Les langages à objets

version du 23 novembre 2004 – 09 : 51

Les primitives du ς-calcul

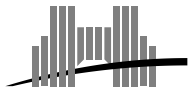
Le ς-calcul est un calcul d'objets à la manière du λ-calcul comme calcul de fonctions.

Le ς-calcul est un calcul minimaliste au même sens que le λ-calcul.

Un objet est composé de méthodes et de champs.

On accède à un champs ou on le modifie

On accède à une méthode ou on la modifie



Les primitives du ς-calcul

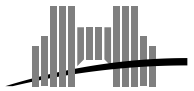
En première approximation on peut considérer les champs comme des méthodes particulières.

Un objet sera donc une collection de méthodes avec deux opérations

– invocation,

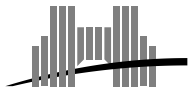
– modification,

self est une entité qui apparaît dans une méthode et représente l'objet auquel la méthode appartient.



Les primitives du ς-calcul

$\varsigma(x)b$	méthode avec comme paramètre x et comme corps b
$[l_1 = \varsigma(x_1)b_1, ..., l_n = \varsigma(x_n)b_n]$	objet contenant n méthodes $l_1, ..., l_n$
$o.l$	invocation de la méthode l de l'objet o
$o.l \Leftarrow \varsigma(x)b$	modifi cation de la méthode l de l'objet o avec remplacement par la méthode $\varsigma(x)b$



Les primitives du ς-calcul

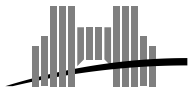
Un objet est donc composé de n composants $l_i = \varsigma(x_i)b_i$.

Les étiquettes l_i sont tous distinctes.

L'ordre des composants n'a pas d'importance.

L'invocation d'un méthode s'écrit $o.l$ où l est une étiquette de o.

Le but est d'exécuter la méthode nommée l de o auquel on attache le paramètre *self* et de retourner le résultat de cette exécution.



Les primitives du ς-calcul

Une modification de méthode est écrite $o.l \Leftarrow \varsigma(x)b$.

On produit une copie de o

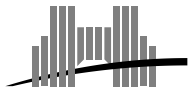
où la méthode l est remplacée par $\varsigma(x)b$.

Il s'agit d'une modification sans mutation, dite aussi fonctionnelle.

L'objet o est recopié avant qu'on lui change sa méthode l.

Les objets sont immuables.

Il n'y a donc pas de «modification en place» des objets.



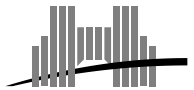
Un exemple

$$[l_1 = \varsigma(x_1)[\,], l_2 = \varsigma(x_2)x_2.l_1]$$

est un objet qui contient deux méthodes.

La première méthode, nommée l_1 retourne un objet vide $[\,]$.

La seconde méthode l_2 invoque la première méthode à travers le paramètre x_2.



La sémantique primitive

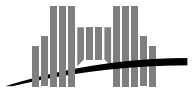
On représente les objets par $o \equiv [l_i = \varsigma(x_i)b_i \ ^{i \in 1..n}]$.

Réduction d'une invocation

$$o.l \quad \rightarrowtail \quad b_i\{\!\{x_i \leftarrow o\}\!\}$$

Réduction d'une modification

$$o.l_j \Leftarrow \varsigma(y)b \quad \rightarrowtail \quad [l_j = \varsigma(y)b, l_i = \varsigma(x_i)b_i \ ^{i \in 1..n-\{j\}}]$$



Un exemple

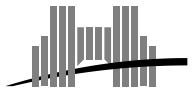
Soit l'objet

$$o_1 \triangleq [l = \varsigma(x)[\,]]$$

alors

$$o_1.l \rightarrowtail [\,]\{\!\{x \leftarrow o_1\}\!\} \equiv [\,]$$

$$o_1.l \Leftarrow \varsigma(y)o_1 \rightarrowtail [l = \varsigma(y)o_1]$$



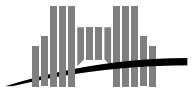
On peut facilement écrire des programmes qui ne terminent pas sans utiliser explicitement la récursion.

