INF 544 Luc Maranget

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,

$$\frac{E \oplus [x:A] \vdash t : B}{E \vdash (\textbf{Fun}\ (x:A)\ \texttt{->}\ t) : A\ \texttt{->}\ B} \qquad \frac{E \oplus [x:A] \vdash t : A}{E \vdash (\textbf{Fix}\ (x:A)\ \texttt{->}\ t) : A}$$

deviennent

$$\frac{E \oplus [x:A] \vdash t : B}{E \vdash (\textbf{Fun}\ x\ \texttt{->}\ t) : A\ \texttt{->}\ B} \qquad \frac{E \oplus [x:A] \vdash t : A}{E \vdash (\textbf{Fix}\ x\ \texttt{->}\ t) : A}$$

Et c'est tout !

Pour être complet

Voici les autres règles qui ne changent pas.

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

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

$$\frac{E \vdash t_1 : A \texttt{ -> } B \qquad E \vdash t_2 : A}{E \vdash t_1\ t_2 : B}$$

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

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.

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

Et

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

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

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

Exemple

Fun `x -> x+1` est bien de type **Nat** `->` **Nat**.

$$\frac{\dfrac{[\texttt{x} : \mathbf{Nat}] \vdash \texttt{x} : \mathbf{Nat} \qquad [\texttt{x} : \mathbf{Nat}] \vdash \texttt{1} : \mathbf{Nat}}{[\texttt{x} : \mathbf{Nat}] \vdash \texttt{x+1} : \mathbf{Nat}}}{\vdash (\mathbf{Fun}\ \texttt{x -> x+1}) : \mathbf{Nat} \texttt{ -> } \mathbf{Nat}}$$

- 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 $[\texttt{x} : \mathbf{Nat}]$.

Variable de type

Fun `x -> x+1` est de type ? `X` `->` **Nat**, avec $[\texttt{X} = \mathbf{Nat}]$.

$$\frac{\dfrac{[\texttt{x} : \texttt{X}] \vdash \texttt{x} : \texttt{X} \qquad [\texttt{x} : \texttt{X}] \vdash \texttt{1} : \mathbf{Nat}}{[\texttt{x} : \texttt{X}] \vdash \texttt{x+1} : \mathbf{Nat}, [\texttt{X} = \mathbf{Nat}]}}{\vdash (\mathbf{Fun}\ \texttt{x -> x+1}) : \texttt{X} \texttt{ -> } \mathbf{Nat}, [\texttt{X} = \mathbf{Nat}]}$$

- 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 $[\texttt{X} = \mathbf{Nat}]$.
- On type la fonction.

Idée générale de l'inférence (synthèse) de type

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

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

  - ▷ Et posant des équations.

$$\frac{E \vdash t_1 : A_1 \qquad E \vdash t_2 : A_2}{E \vdash t_1\ \texttt{+}\ t_2 : \textbf{Nat}, [A_1 = \textbf{Nat}, A_2 = \textbf{Nat}]}$$

- 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.

$$\textbf{Nat} \in \mathcal{T} \qquad X \in \mathcal{T} \qquad \frac{A_1 \in \mathcal{T} \qquad A_2 \in \mathcal{T}}{A_1\ \texttt{->}\ A_2 \in \mathcal{T}}$$

Avec les nouveaux types on a :

$$\frac{[\texttt{x} : \texttt{X}] \vdash \texttt{x} : \texttt{X}}{\vdash (\textbf{Fun}\ \texttt{x -> x}) : \texttt{X -> X}}$$

Quel sens donner à un tel type ?

`id` : `X -> X` veut dire que `id` : A `->` A pour tout type A.

Et en effet on peut donner les types suivants à **Fun** `x -> x`.

- **Nat -> Nat**, (**Nat -> Nat**) **-> Nat -> Nat**, etc.
- Et même (`Y -> Z`) `-> Y -> Z`, mais pas `Y ->` **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 : $\sigma(A\ \texttt{->}\ B) = \sigma(A)\ \texttt{->}\ \sigma(B)$.
- 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

$$\dfrac{\dfrac{\dfrac{\dfrac{[\texttt{f} : \texttt{F}, \texttt{x} : \texttt{X}] \vdash \texttt{f} : \texttt{F} \qquad [\texttt{f} : \texttt{F}, \texttt{x} : \texttt{X}] \vdash \texttt{x} : \texttt{X}}{[\texttt{f} : \texttt{F}, \texttt{x} : \texttt{X}] \vdash \texttt{f x} : \texttt{Y}} \qquad [\texttt{f} : \texttt{F}, \texttt{x} : \texttt{X}] \vdash \texttt{x} : \texttt{X}}{[\texttt{f} : \texttt{F}, \texttt{x} : \texttt{X}] \vdash \texttt{f x+x} : \mathbf{Nat}}}{[\texttt{f} : \texttt{F}] \vdash (\mathbf{Fun}\ \texttt{x} \texttt{->} \mathbf{Fun}\ \texttt{x} \texttt{->}\ \texttt{f x+x}) : \texttt{X} \texttt{->} \mathbf{Nat}}}{\vdash (\mathbf{Fun}\ \texttt{f} \texttt{->} \mathbf{Fun}\ \texttt{x} \texttt{->}\ \texttt{f x+x}) : \texttt{F} \texttt{->} \texttt{X} \texttt{->} \mathbf{Nat}}$$

