L'inférence de types

on veut, étant donné un programme implicitement typé, être capable de dire s'il est typable. . . et, en général, en donner le type . . . le ou un type, suivant les cas.

Remarque: des langages comme Pascal, C ou Java demandent que les arguments des fonctions soient explicitement typés, ainsi que le résultat

Les étapes de la compilation

programme source

↓ analyse lexicale, analyse syntaxique

arbre de syntaxe abstraite

↓ analyses statiques (typage, . . .)

arbre décoré

↓ traduction**s**, optimisations

code exécutable (langage machine)

- on analyse le programme sans l'exécuter
- on travaille sur un arbre représentant le code source

Premier exemple

```
let rec f x = fun y ->
    if x then (string_of_int (9*y)) else f (y>11) (y-3)
f :
```

Premier exemple

```
let rec f x = fun y ->
    if x then (string_of_int (9*y)) else f (y>11) (y-3)
```

`f : bool->int->string`

ingrédients:

- on part des valeurs constantes
- on manipule des *contraintes* entre types

Inférence – esquisse

étapes de la méthode:

- partir du *programme*, et le parcourir en écrivant des contraintes de typage qui doivent être satisfaites suivant les différentes constructions du langage
    - constantes `(3, +, >, ...)`
    - constructions du langage `if then else`
    - fonctions: déclaration, application

Inférence – esquisse

étapes de la méthode:

- partir du *programme*, et le parcourir en écrivant des contraintes de typage qui doivent être satisfaites suivant les différentes constructions du langage
  - constantes `(3, +, >, ...)`
  - constructions du langage `if then else`
  - fonctions: déclaration, application
- “raisonner” sur les contraintes
  - propager l’information
  - arrêter en cas de conflit
    (p.ex. $\texttt{int} = \texttt{int} \rightarrow \texttt{int},\ \ '\texttt{a} \rightarrow \texttt{bool} = \texttt{int}, \ldots$)

Engendrer le problème

Récolter l'information

- on écrit des *contraintes* (égalités entre types) qui doivent être satisfaites pour que l'expression soit typable
- exemple:

$$\texttt{let f g x = }\underbrace{\texttt{if }\underbrace{\overbrace{\texttt{x}}^{A_x} > 0}_{A_1}\texttt{ then }\underbrace{3}_{A_3}\texttt{ else }\overbrace{(\underbrace{\texttt{g}}_{A_g}\ \ \underbrace{\texttt{x}}_{A_x})}^{A_2}}_{A_0}$$

  - A_i: types pour toutes les sous-expressions *(il en manque ci-dessus)*

Récolter l'information

- on écrit des *contraintes* (égalités entre types) qui doivent être satisfaites pour que l'expression soit typable
- exemple:

$$\texttt{let f g x = } \underbrace{\texttt{if } \underbrace{\overbrace{\texttt{x}}^{A_x} > 0}_{A_1} \texttt{ then } \underbrace{3}_{A_3} \texttt{ else } \overbrace{(\underbrace{\texttt{g}}_{A_g}\ \underbrace{\texttt{x}}_{A_x})}^{A_2}}_{A_0}$$

  - A_i: types pour toutes les sous-expressions *(il en manque ci-dessus)*
  - contraintes:

Récolter l'information

- on écrit des *contraintes* (égalités entre types) qui doivent être satisfaites pour que l'expression soit typable
- exemple:

$$\texttt{let f g x = }\underbrace{\texttt{if }\underbrace{\overbrace{\texttt{x}}^{A_x} > 0}_{A_1}\texttt{ then }\underbrace{3}_{A_3}\texttt{ else }\overbrace{(\underbrace{\texttt{g}}_{A_g}\quad\underbrace{\texttt{x}}_{A_x})}^{A_2}}_{A_0}$$

  - A_i: types pour toutes les sous-expressions *(il en manque ci-dessus)*
  - contraintes: $A_3 = \texttt{int}$, $A_x = \texttt{int}$,

Récolter l'information

- on écrit des *contraintes* (égalités entre types) qui doivent être satisfaites pour que l'expression soit typable
- exemple:

$$\texttt{let f g x = }\underbrace{\texttt{if }\underbrace{\overbrace{\texttt{x}}^{A_x} > 0}_{A_1}\texttt{ then }\underbrace{3}_{A_3}\texttt{ else }\overbrace{(\underbrace{\texttt{g}}_{A_g}\quad\underbrace{\texttt{x}}_{A_x})}^{A_2}}_{A_0}$$

  - A_i: types pour toutes les sous-expressions *(il en manque ci-dessus)*
  - contraintes: $A_3 = \texttt{int}$, $A_x = \texttt{int}$, $\overbrace{A_1 = \texttt{bool}, A_0 = A_3 = A_2}^{\texttt{if then else}}$

Récolter l'information

- on écrit des *contraintes* (égalités entre types) qui doivent être satisfaites pour que l'expression soit typable
- exemple:

$$\texttt{let f g x = if } \underbrace{\underbrace{\overbrace{\texttt{x}}^{A_x} > 0}_{A_1} \texttt{ then } \underbrace{3}_{A_3} \texttt{ else } \overbrace{(\underbrace{\texttt{g}}_{A_g}\ \ \underbrace{\texttt{x}}_{A_x})}^{A_2}}_{A_0}$$

  - A_i: types pour toutes les sous-expressions *(il en manque ci-dessus)*
  - contraintes: $A_3 = \texttt{int}$, $A_x = \texttt{int}$,

$$\overbrace{A_1 = \texttt{bool}, A_0 = A_3 = A_2}^{\texttt{if then else}}, A_g = A_x \to A_2$$