Soit

$$o_2 \triangleq [l = \varsigma(x)x.l]$$

alors

$$o_2.l \rightarrowtail x.l\{\!\{x \leftarrow o_2\}\!\} \equiv o_2.l \rightarrowtail \ldots$$



La substitution de «self» permet à une méthode de retourner ou de modifier «self»

c'est-à-dire de modifier ou de retourner l'objet lui-même.

Soit

$$o_3 \triangleq [l = \varsigma(x)x]$$

alors

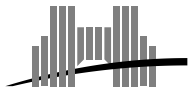
$$o_3.l \rightarrowtail x\{\!\{x \leftarrow o_3\}\!\} \equiv o_3$$

Soit

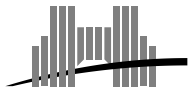
$$o_4 \triangleq [l = \varsigma(y)(y.l \Leftarrow \varsigma(x)x)]$$

alors

$$o_4.l \rightarrowtail (o_4.l \Leftarrow \varsigma(x)x) \rightarrowtail o_3$$

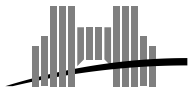


Cette faculté de l’objet de se modifier lui-même est une caractéristique des langages à prototype.



La syntaxe

$$a, b \quad ::= \quad x \mid [l_i = \varsigma(x_i)b_i \; ^{i \in 1..n}] \mid a.l \mid a.l \Leftarrow \varsigma(x)b$$



Notations

Si dans une méthode $\varsigma(x)b$, b ne contient pas x, on peut considérer qu'il s'agit d'un champ.

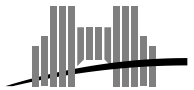
Si $x \notin FV(b)$, on écrit $o.l := b$ à la place de $o.l \Leftarrow \varsigma(x)b$.

Dans les objets, on écrira

$$[..., l = o, ...]$$

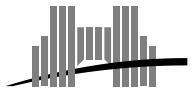
au lieu de

$$[..., l = \varsigma(s)o, ...].$$



Variables libres

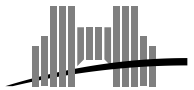
$$
\begin{aligned}
FV(\varsigma(y)b) &\triangleq FV(b) - \{y\} \\
FV(x) &\triangleq \{x\} \\
FV([l_i = \varsigma(x_i)b_i\ ^{i\in 1..n}]) &\triangleq \bigcup_{i\in 1..n} FV(\varsigma(x_i)b_i) \\
FV(a.l) &\triangleq FV(a) \\
FV(a.l \Leftarrow \varsigma(y)b) &\triangleq FV(a) \cup FV(\varsigma(y)b)
\end{aligned}
$$



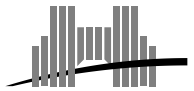
Substitution dans les objets

On applique la convention de Barendregt.

$$
\begin{array}{rcl}
(\varsigma(y)b)\{\!\{x \leftarrow c\}\!\} & \triangleq & \varsigma(y)(b\{\!\{x \leftarrow c\}\!\}) \\
x\{\!\{x \leftarrow c\}\!\} & \triangleq & c \\
y\{\!\{x \leftarrow c\}\!\} & \triangleq & y \\
[l_i = \varsigma(x_i)b_i\ ^{i\in 1..n}]\{\!\{x \leftarrow c\}\!\} & \triangleq & [l_i = \varsigma(x_i)b_i\{\!\{x \leftarrow c\}\!\}\ ^{i\in 1..n}] \\
(a.l)\{\!\{x \leftarrow c\}\!\} & \triangleq & (a\{\!\{x \leftarrow c\}\!\}).l \\
(a.l \Leftarrow \varsigma(y)b)\{\!\{x \leftarrow c\}\!\} & \triangleq & (a\{\!\{x \leftarrow c\}\!\}).l \Leftarrow ((\varsigma(y)b)\{\!\{x \leftarrow c\}\!\})
\end{array}
$$



Exemples

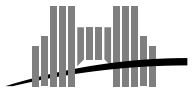


Traduction du λ-calcul

Le ς-calcul est au moins aussi puissant que le λ-calcul !

