

Inférence de type

`Luc.Maranget@inria.fr`

`http://www.enseignement.polytechnique.fr/profs/
informatique/Luc.Maranget/TLP/`

A Typage implicite.

B Inférence de type.

C Unification.

Types implicites

Il est très pénible de devoir écrire

Fun (x:Nat) -> x+1 Fun (x:Nat) -> Fun (y:Nat) -> x+y

Alors que les types sont ici « évidents ». Si on écrit

Fun x -> x+1 Fun x -> Fun y -> x+y

Il est clair que l'on peut facilement deviner que **x** et **y** doivent être de type **Nat**.

On peut ensuite vérifier le typage comme il y a quinze jours.

Typage implicite (à la Curry)

On préfère deviner et vérifier en même temps.

Les règles,

$$\frac{E \oplus [x : A] \vdash t : B}{E \vdash (\mathbf{Fun} (x : A) \rightarrow t) : A \rightarrow B}$$

$$\frac{E \oplus [x : A] \vdash t : A}{E \vdash (\mathbf{Fix} (x : A) \rightarrow t) : A}$$

deviennent

$$\frac{E \oplus [x : A] \vdash t : B}{E \vdash (\mathbf{Fun} x \rightarrow t) : A \rightarrow B}$$

$$\frac{E \oplus [x : A] \vdash t : A}{E \vdash (\mathbf{Fix} x \rightarrow t) : A}$$

Et c'est tout !

Pour être complet

Voici les autres règles qui ne changent pas.

$$E \vdash n : \mathbf{Nat} \qquad \frac{E(x) = A}{E \vdash x : A} \qquad \frac{E \vdash t_1 : \mathbf{Nat} \quad E \vdash t_2 : \mathbf{Nat}}{E \vdash t_1 \text{ op } t_2 : \mathbf{Nat}}$$

$$\frac{E \vdash t_1 : \mathbf{Nat} \quad E \vdash t_2 : A \quad E \vdash t_3 : A}{E \vdash \mathbf{Ifz } t_1 \mathbf{ Then } t_2 \mathbf{ Else } t_3 : A}$$

$$\frac{E \vdash t_1 : A \rightarrow B \quad E \vdash t_2 : A}{E \vdash t_1 t_2 : B}$$

$$\frac{E \vdash t_1 : A \quad E \oplus [x : A] \vdash t_2 : B}{E \vdash \mathbf{Let } x = t_1 \mathbf{ In } t_2 : B}$$

En effet, il n'y a rien à deviner ici (cf. le **Let**).

Une vraie relation

Contrairement au typage explicite (à la Church).

On peut avoir $E \vdash t : A_1$ et $E \vdash t : A_2$ pour $A_1 \neq A_2$.

Par ex.

$$\frac{[x : \mathbf{Nat}] \vdash x : \mathbf{Nat}}{\vdash (\mathbf{Fun} \ x \ -> \ x) : \mathbf{Nat} \ -> \ \mathbf{Nat}}$$

Et

$$\frac{[x : \mathbf{Nat} \ -> \ \mathbf{Nat}] \vdash x : \mathbf{Nat} \ -> \ \mathbf{Nat}}{\vdash (\mathbf{Fun} \ x \ -> \ x) : (\mathbf{Nat} \ -> \ \mathbf{Nat}) \ -> \ \mathbf{Nat} \ -> \ \mathbf{Nat}}$$

Et à vrai dire, pour tout type A , on a facilement

$$\frac{[x : A] \vdash x : A}{\vdash (\mathbf{Fun} \ x \ -> \ x) : A \ -> \ A}$$

Exemple

Fun $x \rightarrow x+1$ est bien de type $\text{Nat} \rightarrow \text{Nat}$.

$$\frac{\frac{[x : \text{Nat}] \vdash x : \text{Nat} \quad [x : \text{Nat}] \vdash 1 : \text{Nat}}{[x : \text{Nat}] \vdash x+1 : \text{Nat}}}{\vdash (\mathbf{Fun} \ x \rightarrow x+1) : \text{Nat} \rightarrow \text{Nat}}$$

- ▶ On cherche à typer la fonction.
- ▶ On devine le bon type pour x (Nat).
- ▶ On cherche à typer $x+1$.
- ▶ On type les arguments de l'addition.
- ▶ On vérifie l'addition.
- ▶ On construit le type de la fonction.