Récolter l'information

- on écrit des *contraintes* (égalités entre types) qui doivent être satisfaites pour que l'expression soit typable
- exemple:

$$\texttt{let f g x = }\underbrace{\texttt{if }\underbrace{\overbrace{\texttt{x}}^{A_x} > 0}_{A_1}\texttt{ then }\underbrace{3}_{A_3}\texttt{ else }\overbrace{(\underbrace{\texttt{g}}_{A_g}\quad\underbrace{\texttt{x}}_{A_x})}^{A_2}}_{A_0}$$

  - A_i: types pour toutes les sous-expressions *(il en manque ci-dessus)*
  - contraintes: $A_3 = \texttt{int}$, $A_x = \texttt{int}$,

$$\overbrace{A_1 = \texttt{bool}, A_0 = A_3 = A_2}^{\texttt{if then else}}, A_g = A_x \to A_2, A_0 = \texttt{int}$$

Récolter l'information

- on écrit des *contraintes* (égalités entre types) qui doivent être satisfaites pour que l'expression soit typable
- exemple:

$$\texttt{let f g x = } \underbrace{\texttt{if } \underbrace{\overbrace{\texttt{x}}^{A_x} > 0}_{A_1} \texttt{ then } \underbrace{3}_{A_3} \texttt{ else } \overbrace{(\underbrace{\texttt{g}}_{A_g}\ \underbrace{\texttt{x}}_{A_x})}^{A_2}}_{A_0}$$

  - A_i: types pour toutes les sous-expressions *(il en manque ci-dessus)*
  - contraintes: $A_3 = \texttt{int}$, $A_x = \texttt{int}$, $\overbrace{A_1 = \texttt{bool}, A_0 = A_3 = A_2}^{\texttt{if then else}}, A_g = A_x \to A_2, A_0 = \texttt{int}, \ldots$

Récolter l'information

- on écrit des *contraintes* (égalités entre types) qui doivent être satisfaites pour que l'expression soit typable
- exemple:

$$\texttt{let f g x = } \underbrace{\texttt{if } \underbrace{\overbrace{\texttt{x}}^{A_x} > 0}_{A_1} \texttt{ then } \underbrace{3}_{A_3} \texttt{ else } \overbrace{(\underbrace{\texttt{g}}_{A_g}\ \ \underbrace{\texttt{x}}_{A_x})}^{A_2}}_{A_0}$$

  - A_i: types pour toutes les sous-expressions *(il en manque ci-dessus)*
  - contraintes: $A_3 = \texttt{int},\ A_x = \texttt{int},$

$$\overbrace{A_1 = \texttt{bool}, A_0 = A_3 = A_2}^{\texttt{if then else}}, A_g = A_x \to A_2, A_0 = \texttt{int}, \ldots$$

et le type de `f`?

Récolter l'information

- on écrit des *contraintes* (égalités entre types) qui doivent être satisfaites pour que l'expression soit typable
- exemple:

$$\texttt{let f g x = }\underbrace{\texttt{if }\underbrace{\overbrace{\texttt{x}}^{A_x} > 0}_{A_1}\texttt{ then }\underbrace{3}_{A_3}\texttt{ else }\overbrace{(\underbrace{\texttt{g}}_{A_g}\quad\underbrace{\texttt{x}}_{A_x})}^{A_2}}_{A_0}$$

  - A_i: types pour toutes les sous-expressions *(il en manque ci-dessus)*
  - contraintes: $A_3 = \texttt{int},\ A_x = \texttt{int},$

$$\overbrace{A_1 = \texttt{bool}, A_0 = A_3 = A_2}^{\texttt{if then else}}, A_g = A_x \to A_2, A_0 = \texttt{int}, \ldots$$

et le type de f? $\overbrace{\texttt{f}}^{A_f} = \texttt{fun}\ \overbrace{\texttt{g}}^{A_g}\ \texttt{->}\ \underbrace{\texttt{fun}\ \overbrace{\texttt{x}}^{A_x}\ \texttt{->}\ \overbrace{\ldots}^{A_0}}_{A_4}$

$$\rightsquigarrow \quad A_4 = A_x \to A_0,\ A_f = A_g \to A_4$$

Engendrer les contraintes

- on associe une ‘inconnue de type’ (variable de type) à chaque sous-expression *(ou sous-arbre)*
- contraintes = équations entre types

Engendrer les contraintes

- on associe une ‘inconnue de type’ (variable de type) à chaque sous-expression *(ou sous-arbre)*
- contraintes = équations entre types

  ```
  m= e1+e2
  ```

Engendrer les contraintes

- on associe une ‘inconnue de type’ (variable de type) à chaque sous-expression *(ou sous-arbre)*
- contraintes = équations entre types

  `m= e1+e2` $T_{\texttt{m}} = \texttt{int}, T_{\texttt{e1}} = \texttt{int}, T_{\texttt{e2}} = \texttt{int},$

  `m = if e1 then e2 else e3`

Engendrer les contraintes

- on associe une 'inconnue de type' (variable de type) à chaque sous-expression *(ou sous-arbre)*
- contraintes = équations entre types

`m= e1+e2`	$T_{\mathtt{m}} = \mathtt{int}, T_{\mathtt{e1}} = \mathtt{int}, T_{\mathtt{e2}} = \mathtt{int},$
`m = if e1 then e2 else e3`	$T_{\mathtt{e1}} = \mathtt{bool}, T_{\mathtt{e2}} = T_{\mathtt{m}}, T_{\mathtt{e3}} = T_{\mathtt{m}}$
`m = e1 e2`	

