ v^*))) \\
\qquad\qquad\qquad\qquad \textbf{else}\ \langle \mathit{nil}, 0 \rangle \\
\qquad\qquad\quad \textbf{in}\ ((\textbf{fix}\ \mathit{loop})\ v^*)
\end{array}
$$

$$
\mathscr{V}[\![\texttt{var}\ ce\ id]\!] = \mathscr{I}[\![id]\!]
$$

Équations sémantiques : les classes

$$\mathscr{K}^*[\![c^*]\!]\rho = \textbf{let } loop = \lambda f.\lambda c^*.\lambda\rho$$
$$\textbf{if } \neg null(c^*)$$
$$\textbf{then } ((f\ (tail\ c^*))\ \mathscr{K}[\![head(c^*)]\!]\rho)$$
$$\textbf{else } \rho$$
$$\textbf{in } (((\textbf{fix } loop)\ c^*)\ \rho)$$

$$\mathscr{K}[\![\texttt{class } id\ ce\ v^*\ m^*]\!]\rho =$$
$$\textbf{let}* ide = \mathscr{I}[\![id]\!]$$
$$\textbf{and } super = out\textbf{Class}(\rho\ \mathscr{CE}[\![ce]\!])$$
$$\textbf{in } (((extendEnv\ \rho)\ ide)$$
$$in\textbf{DV}_2((((make\text{-}ClassWrapper\ ide)\ \mathscr{V}^*[\![v^*]\!])\ \mathscr{M}^*[\![m^*]\!]super)\ super))$$

Équations sémantiques : les expressions de classes (types)

$$\mathscr{CE}[\![\texttt{cexp } id]\!] = \mathscr{I}[\![id]\!]$$

Équations sémantiques : les méthodes

$$\mathscr{M}^*[\![m^*]\!] = \lambda super.$$
$$\textbf{let } loop = \lambda f.\lambda m^*.\lambda d.$$
$$\textbf{if } \neg null(m^*)$$
$$\textbf{then } ((f\ tail(m^*))\ (\mathscr{M}[\![head(m^*)]\!]super\ d))$$
$$\textbf{else } d$$
$$\textbf{in } (((\textbf{fix } loop)\ m^*)\ emptyDict)$$

$$\mathscr{M}[\![\texttt{method } id\ v_1^*\ ce\ v_2^*\ i]\!] =$$
$$\lambda super.\lambda d.$$
$$\textbf{let } ide = \mathscr{I}[\![id]\!]$$
$$\textbf{in } (((extendDict\ d)\ ide)$$
$$in(\textbf{Method} \oplus \textbf{U})_1((((make\text{-}method\ \mathscr{V}^*[\![v_1^*]\!])\ \mathscr{V}^*[\![v_2^*]\!])\ i)\ super)$$

Équations sémantiques : les instructions `seq` et `assign`

$$\mathscr{C}[\![\texttt{seq } i_1\ i_2]\!]\rho\sigma o = \textbf{let } \langle\sigma_1, o_1\rangle = \mathscr{C}[\![i_1]\!]\rho\sigma o \textbf{ in } \mathscr{C}[\![i_2]\!]\rho\sigma_1 o_1$$
$$\mathscr{C}[\![\texttt{assign } id\ e]\!]\rho\sigma o = \textbf{let } \langle ev, \sigma_1, o_1\rangle = \mathscr{E}[\![e]\!]\rho\sigma o$$
$$\textbf{in } \langle((((updateStore\ \sigma_1)\ out\textbf{Loc}(out\textbf{LV}(\rho\ \mathscr{I}[\![id]\!])))\ \textbf{EV}to\textbf{SV}(ev))),$$
$$o_1\rangle$$