Mais il y a encore une étape magique : deviner $[x : \text{Nat}]$.

Variable de type

Fun $x \rightarrow x+1$ est de type ? $X \rightarrow \mathbf{Nat}$, avec $[X = \mathbf{Nat}]$.

$$\frac{\frac{[x : X] \vdash x : X \quad [x : X] \vdash 1 : \mathbf{Nat}}{[x : X] \vdash x+1 : \mathbf{Nat}, [X = \mathbf{Nat}]}}{\vdash (\mathbf{Fun} \ x \rightarrow x+1) : X \rightarrow \mathbf{Nat}, [X = \mathbf{Nat}]}$$

- ▶ On cherche à typer la fonction.
- ▶ On pose que le type de x est X
- ▶ On cherche à typer l'addition.
- ▶ Premier argument, second argument.
- ▶ Pour typer d'addition il faut $[X = \mathbf{Nat}]$.
- ▶ On type la fonction.

Idée générale de l'inférence (synthèse) de type

- ▶ Procéder à un essai de vérification, en
 - ▷ Introduisant des variables,

$$\frac{E \oplus [x : X] \vdash t : A}{E \vdash (\mathbf{Fun} \ x \ -> \ t) : X \ -> \ A}$$

- ▷ Et posant des équations.

$$\frac{E \vdash t_1 : A_1 \quad E \vdash t_2 : A_2}{E \vdash t_1 + t_2 : \mathbf{Nat}, [A_1 = \mathbf{Nat}, A_2 = \mathbf{Nat}]}$$

- ▶ À la fin, on a un type contenant des variables et des équations.

Retour sur la vérification

Les types sont donc changés, ils peuvent contenir des variables.

$$\text{Nat} \in \mathcal{T} \qquad X \in \mathcal{T} \qquad \frac{A_1 \in \mathcal{T} \quad A_2 \in \mathcal{T}}{A_1 \rightarrow A_2 \in \mathcal{T}}$$

Avec les nouveaux types on a :

$$\frac{[x : X] \vdash x : X}{\vdash (\mathbf{Fun} \ x \rightarrow x) : X \rightarrow X}$$

Quel sens donner à un tel type ?

$\text{id} : X \rightarrow X$ veut dire que $\text{id} : A \rightarrow A$ pour tout type A .

Et en effet on peut donner les types suivants à $\mathbf{Fun} \ x \rightarrow x$.

- ▶ $\text{Nat} \rightarrow \text{Nat}$, $(\text{Nat} \rightarrow \text{Nat}) \rightarrow \text{Nat} \rightarrow \text{Nat}$, etc.
- ▶ Et même $(Y \rightarrow Z) \rightarrow Y \rightarrow Z$, mais pas $Y \rightarrow \text{Nat}$.

Variables dans les types

Sémantique :

Si X apparaît dans A , type de t , alors $[X \mapsto B]A$ est aussi un type de t pour tout type B .

Note : $[X \mapsto B]$ est une nouvelle notation pour la substitution, cette fois appliquée aux types.

Rappels/Remarques.

- ▶ Substitution : fonction des variables dans les termes.
- ▶ Facilement, étendue aux termes : $\sigma(A \rightarrow B) = \sigma(A) \rightarrow \sigma(B)$.
- ▶ Substitutions de support fini $\{X \mid \sigma(X) \neq X\}$.
- ▶ Pas de problèmes de capture ici.

Un sens pour les variables de type

Si X apparaît dans A , type de t , alors $[X \mapsto B]A$ est aussi un type de t pour tout type B .

Définition justifiée par le lemme (facile).

Soit σ une substitution, alors $E \vdash t : A \Rightarrow \sigma(E) \vdash t : \sigma(A)$.

(Avec bien entendu $\sigma(E)$ défini comme associant $\sigma(E(x))$ à x .)