- 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 $[\texttt{F} = \texttt{X} \texttt{->} \texttt{Y}]$.
- Le type de `f x+x` est **Nat**, avec $[\texttt{Y} = \mathbf{Nat}, \texttt{X} = \mathbf{Nat}]$.
- Type final, avec les équations $[\texttt{F} = \texttt{X} \texttt{->} \texttt{Y}, \texttt{Y} = \mathbf{Nat}, \texttt{X} = \mathbf{Nat}]$.

Résultat de l'inférence

Le type :

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

Avec les équations :

$$[\texttt{F} = \texttt{X} \texttt{->} \texttt{Y}, \texttt{Y} = \mathbf{Nat}, \texttt{X} = \mathbf{Nat}]$$

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

$$[\texttt{X} \mapsto \mathbf{Nat}, \texttt{Y} \mapsto \mathbf{Nat}, \texttt{F} \mapsto \mathbf{Nat} \texttt{->} \mathbf{Nat}]$$

On substitue dans le type :

$$\vdash \mathbf{Fun}\ \texttt{f} \texttt{->} \mathbf{Fun}\ \texttt{x} \texttt{->}\ \texttt{f x+x} : (\mathbf{Nat} \texttt{->} \mathbf{Nat}) \texttt{->} \mathbf{Nat} \texttt{->} \mathbf{Nat}$$

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 importent les doublons) d'équations.

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

Avec les notations de définition de prédicat par induction.

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

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

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

Production des équations II

Les opérations sur les entiers.

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

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

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

Équations III

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

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

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 **Fun** `f -> 2+f 1`.

Type : `F -> Nat`.

Équations : $[\texttt{F} = (\textbf{Nat}\ \texttt{->}\ \texttt{Y}), \texttt{Y} = \textbf{Nat}, \textbf{Nat} = \textbf{Nat}]$.

Solution : $[\texttt{Y} \mapsto \textbf{Nat}, \texttt{F} \mapsto (\textbf{Nat}\ \texttt{->}\ \textbf{Nat})]$.

Type : (**Nat** `->` **Nat**) `->` **Nat**.

Résoudre

Équations du type $\mathcal{E} = \{\, A_1 = B_1, \ldots, A_n = B_n \,\}$, 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 :

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

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 : $[\texttt{F} = \textbf{Nat}\ \texttt{->}\ \texttt{Y}, \texttt{Y} = \textbf{Nat}, \textbf{Nat} = \textbf{Nat}]$.

Solution facile : $[\texttt{Y} \mapsto \textbf{Nat}, \texttt{F} \mapsto (\textbf{Nat}\ \texttt{->}\ \textbf{Nat})]$.

Pas de solution

À typer : **Fun** `f -> f 1+f`.

Equations : $[\texttt{F} = \textbf{Nat}\ \texttt{->}\ \texttt{Y}, \texttt{Y} = \textbf{Nat}, \texttt{F} = \textbf{Nat}]$.

Aucune valeur possible pour `F` car **Nat** `->` `Y` $\neq$ **Nat** pour tout `Y`.

C'est donc la résolution qui détecte les errreurs, 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 -> x x`.

Équations : $[\texttt{X} = \texttt{X -> Y}]$.

