Inférence par contraintes pour les GADTs

Olivier Martinot
encadré par Gabriel Scherer et François Pottier

Inria, Université Paris Cité

Lundi 2 décembre 2024

Membres du jury :

Catherine Dubois
Jacques Garrigue
Jean-Christophe Filliâtre
Adrien Guatto
Caterina Urban

Problème

- Le typeur d'OCaml est difficile à comprendre et à maintenir.
- Une autre approche : l'inférence par résolution de contraintes
- Est-ce qu'on saurait faire basculer le typeur entier vers cette approche ?
- Réponse partielle : on présente le typage d'une des fonctionnalités avancés (types de données algébriques généralisés) avec cette approche.

1. Programme, typage et inférence de types

2. Inférence de types par résolution de contraintes

3. Inférence par contrainte
pour les types algébriques généralisés

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 :
  - format de donnée invalide
  - problème d'alignement mémoire
  - erreur d'approximation
  - ...
- On veut des garanties sur les exécutions de nos programmes
- Une approche : s'intéresser à la façon dont on manipule les données dans nos programmes, annoter les programmes avec des types

Sur un exemple

- Casserole : Ustensile
- 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

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 implicites d'un programme.

Améliorer le confort de programmation :

- gagner du temps
- code plus agréable

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 contraintes

- W = X $\rightarrow$ Y

Exemple : type de succ ?

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

Nos contraintes

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

Ce qu'on sait

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

Donc

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

Exemple : type de succ ?

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

Nos contraintes

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

Exemple : type de succ ?

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

Nos contraintes

- $\texttt{W} = \texttt{X} \to \texttt{Y}$
- `n : int`
- `Y = int`

Exemple : type de succ ?

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

Nos contraintes

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

Exemple : type de succ ?

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

Nos contraintes

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

Section 2

Inférence de types par résolution de contraintes

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 $\quad W = X \to Y \quad \wedge \quad X = \ldots \quad \wedge \quad \ldots$

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	$W = \mathtt{int} \to \mathtt{int}\ ;\ X = \mathtt{int}\ ;\ \ldots$

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	$W = \mathtt{int} \to \mathtt{int}\ ;\ X = \mathtt{int}\ ;\ \dots$
Élaboration d'un terme annoté	`fun (n : int) → n + 1`

Sémantique des contraintes

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

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

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

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

Solveur de contraintes

Pottier and Rémy (2005)

Règles de réécriture

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

$$S\ ;\ U\ ;\ C \to S'\ ;\ U'\ ;\ C'$$

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

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

Solveur de contraintes

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

$$
\begin{array}{ll}
S\ ;\ U\ ;\ C_1 \wedge C_2 & \rightarrow S[[] \wedge C_2]\ ;\ U\ ;\ C_1 \\
S[[] \wedge C]\ ;\ U\ ;\ \text{true} & \rightarrow S\ ;\ U\ ;\ C \\
\ldots &
\end{array}
$$

Solveur de contraintes

Multi-équations

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

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

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

Inférence par contrainte pour les types algébriques généralisés

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.

Autres avantages : typage plus fin, performance.

Garrigue and Rémy (2013)

Generalized Algebraic Datatype (GADT)

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

Programme OCaml :

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

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

Avec égalité explicite :

```
type 'a 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, Refl) -> n
  | Bool (b, Refl) -> 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
       This instance of a is ambiguous:
       it would escape the scope of its equation
*)
```

Contributions

Présentation par contraintes des GADTs ambivalents (noyau)

- Hypothèses d'égalité ambivalentes
- Sémantique ambivalente
- Solveur (non prouvé)
- Implémentation dans Inferno
- Structures abstraites

Contrainte d'hypothèse d'égalité

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

Contrainte d'hypothèse d'égalité

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

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

Contrainte (simplifiée)

$\forall a.\ (a = \mathtt{int}) \Rightarrow \exists Y.Y \text{ is } a \wedge Y \text{ is } \mathtt{int}$

Contrainte d'hypothèse d'égalité

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

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

Contrainte (simplifiée)

