Initiation à la sémantique (1)

Solange Coupet-Grimal

April 1, 2010

Présentation informelle

Alonzo Church (Princeton, 1935-1940)
Paradigme de la programmation fonctionnelle

- **Termes**
  - λx.M représente *fun* $x \rightarrow M$
  - (M N) représente *l'application de M à N*

Présentation informelle

Alonzo Church (Princeton, 1935-1940)
Paradigme de la programmation fonctionnelle

- **Termes**
  - λx.M représente *fun* $x \rightarrow M$
  - (M N) représente *l'application de M à N*
- **Types**
  - $M : t$
  - Propriétés des termes - contraintes sur leur utilisation
    Exemple $M: s \rightarrow t$ ne peut s'appliquer qu'à $N : s$
    Dans ce cas $(M\ N) : t$

Présentation informelle

Alonzo Church (Princeton, 1935-1940)
Paradigme de la programmation fonctionnelle

- **Termes**
  - λx.M représente *fun* $x \rightarrow M$
  - (M N) représente *l'application de M à N*
- **Types**
  - $M : t$
  - Propriétés des termes - contraintes sur leur utilisation
    Exemple $M: s \rightarrow t$ ne peut s'appliquer qu'à $N : s$
    Dans ce cas $(M\ N) : t$
- **Règles de calcul**
  - (λx.M N) se convertit en $M[x := N]$ (avec précaution)

Syntaxe

- **Types** `type ::= o | type→type`
  - Exemples : $o \rightarrow (o \rightarrow o)$
  - Opérateur $\rightarrow$: association par la droite

    $o \rightarrow o \rightarrow o$ pour $o \rightarrow (o \rightarrow o)$

Syntaxe

- **Types** $\texttt{type ::= o | type}\rightarrow\texttt{type}$
  - Exemples : $o \rightarrow (o \rightarrow o)$
  - Opérateur $\rightarrow$: association par la droite
    $$o \rightarrow o \rightarrow o \quad \text{pour} \quad o \rightarrow (o \rightarrow o)$$
- **Termes** $\texttt{terme::= x} \mid \lambda \texttt{x}_{:type}\texttt{.terme} \mid \texttt{(terme terme)}$
  - Opérateur d'*application*: association par la gauche
    $$M\,N\,P \quad \text{pour} \quad ((M\,N)\,P)$$

Syntaxe

- **Types** $\texttt{type ::= o | type}\rightarrow\texttt{type}$
  - Exemples : $o \rightarrow (o \rightarrow o)$
  - Opérateur $\rightarrow$: association par la droite
    $o \rightarrow o \rightarrow o$ pour $o \rightarrow (o \rightarrow o)$
- **Termes** $\texttt{terme::= x} \mid \lambda \texttt{x}_{:type}\texttt{.terme} \mid \texttt{(terme terme)}$
  - Opérateur d'*application*: association par la gauche
    $M\,N\,P$ pour $((M\,N)\,P)$
- **Exemple courant**

$$M = \lambda f_{:o\rightarrow o\rightarrow o}.\lambda x_{:o}.\lambda y_{:o}.\,((f\ y)\ x)$$

cf. Caml : avec la déclaration $type\ o = int$

$let\ M = fun(f : o \rightarrow o \rightarrow o) \rightarrow fun(x : o) \rightarrow fun(y : o) \rightarrow (f\ y)\ x$

$let\ M\ (f : o \rightarrow o \rightarrow o)\ x\ y = (f\ y)\ x$

Syntaxe

- **Types** $\texttt{type ::= o | type}\rightarrow\texttt{type}$
  - Exemples : $o \rightarrow (o \rightarrow o)$
  - Opérateur $\rightarrow$: association par la droite
    $o \rightarrow o \rightarrow o$ pour $o \rightarrow (o \rightarrow o)$
- **Termes** $\texttt{terme::= x} \mid \lambda \texttt{x}_{:type}\texttt{.terme} \mid \texttt{(terme terme)}$
  - Opérateur d'*application*: association par la gauche
    $M\ N\ P$ pour $((M\ N)\ P)$
- **Exemple courant**

$$M = \lambda f_{:o\rightarrow o\rightarrow o}.\lambda x_{:o}.\lambda y_{:o}.\ ((f\ y)\ x)$$

cf. Caml : avec la déclaration $type\ o = int$

$let\ M = fun(f : o \rightarrow o \rightarrow o) \rightarrow fun(x : o) \rightarrow fun(y : o) \rightarrow (f\ y)\ x$

$let\ M\ (f : o \rightarrow o \rightarrow o)\ x\ y = (f\ y)\ x$

Quelle est la fonction M?