Autre exemple d'inférence

$$\begin{array}{c}
 [f : F, x : X] \vdash f : F \quad [f : F, x : X] \vdash x : X \\
 \hline
 [f : F, x : X] \vdash f \ x : Y \quad [f : F, x : X] \vdash x : X \\
 \hline
 [f : F, x : X] \vdash f \ x+x : \mathbf{Nat} \\
 \hline
 [f : F] \vdash (\mathbf{Fun} \ x \rightarrow f \ x+x) : X \rightarrow \mathbf{Nat} \\
 \hline
 \vdash (\mathbf{Fun} \ f \rightarrow \mathbf{Fun} \ x \rightarrow f \ x+x) : F \rightarrow X \rightarrow \mathbf{Nat}
 \end{array}$$

- ▶ On cherche à typer les deux **Fun**
- ▶ On cherche à typer $f \ x+x$, puis l'application $f \ x$.
- ▶ On a maintenant le type de toutes les feuilles.
- ▶ Le type de $f \ x$ est Y avec l'équation $[F = X \rightarrow Y]$.
- ▶ Le type de $f \ x+x$ est \mathbf{Nat} , avec $[Y = \mathbf{Nat}, X = \mathbf{Nat}]$.
- ▶ Type final, avec les équations $[F = X \rightarrow Y, Y = \mathbf{Nat}, X = \mathbf{Nat}]$.

Résultat de l'inférence

Le type :

$$\vdash (\mathbf{Fun} \mathbf{f} \rightarrow \mathbf{Fun} \mathbf{x} \rightarrow \mathbf{f} \mathbf{x}+\mathbf{x}) : \mathbf{F} \rightarrow \mathbf{X} \rightarrow \mathbf{Nat}$$

Avec les équations :

$$[\mathbf{F} = \mathbf{X} \rightarrow \mathbf{Y}, \mathbf{Y} = \mathbf{Nat}, \mathbf{X} = \mathbf{Nat}]$$

On résout (en substituant X et Y par leurs valeurs) :

$$[\mathbf{X} \mapsto \mathbf{Nat}, \mathbf{Y} \mapsto \mathbf{Nat}, \mathbf{F} \mapsto \mathbf{Nat} \rightarrow \mathbf{Nat}]$$

On substitue dans le type :

$$\vdash \mathbf{Fun} \mathbf{f} \rightarrow \mathbf{Fun} \mathbf{x} \rightarrow \mathbf{f} \mathbf{x}+\mathbf{x} : (\mathbf{Nat} \rightarrow \mathbf{Nat}) \rightarrow \mathbf{Nat} \rightarrow \mathbf{Nat}$$

Nous allons systématiser l'algorithme d'inférence de type.

Production des équations (et du type)

À un environnement de typage E et un terme t , on associe un type A et un ensemble (une liste, peu important les doublons) d'équations.

$$E \vdash t \rightsquigarrow A, \mathcal{E}$$

Avec les notations de définition de prédicat par induction.

$$\frac{E(X) = A}{E \vdash x \rightsquigarrow A, \emptyset}$$

$$\frac{E \vdash t_1 \rightsquigarrow A_1, \mathcal{E}_1 \quad E \vdash t_2 \rightsquigarrow A_2, \mathcal{E}_2}{E \vdash t_1 t_2 \rightsquigarrow X, \mathcal{E}_1 \cup \mathcal{E}_2 \cup [A_1 = A_2 \rightarrow X]} (X \text{ frais})$$

$$\frac{E \oplus [x : X] \vdash t \rightsquigarrow A, \mathcal{E}}{E \vdash (\mathbf{Fun} \ x \rightarrow t) \rightsquigarrow X \rightarrow A, \mathcal{E}} (X \text{ frais})$$

Production des équations II

Les opérations sur les entiers.

$$E \vdash n \rightsquigarrow \mathbf{Nat}, \emptyset$$

$$\frac{E \vdash t_1 \rightsquigarrow A_1, \mathcal{E}_1 \quad E \vdash t_2 \rightsquigarrow A_2, \mathcal{E}_2}{E \vdash t_1 \text{ op } t_2 \rightsquigarrow \mathbf{Nat}, \mathcal{E}_1 \cup \mathcal{E}_2 \cup [A_1 = \mathbf{Nat}, A_2 = \mathbf{Nat}]}$$

$$\frac{E \vdash t_1 \rightsquigarrow A_1, \mathcal{E}_1 \quad E \vdash t_2 \rightsquigarrow A_2, \mathcal{E}_2 \quad E \vdash t_3 \rightsquigarrow A_3, \mathcal{E}_3}{E \vdash \mathbf{Ifz } t_1 \mathbf{ Then } t_2 \mathbf{ Else } t_3 \rightsquigarrow A_2, \mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}_3 \cup [A_1 = \mathbf{Nat}, A_2 = A_3]}$$

