Inférence par contraintes pour les GADTs

Olivier Martinot + TODO

TODO

Lundi 2 décembre 2024

Section 1

Programme, typage et inférence de types

Recette / programme

Recette

```
Ingrédients :
100g de riz


Faire bouillir 100cL d'eau
Faire cuire le riz 10 min
Égouter le riz


Plat:
Riz cuit
```

Programme

```
Entrées :
a entier positif
b entier positif

res ← 0
Tant que b > 0:
    res ← res + a
    b ← b - 1

Sortie:
res vaut a × b
```

Des programmes qui posent problème

- Un programme peut planter :
  - manque d'espace mémoire
  - division par zéro
  - erreur d'approximation
  - ...
- On veut des garanties sur les exécutions de nos programmes

Type

- Une approche : s'intéresser à la façon dont on manipule les données dans nos programmes
- On donne des types aux morceaux des programmes

Le typage consiste à s'assurer que les types des différentes morceaux du programme sont cohérents entre eux.

Sur un exemple

- Casserole : Instrument
- Riz : Aliment

Langages de programmation : famille ML, OCaml

- Meta-language (ML) : langage de programmation développé dans les années 1970
- Aujourd'hui : famille de langages de programmation dont OCaml

Exemple : programme annoté par son type

En mathématiques

$$\begin{aligned} succ \; : \; & \mathbb{Z} \to \mathbb{Z} \\ & n \mapsto n + 1 \end{aligned}$$

En OCaml

```
let succ : int -> int =
  fun n -> n + 1
```

Exemple : programme annoté par son type

En mathématiques

$$\begin{aligned} succ \; : \; & \mathbb{Z} \to \mathbb{Z} \\ & n \mapsto n + 1 \end{aligned}$$

En OCaml

```
let succ : int -> int =
  fun n -> n + 1
```

Exemple : programme annoté par son type

En mathématiques

$$\begin{aligned} \mathit{succ} \ :\ & \mathbb{Z} \to \mathbb{Z} \\ & n \mapsto n+1 \end{aligned}$$

En OCaml

```
let succ : int -> int =
  fun n -> n + 1
```

Erreurs de typage

```
let succ : int -> int =
  fun n -> n + 1;;
```

Erreurs de typage

```
let succ : int -> int =
  fun n -> n + 1;;

# succ 0;;
- : int = 1
```

Erreurs de typage

```
let succ : int -> int =
  fun n -> n + 1;;

# succ 0;;
- : int = 1

# succ (0, 3);;
Error: This expression has type 'a * 'b but an expression
       was expected of type int
```

Inférence de type

On cherche à déduire les types des morceaux du programme. On a des indices :

- type d'une fonction : $\mathtt{A} \rightarrow \mathtt{B}$
- regarder les utilisations d'une variable dans le reste du programme
- ...

Exemple

```
let x = expr in
...
f (x, y)
```

En connaissant le type de `f` (en particulier le type de ses arguments), on peut en déduire le type attendu pour `x`.