Engendrer les contraintes

- on associe une ‘inconnue de type’ (variable de type) à chaque sous-expression *(ou sous-arbre)*
- contraintes = équations entre types

`m= e1+e2`	$T_{\mathtt{m}} = \mathtt{int}$, $T_{\mathtt{e1}} = \mathtt{int}$, $T_{\mathtt{e2}} = \mathtt{int}$,
`m = if e1 then e2 else e3`	$T_{\mathtt{e1}} = \mathtt{bool}$, $T_{\mathtt{e2}} = T_{\mathtt{m}}$, $T_{\mathtt{e3}} = T_{\mathtt{m}}$
`m = e1 e2`	$T_{\mathtt{e1}} = T_{\mathtt{e2}} \rightarrow T_{\mathtt{m}}$
`m = fun x -> e`	

Engendrer les contraintes

- on associe une ‘inconnue de type’ (variable de type) à chaque sous-expression *(ou sous-arbre)*
- contraintes = équations entre types

`m= e1+e2`	$T_{\mathtt{m}} = \mathtt{int}, T_{\mathtt{e1}} = \mathtt{int}, T_{\mathtt{e2}} = \mathtt{int},$
`m = if e1 then e2 else e3`	$T_{\mathtt{e1}} = \mathtt{bool}, T_{\mathtt{e2}} = T_{\mathtt{m}}, T_{\mathtt{e3}} = T_{\mathtt{m}}$
`m = e1 e2`	$T_{\mathtt{e1}} = T_{\mathtt{e2}} \rightarrow T_{\mathtt{m}}$
`m = fun x -> e`	$T_{\mathtt{m}} = T_{\mathtt{x}} \rightarrow T_{\mathtt{e}}$

Engendrer les contraintes

- on associe une ‘inconnue de type’ (variable de type) à chaque sous-expression *(ou sous-arbre)*
- contraintes = équations entre types

`m= e1+e2`	$T_{\mathtt{m}} = \mathtt{int}$, $T_{\mathtt{e1}} = \mathtt{int}$, $T_{\mathtt{e2}} = \mathtt{int}$,
`m = if e1 then e2 else e3`	$T_{\mathtt{e1}} = \mathtt{bool}$, $T_{\mathtt{e2}} = T_{\mathtt{m}}$, $T_{\mathtt{e3}} = T_{\mathtt{m}}$
`m = e1 e2`	$T_{\mathtt{e1}} = T_{\mathtt{e2}} \rightarrow T_{\mathtt{m}}$
`m = fun x -> e`	$T_{\mathtt{m}} = T_{\mathtt{x}} \rightarrow T_{\mathtt{e}}$

ainsi, pour `fun x -> e`, on engendre $T_{\mathtt{x}}$, et (en principe) à chaque occurrence de `x` dans `e`, on engendre T_i et on écrit $T_i = T_{\mathtt{x}}$

Engendrer les contraintes

- on associe une ‘inconnue de type’ (variable de type) à chaque sous-expression *(ou sous-arbre)*
- contraintes = équations entre types

`m= e1+e2`	$T_{\mathtt{m}} = \mathtt{int}$, $T_{\mathtt{e1}} = \mathtt{int}$, $T_{\mathtt{e2}} = \mathtt{int}$,
`m = if e1 then e2 else e3`	$T_{\mathtt{e1}} = \mathtt{bool}$, $T_{\mathtt{e2}} = T_{\mathtt{m}}$, $T_{\mathtt{e3}} = T_{\mathtt{m}}$
`m = e1 e2`	$T_{\mathtt{e1}} = T_{\mathtt{e2}} \rightarrow T_{\mathtt{m}}$
`m = fun x -> e`	$T_{\mathtt{m}} = T_{\mathtt{x}} \rightarrow T_{\mathtt{e}}$

  ainsi, pour `fun x -> e`, on engendre $T_{\mathtt{x}}$, et (en principe) à chaque occurrence de `x` dans `e`, on engendre T_i et on écrit $T_i = T_{\mathtt{x}}$

- parcours récursif de l’arbre en appliquant ces règles

Engendrer les contraintes – exemples

- exemple: `let f = fun g -> fun x -> (g (x*2))-3`

  $A_f = A_g \to A_0,\ A_0 = A_x \to A_1,\ A_1 = \texttt{int},\ A_2 = \texttt{int},\ A_g = A_3 \to A_2,\ A_3 = \texttt{int},\ A_x = \texttt{int}$

Engendrer les contraintes – exemples

- exemple: `let f = fun g -> fun x -> (g (x*2))-3`

  $A_f = A_g \to A_0,\ \ A_0 = A_x \to A_1,\ \ A_1 = \mathtt{int},\ \ A_2 = \mathtt{int},\ \ A_g = A_3 \to A_2,\ \ A_3 = \mathtt{int},\ \ A_x = \mathtt{int}$

- $\texttt{fun x ->}\overbrace{\texttt{fun f -> }\underbrace{\texttt{(f x)}}_{u}}^{v}$

  $$T_{\mathtt{f}} = T_{\mathtt{x}} \to T_u \qquad T_v = T_{\mathtt{f}} \to T_u \qquad T_0 = T_{\mathtt{x}} \to T_v$$

Engendrer les contraintes – exemples

- exemple: `let f = fun g -> fun x -> (g (x*2))-3`

  $A_f = A_g \to A_0,\ A_0 = A_x \to A_1,\ A_1 = \texttt{int},\ A_2 = \texttt{int},\ A_g = A_3 \to A_2,\ A_3 = \texttt{int},\ A_x = \texttt{int}$