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
```


Plusieures solutions

Terme : **Fun** `f -> Fun x -> f x`.

Equations : $[\texttt{F} = \texttt{X -> Y}]$.

Une solution ϕ : $[\texttt{X} \mapsto \mathbf{Nat}, \texttt{F} \mapsto (\mathbf{Nat}\ \texttt{->}\ \mathbf{Nat}), \texttt{Y} \mapsto \mathbf{Nat}]$.

Une autre σ : $[\texttt{F} \mapsto \texttt{X -> Y}]$ (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 $\psi = [\texttt{X} \mapsto \mathbf{Nat}, \texttt{Y} \mapsto \mathbf{Nat}]$.

En effet,

$$\psi(\sigma(\texttt{F})) = \psi(\texttt{X -> Y}) = \mathbf{Nat}\ \texttt{->}\ \mathbf{Nat}$$

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

$$\psi(\sigma(\texttt{Z})) = \psi(\texttt{Z}) = \texttt{Z}$$

Plusieures solutions

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

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

Equations : $[\texttt{B} = \mathbf{Nat}, \texttt{X} = \texttt{Y}]$.

Une solution σ : $[\texttt{B} \mapsto \mathbf{Nat}, \texttt{X} \mapsto \texttt{Y}]$.

Une autre solution σ' : $[\texttt{B} \mapsto \mathbf{Nat}, \texttt{Y} \mapsto \texttt{X}]$.

Les deux solutions sont principales :

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

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 `->` $B = C$ `->` D, remplacer par $[A = C, B = D]$.
2. $\mathbf{Nat} = \mathbf{Nat}$ ou $X = X$, supprimer.
3. $\mathbf{Nat} = A$ `->` 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 $(\mathcal{V}(E), |\mathcal{E}|)$.

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

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

- Cas 1. et 2.
  - $\mathcal{V}(\mathcal{E})$ 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 :

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

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

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

1. Si $\mathsf{mgu}(\mathcal{E})$ n'echoue pas, il produit une solution de $\mathcal{E}$.
2. Si il existe une solution de $\mathcal{E}$, alors $\mathsf{mgu}(\mathcal{E})$ 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 -> Fun n -> 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` = **Nat**] (et aussi [**Nat** = **Nat**], mais bon). Type **Nat**.
- De `pow (n-1)`, il vient : [`P` = (**Nat** `->` `Y`)]. Type `Y`.
- De `2*pow (n-1)`, il vient : [`Y` = **Nat**]. Type **Nat**.
- De **Ifz**..., il vient : [`N` = **Nat**] (et aussi [**Nat** = **Nat**]). Type **Nat**.
- Le type de **Fun** `n ->` $\cdots$ est : `N ->` **Nat**.
- Le type de **Fix** `pow ->` $\cdots$ est : `N ->` **Nat**, avec l'équation : [`P` = (`N ->` **Nat**)].

Point fixe

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

Équations,

$$[\texttt{N} = \textbf{Nat}, \texttt{P} = (\textbf{Nat -> Y}), \texttt{Y} = \textbf{Nat}, \texttt{N} = \textbf{Nat}, \texttt{P} = (\texttt{N -> Nat})]$$

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`

$$[\texttt{N} = \textbf{Nat}, \texttt{P} = \textbf{Nat}, \texttt{P} = (\texttt{N -> Nat})]$$

Résolution

$$[\texttt{N} = \textbf{Nat}, \texttt{P} = (\textbf{Nat -> Y}), \texttt{Y} = \textbf{Nat}, \texttt{N} = \textbf{Nat}, \texttt{P} = (\texttt{N -> Nat})]$$

- $[\texttt{N} \mapsto \textbf{Nat}]$ et
  $[\texttt{P} = (\textbf{Nat -> Y}), \texttt{Y} = \textbf{Nat}, \textbf{Nat} = \textbf{Nat}, \texttt{P} = (\textbf{Nat -> Nat})]$.
- $[\texttt{N} \mapsto \textbf{Nat}, \texttt{P} \mapsto (\textbf{Nat -> Y})]$ et
  $[\texttt{Y} = \textbf{Nat}, \textbf{Nat} = \textbf{Nat}, (\textbf{Nat -> Y}) = (\textbf{Nat -> Nat})]$.
- $[\texttt{N} \mapsto \textbf{Nat}, \texttt{P} \mapsto (\textbf{Nat -> Nat}), \texttt{Y} \mapsto \textbf{Nat}]$ et
  $[\textbf{Nat} = \textbf{Nat}, (\textbf{Nat -> Nat}) = (\textbf{Nat -> Nat})]$
- ...
- $[\texttt{N} \mapsto \textbf{Nat}, \texttt{P} \mapsto (\textbf{Nat -> Nat}), \texttt{Y} \mapsto \textbf{Nat}]$

Type du point fixe : `N -> Nat` soit **Nat -> 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 -> X`.
- Soit l'équation

  $$[(\texttt{X -> X}) = (\texttt{(X -> X) -> Y})]$$

  Et donc

  $$[\texttt{X} = (\texttt{X -> X}), \texttt{X} = \texttt{Y}]$$