Syntaxe

$$M = \lambda f_{:o\rightarrow o\rightarrow o}.\lambda x_{:o}.\lambda y_{:o}.\,((f\ y)\ x)$$

Sous-termes de M

$$x,\quad y,\quad f,\quad (f\ y),\quad ((f\ y)\ x),\quad \lambda y_{:o}.\,((f\ y)\ x),$$
$$\lambda x_{:o}.\lambda y_{:o}.\,((f\ y)\ x),\quad \lambda f_{:o\rightarrow o\rightarrow o}.\lambda x_{:o}.\lambda y_{:o}.\,((f\ y)\ x)$$

Variables libres et liées

- x est *libre* dans le terme x
- occurrence *libre* de x dans M ou $N \rightsquigarrow$ *libre* dans $(M\ N)$
- occurrence *libre* de x dans $M \rightsquigarrow$ *liée* dans $\lambda x_{:t}.M$ par λx
- occurrence *libre* de y dans $M \rightsquigarrow$ *libre* dans $\lambda x_{:t}.M$ si $x \neq y$

FV(M): ensemble des variables libres de M

$M = \lambda f_{:o\to o\to o}.\ \lambda x_{:o}.\ \lambda y_{:o}.\ ((f\ y)\ x \qquad FV(M) = \emptyset \qquad (clos)$
$N = \lambda x_{:o}.\ \lambda y_{:o}.\ ((f\ y)\ x) \qquad FV(N) = \{f\} \qquad (non\ clos)$

cf. Caml

$let\ M\ f\ x\ y\ =\ (f\ y)\ x \qquad FV(M) = \emptyset \qquad (M clos)$
$let\ N\ x\ y\ =\ (f\ y)\ x \qquad FV(N) = \{f\} \qquad (N\ non\ clos)$

α-Equivalence sur les termes

Egalité au nom des variables liées près

$$M = \lambda f_{:o\to o\to o}.\, \lambda x_{:o}.\, \lambda y_{:o}.\, ((f\ y)\ x)$$

$$M \sim_\alpha \lambda g_{:o\to o\to o}.\, \lambda a_{:o}.\, \lambda b_{:o}.\, ((g\ b)\ a)$$

cf. Caml

$$\begin{array}{l} let\ m\ x\ y = (x + .y)/.2. \\ let\ n\ a\ b\ = (a + .b)/.2. \end{array} \quad \rightsquigarrow m \sim_\alpha n$$

$$\begin{array}{l} let\ M\ f\ x\ y = (f\ y)\ x \\ let\ N\ g\ \ a\ b\ = (g\ b)\ a \end{array} \quad \rightsquigarrow M \sim_\alpha N$$

Typage

- Terme valide
  - Syntaxiquement correct
  - Typable
    - un type peut lui être attribué selon certaines règles
      $M = \lambda f_{:o\rightarrow o\rightarrow o}.\lambda x :_o .\lambda y_{:o}.(f\ y)\ x$
      $M\ :\ (o \rightarrow o \rightarrow o) \rightarrow o \rightarrow o \rightarrow o$ (cf. Caml)
    - si le type des variables libres est donné:
      $N = \lambda x_{:o}.\lambda y_{:o}.\ ((f\ y)\ x) : o \rightarrow o \rightarrow o$
      si l'on supose que $f\ :\ o \rightarrow o \rightarrow o$

Typage

- Terme valide
  - Syntaxiquement correct
  - Typable
    - un type peut lui être attribué selon certaines règles
      $M = \lambda f_{:o\rightarrow o\rightarrow o}.\lambda x :_o .\lambda y_{:o}.(f\ y)\ x$
      $M\ :\ (o \rightarrow o \rightarrow o) \rightarrow o \rightarrow o \rightarrow o$ (cf. Caml)
    - si le type des variables libres est donné:
      $N = \lambda x_{:o}.\lambda y_{:o}.\ ((f\ y)\ x) : o \rightarrow o \rightarrow o$
      si l'on supose que $f\ :\ o \rightarrow o \rightarrow o$
- Affectation de type : $H = \{x_1 : t_1; \ldots; x_n : t_n\}$
  - x_i: variables distinctes
  - Notation: $H(x_i) = t_i$ ou $x_i : t_i \in H$