Exemple : type de succ ?

```
let succ : ? =
  fun n -> n + 1;;
```

Exemple : type de succ ?

```
let succ : W =
  fun n -> n + 1;;
```

Exemple : type de succ ?

```
let succ : W =
  fun (n : X) -> (n + 1 : Y) ;;
```

Nos hypothèses

- W = X $\rightarrow$ Y

Exemple : type de succ ?

```
let succ : W =
  fun (n : X) -> (n + 1 : Y) ;;
```

Nos hypothèses

- W = X $\rightarrow$ Y

Ce qu'on sait

+ : int $\rightarrow$ int $\rightarrow$ int

Ce qu'on en déduit

- n : int
- 1 : int
- Y = int

Exemple : type de succ ?

```
let succ : W =
  fun (n : X) -> (n + 1 : Y) ;;
```

Nos hypothèses

- W = X $\rightarrow$ Y
- n : int
- Y = int

Exemple : type de succ ?

```
let succ : W =
  fun (n : X) -> (n + 1 : Y) ;;
```

Nos hypothèses

- W = X $\rightarrow$ Y
- n : int
- Y = int

Exemple : type de succ ?

```
let succ : W =
  fun (n : X) -> (n + 1 : Y) ;;
```

Nos hypothèses

- W = X $\rightarrow$ Y
- X = int
- Y = int

Exemple : type de succ ?

```
let succ : W =
  fun (n : X) -> (n + 1 : Y) ;;
```

Nos hypothèses

- W = int → int
- X = int
- Y = int

Section 2

Inférence de types par résolution de contraintes

Hindley-Damas-Milner

```
let first_arg : ? =
  fun (x, y) -> x

# first_arg (0, true);;
- : int = 0

# first_arg (false, 1);;
- : bool = false
```

Hindley-Damas-Milner

Intuition

Quand on essaye d'inférer un type pour `first_arg`, il va rester des inconnues : $W = X \times Y \to X$.

Une solution : polymorphisme du `let`

`first_arg` : $\forall \alpha\beta.\ \alpha \times \beta \to \alpha$

Principalité

Le type T d'un terme est principal si tout autre type possible pour ce terme sont des instances de T.

Inférence de types par résolution de contraintes

- Génération de contraintes
- Résolution de contraintes
- Élaboration d'un terme annoté

Exemple d'inférence par contraintes

```
fun n → n + 1
```

Exemple d'inférence par contraintes

`fun n → n + 1`

Génération de contraintes $\quad W = X \to Y \quad \wedge \quad X = \dots \quad \wedge \quad \dots$

Exemple d'inférence par contraintes

`fun n → n + 1`

Génération de contraintes	$W = X \to Y \quad \wedge \quad X = \ldots \quad \wedge \quad \ldots$
Résolution de contraintes	$W = \texttt{int} \to \texttt{int}\ ;\ X = \texttt{int}\ ;\ \ldots$

Exemple d'inférence par contraintes

`fun n → n + 1`

Génération de contraintes	$W = X \to Y \quad \wedge \quad X = \dots \quad \wedge \quad \dots$
Résolution de contraintes	$W = \texttt{int} \to \texttt{int}\ ;\ X = \texttt{int}\ ;\ \dots$
Élaboration d'un terme annoté	`fun (n : int) → n + 1`

Sémantique des contraintes

$$
\begin{array}{lll}
s & ::= & (\rightarrow) \mid (\times) \mid \mathit{Constr} \mid \ldots \\
\tau & ::= & a \mid s\ \bar{\tau} \\
t & ::= & s\ \bar{t} \\
T & ::= & X \mid a \mid s\ \bar{X}
\end{array}
$$

$$C ::= \text{true} \mid \text{false} \mid C \wedge C \mid \exists X.C \mid X \text{ is } T \mid \ldots$$

$$
\frac{}{E; \gamma \models \text{true}}
\qquad
\frac{E; \gamma \models C_1 \qquad E; \gamma \models C_2}{E; \gamma \models C_1 \wedge C_2}
\qquad
\frac{\exists t, \quad E; \gamma[X \mapsto t] \models C}{E; \gamma \models \exists X.C}
$$

$$
\frac{\gamma(X) = \gamma(T)}{E; \gamma \models X \text{ is } T}
\qquad \ldots
$$