On pose

$$p \bullet q \quad \triangleq \quad (p.arg := q).val$$



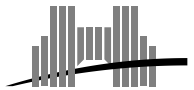
alors

$\lambda -$ termes objets pures

$$
\begin{array}{rcl}
\langle\!\langle x \rangle\!\rangle & \triangleq & x \\
\langle\!\langle b(a) \rangle\!\rangle & \triangleq & \langle\!\langle b \rangle\!\rangle \bullet \langle\!\langle a \rangle\!\rangle \\
\langle\!\langle \lambda x.b \rangle\!\rangle & \triangleq & [arg = \varsigma(x)x.arg, \quad val = \varsigma(x)\langle\!\langle b \rangle\!\rangle\{\!\{x \leftarrow x.arg\}\!\}]
\end{array}
$$

L'idée est que l'application $\langle\!\langle b(a) \rangle\!\rangle$ stocke l'argument dans une place connue (le champ arg).

Puis elle invoque une méthode de $\langle\!\langle b \rangle\!\rangle$ qui peut accéder à l'argument à travers self.

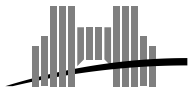


Par exemple,

$$\begin{aligned}\langle\!\langle(\lambda x.x)y\rangle\!\rangle &\equiv ([arg = \varsigma(x)x.arg, \quad val = \varsigma(x)x.arg].arg := y).val \\ &= y\end{aligned}$$

La valeur initiale de l'argument arg n'a pas d'importance.

Ici on prend une valeur qui conduit à une boucle si on tente de l'«évaluer». Ce qui correspond à une valeur non initialisée.

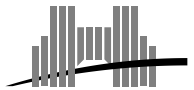


Soit

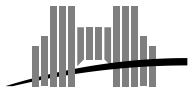
$$o \quad \equiv \quad [arg = \langle\!\langle a \rangle\!\rangle, \quad val = \varsigma(x)\langle\!\langle b \rangle\!\rangle\{\!\{x \leftarrow x.arg\}\!\}]$$

alors

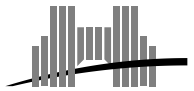
$$\begin{aligned}
\langle\!\langle (\lambda x.b)\ a \rangle\!\rangle \quad &\equiv \quad ([arg = \varsigma(x)x.arg, \\
&\qquad val = \varsigma(x)\langle\!\langle b \rangle\!\rangle\{\!\{x \leftarrow x.arg\}\!\}].arg := \langle\!\langle a \rangle\!\rangle).val \\
&= \quad o.val \\
&= \quad (\langle\!\langle b \rangle\!\rangle\{\!\{x \leftarrow x.arg\}\!\})\{\!\{x \leftarrow o\}\!\} \\
&= \quad \langle\!\langle b \rangle\!\rangle\{\!\{x \leftarrow o.arg\}\!\} \\
&= \quad \langle\!\langle b \rangle\!\rangle\{\!\{x \leftarrow \langle\!\langle a \rangle\!\rangle\}\!\}
\end{aligned}$$



La η-conversion n’est pas préservée, parce qu’il y a des objets qui ne représentent pas des λ-termes.



Point fixe

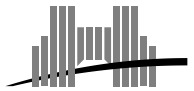


Point fixe

Comme on peut coder le λ-calcul on peut obtenir une version objet de tout ce que l'on peut coder dans le λ-calcul.

On peut cependant obtenir de meilleure version en les encodant directement dans le ς-calcul.

C'est le cas du point fixe.



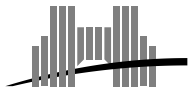
Point fixe

$$\mathsf{fix} \triangleq [arg = \varsigma(x)x.arg, \quad val = \varsigma(x)((x.arg).arg := x.val).val]$$

Vérifions.

D'abord posons

$$\begin{aligned} \mathsf{fix}_f &\triangleq \mathsf{fix}.arg := f \\ &= [arg = f, \quad val = \varsigma(x)((x.arg).arg := x.val).val] \end{aligned}$$

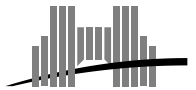


alors

$$\begin{aligned}
\mathsf{fix} \bullet f &\equiv \mathsf{fix}_f.val \\
&= ((\mathsf{fix}_f.arg).arg := \mathsf{fix}_f.val).val \\
&= (f.arg := \mathsf{fix} \bullet f).val \\
&\equiv f \bullet (\mathsf{fix} \bullet f)
\end{aligned}$$

