

Principes des lang. de progr. INE 11

Michel Mauny

ENSTA ParisTech

Prénom.Nom@ensta.fr

Navigation icons

Michel Mauny (ENSTA ParisTech)

INE 11

Prénom.Nom@ensta.fr

1 / 22

Typage et synthèse de types

- 1 Inférence de types de PCF
- 2 Polymorphisme
- 3 Algorithme

Navigation icons

Michel Mauny (ENSTA ParisTech)

INE 11

Prénom.Nom@ensta.fr

2 / 22

To type or not to type?

1/2

Pas de typage

- grande expressivité des programmes
- programmation «proche de la machine» (OS, drivers)
- risques importants (arrêts **après** l'erreur)
- assembleur, C (avec warnings, quelquefois)

Typage dynamique

- grande expressivité des programmes
- beaucoup de tests de type (à chaque pas d'exécution, efficacité, représentation des données)
- Lisp, Scheme, Perl, Python, etc.

Navigation icons

Michel Mauny (ENSTA ParisTech)

INE 11

Prénom.Nom@ensta.fr

3 / 22

Typage statique

- moins d'expressivité, quoique ...
- factorisation des tests (éliminés car statique)
- ML, OCaml, C++, Java, Haskell

Inférence vs. vérification

- écrire les programmes sans de préoccuper des types
- écrire les types, c'est spécifier un peu 😎
- plus facile (pour le concepteur de langage) de vérifier que d'inférer

PCF, typage statique**Rappel : syntaxe**

$$e ::= n \in \mathbb{N} \mid b \in \text{bool} \mid x \mid e_1 e_2 \mid \text{if } e_1 \text{ then } e_2 \text{ else } e_3$$

$$\mid e_1 + e_2 \mid e_1 = e_2$$

$$\mid \text{let } x = e_1 \text{ in } e_2$$

$$\mid \text{letrec } f(x) = e_1 \text{ in } e_2$$

$$\mid \text{fun } x \rightarrow e$$
Éviter les tests dynamiques

- n'additionner / ne tester l'égalité que des entiers!
- ne tester que des valeurs booléennes!
- n'appliquer que des fonctions!

PCF, typage statique**Prédire, c'est faire des choix**

- **if** *booléen* **then** *entier* **else** *fonction*
- peut-on l'appliquer ? l'additionner ?
- la conditionnelle ne peut rendre qu'un type
- «*entier* ou *fonction*» n'est pas un type
- on force les deux types de retour à être identiques, éliminant certains programmes qui, pourtant, peuvent avoir un sens

Les types de PCF

De quoi avons-nous besoin ?

$\tau ::= \text{int} \mid \text{bool} \mid \tau_1 \rightarrow \tau_2$

Calculer les types

- **let rec** $\text{fact}^{\text{int} \rightarrow \text{int}} \ n^{\text{int}} =$
 (if $(n^{\text{int}} = 0^{\text{int}})^{\text{bool}}$ **then**
 1^{int}
 else
 $(n^{\text{int}} * (\text{fact}^{\text{int} \rightarrow \text{int}}(n^{\text{int}} - 1^{\text{int}})^{\text{int}})^{\text{int}})^{\text{int}}$
- permet de les vérifier aussi
 vérifier = calculer, puis comparer
- sens de «comparer» : égalité, compatibilité?
 comparer = exiger, forcer l'égalité

Navigation icons

Les différents cas

1/2

$n \in \mathbb{N}$: du type int

$b \in \text{bool}$: du type bool

if e_1 **then** e_2 **else** e_3

le type de e_1 doit être bool ; e_2 et e_3 doivent être compatibles ;

$e_1 + e_2$ et $e_1 = e_2$

arguments entiers, résultats entier ou booléen

let $x = e_1$ **in** e_2

calculer le type de e_1 et l'associer à x durant le typage de e_2

⇒ environnement de typage Γ

Navigation icons

Les différents cas

2/2

$x : \Gamma(x)$

(fun $x \rightarrow e$) : $\tau_1 \rightarrow \tau_2$

calculer τ_2 pour e en supposant τ_1 pour x

C'est magique ?