Solveur de contraintes

Règles de réécriture

S $\approx$ contexte d'évaluation
U $\approx$ unifications
C $\approx$ contrainte courante
L'état représente $S[U \wedge C]$

$$S \;;\; U \;;\; C \rightarrow S' \;;\; U' \;;\; C'$$

$$U ::= \text{true} \mid \text{false} \mid U \wedge U \mid \exists X.U \mid X = Y = \dots$$

$$S ::= [] \mid S[\exists X.[]] \mid S[[] \wedge C] \mid \dots$$

Solveur de contraintes

- Contrainte initiale : $S[U \wedge C]$ avec $S = []$ et $U = \text{true}$
- Contrainte finale : true ou false

$$S\ ;\ U\ ;\ C_1 \wedge C_2 \quad \rightarrow S[[] \wedge C_2]\ ;\ U\ ;\ C_1$$

$$S[[] \wedge C]\ ;\ U\ ;\ \text{true} \quad \rightarrow S\ ;\ U\ ;\ C$$

$$\ldots$$

Solveur de contraintes

Multi-équations

Dans U, on garde en mémoire des multi-équations de la forme

$$\epsilon ::= X_1 = \cdots = X_n \; [= s \; \bar{Y}]$$

La bibliothèque Inferno

- Bibliothèque pour l'inférence de types par résolution de contraintes
- Développée depuis 2014 par François Pottier
- Combine génération et élaboration

Dans ma thèse

Extension du typeur à d'autres constructions (types algébriques, GADTs)

Section 3

Introduire les GADTs dans le typeur

Generalized Algebraic Datatype (GADT)

```
type _ expr =
| Int  : int -> int expr
| Bool : bool -> bool expr

let binop (type a) (e1 : a expr) (e2 : a expr) : a =
  match (e1, e2) with
  | (Bool b1, Bool b2) -> b1 && b2
  | (Int i1, Int i2) -> i1 + i2
```

Exhaustivité du filtrage par motif

Grâce au typage, on sait que les cas `(Int, Bool)` et `(Bool, Int)` sont impossibles.

Generalized Algebraic Datatype (GADT)

Programme OCaml :

```
type _ expr =
| Int : int -> int expr
| Bool : bool -> bool expr
```

Avec égalité explicite :

```
type (_, _) eq = Refl : ('a, 'a) eq

type _ expr =
| Int of int * ('a, int) eq
| Bool of bool * ('a, bool) eq

let f (type a) (e : a expr) : a =
  match e with
  | Int (n, hyp) -> n
  | Bool (b, hyp) -> b
```

Generalized Algebraic Datatype (GADT)

Ambiguïté

```
let f (type a) (hyp : (a,int) eq) (y : a) =
  match hyp with Refl ->
    (* a = int *)
    if y > 0 then y else 0
(*
   Error: This expression has type int but an expression
       was expected of type
         a = int
       This instance of int is ambiguous:
       it would escape the scope of its equation
*)
```

Contribution : contrainte d'hypothèse d'égalité

$$C ::= \cdots \mid (\tau_1 = \tau_2) \Rightarrow C$$

Contribution : contrainte d'hypothèse d'égalité

$$C ::= \cdots \mid (\tau_1 = \tau_2) \Rightarrow C$$

```
let f (type a) (x : (a,int) eq) (y : a) : int =
  match x with Refl -> (* a = int *) y
```

Contrainte (simplifiée)

$\forall a.\ (\phi :\ a\ =\ \texttt{int}) \Rightarrow \exists X.X \text{ is } a \wedge X \text{ is } \texttt{int}$

Contribution : contrainte d'hypothèse d'égalité

$$C ::= \cdots \mid (\tau_1 = \tau_2) \Rightarrow C$$

```
let f (type a) (x : (a,int) eq) (y : a) : int =
  match x with Refl -> (* a = int *) y
```

Contrainte (simplifiée)

$\forall a. \ (\phi : \ a \ = \ \texttt{int}) \Rightarrow \exists X.X \text{ is } a \wedge X \text{ is } \texttt{int}$

Résolution

$$\phi : a = \texttt{int}$$

$$X = a \ \wedge \ X = \texttt{int} \qquad \rightarrow \qquad X = a = \texttt{int} \qquad \rightarrow \qquad \phi \vdash X = a$$