Équations sémantiques : l'instruction `writefield`

$$\begin{aligned}
&\mathscr{C}[\![\texttt{writefield}\ e_1\ id\ e_2]\!]\rho\sigma o = \\
&\qquad \textbf{let}* \langle ev, \sigma_1, o_1\rangle = \mathscr{E}[\![e_1]\!]\rho\sigma o \\
&\qquad \textbf{and}\ obj = out\textbf{Object}(\sigma_1\ \textbf{Oid}to\textbf{Loc}(out\textbf{Oid}(out\textbf{RV}(ev)))) \\
&\qquad \textbf{and}\ \langle ev_2, \sigma_2, o_2\rangle = \mathscr{E}[\![e_2]\!]\rho\sigma_1 o_1 \\
&\qquad \textbf{in}\ \langle(((((obj\ 1)\ \mathscr{I}[\![id]\!])\ \textbf{EV}to\textbf{SV}(ev_2))\ \sigma_2), o_2\rangle \\
&\mathscr{C}[\![\texttt{writefield self}\ id\ e]\!]\rho\sigma o = \\
&\qquad \textbf{let}* obj = out\textbf{Object}(\rho\ \mathscr{I}[\![\texttt{self}]\!]) \\
&\qquad \textbf{and}\ \langle ev_1, \sigma_1, o_1\rangle = \mathscr{E}[\![e]\!]\rho\sigma o \\
&\qquad \textbf{in}\ \langle(((((obj\ 1)\ \mathscr{I}[\![id]\!])\ \textbf{EV}to\textbf{SV}(ev))\ \sigma_1), o_1\rangle
\end{aligned}$$

Équations sémantiques : les instructions `if` et `while`

$$\begin{aligned}
&\mathscr{C}[\![\texttt{if}\ e\ i_1\ i_2]\!]\rho\sigma o = \textbf{let}\ \langle ev, \sigma_1, o_1\rangle = \mathscr{E}[\![e]\!]\rho\sigma o \\
&\qquad \textbf{in}\ \ \textbf{if}\ out\textbf{T}(out\textbf{V}(out\textbf{RV}(ev))) \\
&\qquad\quad \textbf{then}\ \mathscr{C}[\![i_1]\!]\rho\sigma_1 o_1 \\
&\qquad\quad \textbf{else}\ \mathscr{C}[\![i_2]\!]\rho\sigma_1 o_1 \\
&\mathscr{C}[\![\texttt{while}\ e\ i]\!]\rho\sigma o = \textbf{let}\ loop = \lambda f.\lambda\sigma.\lambda o. \\
&\qquad\qquad \textbf{let}\ \langle ev, \sigma_1, o_1\rangle = \mathscr{E}[\![e]\!]\rho\sigma o \\
&\qquad\qquad \textbf{in if}\ out\textbf{T}(out\textbf{V}(out\textbf{RV}(ev))) \\
&\qquad\qquad\quad \textbf{then let}\ \langle \sigma_2, o_2\rangle = \mathscr{C}[\![i]\!]\rho\sigma_1 o_1 \\
&\qquad\qquad\qquad\ \textbf{in}\ ((f\ \sigma_2)\ o_2) \\
&\qquad\qquad\quad \textbf{else}\ \langle \sigma_1, o_1\rangle \\
&\qquad \textbf{in}\ (((\textbf{fix}\ loop)\ \sigma)\ o)
\end{aligned}$$

Équations sémantiques : les instructions `return` et `writeln`

$$\begin{aligned}
&\mathscr{C}[\![\texttt{return}\ e]\!]\rho\sigma o = \textbf{let}\ \langle ev, \sigma_1, o_1\rangle = \mathscr{E}[\![e]\!]\rho\sigma o \\
&\qquad \textbf{in}\ \langle((((updateStore\ \sigma_1)\ out\textbf{Loc}(out\textbf{LV}(\rho\ \mathscr{I}[\![\texttt{return}]\!])))) \\
&\qquad\qquad \textbf{EV}to\textbf{SV}(ev)), o_1\rangle \\
&\mathscr{C}[\![\texttt{writeln}\ e]\!]\rho\sigma o = \textbf{let}\ \langle ev, \sigma_1, o_1\rangle = \mathscr{E}[\![e]\!]\rho\sigma o \\
&\qquad \textbf{in}\ \langle \sigma_1, (affix\ o_1\ \textbf{EV}\text{->string}(ev))\rangle
\end{aligned}$$