  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 :

$$[\ldots, \texttt{X} = \mathbf{Nat}, \ldots]$$

C'est remplacer `X` par **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 :

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

On a :

$$\frac{E, \sigma \vdash t_1 \rightsquigarrow A_1, \sigma_1 \qquad E, \sigma_1 \vdash t_2 \rightsquigarrow A_2, \sigma_2}{E, \sigma \vdash t_1\ t_2 \rightsquigarrow X, \mathsf{mgu}[A_1 = (A_2\ \texttt{->}\ X), \sigma_2]}(X \text{ frais})$$

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 $\sigma = \{\, X_1 \mapsto A_1, \ldots, X_n \mapsto A_n \,\}$, 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 $\mathsf{mgu}(\mathcal{E})$, mais ça serait dommage d'oublier que σ est déjà résolu.

Une vue incrémentale de l'unification

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

1. Remplacer les X_i par les A_i dans B et C — les X_i n'apparaissent plus.
2. Résoudre la nouvelle équation : $\sigma' = \{ Y_1 \mapsto B_1, \ldots, Y_m \mapsto B_m \}$ — 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 $\{ Y_1 \mapsto B_1, \ldots, Y_m \mapsto B_m, X_1 \mapsto C_1, \ldots, X_n \mapsto C_m \}$.

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

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

Toutes les règles

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

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

$$\frac{E, \sigma \vdash t_1 \rightsquigarrow A, \sigma_1 \qquad E, \sigma_1 \vdash t_2 \rightsquigarrow B, \sigma_2}{E, \sigma \vdash t_1\ t_2 \rightsquigarrow X, \mathsf{mgu}[\sigma_2(A) = \sigma_2(B \texttt{ -> } X)] \circ \sigma_2} (X \text{ frais})$$

$$\frac{E \oplus [x = X], \sigma \vdash t \rightsquigarrow A, \sigma'}{E, \sigma \vdash (\textbf{Fix } x \texttt{ -> } t) \rightsquigarrow A, \mathsf{mgu}[\sigma'(X) = \sigma'(A)] \circ \sigma'} (X \text{ frais})$$

Vraiment toutes les règles

$$E, \sigma \vdash \texttt{n} \rightsquigarrow \textbf{Nat}, \sigma$$

$$\frac{E, \sigma \vdash t_1 \rightsquigarrow A_1, \sigma_1 \qquad \sigma_1' = \mathsf{mgu}[\sigma_1(A_1) = \textbf{Nat}] \circ \sigma_1 \qquad E, \sigma_1' \vdash t_2 \rightsquigarrow A_2, \sigma_2 \qquad \sigma_2' = \mathsf{mgu}[\sigma_2(A_2) = \textbf{Nat}] \circ \sigma_2}{E, \sigma \vdash t_1 \texttt{ op } t_2 \rightsquigarrow \textbf{Nat}, \sigma_2'}$$

$$\frac{E, \sigma \vdash t_1 \rightsquigarrow A_1, \sigma_1 \qquad \sigma_1' = \mathsf{mgu}[\sigma_1(A_1) = \textbf{Nat}] \circ \sigma_1 \qquad E, \sigma_1' \vdash t_2 \rightsquigarrow A_2, \sigma_2 \qquad E, \sigma_2 \vdash t_3 \rightsquigarrow A_3, \sigma_3 \qquad \sigma_3' = \mathsf{mgu}[\sigma_3(A_2) = \sigma_3(A_3)] \circ \sigma_3}{E, \sigma \vdash \textbf{Ifz } t_1 \textbf{ Then } t_2 \textbf{ Else } t_3 \rightsquigarrow A_2, \sigma_3'}$$

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

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

- De **Ifz**, il vient d'abord, $\sigma = [\texttt{N} \mapsto \textbf{Nat}]$, puis.
  - Pour le **Then** $E, \sigma \vdash \texttt{1} \rightsquigarrow \textbf{Nat}, \sigma$.
  - Et le **Else**
    - D'abord $E, \sigma \vdash \texttt{2} \rightsquigarrow \textbf{Nat}, \sigma$,
    - Puis `n-1` donne $\mathsf{mgu}[\sigma(\texttt{N}) = \textbf{Nat}] \circ \sigma$ qui ne change rien
    - Puis surtout (application `pow (n-1)`) $\mathsf{mgu}[\sigma(\texttt{P}) = \sigma(\textbf{Nat } \texttt{->}\ \texttt{Y})] \circ \sigma$. Qui vaut $\sigma' = [\texttt{P} \mapsto (\textbf{Nat } \texttt{->}\ \texttt{Y}), \texttt{X} \mapsto \textbf{Nat}]$.
    - Et enfin le deuxième argument de « `*` » conduit au calcul

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

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

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

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

- Le type de **Fun** est `N -> Nat`,
- C'est donc le type de **Fix**, avec à calculer

$$\mathsf{mgu}[\sigma''(\texttt{P}) = \sigma''(\texttt{N}\ \texttt{->}\ \mathbf{Nat})] \circ \sigma''$$

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

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