Équations III

$$\frac{E \oplus [x : X] \vdash t \rightsquigarrow A, \mathcal{E}}{E \vdash (\mathbf{Fix} \ x \ -> \ t) \rightsquigarrow A, \mathcal{E} \cup [X = A]} \text{ (} X \text{ frais)}$$

$$\frac{E \vdash t_1 \rightsquigarrow A_1, \mathcal{E}_1 \quad E \oplus [x : A_1] \vdash t_2 \rightsquigarrow A_2, \mathcal{E}_2}{E \vdash (\mathbf{Let} \ x = t_1 \ \mathbf{In} \ t_2) \rightsquigarrow A_2, \mathcal{E}_1 \cup \mathcal{E}_2}$$

La relative complexité des notations cache un peu que l'on définit en fait une fonction :

- ▶ Arguments : E et t .
- ▶ Résultat : la paire (A, \mathcal{E}) .
- ▶ Technique : induction sur la structure de t .

Exemple

À typer $\mathbf{Fun} \ f \rightarrow 2+f \ 1$.

Type : $\mathbf{F} \rightarrow \mathbf{Nat}$.

Équations : $[\mathbf{F} = (\mathbf{Nat} \rightarrow \mathbf{Y}), \mathbf{Y} = \mathbf{Nat}, \mathbf{Nat} = \mathbf{Nat}]$.

Solution : $[\mathbf{Y} \mapsto \mathbf{Nat}, \mathbf{F} \mapsto (\mathbf{Nat} \rightarrow \mathbf{Nat})]$.

Type : $(\mathbf{Nat} \rightarrow \mathbf{Nat}) \rightarrow \mathbf{Nat}$.

Résoudre

Équations du type $\mathcal{E} = \{ A_1 = B_1, \dots, A_n = B_n \}$, avec un langage de termes simple (sans variables liées).

Une solution : une substitution σ tq, $\sigma(A_i) = \sigma(B_i)$. On note les substitutions :

$$[X_1 \mapsto C_1, \dots, X_m \mapsto C_m]$$

Et on impose des X_i distincts, tq. X_i n'apparaît dans aucun C_j (substitution idempotente $\sigma \circ \sigma = \sigma$).

Retour sur l'exemple : $[F = \mathbf{Nat} \rightarrow Y, Y = \mathbf{Nat}, \mathbf{Nat} = \mathbf{Nat}]$.

Solution facile : $[Y \mapsto \mathbf{Nat}, F \mapsto (\mathbf{Nat} \rightarrow \mathbf{Nat})]$.

Pas de solution

À typer : **Fun** $f \rightarrow f \ 1+f$.

Equations : $[F = \mathbf{Nat} \rightarrow Y, Y = \mathbf{Nat}, F = \mathbf{Nat}]$.

Aucune valeur possible pour F car $\mathbf{Nat} \rightarrow Y \neq \mathbf{Nat}$ pour tout Y .

C'est donc la résolution qui détecte les erreurs, avec un message du style :

```
% ocaml
      Objective Caml version 3.10.0

# (fun f -> f 1 + f);;
      ^

This expression has type int -> int
but is here used with type int
```

Pas de solution

À typer : **Fun** $x \rightarrow x\ x$.

Équations : $[X = X \rightarrow Y]$.

Aucune valeur possible pour X , car les types sont des termes finis.

La résolution detecte un autre style d'erreur, avec :

```
% ocaml
      Objective Caml version 3.10.0

# (fun x -> x x);;
      ^

This expression has type 'a -> 'b
but is here used with type 'a
```

Plusieurs solutions

Terme : $\mathbf{Fun} \ f \rightarrow \mathbf{Fun} \ x \rightarrow f \ x.$

Equations : $[F = X \rightarrow Y].$

Une solution $\phi : [X \mapsto \mathbf{Nat}, F \mapsto (\mathbf{Nat} \rightarrow \mathbf{Nat}), Y \mapsto \mathbf{Nat}].$

Une autre $\sigma : [F \mapsto X \rightarrow Y]$ (il reste des variables !).

Quelle solution ? σ car plus générale (*principale*).