$(e_1 e_2)$:

calcul de $(\tau_2 \rightarrow \tau)$ pour e_1 ; calcul de τ_2' pour e_2 ; identifier τ_2 et τ_2'

letrec $f(x) = e_1$ **in** e_2

- calculer le type $(\tau_f \rightarrow \tau_x \rightarrow \tau_{e_1})$ de **(fun** $f \rightarrow$ **(fun** $x \rightarrow e_1)$) en forçant l'égalité entre τ_f et $\tau_x \rightarrow \tau_{e_1}$;
- puis calculer le type de e_2 en supposant $f : \tau_x \rightarrow \tau_{e_1}$

Navigation icons

Types

$$\tau ::= \alpha \mid \text{int} \mid \text{bool} \mid \tau_1 \rightarrow \tau_2$$

Règles

$$\Gamma \vdash n : \text{int} \quad \Gamma \vdash b : \text{bool} \quad \frac{x \in \text{dom}(\Gamma) \quad \Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash (\text{if } e_1 \text{ then } e_2 \text{ else } e_3) : \tau} \quad \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash (e_1 + e_2) : \text{int}}$$

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash (e_1 = e_2) : \text{bool}}$$

Règles (suite)

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \oplus [x : \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash (\text{let } x = e_1 \text{ in } e_2) : \tau_2} \quad \frac{\Gamma \oplus [x : \tau_1] \vdash e : \tau_2}{\Gamma \vdash (\text{fun } x \rightarrow e) : \tau_1 \rightarrow \tau_2}$$

$$\frac{\Gamma \vdash e_1 : \tau_2 \rightarrow \tau \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1 e_2) : \tau}$$

$$\frac{\Gamma \oplus [f : \tau \rightarrow \tau_1; x : \tau] \vdash e_1 : \tau_1 \quad \Gamma \oplus [f : \tau \rightarrow \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash (\text{letrec } f(x) = e_1 \text{ in } e_2) : \tau_2}$$

Pas de règles d'erreur

si on peut pas construire d'arbre de dérivation pour un jugement, il n'est pas valide

Une règle par construction

on est proche de l'algorithme!

Plusieurs occurrences de la même méta-variable (τ_i)

unification pour résoudre ces équations

Polymorphisme à la ML

Jugement pas prouvable

- $\Gamma \vdash (\text{let } f = \text{fun } x \rightarrow x \text{ in } (f f)(1)) : \text{int}$

Pourquoi ?

- $\text{let } f^{\tau \rightarrow \tau} = \text{fun } x^{\tau} \rightarrow x^{\tau} \text{ in } (f^{(i \rightarrow i) \rightarrow (i \rightarrow i)} f^{i \rightarrow i})^{i \rightarrow i(1^i)}$
- $\nexists \tau$ tel que $\tau = i \rightarrow i$ et $\tau = (i \rightarrow i) \rightarrow (i \rightarrow i)$
 f demande à avoir deux types incompatibles, **non unifiables**

Solution : schémas de types

- $f : \forall \alpha. (\alpha \rightarrow \alpha)$
- instanciable en tous les types requis
- polymorphisme **paramétrique**

Navigation icons

Polymorphisme à la ML

Types et schémas

$\tau ::= \alpha \mid \text{int} \mid \text{bool} \mid \tau_1 \rightarrow \tau_2$
 $\sigma ::= \forall \vec{\alpha}. \tau$

Un type est aussi un schéma de types (quand $\vec{\alpha} = \emptyset$)

Instanciation générique

$\tau_2 \leq \forall \vec{\alpha}. \tau_1$, si il existe une substitution θ dont le domaine est $\vec{\alpha}$ et telle que $\tau_2 = \theta(\tau_1)$

Généralisation

$\text{gen}(\tau, \Gamma) = \forall \vec{\alpha}. \tau$ où $\vec{\alpha} = \text{vars}(\tau) \setminus \text{fv}(\Gamma)$

correspond à identifier un ensemble de solutions à un sous-système d'équations

Navigation icons

Polymorphisme à la ML

Règles

[règles grisées : inchangées par rapport au système monomorphe]

$\Gamma \vdash n : \text{int}$ $\Gamma \vdash b : \text{bool}$ $\frac{x \in \text{dom}(\Gamma) \quad \tau \leq \Gamma(x)}{\Gamma \vdash x : \tau}$