La traduction de $\lambda x.xx$ est

$$\begin{aligned}
\langle\!\langle \lambda x.xx \rangle\!\rangle \equiv\ & [arg = \varsigma(x)x.arg, \\
& val = \varsigma(x)((x.arg).arg := x.arg).val]
\end{aligned}$$



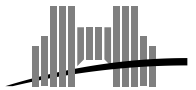
Le point fixe de Fisher, Honsell et Mitchell

Fisher, Honsell et Mitchell ont proposé

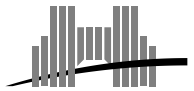
$$\mathsf{fix}' \triangleq \lambda f[rec = \varsigma(s) f(s.rec)].rec.$$

On peut le traduire de façon proche mais pas identique :

$$\begin{aligned}
\langle\!\langle \mathsf{fi\,x}' \rangle\!\rangle &= [arg = \varsigma(x)x.arg, \\
&\quad val = \varsigma(f)[rec = \varsigma(s) f \bullet (s.rec)].rec\{\!\{f \leftarrow f.arg\}\!\}] \\
&= [arg = \varsigma(x)x.arg, \\
&\quad val = \varsigma(f)[rec = \varsigma(s)(f.arg := s.rec).val].rec\{\!\{f \leftarrow f.arg\}\!\}] \\
&= [arg = \varsigma(x)x.arg, \\
&\quad val = \varsigma(f)[rec = \varsigma(s)((f.arg).arg := s.rec).val].rec]
\end{aligned}$$



Le point géométrique

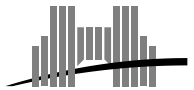


L'objet point

Un grand classique de la théorie des langages à objets est le point géométrique qui contient un méthode move qui déplace le point c'est-à-dire qui modifie les coordonnées du point.

$$origine_1 \triangleq [x = 0, mv\_x = \varsigma(s)\lambda\, dx(s.x := s.x + dx)]$$

$$origine_2 \triangleq [x = 0, y = 0, mv\_x = \varsigma(s)\lambda\, dx(s.x := s.x + dx), mv\_y = \varsigma(s)\lambda\, dy(s.y := s.y + dy)]$$

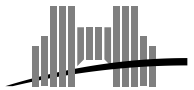


On peut définir

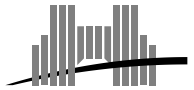
$$unit_2 \triangleq origin_2.mv\_x(1).mv\_y(1)$$

On peut montrer que l'abcisse de $unit_2$ est 1, autrement dit que

$$unit_2.x \quad = \quad 1.$$



Méthodes archivantes



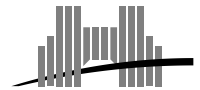
Auto-archive

Une illustration simple de la technique qui permet à un objet de se stocker soi-même.

Un objet peut garder des copies de lui-même.

Cet objet a

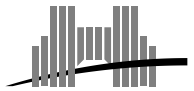
- une méthode d'archivage backup,
- et une méthode de restauration retrieve qui retourne la dernière archive.



$$
\begin{array}{rcl}
o & \triangleq & [retrieve = \varsigma(s_1)s_1, \\
 & & backup = \varsigma(s_2)s_2.retrieve \Leftarrow \varsigma(s_1)s_2, \\
 & & \text{et plein d'autres choses }].
\end{array}
$$

On a donc

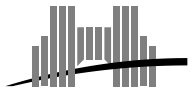
$$
\begin{array}{rcl}
o' & \triangleq & o.backup \\
 & = & [retrieve = \sigma(s_1)o, ...]
\end{array}
$$



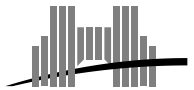
Plus tard après avoir modifier des attributs de o' on peut restaurer l'objet qui a été archivé et l'on retrouve l'objet qui était l'objet courant au moment de l"archivage.

$$o'.retrieve \quad = \quad o$$

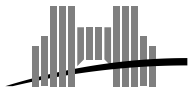
On peut itérer des invocations de la méthode retrieve pour retrouver des objets de plus en plus anciennement archivés, jusqu'à converger finalement vers l'objet initial.



Cellules



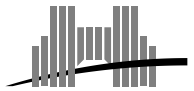
$$
\begin{aligned}
myCell \quad \triangleq \quad & [contents = 0, \\
& \quad get = \varsigma(s)s.contents, \\
& \quad set = \varsigma(s)\lambda\, n\, (s.contents := n)]
\end{aligned}
$$



Une cellule à restauration

- une méthode backup
- et une méthode restore pour restaurer l'objet en copiant l'archive dans le contenu.