Typage

- Terme valide
  - Syntaxiquement correct
  - Typable
    - un type peut lui être attribué selon certaines règles
      $M = \lambda f_{:o\rightarrow o\rightarrow o}.\lambda x :_o .\lambda y_{:o}.(f\ y)\ x$
      $M\ :\ (o \rightarrow o \rightarrow o) \rightarrow o \rightarrow o \rightarrow o$ (cf. Caml)
    - si le type des variables libres est donné:
      $N = \lambda x_{:o}.\lambda y_{:o}.\ ((f\ y)\ x) : o \rightarrow o \rightarrow o$
      si l'on supose que $f\ :\ o \rightarrow o \rightarrow o$
- Affectation de type : $H = \{x_1 : t_1; \ldots; x_n : t_n\}$
  - x_i: variables distinctes
  - Notation: $H(x_i) = t_i$ ou $x_i : t_i \in H$
- Jugement:
  $\{f : o \rightarrow o \rightarrow o\} \vdash N : o \rightarrow o \rightarrow o$

Règles de typage

- $H \cup \{x : t\} \vdash x : t$ (1) *projection*

Règles de typage

- $H \cup \{x : t\} \vdash x : t$ (1) *projection*
- $\dfrac{H \vdash M : s \rightarrow t \quad H \vdash N : s}{H \vdash (M\ N) : t}$ (2) *application*

Règles de typage

- $H \cup \{x : t\} \vdash x : t$ (1) *projection*
- $\dfrac{H \vdash M : s \rightarrow t \quad H \vdash N : s}{H \vdash (M\,N) : t}$ (2) *application*
- $\dfrac{H \cup \{x : s\} \vdash M : t}{H \vdash \lambda x_{:s}.M : s \rightarrow t}$ (3) *abstraction*

Exemple d'inférence de type

$N = \lambda x_{:o}.\lambda y_{:o}.((f\ y)\ x)$ $\qquad$ $H = \{f : o \to o \to o\ , x : o\ , y : o\}$

Etablir $\qquad$ $H \vdash N : o \to o \to o$

$$\dfrac{\dfrac{}{H \vdash f : o \to o \to o}\ proj \qquad \dfrac{}{H \vdash y : o}\ proj}{\dfrac{\dfrac{H \vdash (f\ y) : o \to o}{}\ app \qquad \dfrac{}{H \vdash x : o}\ proj}{H \vdash ((f\ y)\ x) : o}\ appl}$$

$$\dfrac{\dfrac{}{\{f : o \to o \to o, x : o\} \vdash \lambda y_{:o}.((f\ y)\ x) : o \to o}\ abs}{\{f : o \to o \to o\} \vdash \lambda x_{:o}.\lambda y_{:o}.((f\ y)\ x) : o \to o \to o}\ abs$$

le système $\lambda^{\rightarrow}$

- Un terme non clos n'est pas complètement défini:
  on ne connait pas le type de ses variables libres

le système $\lambda^{\rightarrow}$

- Un terme non clos n'est pas complètement défini:
  on ne connait pas le type de ses variables libres

- Les objets du calcul sont donc les jugements:

$$\boxed{H \vdash M : t}$$

Triplet : affectation H (type des FV) + terme + type

le système $\lambda^{\rightarrow}$

- Un terme non clos n'est pas complètement défini:
  on ne connait pas le type de ses variables libres

- Les objets du calcul sont donc les jugements:

$$\boxed{H \vdash M : t}$$

Triplet : affectation H (type des FV) + terme + type

- On a ainsi défini un système de calcul $\lambda^{\rightarrow}$:
  - la syntaxe des types et des termes
  - les règles de typage

β-conversion

Réduction des β-*redex* ou β-*radicaux*

β-redex: expressions de la forme $(\lambda x_{:s}.M)\ N$

$$\boxed{(\lambda x_{:s}.M)\ N \rightarrow_\beta M[x := N]}$$

Signification de M[x := N]

- M dans lequel les occurrences libres de x sont remplacées par N
- si N contient des variables libres qui se retrouvent liées après substitution (capture) $\rightsquigarrow$ α-conversions dans M