Multi-équations avec égalités

La cohérence des multi-équations dépend désormais des égalités introduites :

$$\Phi \vdash \epsilon$$

$$\Phi ::= \phi_1, \ldots \phi_n$$

$$\epsilon ::= X_1 = \cdots = X_n = a_1 = \cdots = a_m = s_1\ \bar{Y}_1 = \cdots = s_p\ \bar{Y}_p$$

Résolution

$$\dfrac{U \# \bar{X}, \phi}{S[(\phi : \ \tau_1 \ = \ \tau_2) \Rightarrow \exists \bar{X}.[]] \ ; \ U \ ; \ \mathsf{true} \rightarrow S \ ; \ U \ ; \ \mathsf{true}} \ \text{POP-EQ(SIMPLIFIÉE)}$$

Exemple

$$S[(\phi : \ a \ = \ \texttt{int}) \Rightarrow \exists X.[]] \ ; \ \vdash Y = \texttt{bool} \ ; \ \mathsf{true}$$

$$\rightarrow S \ ; \ \vdash Y = \texttt{bool} \ ; \ \mathsf{true}$$

Résolution

POP-EQ

$$\frac{U_1 \# \bar{X}, \phi \quad \big(\mathsf{Eqs}(S), \phi \Rightarrow \exists \bar{X}. U_2\big) \equiv \mathsf{true}}{S[(\phi :\ \tau_1\ =\ \tau_2) \Rightarrow \exists \bar{X}.[]] \ ;\ U_1 \wedge U_2\ ;\ \mathsf{true} \quad \rightarrow \quad S\ ;\ U_1\ ;\ \mathsf{true}}$$

Exemple

$$S[(\phi :\ a\ =\ \mathtt{int}) \Rightarrow \exists X.[]] \ ;\ (\vdash Y = \mathtt{bool}) \wedge (\phi \vdash X = a)\ ;\ \mathsf{true}$$

$$\rightarrow \quad S\ ;\ \vdash Y = \mathtt{bool}\ ;\ \mathsf{true}$$

Résolution

$$\frac{U_1 \# \bar{X}, \phi \qquad \color{red}{(\mathsf{Eqs}(S), \phi \Rightarrow \exists \bar{X}. U_2) \equiv \mathsf{true}}}{S[(\phi : \tau_1 = \tau_2) \Rightarrow \exists \bar{X}.[]] \ ; \ U_1 \wedge U_2 \ ; \ \mathsf{true} \quad \rightarrow \quad S \ ; \ U_1 \ ; \ \mathsf{true}}\ \text{POP-EQ}$$

Exemple

- $(\tau_1 = \tau_2) \Rightarrow \exists X.\ X \text{ is } \tau_1 \ \equiv \ \mathsf{true}$
- $(\tau_1 = \tau_2) \Rightarrow \exists X.\ Z \text{ is } \tau_1 \ \not\equiv \ \mathsf{true}$

Échappement d'hypothèse d'égalité

$$\dfrac{\phi \in \Phi \qquad \big(\mathsf{Eqs}(S), \phi \Rightarrow \exists \bar{X}.\epsilon\big) \not\equiv \mathsf{true}}{S[(\phi : \tau_1 = \tau_2) \Rightarrow \exists \bar{X}.[]] \;;\; U \wedge (\Phi \vdash \epsilon) \;;\; \mathsf{true} \quad \rightarrow \quad \mathsf{false}}\ \textsc{Scope-Escape}$$

Exemple

$$S[\exists X.(\phi : a = \mathtt{int}) \Rightarrow []] \;;\; (\phi \vdash X = a) \;;\; \mathsf{true}$$

$$\rightarrow \quad \mathsf{false}$$

Échappement d'hypothèse d'égalité

Dans l'implémentation

Niveaux et portées

Exemples

$$\exists X.(\phi : \ \tau_1 \ = \ \tau_2) \Rightarrow \exists Y.Y \text{ is } \tau_1 \wedge Y \text{ is } X$$