Équations sémantiques : les expressions I

$$\begin{aligned}
\mathscr{E}[\![n]\!]\rho\sigma o &= \langle in\textbf{EV}_1(in\textbf{RV}_1(in\textbf{V}_2(\mathscr{L}[\![n]\!]))), \sigma, o\rangle \\
\mathscr{E}[\![\texttt{true}]\!]\rho\sigma o &= \langle in\textbf{EV}_1(in\textbf{RV}_1(in\textbf{V}_1(true))), \sigma, o\rangle \\
\mathscr{E}[\![\texttt{false}]\!]\rho\sigma o &= \langle in\textbf{EV}_1(in\textbf{RV}_1(in\textbf{V}_1(false))), \sigma, o\rangle \\
\mathscr{E}[\![\texttt{not}\ e]\!]\rho\sigma o &= \langle in\textbf{EV}_1(in\textbf{RV}_1(in\textbf{V}_1(\neg out\textbf{T}(out\textbf{V}(out\textbf{RV}(\mathscr{E}[\![e]\!]\rho\sigma o)))))), \sigma, o\rangle \\
\mathscr{E}[\![\texttt{nil}]\!]\rho\sigma o &= \langle in\textbf{EV}_1(in\textbf{RV}_2(in\textbf{Oid}_2(nil))), \sigma, o\rangle
\end{aligned}$$

Équations sémantiques : les expressions II

$$\mathscr{E}[\![\texttt{new}\ ce]\!]\rho\sigma o = \textbf{let}\ \langle oid, \sigma_1\rangle = ((out\textbf{Class}(\rho\ \mathscr{CE}[\![ce]\!])\ 2)\ \sigma)$$
$$\textbf{in}\ \langle in\textbf{EV}_1(in\textbf{RV}_2(oid)), \sigma_1, o\rangle$$

$$\mathscr{E}[\![\texttt{instanceof}\ e\ ce]\!]\rho\sigma o = \textbf{let*}\ \langle ev_1, \sigma_1, o_1\rangle = \mathscr{E}[\![e]\!]\rho\sigma o$$
$$\textbf{and}\ obj = out\textbf{Object}(\sigma_1\ \textbf{Oid}to\textbf{Loc}(out\textbf{RV}(ev_1))))$$
$$\textbf{in}\ \langle((obj\ 3)\ \mathscr{CE}[\![ce]\!]), \sigma_1, o_1\rangle$$

Équations sémantiques : les expressions III

$$\mathscr{E}[\![\texttt{methodcall self}\ id\ e^*]\!]\rho\sigma o = \textbf{let*}\ \langle ev_1^*, \sigma_1, o_1\rangle = \mathscr{E}^*[\![e^*]\!]\rho\sigma o$$
$$\textbf{and}\ obj = out\textbf{Object}(\rho\ \mathscr{I}[\![\texttt{self}]\!])$$
$$\textbf{in}\ (((((((obj\ 2)\ \mathscr{I}[\![id]\!])\ \rho)\ \rho)\ \sigma_2)\ o_2)\ \textbf{EV}^*to\textbf{PV}^*(ev_1^*))$$

$$\mathscr{E}[\![\texttt{methodcall super}\ id\ e^*]\!]\rho\sigma o = \textbf{let*}\ \langle ev_1^*, \sigma_1, o_1\rangle = \mathscr{E}^*[\![e^*]\!]\rho\sigma o$$
$$\textbf{and}\ obj = out\textbf{Object}(\rho\ \mathscr{I}[\![\texttt{self}]\!])$$
$$\textbf{and}\ s = out\textbf{Class}(\rho\ \mathscr{I}[\![\texttt{super}]\!])$$
$$\textbf{in}\ (((((((s\ 0)\ \mathscr{I}[\![id]\!])\ obj)\ \rho)\ \sigma_2)\ o_2)\ \textbf{EV}^*to\textbf{PV}^*(ev_1^*))$$

$$\mathscr{E}[\![\texttt{methodcall}\ e\ id\ e^*]\!]\rho\sigma o = \textbf{let*}\ \langle ev_1^*, \sigma_1, o_1\rangle = \mathscr{E}^*[\![e^*]\!]\rho\sigma o$$
$$\textbf{and}\ \langle ev_2, \sigma_2, o_2\rangle = \mathscr{E}[\![e]\!]\rho\sigma_1 o_1$$
$$\textbf{and}\ obj = out\textbf{Object}(\sigma_2\ \textbf{Oid}to\textbf{Loc}(out\textbf{Oid}(out\textbf{RV}(ev_2))$$
$$\textbf{in}\ (((((((obj\ 2)\ \mathscr{I}[\![id]\!])\ \rho)\ \rho)\ \sigma_2)\ o_2)\ \textbf{EV}^*to\textbf{PV}^*(ev_1^*))$$