β-conversion

$$\boxed{(\lambda x_{:s}.M)\ N \rightarrow_\beta M[x := N]}$$

Analogies avec la programmation

(fun x $\rightarrow$ x*x+y)10 $\rightarrow_\beta$ (x*x+y) [x:=10] = 10*10+y

β-conversion

$$\boxed{(\lambda x_{:s}.M)\ N \rightarrow_\beta M[x := N]}$$

Analogies avec la programmation
(captures)

(fun x y $\rightarrow$ x*x+y)(2*y) $\rightarrow_\beta$ (fun y $\rightarrow$ x*x+y) [x:=2*y]

~~(2*y) *(2*y) +y~~

*(fun y $\rightarrow$ x*x+y)* $\sim_\alpha$ *(fun z $\rightarrow$ x*x+z)*

(fun z $\rightarrow$ x*x+z) [x:=2*y] $\equiv$ fun z $\rightarrow$ (2*y)*(2*y)+z

β-conversion

- $N = \lambda i_{:o}.\lambda j_{:o}.y\ i\ j$: y libre dans N
  N typable pour $H = \{y : o \rightarrow o \rightarrow o\}$

β-conversion

- $N = \lambda i_{:o}.\lambda j_{:o}.y\ i\ j$: y libre dans N
  N typable pour $H = \{y : o \rightarrow o \rightarrow o\}$
- $M = (\lambda x_{:o}.\lambda y_{:o}.\ \mathrm{f}\ y\ x)$: y liée dans M

β-conversion

- $N = \lambda i_{:o}.\lambda j_{:o}.y\ i\ j$: y libre dans N
  N typable pour $H = \{y : o \rightarrow o \rightarrow o\}$
- $M = (\lambda x_{:o}.\lambda y_{:o}.\ \mathrm{f}\ y\ x)$: y liée dans M
- pas de rapport entre y dans M et y dans N (même leurs types diffèrent!)

β-conversion

- $N = \lambda i_{:o}.\lambda j_{:o}.y\ i\ j$: y libre dans N
  N typable pour $H = \{y : o \rightarrow o \rightarrow o\}$
- $M = (\lambda x_{:o}.\lambda y_{:o}.\ \mathrm{f}\ y\ x)$: y liée dans M
- pas de rapport entre y dans M et y dans N (même leurs types diffèrent!)

- $\boxed{(\lambda f_{o\rightarrow o\rightarrow o}.M)\ N \quad \rightarrow_\beta \quad M[f := N]}$

β-conversion

- $N = \lambda i_{:o}.\lambda j_{:o}.y\ i\ j$: y libre dans N
  N typable pour $H = \{y : o \rightarrow o \rightarrow o\}$
- $M = (\lambda x_{:o}.\lambda y_{:o}.\ \mathrm{f}\ y\ x)$: y liée dans M
- pas de rapport entre y dans M et y dans N (même leurs types diffèrent!)

- $\boxed{(\lambda f_{o\rightarrow o\rightarrow o}.M)\ N \quad \rightarrow_\beta \quad M[f := N]}$

- Substitution sans précaution de f par N dans M, le y de N est *capturé*:
  $$(\lambda x_{:o}.\lambda y_{:o}.\underline{(\lambda i_{:o}.\lambda j_{:o}.y\ i\ j)}\ y\ x)$$
  non typable, erroné!

β-conversion

- $N = \lambda i_{:o}.\lambda j_{:o}.y\ i\ j$: y libre dans N
  N typable pour $H = \{y : o \rightarrow o \rightarrow o\}$
- $M = (\lambda x_{:o}.\lambda y_{:o}.\ \mathrm{f}\ y\ x)$: y liée dans M
- pas de rapport entre y dans M et y dans N (même leurs types diffèrent!)

- $\boxed{(\lambda f_{o\rightarrow o\rightarrow o}.M)\ N \quad \rightarrow_\beta \quad M[f := N]}$

- Solution: renommer y dans M par une variable *fraiche* (par ex: z)$\rightsquigarrow$ terme α-équivalent
- on obtient $(\lambda x_{:o}.\lambda z_{:o}.\underline{(\lambda i_{:o}.\lambda j_{:o}.y\ i\ j)}\ z\ x)$

β-conversion

Plus généralement :

$$M \rightarrow_\beta N$$

si N est obtenu en faisant subir une β-conversion à un β-redex, sous-terme de M.