$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash (\text{if } e_1 \text{ then } e_2 \text{ else } e_3) : \tau}$ $\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash (e_1 + e_2) : \text{int}}$

$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash (e_1 = e_2) : \text{bool}}$

Navigation icons

Polymorphisme à la ML

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \oplus [x : \text{gen}(\tau_1, \Gamma)] \vdash e_2 : \tau_2}{\Gamma \vdash (\text{let } x = e_1 \text{ in } e_2) : \tau_2}$$
$$\frac{\Gamma \oplus [x : \tau_1] \vdash e : \tau_2}{\Gamma \vdash (\text{fun } x \rightarrow e) : \tau_1 \rightarrow \tau_2} \quad \frac{\Gamma \vdash e_1 : \tau_2 \rightarrow \tau \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1 e_2) : \tau}$$
$$\frac{\Gamma \oplus [f : \tau \rightarrow \tau_1; x : \tau] \vdash e_1 : \tau_1 \quad \Gamma \oplus [f : \text{gen}(\tau \rightarrow \tau_1, \Gamma)] \vdash e_2 : \tau_2}{\Gamma \vdash (\text{letrec } f(x) = e_1 \text{ in } e_2) : \tau_2}$$

Navigation icons

Propriétés

Correction

Soit $\Gamma \vdash e : \tau$, et $\rho : \Gamma$.

Si $\rho \vdash e \Rightarrow r$, alors r est une valeur v et son type est τ

Corollaire

Un programme typable s'exécute sans erreur de type.

Well-typed programs do not go wrong

Navigation icons

Algorithme d'inférence

1/2

Pour e et Γ , produit un type τ et une substitution θ tels que :

$$\theta(\Gamma) \vdash e : \tau$$

let rec infer $\Gamma e = \text{match } e \text{ with}$

 «n» $\rightarrow (\text{int}, [])$

 | «x» $\rightarrow (\text{instance}(\Gamma(x), [])$

 | « $e_1 e_2$ » \rightarrow

let $(\tau_1, \theta_1) = \text{infer } \Gamma e_1 \text{ in}$

let $(\tau_2, \theta_2) = \text{infer } (\theta_1(\Gamma)) e_2 \text{ in}$

let $\tau_3 = \text{nouvelle variable in}$

let $\theta_3 = \text{unif } (\theta_2(\tau_1)) (\tau_2 \rightarrow \tau_3) \text{ in}$

$(\theta_3(\tau_3), \theta_3 \circ \theta_2 \circ \theta_1)$

 ...

Navigation icons

```

let rec infer  $\Gamma$  e = match e with
  ...
  | «(fun x  $\rightarrow$  e)»  $\rightarrow$ 
    let  $\tau_0$  = nouvelle variable in
    let ( $\tau_1, \theta_1$ ) = infer ( $\Gamma \oplus [x : \tau_0]$ ) e in
    (( $\theta_1(\tau_0) \rightarrow \tau_1$ ),  $\theta_1$ )
  | «(let x = e1 in e2)»  $\rightarrow$ 
    let ( $\tau_1, \theta_1$ ) = infer  $\Gamma$  e1 in
    let ( $\tau_2, \theta_2$ ) = infer ( $\theta_1(\Gamma) \oplus [x : \text{gen}(\tau_1, \theta_1(\Gamma))]$ ) e2 in
    ( $\tau_2, \theta_2 \circ \theta_1$ )
  | ...
    
```



Propriétés de l'algorithme

Correction

si $\text{infer } \Gamma e = (\tau, \theta)$, alors $\theta(\Gamma) \vdash e : \tau$

Complétude

pour un programme typable, l'algorithme va calculer son type *le plus général*, c'est-à-dire celui dont tous ses autres types sont des instances



Conclusion

Typage statique

- échange de la sécurité contre de la flexibilité
- tenir compte du typage dans l'organisation du code
- éviter la génération de (certains) tests dynamiques

Inférence de types

- évite de spécifier les types des données manipulées
- propage l'information de typage

Propriétés

- correction du typage vis-à-vis de la sémantique
- correction et complétude de l'algorithme