Équations sémantiques : les expressions IV

$$\mathscr{E}[\![\texttt{readfield self}\ id]\!]\rho\sigma o = \textbf{let*}\ \langle ev_1, \sigma_1, o_1\rangle = \mathscr{E}[\![e]\!]\rho\sigma o$$
$$\textbf{and}\ obj = out\textbf{Object}(\rho\ \mathscr{I}[\![\texttt{self}]\!])$$
$$\textbf{in}\ \langle(((obj\ 0)\ \mathscr{I}[\![id]\!])\ \sigma_1), \sigma_1, o_1\rangle$$

$$\mathscr{E}[\![\texttt{readfield}\ e\ id]\!]\rho\sigma o = \textbf{let*}\ \langle ev_1, \sigma_1, o_1\rangle = \mathscr{E}[\![e]\!]\rho\sigma o$$
$$\textbf{and}\ obj = out\textbf{Object}(\sigma_1\ \textbf{Oid}to\textbf{Loc}(out\textbf{Oid}(out\textbf{RV}(ev_1))))$$
$$\textbf{in}\ \langle(((obj\ 0)\ \mathscr{I}[\![id]\!])\ \sigma_1), \sigma_1, o_1\rangle$$

Équations sémantiques : les expressions V

$$\mathscr{E}[\![\texttt{plus}\ e_1\ e_2]\!]\rho\sigma o = \textbf{let*}\ \langle ev_1, \sigma_1, o_1\rangle = \mathscr{E}[\![e_1]\!]\rho\sigma o$$
$$\textbf{and}\ \langle ev_2, \sigma_2, o_2\rangle = \mathscr{E}[\![e_2]\!]\rho\sigma_1 o_1$$
$$\textbf{in}\ \langle in\textbf{EV}_1(in\textbf{RV}_1(in\textbf{V}_2(out\textbf{Z}(out\textbf{V}(out\textbf{RV}(ev_1)))) +$$
$$out\textbf{Z}(out\textbf{V}(out\textbf{RV}(ev_2))))))), \sigma_2, o_2\rangle$$

$$\mathscr{E}[\![\texttt{minus}\ e_1\ e_2]\!]\rho\sigma o = \textbf{let*}\ \langle ev_1, \sigma_1, o_1\rangle = \mathscr{E}[\![e_1]\!]\rho\sigma o$$
$$\textbf{and}\ \langle ev_2, \sigma_2, o_2\rangle = \mathscr{E}[\![e_2]\!]\rho\sigma_1 o_1$$
$$\textbf{in}\ \langle in\textbf{EV}_1(in\textbf{RV}_1(in\textbf{V}_2(out\textbf{Z}(out\textbf{V}(out\textbf{RV}(ev_1)))) -$$
$$out\textbf{Z}(out\textbf{V}(out\textbf{RV}(ev_2))))))), \sigma_2, o_2\rangle$$

$$\mathscr{E}[\![\texttt{times}\ e_1\ e_2]\!]\rho\sigma o = \textbf{let*}\ \langle ev_1, \sigma_1, o_1\rangle = \mathscr{E}[\![e_1]\!]\rho\sigma o$$
$$\textbf{and}\ \langle ev_2, \sigma_2, o_2\rangle = \mathscr{E}[\![e_2]\!]\rho\sigma_1 o_1$$
$$\textbf{in}\ \langle in\textbf{EV}_1(in\textbf{RV}_1(in\textbf{V}_2(out\textbf{Z}((out\textbf{V}(out\textbf{RV}(ev_1)))) \times$$
$$out\textbf{Z}(out\textbf{V}(out\textbf{RV}(ev_2))))))), \sigma_2, o_2\rangle$$