La solution = une solution *principale* : ϕ solution, entraîne il existe ψ avec $\phi = \psi \circ \sigma.$

Ici $\psi = [X \mapsto \mathbf{Nat}, Y \mapsto \mathbf{Nat}].$

En effet,

$$\psi(\sigma(F)) = \psi(X \rightarrow Y) = \mathbf{Nat} \rightarrow \mathbf{Nat}$$

$$\psi(\sigma(X)) = \psi(X) = \mathbf{Nat} \qquad \psi(\sigma(Y)) = \psi(Y) = \mathbf{Nat}$$

$$\psi(\sigma(Z)) = \psi(Z) = Z$$

Plusieurs solutions

Terme : **Fun** b -> **Fun** x -> **Fun** y -> **Ifz** b **Then** x **Else** y.

Type : B -> X -> Y -> X.

Equations : [B = **Nat**, X = Y].

Une solution σ : [B \mapsto **Nat**, X \mapsto Y].

Une autre solution σ' : [B \mapsto **Nat**, Y \mapsto X].

Les deux solutions sont principales :

$$\sigma' = [Y \mapsto X] \circ \sigma \qquad \sigma = [X \mapsto Y] \circ \sigma'$$

Algorithme de Robinson

Produit une solution principale σ d'un système \mathcal{E} .

Informellement : par réécriture de l'ensemble \mathcal{E} , jusqu'à ce qu'il soit une substitution (idempotente).

On choisit une équation de \mathcal{E} non-encore examinée.

1. $A \rightarrow B = C \rightarrow D$, remplacer par $[A = C, B = D]$.
2. $\text{Nat} = \text{Nat}$ ou $X = X$, supprimer.
3. $\text{Nat} = A \rightarrow B$ (ou symétrique), échouer.
4. $X = A$ (ou symétrique), où X apparaît dans A , échouer.
5. $X = A$ (ou symétrique) et X n'apparaît pas dans A , remplacer X par A dans les autres équations.

Terminaison de l'algorithme de Robinson

À l'étape 5. on garde l'équation $X = A$ qui est maintenant examinée. La variable X est *résolue*, elle n'apparaît plus dans les équations non-examinées.

Soit la paire d'entiers naturels $(\mathcal{V}(E), |\mathcal{E}|)$.

- ▶ $\mathcal{V}(E)$ nombre de variables non-résolues.
- ▶ $|\mathcal{E}|$ taille des équations.

L'algo termine, car $(\mathcal{V}(E), |\mathcal{E}|)$ décroît strictement (ordre lexicographique).

- ▶ Cas 1. et 2.
 - ▷ $\mathcal{V}(E)$ stable (peut décroître si $X = X$ unique équation qui contient X),
 - ▷ $|\mathcal{E}|$ décroît.
- ▶ Cas 5. $\mathcal{V}(E)$ décroît.

Correction

L'algorithme produit un système de la forme :

$$\text{mgu}(\mathcal{E}) = [X_1 = C_1, \dots, X_m = C_m]$$

Où les X_i sont deux à deux distincts et n'apparaissent pas dans les C_j . Autrement dit une substitution (idempotente)

$$[X_1 \mapsto C_1, \dots, X_m \mapsto C_m]$$

1. Si $\text{mgu}(\mathcal{E})$ n'échoue pas, il produit une solution de \mathcal{E} .
2. Si il existe une solution de \mathcal{E} , alors $\text{mgu}(\mathcal{E})$ est une solution principale de \mathcal{E} .

Algo de synthèse complet

Synthèse du type de t . En deux étapes :

- ▶ Calculer $\vdash t \rightsquigarrow A, \mathcal{E}$.
- ▶ Puis, résoudre \mathcal{E} , ce qui donne σ

Correction de la synthèse

1. (Correction) Si pas d'échec $\vdash t : \sigma(A)$.
2. (Complétude) Si $\vdash t : B$, alors il existe ψ , tq. $B = \psi(\sigma(A))$
($\sigma(A)$ type principal).

Le **Fix** n'a rien de mystérieux

Fix pow \rightarrow **Fun** n \rightarrow **Ifz** n **Then** 1 **Else** 2*pow (n-1)

Soit P type de pow et N type de n.