- $\texttt{fun x ->}\overbrace{\texttt{fun f -> }\underbrace{\texttt{(f x)}}_{u}}^{v}$

$$T_{\texttt{f}} = T_{\texttt{x}} \to T_u \qquad T_v = T_{\texttt{f}} \to T_u \qquad T_0 = T_{\texttt{x}} \to T_v$$

(ce qui se résoud en $T_0 \ = \ T_{\texttt{x}} \to (T_{\texttt{x}} \to T_u) \to T_u$)

Le système de types

- la manière dont les contraintes sont engendrées découle de la définition du système de types, qui à son tour est décrit par des *règles de typage*

  on définit la relation $\Gamma \vdash \mathtt{e}\colon T$, où Γ est une liste d'*hypothèses de typage* de la forme $\mathtt{x}\colon T_{\mathtt{x}}$, “pour `x` variable libre de `e`”

Le système de types

- la manière dont les contraintes sont engendrées découle de la définition du système de types, qui à son tour est décrit par des *règles de typage*

  on définit la relation $\Gamma \vdash \texttt{e} \colon T$, où Γ est une liste d'*hypothèses de typage* de la forme $\texttt{x} \colon T_{\texttt{x}}$, “pour $\texttt{x}$ variable libre de $\texttt{e}$”

$$\begin{array}{l}\textsc{TCst}_0\\ \Gamma \vdash 0 : \texttt{int}\end{array} \qquad \begin{array}{l}\textsc{TCst}_1\\ \Gamma \vdash 1 : \texttt{int}\end{array} \qquad \begin{array}{l}\textsc{TCst}_+\\ \Gamma \vdash (+) : \texttt{int} \to \texttt{int} \to \texttt{int}\end{array} \qquad \textit{etc.}\ldots$$

$$\begin{array}{l}\textsc{TIf}\\ \dfrac{\Gamma \vdash \texttt{b} : \texttt{bool} \qquad \Gamma \vdash \texttt{e} : T \qquad \Gamma \vdash \texttt{e}' : T}{\Gamma \vdash \texttt{if b then e else e}' : T}\end{array}$$

$$\begin{array}{l}\textsc{TApp}\\ \dfrac{\Gamma \vdash \texttt{f} : T \to U \qquad \Gamma \vdash \texttt{e} : T}{\Gamma \vdash \texttt{f e} : U}\end{array} \qquad \begin{array}{l}\textsc{TFun}\\ \dfrac{\Gamma, \texttt{x} : T \vdash \texttt{e} : U}{\Gamma \vdash \texttt{fun x -> e} : T \to U}\end{array} \qquad \begin{array}{l}\textsc{TVar}\\ \dfrac{\texttt{x} : T \in \Gamma}{\Gamma \vdash \texttt{x} : T}\end{array}$$

Dérivation de typage – exemple

$$\textsc{TCst}_0\ \frac{}{\Gamma \vdash 0 : \texttt{int}} \qquad \textsc{TCst}_1\ \frac{}{\Gamma \vdash 1 : \texttt{int}} \qquad \textsc{TCst}_+\ \frac{}{\Gamma \vdash (+) : \texttt{int} \longrightarrow \texttt{int} \longrightarrow \texttt{int}}$$

$$\textsc{TIf}\ \frac{\Gamma \vdash \texttt{b} : \texttt{bool} \qquad \Gamma \vdash \texttt{e} : T \qquad \Gamma \vdash \texttt{e}' : T}{\Gamma \vdash \texttt{if b then e else e}' : T}$$

$$\textsc{TApp}\ \frac{\Gamma \vdash \texttt{f} : T \longrightarrow U \qquad \Gamma \vdash \texttt{e} : T}{\Gamma \vdash \texttt{f e} : U} \qquad \textsc{TFun}\ \frac{\Gamma, \texttt{x} : T \vdash \texttt{e} : U}{\Gamma \vdash \texttt{fun x -> e} : T \longrightarrow U} \qquad \textsc{TVar}\ \frac{\texttt{x} : T \in \Gamma}{\Gamma \vdash \texttt{x} : T}$$

- ces règles permettent de construire des *dérivations de typage* (arbres dont la conclusion est un *jugement de typage*)
- exemple:

  $\emptyset \vdash$ `fun g -> fun x -> (g (x*2))-3 : (int->int)->int->int`

  DÉMO au tableau

Dérivation de typage – exemple

$$\textsc{TCst}_0 \quad \Gamma \vdash 0 : \mathtt{int} \qquad \textsc{TCst}_1 \quad \Gamma \vdash 1 : \mathtt{int} \qquad \textsc{TCst}_+ \quad \Gamma \vdash (+) : \mathtt{int} \to \mathtt{int} \to \mathtt{int}$$

$$\textsc{TIf} \quad \frac{\Gamma \vdash \mathtt{b} : \mathtt{bool} \qquad \Gamma \vdash \mathtt{e} : T \qquad \Gamma \vdash \mathtt{e}' : T}{\Gamma \vdash \mathtt{if\ b\ then\ e\ else\ e}' : T}$$

$$\textsc{TApp} \quad \frac{\Gamma \vdash \mathtt{f} : T \to U \qquad \Gamma \vdash \mathtt{e} : T}{\Gamma \vdash \mathtt{f\ e} : U} \qquad \textsc{TFun} \quad \frac{\Gamma, \mathtt{x} : T \vdash \mathtt{e} : U}{\Gamma \vdash \mathtt{fun\ x\ \text{->}\ e} : T \to U} \qquad \textsc{TVar} \quad \frac{\mathtt{x} : T \in \Gamma}{\Gamma \vdash \mathtt{x} : T}$$