Exemple: $(M\ ((\lambda x.E)\ A)) \rightarrow_\beta (M\ E[x := A])$

η-conversion

- η-redex : $\boxed{\lambda x_{:s}.(M\ x) \qquad x \notin FV(M)}$

η-conversion

- η-redex : $\boxed{\lambda x_{:s}.(M\ x) \qquad x \notin FV(M)}$
- η-réduction du redex : $\boxed{\lambda x_{:s}.(M\ x) \rightarrow_{\eta} M}$

η-conversion

- η-redex : $\boxed{\lambda x_{:s}.(M\ x) \qquad x \notin FV(M)}$
- η-réduction du redex : $\boxed{\lambda x_{:s}.(M\ x) \rightarrow_\eta M}$
- Généralisation

$$M \rightarrow_\eta N$$

si N est obtenu en faisant subir une η-conversion à un η-redex, sous-terme de M.

η-conversion

Exemple

$$\lambda i_{:o}.\lambda j_{:o}.f\ i\ j \quad \rightarrow_\eta \quad \lambda i_{:o}.f\ i \quad \rightarrow_\eta \quad f$$

cf. Caml

let exemple = fun i $\rightarrow$ fun j $\rightarrow$ f i j

let exemple i j = f i j

let exemple i = f i

let exemple = f

$\beta\eta\alpha$-conversion

$$M \quad \rightarrow_{\alpha\beta\eta} \quad N$$

quand N est obtenu en faisant subir des
η, β, α-conversions successives
à des sous-termes à partir de M.

La conversion $\rightarrow_{\alpha\beta\eta}$ est indépendante du typage:
transformation syntaxique d'un terme

Lien avec le typage?

Propriété de réduction du sujet

Le type est conservé par $\beta\eta$-réduction

$$\left.\begin{array}{l} H \vdash M : t \\ M \rightarrow_{\beta,\eta} N \end{array}\right\} \Rightarrow H \vdash N : t$$

Forme normale - Forte normalisation

- **Forme normale**

Forme normale - Forte normalisation

- **Forme normale**
  - Un terme qui ne contient plus aucun $\beta\eta$ redex

Forme normale - Forte normalisation

- **Forme normale**
  - Un terme qui ne contient plus aucun $\beta\eta$ redex

- **Théorème de forte normalistion**

Forme normale - Forte normalisation

- **Forme normale**
  - Un terme qui ne contient plus aucun $\beta\eta$ redex

- **Théorème de forte normalistion**

  - Tout terme de $\lambda^{\rightarrow}$ a une unique forme normale,

Forme normale - Forte normalisation

- **Forme normale**
  - Un terme qui ne contient plus aucun $\beta\eta$ redex
- **Théorème de forte normalistion**
  - Tout terme de $\lambda^{\rightarrow}$ a une unique forme normale,
  - obtenue après un nombre fini de $\beta\eta$-réductions

Forme normale - Forte normalisation

- **Forme normale**
  - Un terme qui ne contient plus aucun $\beta\eta$ redex

- **Théorème de forte normalistion**
  - Tout terme de $\lambda^{\rightarrow}$ a une unique forme normale,
  - obtenue après un nombre fini de $\beta\eta$-réductions
  - quelle que soit la stratégie utilisée
    (choix du redex à transformer à chaque étape)

Forme normale - Forte normalisation

Nécessité du typage pour la normalisation

$$(\lambda x.x\,x)\,(\lambda x.x\,x) \quad \rightarrow_\beta \quad (\lambda x.x\,x)\,(\lambda x.x\,x)$$

Règles équationnelles

$$H \triangleright M = N : t$$

Pour l'affectation de types H,
M et N sont deux termes de type t,
considérés comme équivalents

De quel point de vue des termes sont-ils considérés comme équivalents?

Règles équationnelles

Règles β et η

$$\frac{H \cup \{x : s\} \vdash M : t \qquad H \vdash N : s}{H \rhd (\lambda x_{:s}.M)N = M[x := N] : t} \quad (\beta)$$

$$\frac{H \vdash M : s \to t \qquad x \notin FV(M)}{H \rhd (\lambda x_{:s}.M\ x) = M : s \to t} \quad (\eta)$$

Règles équationnelles

Relation d'équivalence

$$\frac{H \vdash M : t}{H \triangleright M = M : t} \qquad \frac{H \triangleright M = N : t}{H \triangleright N = M : t} \qquad \frac{H \triangleright M = N : t \quad H \triangleright N = P : t}{H \triangleright M = P : t}$$