$\forall a.\ (a = \mathtt{int}) \Rightarrow \exists Y.Y \text{ is } a \wedge Y \text{ is } \mathtt{int}$

Résolution

$$Y = a \ \wedge \ Y = \mathtt{int} \quad \rightarrow \quad Y = a = \mathtt{int} \quad \rightarrow \quad \phi : a = \mathtt{int} \vdash Y = a$$

Multi-équations avec égalités

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

$$\Phi \vdash \epsilon$$

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

Résolution

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

Exemple

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

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

Résolution

POP-EQ

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

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

POP-EQ

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

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

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

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

Niveau $\approx$ profondeur de définition d'une variable d'inférence
Portée $\approx$ profondeur de définition d'une hypothèse d'égalité
Exemple :

$$\exists X.\ (\phi :\ \tau_1\ =\ \tau_2) \Rightarrow \exists Y_1 Y_2 \ldots$$

Multi-équation $\Phi \vdash \bar{X}\ [= s\,\bar{Y}]$:
Niveau $\approx$ min des niveaux de $\bar{X}$
Portée $\approx$ max des portées de Φ
Si portée $>$ niveau : erreur

Variables rigides et problèmes de partage

Résoudre $(X = a) \wedge (Z = a) \wedge (Z = \texttt{int})$ dans le contexte $\phi : a = \texttt{int}$

a est une variable ?

$\phi \vdash X = Z = a$

a est une structure ?

$(\vdash X = a) \quad \wedge \quad (\phi \vdash Z = a)$

Nouvelle construction : structure abstraite locale.

Comment adapter la généralisation ?

Structure abstraite locale

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

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

Exemples

$$(\forall a . \exists X . X = a \to a) \equiv \mathsf{true}$$
$$(\forall a . \exists X . X = a = \mathtt{int}) \not\equiv \mathsf{true}$$

Sémantique de la contrainte d'hypothèse d'égalité

Pour finir : quelle sémantique donner à ces nouvelles constructions ?

Sémantique naturelle

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

Avantages : simple, le solveur est correct (conjecture)

Inconvénient : le solveur n'est pas complet

Exemple :

$$\frac{\dfrac{\forall t,\ \exists t', \quad \emptyset; [a \mapsto t, X \mapsto t'] \models (a = \mathtt{int}) \Rightarrow X \text{ is } a \wedge X \text{ is } \mathtt{int}}{\forall t, \quad \emptyset; [a \mapsto t] \models \exists X.(a = \mathtt{int}) \Rightarrow X \text{ is } a \wedge X \text{ is } \mathtt{int}}}{\emptyset; \emptyset \models \forall a\, \exists X.\ (a = \mathtt{int}) \Rightarrow X \text{ is } a \wedge X \text{ is } \mathtt{int}}$$

Sémantique ambivalente

$$\psi ::= \emptyset \mid t \approx \psi \qquad\qquad \kappa ::= \mathsf{true} \mid \mathsf{false}$$

$$\frac{\kappa \ \wedge \ (\gamma(\tau_1) = \gamma(\tau_2)) \ ; \ E \ ; \ \gamma \quad \models^{\mathsf{amb}} \quad C}{\kappa; E; \gamma \models^{\mathsf{amb}} (\tau_1 = \tau_2) \Rightarrow C}$$

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

Exemple :

$$\frac{\dfrac{\forall t,\ \exists \psi, \quad t = \mathtt{int}; \emptyset; [a \mapsto t, X \mapsto \psi] \models X \text{ is } a \wedge X \text{ is } \mathtt{int}}{\forall t, \quad t = \mathtt{int}; \emptyset; [a \mapsto t] \models \exists X. X \text{ is } a \wedge X \text{ is } \mathtt{int}}}{\mathsf{true}; \emptyset; \emptyset \models \forall a.\ (a = \mathtt{int}) \Rightarrow \exists X.\ X \text{ is } a \wedge X \text{ is } \mathtt{int}}$$

On peut choisir $\psi = \{t, \mathtt{int}\}$.
Mais $\exists X.(a = \mathtt{int}) \Rightarrow \ldots$ ne va plus marcher.

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