Équations sémantiques : les expressions VI

$$\mathscr{E}[\![\texttt{equal}\ e_1\ e_2]\!]\rho\sigma o = \textbf{let}* \langle ev_1, \sigma_1, o_1\rangle = \mathscr{E}[\![e_1]\!]\rho\sigma o$$
$$\textbf{and}\ \langle ev_2, \sigma_2, o_2\rangle = \mathscr{E}[\![e_2]\!]\rho\sigma_1 o_1$$
$$\textbf{in}\ \langle in\mathbf{EV}_1(in\mathbf{RV}_1(in\mathbf{V}_1(out\mathbf{V}(out\mathbf{RV}(ev_1)) =$$
$$out\mathbf{V}(out\mathbf{RV}(ev_2))))), \sigma_2, o_2\rangle$$

$$\mathscr{E}[\![\texttt{and}\ e_1\ e_2]\!]\rho\sigma o = \textbf{let}* \langle ev_1, \sigma_1, o_1\rangle = \mathscr{E}[\![e_1]\!]\rho\sigma o$$
$$\textbf{and}\ \langle ev_2, \sigma_2, o_2\rangle = \mathscr{E}[\![e_2]\!]\rho\sigma_1 o_1$$
$$\textbf{in}\ \langle in\mathbf{EV}_1(in\mathbf{RV}_1(in\mathbf{V}_1(out\mathbf{T}(out\mathbf{V}(out\mathbf{RV}(ev_1)))\wedge$$
$$out\mathbf{T}(out\mathbf{V}(out\mathbf{RV}(ev_2))))), \sigma_2, o_2\rangle$$

Équations sémantiques : les expressions VII

$$\mathscr{E}[\![\texttt{or}\ e_1\ e_2]\!]\rho\sigma o = \textbf{let}* \langle ev_1, \sigma_1, o_1\rangle = \mathscr{E}[\![e_1]\!]\rho\sigma o$$
$$\textbf{and}\ \langle ev_2, \sigma_2, o_2\rangle = \mathscr{E}[\![e_2]\!]\rho\sigma_1 o_1$$
$$\textbf{in}\ \langle in\mathbf{EV}_1(in\mathbf{RV}_1(in\mathbf{V}_1(out\mathbf{T}(out\mathbf{V}(out\mathbf{RV}(ev_1))))\vee$$
$$out\mathbf{T}(out\mathbf{V}(out\mathbf{RV}(ev_2))))), \sigma_2, o_2\rangle$$

$$\mathscr{E}[\![\texttt{less}\ e_1\ e_2]\!]\rho\sigma o = \textbf{let}* \langle ev_1, \sigma_1, o_1\rangle = \mathscr{E}[\![e_1]\!]\rho\sigma o$$
$$\textbf{and}\ \langle ev_2, \sigma_2, o_2\rangle = \mathscr{E}[\![e_2]\!]\rho\sigma_1 o_1$$
$$\textbf{in}\ \langle in\mathbf{EV}_1(in\mathbf{RV}_1(in\mathbf{V}_1(out\mathbf{Z}(out\mathbf{V}(out\mathbf{RV}(ev_1))) <$$
$$out\mathbf{Z}(out\mathbf{V}(out\mathbf{RV}(ev_2))))), \sigma_2, o_2\rangle$$

Équations sémantiques : les listes d'expressions

$$\mathscr{E}^*[\![e^*]\!]\rho\sigma o = \textbf{let}\ loop = \textbf{fix}(\lambda f.\lambda e^*.\lambda\sigma.\lambda o.$$
$$\textbf{if}\ \neg(null\ e^*)\ \textbf{then}$$
$$\textbf{let}* \langle ev, \sigma_1, o_1\rangle = \mathscr{E}[\![(head\ e^*)]\!]\rho\sigma o$$
$$\textbf{and}\ \langle ev^*, \sigma_2, o_2\rangle = ((((f\ (tail\ e^*))\ \rho)\ \sigma_1)\ o_1)$$
$$\textbf{in}\ \langle (prefix\ ev\ ev^*), \sigma_2, o_2\rangle$$
$$\textbf{else}$$
$$\langle\langle nil, 0\rangle, \sigma, o\rangle)$$
$$\textbf{in}\ (((loop\ e^*)\ \sigma)\ o)$$