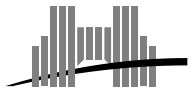
$$
\begin{aligned}
myReCell \quad \triangleq \quad & [contents = 0, \\
& \quad get = \varsigma(s)s.contents, \\
& \quad set = \varsigma(s)\lambda\, n\, (s.backup := s.contents).contents := n, \\
& \quad bakcup = 0, \\
& \quad restore = \varsigma(s)s.contents := s.backup]
\end{aligned}
$$



On peut définir une cellule avec restauration sans champ backup, en s'appuyant sur la modification de méthode.

C'est la méthode set qui est en charge de mettre à jour la méthode restore.

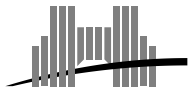
Chaque fois que set est invoquée elle modifie restore en $\varsigma(z)z.contents := m$ où m est le contenue de la cellule avant l'invocation.



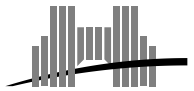
$$
\begin{aligned}
myOtherReCell \quad &\triangleq \\
&[contents = 0, \\
&\quad get = \varsigma(s)s.contents, \\
&\quad set = \varsigma(s)\lambda\, n\, (s.restore \Leftarrow \varsigma(z)(z.contents := s.contents).contents := n), \\
&\quad restore = \varsigma(s)s.contents := 0]
\end{aligned}
$$

Ceci montre qu'il est primordial que les self soient des paramètres nommés.

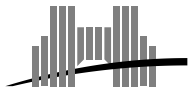
Fournir un mot-clé **self** comme cela est fait dans beaucoup de langages de programmation par objet n'est pas suffisant.



Un système de types



Typage des objets

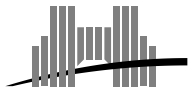


$$\frac{\Gamma \vdash B_i \qquad \forall i \in 1..n}{\Gamma \vdash [l_i : B_i \ ^{i \in 1..n}]}$$

les l_i sont distincts $\quad Type\ Objet$

$$\frac{\Gamma, x_i : A \vdash b_i : B_i \qquad \forall i \in 1..n}{\Gamma \vdash [l_i = \varsigma(x_i : A) b_i \ ^{i \in 1..n}] : A}$$

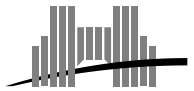
où $A \equiv [l_i : B_i \ ^{i \in 1..n}] \quad Val\ Objet$



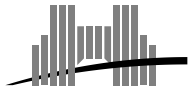
Pour $j \in 1..n$,

$$\frac{\Gamma \vdash a : [l_i : B_i \,^{i \in 1..n}]}{\Gamma \vdash a.l_j : B_j} \quad ValSelect$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma, x : A \vdash b : B_j}{\Gamma \vdash a.l_j \Leftarrow \varsigma(x : A)b : A} \quad \text{où } A \equiv [l_i : B_i \,^{i \in 1..n}] \quad TypeObjet$$



Bonne formation des types

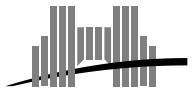


$\Gamma \vdash \diamond$ doit se lire «le contexte Γ est cohérent».

$$\frac{}{\emptyset \vdash \diamond} \; Env\ \emptyset$$

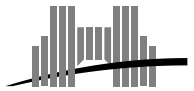
$$\frac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x : A \vdash \diamond} \; Env\ x$$

$$\frac{\Gamma', x : A, \Gamma'' \vdash \diamond}{\Gamma', x : A, \Gamma'' \vdash x : A} \; Env\ x$$



Tout cela se lit :

- le contexte vide est cohérent,
- si A est un type bien formé dans Γ, alors le contexte $\Gamma, x : A$ est cohérent,
- si le contexte $\Gamma', x : A, \Gamma''$ est cohérent, alors x est de type A dans ce contexte.



Unicité du type

Proposition : Si $\Gamma \vdash a : A$ et $\Gamma \vdash a : A'$ sont dérivables

alors $A \equiv A'$.

Ce système de type ne garantit pas la forte normalisation.

Il garantit seulement la correction.

On démontre aussi une propriété de réduction du sujet.