- ces règles permettent de construire des *dérivations de typage* (arbres dont la conclusion est un *jugement de typage*)
- exemple:

  $\emptyset \vdash$ `fun g -> fun x -> (g (x*2))-3 : (int->int)->int->int`

  DÉMO au tableau

- une règle par construction du langage

  ↪ pour l'inférence, on raisonne par cas

  "on fait un `match with`"

Retour de l'unification

- les contraintes engendrées par un programme sont vues comme un **problème d'unification**

  on résoud des équations symboliques sur les *types* de Caml

  p.ex. $A_1 \to (\texttt{int} \to A_2) \overset{?}{=} (A_3 \to \texttt{bool}) \to A_1$

  ou si on préfère

  $\mathit{fleche}(A_1, \mathit{fleche}(\texttt{int}, A_2)) \overset{?}{=} \mathit{fleche}(\mathit{fleche}(A_3, \texttt{bool}), A_1)$

Retour de l'unification

- les contraintes engendrées par un programme sont vues comme un **problème d'unification**

  on résoud des équations symboliques sur les *types* de Caml

  p.ex. $A_1 \rightarrow (\mathtt{int} \rightarrow A_2) \stackrel{?}{=} (A_3 \rightarrow \mathtt{bool}) \rightarrow A_1$

  ou si on préfère

  $\mathit{fleche}(A_1, \mathit{fleche}(\mathtt{int}, A_2)) \stackrel{?}{=} \mathit{fleche}(\mathit{fleche}(A_3, \mathtt{bool}), A_1)$

  - si l'unification donne une substitution S, on renvoie le type $S(A_f)$ *(on est en train de typer `let f = ...`)*
  - sinon, on proteste *(Caml raconte où l'unification a planté)*

Retour de l'unification

- les contraintes engendrées par un programme sont vues comme un **problème d'unification**

  on résoud des équations symboliques sur les *types* de Caml

  p.ex. $A_1 \to (\mathtt{int} \to A_2) \stackrel{?}{=} (A_3 \to \mathtt{bool}) \to A_1$

  ou si on préfère

  $\mathit{fleche}(A_1, \mathit{fleche}(\mathtt{int}, A_2)) \stackrel{?}{=} \mathit{fleche}(\mathit{fleche}(A_3, \mathtt{bool}), A_1)$

  - si l'unification donne une substitution S, on renvoie le type $S(A_f)$ *(on est en train de typer `let f = ...`)*
  - sinon, on proteste *(Caml raconte où l'unification a planté)*
- et voilà

Inférence de types – propriétés

$$\text{terme } \mathtt{m} \rightarrow \text{système d'équations } \mathcal{C}(\mathtt{m}) \xrightarrow{\text{unification}} \text{unificateur } S$$

Propriétés:

- correction: un unificateur S de $\mathcal{C}(\mathtt{m})$ permet d'inférer
$$\emptyset \vdash m : S(A_{\mathtt{m}})$$
- complétude: si l'on peut dériver $\emptyset \vdash \mathtt{m} : T$, alors $\mathcal{C}(\mathtt{m})$ admet une solution S t.q. $S(A_{\mathtt{m}}) = T$

Déroulons un exemple

```
let h f b = if b then 52 else (f b)+32
```

on engendre le problème d'unification

$A_h \overset{?}{=} A_f \rightarrow A_1,\ A_1 \overset{?}{=} A_b \rightarrow A_2,\ A_2 \overset{?}{=} \texttt{int},\ A_b \overset{?}{=} \texttt{bool},\ A_3 \overset{?}{=} \texttt{int},\ A_f \overset{?}{=} A_b \rightarrow A_3$

Déroulons un exemple

```
let h f b = if b then 52 else (f b)+32
```

on engendre le problème d'unification

$A_h \overset{?}{=} A_f \rightarrow A_1,\ A_1 \overset{?}{=} A_b \rightarrow A_2,\ A_2 \overset{?}{=} \texttt{int},\ A_b \overset{?}{=} \texttt{bool},\ A_3 \overset{?}{=} \texttt{int},\ A_f \overset{?}{=} A_b \rightarrow A_3$

$\Rightarrow \quad A_1 \overset{?}{=} A_b \rightarrow A_2,\ A_2 \overset{?}{=} \texttt{int},\ A_b \overset{?}{=} \texttt{bool},\ A_3 \overset{?}{=} \texttt{int},\ A_f \overset{?}{=} A_b \rightarrow A_3,\ \{A_h \longmapsfrom A_f \rightarrow A_1\}$

Déroulons un exemple

```
let h f b = if b then 52 else (f b)+32
```

on engendre le problème d'unification

$A_h \overset{?}{=} A_f \to A_1,\ A_1 \overset{?}{=} A_b \to A_2,\ A_2 \overset{?}{=} \texttt{int},\ A_b \overset{?}{=} \texttt{bool},\ A_3 \overset{?}{=} \texttt{int},\ A_f \overset{?}{=} A_b \to A_3$

$\Rightarrow \quad A_1 \overset{?}{=} A_b \to A_2,\ A_2 \overset{?}{=} \texttt{int},\ A_b \overset{?}{=} \texttt{bool},\ A_3 \overset{?}{=} \texttt{int},\ A_f \overset{?}{=} A_b \to A_3,\ \{A_h \longmapsfrom A_f \to A_1\}$