Modification de l'affectation de type

$$\frac{H \triangleright M = N : t \quad x \notin H}{H \cup \{x : s\} \triangleright N = M : t}$$

$$\frac{H \cup \{x : s\} \triangleright M = N : t \quad x \notin FV(M) \cup FV(N)}{H \triangleright M = N : t}$$

Règles équationnelles

Règles de congruence

$$\frac{H \rhd M = M' : s \to t \qquad H \rhd N = N' : s}{H \rhd (M\,N) = (M'\,N') : t}$$

$$\frac{H \cup \{x : s\} \rhd M = M' : t}{H \rhd \lambda x_{:s}.M = \lambda x_{:s}.M' : t} \qquad (\xi)$$

Modèles

- **But** Définir précisément pourquoi ce calcul a été introduit
  - avec ses termes et ses types
  - ses règles de typage et de conversion
  - son équivalence

- Donner un sens aux objets de $\lambda^{\to}$

  i.e. aux jugements de type

- **Interprétation**: objet mathématique associé à $H \vdash M : t$

  cf. ensemble représenté par un type Caml t

  cf. valeur d'une expression Caml M dans un environnement

Interprétation des types

$[\![t]\!]$	*interprétation d'un type t*
$[\![o]\!] = X$	*un ensemble fixé*
$[\![s \rightarrow t]\!] = [\![t]\!]^{[\![s]\!]}$	*ensemble des fonctions de* $[\![s]\!]$ *dans* $[\![t]\!]$

cf. Caml: $[\![int]\!] = \mathbb{Z}$ et $[\![int \rightarrow int]\!] = \mathbb{Z}^{\mathbb{Z}}$, l'ensemble des fonctions sur les entiers relatifs, à valeurs entières.

Environnements: interprétation des variables libres

Affectation de types: $H = \{x_1 : t_1, \ \ldots \ , x_n : t_n\}$.

H-Environnement: ρ qui à x_i associe une valeur v_i

Notations: $\rho = [x_1 \rightarrow v_1, \ldots, x_n \rightarrow v_n]$
$v_i = \rho(x_i) = [\![x_i]\!]_\rho$

Cohérence: $\forall i \in \{1, \ \ldots, \ n\}, \ [\![x_i]\!]_\rho \in [\![t_i]\!]$

Condition imposée à ρ pour être un H-environnement

cf. Caml: avec x : *int*, on peut écrire *let* $x = 10$.
Dans le nouvel environnement ρ, x est interprété par

$$[\![x]\!]_\rho \ = \ 10 \in \mathbb{Z}$$

Interprétation des jugements de type

Induction sur la dérivation du jugement

Données: H une affectation de type, ρ un H-environnement

$$\boxed{[\![H \vdash x : t]\!]_\rho = [\![x]\!]_\rho} \qquad (1)$$

Interprétation des jugements de type

Induction sur la dérivation du jugement

L'application: étude informelle d'un exemple Caml
Données: $H = \{v : int, n : int\}$ et $\rho = [v \rightarrow 10, n \rightarrow 2]$
Interpréter: $[\![H \vdash ((fun\ x -> x * x + v)\ n) : int]\!]_\rho$

$$\frac{H \vdash (\lambda x_{:int}.x * x + v) : int \rightarrow int \qquad H \vdash n : int}{H \vdash ((\lambda x_{:int}.x * x + v)\ \ n) : int}\ (app)$$

Hypothèses: On a interprété le numérateur:

$[\![n]\!]_\rho = 2$ et $[\![H \vdash (\lambda x_{:int}.x * x + v) : int \rightarrow int]\!]_\rho = f$

avec $f : \mathbb{Z} \rightarrow \mathbb{Z}$ t. q. $f(d) = d^2 + 10$

Conclusion: on interprète le dénominateur par 14 (i.e. f(2))

Interprétation des jugements de type

Induction sur la dérivation du jugement

L'application: étude informelle d'un exemple Caml
Données: $H = \{v : int, n : int\}$ et $\rho = [v \rightarrow 10, n \rightarrow 2]$
Interpréter: $[\![H \vdash ((fun\ x -> x * x + v)\ n) : int]\!]_\rho$

$$\frac{H \vdash (\lambda x_{:int}.x * x + v) : int \rightarrow int \qquad H \vdash n : int}{H \vdash ((\lambda x_{:int}.x * x + v)\ \ n) : int}\ (app)$$

$$\boxed{[\![H \vdash ((\lambda x_{:int}.x * x + v)\ \ n) : int]\!]_\rho = f(d)}$$

$$f = [\![H \vdash (\lambda x_{:int}.x * x + v) : int \rightarrow int]\!]_\rho \qquad d = [\![n]\!]_\rho$$

