Inférence par contraintes pour les GADTs

Olivier Martinot encadré par Gabriel Scherer

Picube / Université Paris Cité

Lundi 2 décembre 2024

1. Programme, typage et inférence de types

2. Inférence de types par résolution de contraintes

3. Introduire les GADTs dans le typeur

Section 1

Programme, typage et inférence de types

Qu'est-ce-qu'une recette ?

- Suite d'instructions
- Des ingrédients $\rightarrow$ un plat

Qu'est-ce-qu'un programme ?

- Suite d'instructions pour réaliser un calcul
- Des entrées $\rightarrow$ une sortie
- Formalisation de ces instructions : un langage de programmation

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
  - bloc d'instructions qui boucle à l'infini
  - ...
- 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.

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

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

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 : `A` $\longrightarrow$ `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

- $\mathtt{W} = \mathtt{X} \to \mathtt{Y}$

Ce qu'on sait

$+ : \mathtt{int} \to \mathtt{int} \to \mathtt{int}$

Ce qu'on en déduit

- $\mathtt{n} : \mathtt{int}$
- $1 : \mathtt{int}$
- $\mathtt{Y} = \mathtt{int}$

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 $\longrightarrow$ Y
- X = int
- Y = int

Exemple : type de succ ?

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

Nos hypothèses

- `W = int` $\rightarrow$ `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 (true, 0);;
- : bool = true
```

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

Intuition

La résolution de contrainte ressemble à la résolution d'un système d'équations.

Exemple d'inférence par contraintes

```
fun n → n + 1
```

Exemple d'inférence par contraintes

```
fun n → n + 1
```

Génération de contraintes $W = X \to Y \land X = \cdots \land \ldots$

Exemple d'inférence par contraintes

```
fun n → n + 1
```

Génération de contraintes	$W = X \to Y \wedge X = \cdots \wedge \dots$
Résolution de contraintes	$W = \mathtt{int} \to \mathtt{int}; X = \mathtt{int}; Y = \mathtt{int}; \dots$

Exemple d'inférence par contraintes

`fun n → n + 1`

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

Sémantique des contraintes

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

$$\frac{}{E;\gamma \models \text{true}} \qquad \frac{E;\gamma \models C_1 \qquad E;\gamma \models C_2}{E;\gamma \models C_1 \wedge C_2}$$

$$\frac{\exists t, \quad E;\gamma[X \mapsto t] \models C}{E;\gamma \models \exists X.C} \qquad \ldots$$

Solveur de contraintes

Règles de réécriture

S $\approx$ contexte d'évaluation
U $\approx$ unifications
C $\approx$ contrainte courante

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

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

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

Solveur de contraintes

$$
\begin{aligned}
S\ ;\ U\ ;\ C_1 \wedge C_2 &\quad \rightarrow S[[] \wedge C_2]\ ;\ U\ ;\ C_1 \\
S[[] \wedge C]\ ;\ U\ ;\ \mathsf{true} &\quad \rightarrow S\ ;\ U\ ;\ C \\
\ldots &
\end{aligned}
$$

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 en 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
```

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, eq) -> n
  | Bool (b, eq) -> b
```

Generalized Algebraic Datatype (GADT)

Ambiguité

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

Contrainte d'hypothèse d'égalité

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

Contrainte d'hypothèse d'égalité

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

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

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

Multi-équations avec égalités

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

$$\Phi \vdash \epsilon$$

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

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

Multi-équations avec égalités

POP-EQ(simplifiée)

$$\frac{U \# \bar{X}, \phi}{S[(\phi : \tau_1 = \tau_2) \Rightarrow \exists \bar{X}.[]] \; ; \; U \; ; \; \mathsf{true} \rightarrow S \; ; \; U \; ; \; \mathsf{true}}$$

Exemple

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

Multi-équations avec égalités

POP-EQ

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

Exemple

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

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

Échappement d'hypothèse d'égalité

$$\dfrac{\phi \in \Phi \qquad \left(\mathsf{Eqs}(S), \phi \Rightarrow \exists \bar{X}.\epsilon\right) \not\equiv \mathsf{true}}{S[(\phi : \tau_1 = \tau_2) \Rightarrow \exists \bar{X}.[]] \ ; \ U \wedge (\Phi \vdash \epsilon) \ ; \ \mathsf{true} \rightarrow \mathsf{false}}\ \text{SCOPE-ESCAPE}$$

Exemple

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

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

Échappement d'hypothèse d'égalité

Dans l'implémentation

Niveaux et portées

Variables rigides et problèmes de partage

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

$$\dfrac{\text{BUILD-RIGID-SCHEME(simplifiée)} \qquad \forall \bar{a} \exists X \bar{Y} . U \equiv \text{true}}{\begin{array}{l} S[\text{letr } x = \forall \bar{a} \lambda X . \exists \bar{Y} . [] \text{ in } C] \; ; \; U \; ; \; \text{true} \\ \rightarrow S[\text{letr } x = \forall \bar{a} \lambda X . \exists \bar{Y} . U \text{ in } []] \; ; \; \text{true} \; ; \; C \end{array}}$$

Variables rigides et problèmes de partage

$$\frac{\text{BUILD-RIGID-SCHEME(simplifiée)} \quad \forall\bar{a}\exists X\bar{Y}.U \equiv \text{true}}{\begin{array}{l} S[\text{letr } x = \forall\bar{a}\lambda X.\exists\bar{Y}.[] \text{ in } C] \,;\, U \,;\, \text{true} \\ \to S[\text{letr } x = \forall\bar{a}\lambda X.\exists\bar{Y}.U \text{ in } []] \,;\, \text{true} \,;\, C \end{array}}$$

Exemple

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

Variables rigides et problèmes de partage

$$\dfrac{\textsc{Build-Rigid-Scheme} \\ X\bar{Y} \# ftv(U_1) \wedge a \# U_1 \wedge \forall \bar{a} \exists X \bar{Y}.U_2 \equiv \mathrm{true}}{\begin{array}{c} S[\mathrm{letr}\ x = \forall \bar{a} \lambda X.\exists \bar{Y}.[]\ \mathrm{in}\ C]\ ;\ U_1 \wedge U_2\ ;\ \mathrm{true} \\ \rightarrow S[\mathrm{letr}\ x = \forall \bar{a} \lambda X.\exists \bar{Y}.U_2\ \mathrm{in}\ []]\ ;\ U_1\ ;\ C \end{array}}$$

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