- ▶ De (n-1), il vient : $[N = \mathbf{Nat}]$ (et aussi $[\mathbf{Nat} = \mathbf{Nat}]$, mais bon). Type **Nat**.
- ▶ De pow (n-1), il vient : $[P = (\mathbf{Nat} \rightarrow Y)]$. Type Y.
- ▶ De 2*pow (n-1), il vient : $[Y = \mathbf{Nat}]$. Type **Nat**.
- ▶ De **Ifz**..., il vient : $[N = \mathbf{Nat}]$ (et aussi $[\mathbf{Nat} = \mathbf{Nat}]$). Type **Nat**.
- ▶ Le type de **Fun** n \rightarrow ... est : $N \rightarrow \mathbf{Nat}$.
- ▶ Le type de **Fix** pow \rightarrow ... est : $N \rightarrow \mathbf{Nat}$, avec l'équation : $[P = (N \rightarrow \mathbf{Nat})]$.

Point fixe

Fix pow -> **Fun** n -> **Ifz** n **Then** 1 **Else** 2*pow (n-1)

Équations,

[N = Nat, P = (Nat -> Y), Y = Nat, N = Nat, P = (N -> Nat)]

Type N -> Nat.

Il n'y a pas « *trop* » d'équations, chaque équation correspond à une vérification indispensable.

Par ex. la dernière équation contrôle le bon usage de **pow** relativement à sa définition.

Fix pow -> **Fun** n -> **Ifz** n **Then** 1 **Else** 2*pow

[N = Nat, P = Nat, P = (N -> Nat)]

Résolution

$[N = \text{Nat}, P = (\text{Nat} \rightarrow Y), Y = \text{Nat}, N = \text{Nat}, P = (N \rightarrow \text{Nat})]$

► $[N \mapsto \text{Nat}]$ et

$[P = (\text{Nat} \rightarrow Y), Y = \text{Nat}, \text{Nat} = \text{Nat}, P = (\text{Nat} \rightarrow \text{Nat})]$.

► $[N \mapsto \text{Nat}, P \mapsto (\text{Nat} \rightarrow Y)]$ et

$[Y = \text{Nat}, \text{Nat} = \text{Nat}, (\text{Nat} \rightarrow Y) = (\text{Nat} \rightarrow \text{Nat})]$.

► $[N \mapsto \text{Nat}, P \mapsto (\text{Nat} \rightarrow \text{Nat}), Y \mapsto \text{Nat}]$ et

$[\text{Nat} = \text{Nat}, (\text{Nat} \rightarrow \text{Nat}) = (\text{Nat} \rightarrow \text{Nat})]$

► ...

► $[N \mapsto \text{Nat}, P \mapsto (\text{Nat} \rightarrow \text{Nat}), Y \mapsto \text{Nat}]$

Type du point fixe : $N \rightarrow \text{Nat}$ soit $\text{Nat} \rightarrow \text{Nat}$.

L'unification de ML

Il y a deux différences importantes.

- ▶ Le polymorphisme.
- ▶ L'unification est incrémentale (et en place).

Polymorphisme

Notre système de type ne sait pas typer

```
Let id = Fun x -> x In  
id id
```

► id a le type (principal) $X \rightarrow X$.

► Soit l'équation

$$[(X \rightarrow X) = ((X \rightarrow X) \rightarrow Y)]$$

Et donc

$$[X = (X \rightarrow X), X = Y]$$

Qui n'a pas de solution.

Nous verrons cela la semaine prochaine.

Unification en place

Les types sont en fait un seul graphe que la résolution modifie.

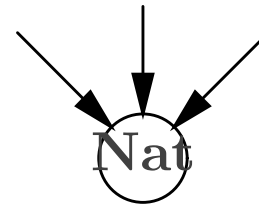
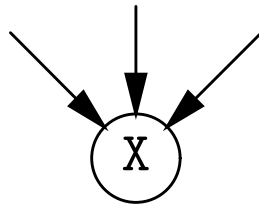
Résoudre :

[... , $X = \mathbf{Nat}$, ...]

C'est remplacer X par \mathbf{Nat} partout.

Possible en une opération sur un graphe :

\Rightarrow



La résolution immédiate

Au lieu d'accumuler les équations puis de les résoudre.

Les résoudre dès qu'elles sont posées.

Quel intérêt : détection plus précoce de l'échec, et messages d'erreurs en rapport avec le source.

```
# (fun f -> f 1 + f);;  
      ^
```

This expression has type `int -> int`
but is here used with type `int`

Mais aussi plus proche de l'unification en place pratiquée par ML.