$\Rightarrow \quad A_2 \overset{?}{=} \texttt{int},\ A_b \overset{?}{=} \texttt{bool},\ A_3 \overset{?}{=} \texttt{int},\ A_f \overset{?}{=} A_b \to A_3,$
$\{A_h \longmapsfrom A_f \to (A_b \to A_2),\ A_1 \longmapsfrom A_b \to A_2\}$

Déroulons un exemple

```
let h f b = if b then 52 else (f b)+32
```

on engendre le problème d'unification

$$A_h \overset{?}{=} A_f \to A_1,\ A_1 \overset{?}{=} A_b \to A_2,\ A_2 \overset{?}{=} \texttt{int},\ A_b \overset{?}{=} \texttt{bool},\ A_3 \overset{?}{=} \texttt{int},\ A_f \overset{?}{=} A_b \to A_3$$

$$\Rightarrow \quad A_1 \overset{?}{=} A_b \to A_2,\ A_2 \overset{?}{=} \texttt{int},\ A_b \overset{?}{=} \texttt{bool},\ A_3 \overset{?}{=} \texttt{int},\ A_f \overset{?}{=} A_b \to A_3,\ \{A_h \longmapsfrom A_f \to A_1\}$$

$$\Rightarrow \quad A_2 \overset{?}{=} \texttt{int},\ A_b \overset{?}{=} \texttt{bool},\ A_3 \overset{?}{=} \texttt{int},\ A_f \overset{?}{=} A_b \to A_3,$$
$$\{A_h \longmapsfrom A_f \to (A_b \to A_2),\ A_1 \longmapsfrom A_b \to A_2\}$$

$$\Rightarrow \quad A_b \overset{?}{=} \texttt{bool},\ A_3 \overset{?}{=} \texttt{int},\ A_f \overset{?}{=} A_b \to A_3,$$
$$\{A_h \longmapsfrom A_f \to (A_b \to \underline{\texttt{int}}),\ A_1 \longmapsfrom A_b \to \underline{\texttt{int}},\ A_2 \longmapsfrom \texttt{int}\}$$

Déroulons un exemple

```
let h f b = if b then 52 else (f b)+32
```

on engendre le problème d'unification

$A_h \overset{?}{=} A_f \to A_1,\ A_1 \overset{?}{=} A_b \to A_2,\ A_2 \overset{?}{=} \mathtt{int},\ A_b \overset{?}{=} \mathtt{bool},\ A_3 \overset{?}{=} \mathtt{int},\ A_f \overset{?}{=} A_b \to A_3$

$\Rightarrow \quad A_1 \overset{?}{=} A_b \to A_2,\ A_2 \overset{?}{=} \mathtt{int},\ A_b \overset{?}{=} \mathtt{bool},\ A_3 \overset{?}{=} \mathtt{int},\ A_f \overset{?}{=} A_b \to A_3,\ \{A_h \longmapsfrom A_f \to A_1\}$

$\Rightarrow \quad A_2 \overset{?}{=} \mathtt{int},\ A_b \overset{?}{=} \mathtt{bool},\ A_3 \overset{?}{=} \mathtt{int},\ A_f \overset{?}{=} A_b \to A_3,$
$\{A_h \longmapsfrom A_f \to (A_b \to A_2),\ A_1 \longmapsfrom A_b \to A_2\}$

$\Rightarrow \quad A_b \overset{?}{=} \mathtt{bool},\ A_3 \overset{?}{=} \mathtt{int},\ A_f \overset{?}{=} A_b \to A_3,$
$\{A_h \longmapsfrom A_f \to (A_b \to \underline{\mathtt{int}}),\ A_1 \longmapsfrom A_b \to \underline{\mathtt{int}},\ A_2 \longmapsfrom \mathtt{int}\}$

$\Rightarrow \quad A_3 \overset{?}{=} \mathtt{int},\ A_f \overset{?}{=} \underline{\mathtt{bool}} \to A_3,$
$\{A_h \longmapsfrom A_f \to (\underline{\mathtt{bool}} \to \mathtt{int}),\ A_1 \longmapsfrom \underline{\mathtt{bool}} \to \mathtt{int},\ A_2 \longmapsfrom \mathtt{int},\ A_b \longmapsfrom \mathtt{bool}\}$

Déroulons un exemple

`let h f b = if b then 52 else (f b)+32`

on engendre le problème d'unification

$A_h \stackrel{?}{=} A_f \rightarrow A_1,\ A_1 \stackrel{?}{=} A_b \rightarrow A_2,\ A_2 \stackrel{?}{=} \texttt{int},\ A_b \stackrel{?}{=} \texttt{bool},\ A_3 \stackrel{?}{=} \texttt{int},\ A_f \stackrel{?}{=} A_b \rightarrow A_3$

$\Rightarrow\ \ A_1 \stackrel{?}{=} A_b \rightarrow A_2,\ A_2 \stackrel{?}{=} \texttt{int},\ A_b \stackrel{?}{=} \texttt{bool},\ A_3 \stackrel{?}{=} \texttt{int},\ A_f \stackrel{?}{=} A_b \rightarrow A_3,\ \{A_h \longmapsfrom A_f \rightarrow A_1\}$

$\Rightarrow\ \ A_2 \stackrel{?}{=} \texttt{int},\ A_b \stackrel{?}{=} \texttt{bool},\ A_3 \stackrel{?}{=} \texttt{int},\ A_f \stackrel{?}{=} A_b \rightarrow A_3,$
$\{A_h \longmapsfrom A_f \rightarrow (A_b \rightarrow A_2),\ A_1 \longmapsfrom A_b \rightarrow A_2\}$