Variables rigides et problèmes de partage

Contribution : structure abstraite locale

$$C ::= \cdots \mid \mathsf{letr}\ x = \forall \bar{a} \lambda X . C_1\ \mathsf{in}\ C_2$$

$$\frac{\forall \bar{a} \exists X \bar{Y} . U \equiv \mathsf{true}}{\begin{array}{c} S[\mathsf{letr}\ x = \forall \bar{a} \lambda X . \exists \bar{Y} . []\ \mathsf{in}\ C]\ ;\ U\ ;\ \mathsf{true} \\ \to S[\mathsf{letr}\ x = \forall \bar{a} \lambda X . \exists \bar{Y} . U\ \mathsf{in}\ []]\ ;\ \mathsf{true}\ ;\ C \end{array}}\ \textsc{Build-Rigid-Scheme}(\text{simplifiée})$$

Contribution : structure abstraite locale

$$\dfrac{\forall \bar{a} \exists X \bar{Y}. U \equiv \mathsf{true}}{\begin{array}{c} S[\mathsf{letr}\ x = \forall \bar{a} \lambda X. \exists \bar{Y}. [] \ \mathsf{in}\ C]\ ;\ U\ ;\ \mathsf{true} \\ \to S[\mathsf{letr}\ x = \forall \bar{a} \lambda X. \exists \bar{Y}. U \ \mathsf{in}\ []]\ ;\ \mathsf{true}\ ;\ C \end{array}}\ \textsc{Build-Rigid-Scheme}(\text{simplifiée})$$

Exemple

$$\begin{array}{rl} & S[\mathsf{letr}\ x = \forall a \lambda X. [] \ \mathsf{in}\ \mathsf{true}]\ ;\ X \ \mathsf{is}\ a \to a\ ;\ \mathsf{true} \\ \to & S[\mathsf{letr}\ x = \forall a \lambda X. X \ \mathsf{is}\ a \to a \ \mathsf{in}\ []]\ ;\ \mathsf{true}\ ;\ \mathsf{true} \end{array}$$

Contribution : structure abstraite locale

$$\dfrac{\begin{array}{l}\text{BUILD-RIGID-SCHEME}\\ X\bar{Y}\#U_1 \quad \wedge \quad a\#U_1 \quad \wedge \quad \forall\bar{a}\exists X\bar{Y}.U_2 \equiv \mathsf{true}\end{array}}{\begin{array}{c}S[\mathsf{letr}\ x = \forall\bar{a}\lambda X.\exists\bar{Y}.[]\ \mathsf{in}\ C]\ ;\ U_1 \wedge U_2\ ;\ \mathsf{true}\\ \rightarrow S[\mathsf{letr}\ x = \forall\bar{a}\lambda X.\exists\bar{Y}.U_2\ \mathsf{in}\ []]\ ;\ U_1\ ;\ C\end{array}}$$

Sémantique de la contrainte d'hypothèse d'égalité

$$\frac{\gamma(\tau_1) = \gamma(\tau_2) \quad \Longrightarrow \quad E; \gamma \models C}{E; \gamma \models (\tau_1 = \tau_2) \Rightarrow C}$$

Sémantique de la contrainte d'hypothèse d'égalité

$$\kappa ::= \mathsf{true} \mid \mathsf{false}$$

$$\frac{\exists\psi \quad \text{if } \kappa \text{ then } |\psi| = 1 \quad \kappa; E; \gamma[X \mapsto \psi] \models^{\mathsf{amb}} C}{\kappa; E; \gamma \models^{\mathsf{amb}} \exists X.C}$$

Conclusion

Contributions

- Étendre Inferno pour prendre en charge les GADTs
- Solveur pour les contraintes d'hypothèse d'égalité

Des suites possibles ?

- Des preuves à faire sur le solveur
- Étendre le support pour les GADTs hors du langage "noyau"
- Prendre en charge d'autres fonctionnalités d'OCaml