Une règle de résolution incrémentale

On avait :

$$\frac{E \vdash t_1 \rightsquigarrow A_1, \mathcal{E}_1 \quad E \vdash t_2 \rightsquigarrow A_2, \mathcal{E}_2}{E \vdash t_1 t_2 \rightsquigarrow X, \mathcal{E}_1 \cup \mathcal{E}_2 \cup [A_1 = A_2 \rightarrow X]} (X \text{ frais})$$

On a :

$$\frac{E, \sigma \vdash t_1 \rightsquigarrow A_1, \sigma_1 \quad E, \sigma_1 \vdash t_2 \rightsquigarrow A_2, \sigma_2}{E, \sigma \vdash t_1 t_2 \rightsquigarrow X, \text{mgu}[A_1 = (A_2 \rightarrow X), \sigma_2]} (X \text{ frais})$$

La forme générale du jugement est

$$E, \sigma \vdash t \rightsquigarrow A, \sigma'$$

Le composant σ représente la solution courante des équations

« vues ».

Propriété : σ' étend σ ($\sigma' = \sigma'' \circ \sigma$) et $\sigma'(E) \vdash t : \sigma'(A)$ (et même type principal).

L'unification incrémentale

Soit σ la solution d'un problème d'unification. C'est à dire $\sigma = \{ X_1 \mapsto A_1, \dots, X_n \mapsto A_n \}$, avec :

- ▶ Les X_i sont deux à deux distincts.
- ▶ Les X_i n'apparaissent pas dans les A_i .

Et soit une nouvelle équation $B = C$.

On veut résoudre les équation $\mathcal{E} = \{ B = C \} \cup \sigma$.

On pourrait simplement poser $\text{mgu}(\mathcal{E})$, mais ça serait dommage d'oublier que σ est déjà résolu.

Une vue incrémentale de l'unification

Pour résoudre $\{ B = C, X_1 \mapsto A_1, \dots, X_n \mapsto A_n \}$.

1. Remplacer les X_i par les A_i dans B et C — les X_i n'apparaissent plus.
2. Résoudre la nouvelle équation : $\sigma' = \{ Y_1 \mapsto B_1, \dots, Y_m \mapsto B_m \}$ — les X_i n'apparaissent ni dans les Y_j , ni dans les B_j .
3. Remplacer les Y_i par les B_i dans les A_j , ce qui donne les C_j — les Y_i n'apparaissent plus.
4. Renvoyer $\{ Y_1 \mapsto B_1, \dots, Y_m \mapsto B_m, X_1 \mapsto C_1, \dots, X_n \mapsto C_m \}$.

Cela s'abrège en $\text{mgu}[\sigma(B) = \sigma(C)] \circ \sigma$.

La notation \circ exprimant les étapes 3. et 4. mais de façon trop générale (quid si $X_i = Y_j$?).

Toutes les règles

$$\frac{E, \sigma \vdash t_1 \rightsquigarrow A, \sigma_1 \quad E \oplus [x = A], \sigma \vdash t_2 \rightsquigarrow B, \sigma_2}{E, \sigma \vdash \mathbf{Let} \ x = t_1 \ \mathbf{In} \ t_2 \rightsquigarrow B, \sigma_2} \qquad \frac{E(x) = A}{E, \sigma \vdash x \rightsquigarrow A, \sigma}$$

$$\frac{E \oplus [x = X], \sigma \vdash t \rightsquigarrow A, \sigma'}{E, \sigma \vdash (\mathbf{Fun} \ x \rightarrow t) \rightsquigarrow X \rightarrow A, \sigma'} (X \text{ frais})$$

$$\frac{E, \sigma \vdash t_1 \rightsquigarrow A, \sigma_1 \quad E, \sigma_1 \vdash t_2 \rightsquigarrow B, \sigma_2}{E, \sigma \vdash t_1 \ t_2 \rightsquigarrow X, \text{mgu}[\sigma_2(A) = \sigma_2(B \rightarrow X)] \circ \sigma_2} (X \text{ frais})$$

$$\frac{E \oplus [x = X], \sigma \vdash t \rightsquigarrow A, \sigma'}{E, \sigma \vdash (\mathbf{Fix} \ x \rightarrow t) \rightsquigarrow A, \text{mgu}[\sigma'(X) = \sigma'(A)] \circ \sigma'} (X \text{ frais})$$