$\Rightarrow\ \ A_b \stackrel{?}{=} \texttt{bool},\ A_3 \stackrel{?}{=} \texttt{int},\ A_f \stackrel{?}{=} A_b \rightarrow A_3,$
$\{A_h \longmapsfrom A_f \rightarrow (A_b \rightarrow \underline{\texttt{int}}),\ A_1 \longmapsfrom A_b \rightarrow \underline{\texttt{int}},\ A_2 \longmapsfrom \texttt{int}\}$

$\Rightarrow\ \ A_3 \stackrel{?}{=} \texttt{int},\ A_f \stackrel{?}{=} \underline{\texttt{bool}} \rightarrow A_3,$
$\{A_h \longmapsfrom A_f \rightarrow (\underline{\texttt{bool}} \rightarrow \texttt{int}),\ A_1 \longmapsfrom \underline{\texttt{bool}} \rightarrow \texttt{int},\ A_2 \longmapsfrom \texttt{int},\ A_b \longmapsfrom \texttt{bool}\}$

$\Rightarrow\ \ A_f \stackrel{?}{=} \texttt{bool} \rightarrow \underline{\texttt{int}}$
$\{A_h \longmapsfrom A_f \rightarrow (\texttt{bool} \rightarrow \texttt{int}),\ A_1 \longmapsfrom \texttt{bool} \rightarrow \texttt{int},\ A_2 \longmapsfrom \texttt{int},\ A_b \longmapsfrom \texttt{bool},\ A_3 \longmapsfrom \texttt{int}\}$

Déroulons un exemple

```
let h f b = if b then 52 else (f b)+32
```

on engendre le problème d'unification

$$A_h \overset{?}{=} A_f \to A_1,\ A_1 \overset{?}{=} A_b \to A_2,\ A_2 \overset{?}{=} \mathtt{int},\ A_b \overset{?}{=} \mathtt{bool},\ A_3 \overset{?}{=} \mathtt{int},\ A_f \overset{?}{=} A_b \to A_3$$

$$\Rightarrow\ A_1 \overset{?}{=} A_b \to A_2,\ A_2 \overset{?}{=} \mathtt{int},\ A_b \overset{?}{=} \mathtt{bool},\ A_3 \overset{?}{=} \mathtt{int},\ A_f \overset{?}{=} A_b \to A_3,\ \{A_h \longmapsfrom A_f \to A_1\}$$

$$\Rightarrow\ A_2 \overset{?}{=} \mathtt{int},\ A_b \overset{?}{=} \mathtt{bool},\ A_3 \overset{?}{=} \mathtt{int},\ A_f \overset{?}{=} A_b \to A_3,$$
$$\{A_h \longmapsfrom A_f \to (A_b \to A_2),\ A_1 \longmapsfrom A_b \to A_2\}$$

$$\Rightarrow\ A_b \overset{?}{=} \mathtt{bool},\ A_3 \overset{?}{=} \mathtt{int},\ A_f \overset{?}{=} A_b \to A_3,$$
$$\{A_h \longmapsfrom A_f \to (A_b \to \underline{\mathtt{int}}),\ A_1 \longmapsfrom A_b \to \underline{\mathtt{int}},\ A_2 \longmapsfrom \mathtt{int}\}$$

$$\Rightarrow\ A_3 \overset{?}{=} \mathtt{int},\ A_f \overset{?}{=} \underline{\mathtt{bool}} \to A_3,$$
$$\{A_h \longmapsfrom A_f \to (\underline{\mathtt{bool}} \to \mathtt{int}),\ A_1 \longmapsfrom \underline{\mathtt{bool}} \to \mathtt{int},\ A_2 \longmapsfrom \mathtt{int},\ A_b \longmapsfrom \mathtt{bool}\}$$

$$\Rightarrow\ A_f \overset{?}{=} \mathtt{bool} \to \underline{\mathtt{int}}$$
$$\{A_h \longmapsfrom A_f \to (\mathtt{bool} \to \mathtt{int}),\ A_1 \longmapsfrom \mathtt{bool} \to \mathtt{int},\ A_2 \longmapsfrom \mathtt{int},\ A_b \longmapsfrom \mathtt{bool},\ A_3 \longmapsfrom \mathtt{int}\}$$

$$\Rightarrow\ \emptyset,$$
$$\{A_h \longmapsfrom \underline{(\mathtt{bool} \to \mathtt{int})} \to (\mathtt{bool} \to \mathtt{int}),\ A_1 \longmapsfrom \mathtt{bool} \to \mathtt{int},\ A_2 \longmapsfrom \mathtt{int},\ A_b \longmapsfrom \mathtt{bool},$$
$$A_3 \longmapsfrom \mathtt{int},\ A_f \longmapsfrom \mathtt{bool} \to \mathtt{int}\}$$

Typage des termes *“purs”*

un type pour `g = fun x f -> (f x)` ?

Typage des termes *“purs”*

un type pour `g = fun x f -> (f x)` ?

- si on déroule l'algorithme d'inférence, on trouve $A_g = A_1 \to (A_2 \to A_3)$ avec la contrainte $A_2 = A_1 \to A_3$, d'où le type $A_1 \to (A_1 \to A_3) \to A_3$

Typage des termes *"purs"*

un type pour `g = fun x f -> (f x)` ?