Interprétation des jugements de type

Induction sur la dérivation du jugement

Données: H une affectation de type, ρ un H-environnement

$H \vdash (M\ N) : t$ provient de $\dfrac{H \vdash M : t \rightarrow s \quad H \vdash N : t}{H \vdash (M\ N) : s}\ (appl)$

$$\boxed{[\![H \vdash (M\ N) : t]\!]_\rho = f(v)} \quad (2)$$

$f = [\![H \vdash M : s \rightarrow t]\!]_\rho$ $\qquad$ $v = [\![H \vdash N : s]\!]_\rho$

Interprétation des jugements de type

Induction sur la dérivation du jugement

Abstraction: étude informelle d'un exemple Caml
Données: $H = \{v : int\}$, $\rho = [v \rightarrow 10]$
Interpréter: $[\![H \vdash (fun\ x -> x * x + v) : int \rightarrow int]\!]_\rho$

$$\frac{H \cup \{x : int\} \vdash (x * x + v) : int}{H \vdash \lambda x_{:int}.(x * x + v) \ : \ int \rightarrow int} \ (abs)$$

Hypothèses: $\forall d \in \mathbb{Z}$ on a interprété le numérateur par:
$[\![H \cup \{x : int\} \vdash x * x + v]\!]_{\rho[x->d]} = d^2 + 10 \in \mathbb{Z}$

Conclusion: on interprète le dénominateur par $f : \mathbb{Z} \rightarrow \mathbb{Z}$ t.q.
$f : d \rightarrow d^2 + 10 = [\![H \cup \{x : int\} \vdash x * x + v]\!]_{\rho[x->d]}$

Interprétation des jugements de type

Induction sur la dérivation du jugement

Abstraction: étude informelle d'un exemple Caml
Données: $H = \{v : int\}$, $\rho = [v \rightarrow 10]$
Interpréter: $[\![H \vdash (fun\ x -> x * x + v) : int \rightarrow int]\!]_\rho$

$$\frac{H \cup \{x : int\} \vdash (x * x + v) : int}{H \vdash \lambda x_{:int}.(x * x + v) \ : \ int \rightarrow int} \ (abs)$$

$$\boxed{[\![H \vdash \lambda x_{:int}.(x * x + v) \ : \ int \rightarrow int]\!]_\rho = f}$$

$$\begin{array}{ccc} f : \mathbb{Z} & \rightarrow & \mathbb{Z} \\ d & & \underbrace{[\![H \cup \{x : int\} \vdash x * x + v]\!]_{\rho[x->d]}}_{d^2 + 10} \end{array}$$

Interprétation des jugements de type

Induction sur la dérivation du jugement

Données: H une affectation de type, ρ un H-environnement

$$H \vdash \lambda x_{:s}.M : s \to t \text{ provient de } \frac{H \cup \{x : s\} \vdash M : t}{H \vdash \lambda x_{:s}.M \ : \ s \to t} \ (abs)$$

$$\boxed{[\![H \vdash \lambda x_{:s}.M : t]\!]_\rho = f} \quad (3)$$

$f : [\![s]\!] \quad \to \quad [\![t]\!]$

$v \qquad\qquad f(d) = [\![H \cup \{x : s\} \vdash M]\!]_{\rho[x \to d]}$

Validité de l'interprétation

Si $H \triangleright M = N : t$ alors $[\![H \vdash N : t]\!]_\rho = [\![H \vdash N : t\ ]\!]_\rho$

Se prouve par induction sur la dérivation de $H \triangleright M = N : t$

$\lambda^{\rightarrow}$ est non trivial

Il existe des termes qui ne sont pas équivalents.

Par exemple:

Si $x \neq y$ on ne peut prouver $\quad H \triangleright x = y : t$

Si on avait pu prouver que $H \triangleright x = y : t$, toute interprétation donnerait des résultats identiques pour $H \vdash x : t$ et $H \vdash y : t$. Or, si l'on interpréte t par un ensemble qui a au moins 2 éléments, on peut interpréter les variables par deux valeurs distintes.