Vraiment toutes les règles

$$E, \sigma \vdash n \rightsquigarrow \mathbf{Nat}, \sigma$$

$$\frac{\begin{array}{l} E, \sigma \vdash t_1 \rightsquigarrow A_1, \sigma_1 \quad \sigma'_1 = \text{mgu}[\sigma_1(A_1) = \mathbf{Nat}] \circ \sigma_1 \\ E, \sigma'_1 \vdash t_2 \rightsquigarrow A_2, \sigma_2 \quad \sigma'_2 = \text{mgu}[\sigma_2(A_2) = \mathbf{Nat}] \circ \sigma_2 \end{array}}{E, \sigma \vdash t_1 \text{ op } t_2 \rightsquigarrow \mathbf{Nat}, \sigma'_2}$$

$$\frac{\begin{array}{l} E, \sigma \vdash t_1 \rightsquigarrow A_1, \sigma_1 \quad \sigma'_1 = \text{mgu}[\sigma_1(A_1) = \mathbf{Nat}] \circ \sigma_1 \\ E, \sigma'_1 \vdash t_2 \rightsquigarrow A_2, \sigma_2 \quad E, \sigma_2 \vdash t_3 \rightsquigarrow A_3, \sigma_3 \quad \sigma'_3 = \text{mgu}[\sigma_3(A_2) = \sigma_3(A_3)] \circ \sigma_3 \end{array}}{E, \sigma \vdash \mathbf{Ifz } t_1 \mathbf{ Then } t_2 \mathbf{ Else } t_3 \rightsquigarrow A_2, \sigma'_3}$$

Une fois encore, ces règles expriment une fonction,

$$E, \sigma \vdash t \rightsquigarrow A, \sigma'$$

dont les arguments sont E, σ, t , et dont le résultat est (A, σ') .

Un exemple

Fix $\text{pow} \rightarrow$ **Fun** $n \rightarrow$ **Ifz** n **Then** 1 **Else** $2 * \text{pow} \ (n-1)$

pow est de type P , n est de type N (E est $[x : X, \text{pow} : P]$).

► De **Ifz**, il vient d'abord, $\sigma = [N \mapsto \mathbf{Nat}]$, puis.

▷ Pour le **Then** E , $\sigma \vdash 1 \rightsquigarrow \mathbf{Nat}, \sigma$.

▷ Et le **Else**

★ D'abord $E, \sigma \vdash 2 \rightsquigarrow \mathbf{Nat}, \sigma$,

★ Puis $n-1$ donne $\text{mgu}[\sigma(N) = \mathbf{Nat}] \circ \sigma$ qui ne change rien

★ Puis surtout (application $\text{pow} \ (n-1)$)

$\text{mgu}[\sigma(P) = \sigma(\mathbf{Nat} \rightarrow Y)] \circ \sigma$. Qui vaut

$\sigma' = [P \mapsto (\mathbf{Nat} \rightarrow Y), X \mapsto \mathbf{Nat}]$.

★ Et enfin le deuxième argument de « $*$ » conduit au calcul

de $\text{mgu}[\sigma'(Y) = \mathbf{Nat}] \circ \sigma'$ qui est

$$\sigma'' = [Y \mapsto \mathbf{Nat}, P \mapsto (\mathbf{Nat} \rightarrow \mathbf{Nat}), N \mapsto \mathbf{Nat}]$$

Nous avons donc le bilan du typage du **Ifz**

$$E, \text{id} \vdash (\mathbf{Ifz} \ n \ \mathbf{Then} \ 1 \ \mathbf{Else} \ 2 * \text{pow} \ (n-1)) \rightsquigarrow \mathbf{Nat}, \sigma''$$

- ▶ Le type de **Fun** est $N \rightarrow \mathbf{Nat}$,
- ▶ C'est donc le type de **Fix**, avec à calculer

$$\text{mgu}[\sigma''(P) = \sigma''(N \rightarrow \mathbf{Nat})] \circ \sigma''$$

L'équation est triviale, et la substitution ne change pas.

- ▶ TP, inférence de type, puis unification.
- ▶ La prochaine fois, polymorphisme.