- si on déroule l'algorithme d'inférence, on trouve $A_g = A_1 \rightarrow (A_2 \rightarrow A_3)$ avec la contrainte $A_2 = A_1 \rightarrow A_3$, d'où le type $A_1 \rightarrow (A_1 \rightarrow A_3) \rightarrow A_3$
- *qui sont ces A_1 et A_3 qui 'restent' ?*

Typage des termes *"purs"*

un type pour `g = fun x f -> (f x)` ?

- si on déroule l'algorithme d'inférence, on trouve $A_g = A_1 \rightarrow (A_2 \rightarrow A_3)$ avec la contrainte $A_2 = A_1 \rightarrow A_3$, d'où le type $A_1 \rightarrow (A_1 \rightarrow A_3) \rightarrow A_3$
- *qui sont ces A_1 et A_3 qui 'restent' ?*
  - des variables de type non contraintes

    exemple encore plus évident: `let f x y = y`, $\rightsquigarrow A_x \rightarrow A_y \rightarrow A_y$

Typage des termes *“purs”*

un type pour `g = fun x f -> (f x)` ?

- si on déroule l’algorithme d’inférence, on trouve $A_g = A_1 \rightarrow (A_2 \rightarrow A_3)$ avec la contrainte $A_2 = A_1 \rightarrow A_3$, d’où le type $A_1 \rightarrow (A_1 \rightarrow A_3) \rightarrow A_3$
- *qui sont ces A_1 et A_3 qui ‘restent’ ?*
  - des variables de type non contraintes

    exemple encore plus évident: `let f x y = y`, $\rightsquigarrow A_x \rightarrow A_y \rightarrow A_y$
  - si `g` avait été appliqué à des arguments, A_1 et A_3 auraient pu subir d’autres contraintes

Limites du typage envisagé

- intéressons-nous à

$$\texttt{(fun f -> (}\underbrace{\texttt{f}}_{T_1}\texttt{(32,"hop"))*(}\underbrace{\texttt{f}}_{T_2}\texttt{(52,false)))}\quad\underbrace{\texttt{(fun (u,v) -> u)}}_{T_0}$$

Limites du typage envisagé

- intéressons-nous à

  $\texttt{(fun f -> (}\underbrace{\texttt{ f }}_{T_1}\texttt{(32,"hop"))*(}\underbrace{\texttt{ f }}_{T_2}\texttt{(52,false)))}\quad\underbrace{\texttt{(fun (u,v) -> u)}}_{T_0}$

- on engendre les contraintes, on mélange un peu:

  $T_1 = \texttt{int} * \texttt{string->int} \quad T_1 = T_0$

  $T_2 = \texttt{int} * \texttt{bool->int} \quad T_2 = T_0$

  $T_0 = T_u * T_v \rightarrow$

  T_u

Limites du typage envisagé

- intéressons-nous à

$$\texttt{(fun f -> (}\underbrace{\texttt{ f }}_{T_1}\texttt{(32,"hop"))*(}\underbrace{\texttt{ f }}_{T_2}\texttt{(52,false)))}\quad\underbrace{\texttt{(fun (u,v) -> u)}}_{T_0}$$

- on engendre les contraintes, on mélange un peu:

$$T_1 = \texttt{int} * \texttt{string->int} \quad T_1 = T_0 \qquad T_0 = T_u * T_v \to$$
$$T_2 = \texttt{int} * \texttt{bool->int} \quad T_2 = T_0$$
$$T_u$$

  - conflit de ressource: T_1 et T_2 ‘veulent’ instancier T_u et T_v
  - d’ailleurs ça ne type pas en Caml

Limites du typage envisagé

- intéressons-nous à

  $$\texttt{(fun f -> (}\underbrace{\texttt{ f }}_{T_1}\texttt{(32,"hop"))*(}\underbrace{\texttt{ f }}_{T_2}\texttt{(52,false)))}\quad\underbrace{\texttt{(fun (u,v) -> u)}}_{T_0}$$

- on engendre les contraintes, on mélange un peu:

  $T_1 = \texttt{int} * \texttt{string->int} \quad T_1 = T_0$

  $T_2 = \texttt{int} * \texttt{bool->int} \quad T_2 = T_0$

  $T_0 = T_u * T_v \rightarrow$

  T_u

  - conflit de ressource: T_1 et T_2 ‘veulent’ instancier T_u et T_v
  - d’ailleurs ça ne type pas en Caml
- on voudrait avoir le droit de donner un type *générique* que l’on puisse *instancier* plusieurs fois
  - une instanciation par utilisation de `f` sur l’exemple

Limites du typage envisagé

- intéressons-nous à

  $$\texttt{(fun f -> (}\underbrace{\texttt{ f }}_{T_1}\texttt{(32,"hop"))*(}\underbrace{\texttt{ f }}_{T_2}\texttt{(52,false)))}\quad\underbrace{\texttt{(fun (u,v) -> u)}}_{T_0}$$

- on engendre les contraintes, on mélange un peu:

  $T_1 = \texttt{int} * \texttt{string->int} \quad T_1 = T_0 \qquad T_0 = T_u * T_v \to$
  $T_2 = \texttt{int} * \texttt{bool->int} \quad T_2 = T_0$
  T_u

  - conflit de ressource: T_1 et T_2 ‘veulent’ instancier T_u et T_v
  - d’ailleurs ça ne type pas en Caml
- on voudrait avoir le droit de donner un type *générique* que l’on puisse *instancier* plusieurs fois
  - une instanciation par utilisation de `f` sur l’exemple
- jusque là les types étaient *monomorphes*, on veut le **polymorphisme**