

Typage

Samuel Mimram

École polytechnique

Première partie I

Réduction

La sémantique à grands pas

- n'est pas modulaire : elle calcule directement le résultat,
- ne permet pas de raisonner sur des programmes avec des évaluations infinies (par exemple, un serveur web).

La sémantique à grands pas

- n'est pas modulaire : elle calcule directement le résultat,
- ne permet pas de raisonner sur des programmes avec des évaluations infinies (par exemple, un serveur web).

En particulier, on aimerait pouvoir distinguer entre

Sémantique à petits pas

La sémantique à grands pas

- n'est pas modulaire : elle calcule directement le résultat,
- ne permet pas de raisonner sur des programmes avec des évaluations infinies (par exemple, un serveur web).

En particulier, on aimerait pouvoir distinguer entre

- des programmes qui bouclent (acceptable) :

```
fix (fun x → x)
```

Sémantique à petits pas

La sémantique à grands pas

- n'est pas modulaire : elle calcule directement le résultat,
- ne permet pas de raisonner sur des programmes avec des évaluations infinies (par exemple, un serveur web).

En particulier, on aimerait pouvoir distinguer entre

- des programmes qui bouclent (acceptable) :

```
fix (fun x → x)
```

- des erreurs (inacceptable) :

```
true + 5
```

Sémantique à petits pas

La sémantique à grands pas

- n'est pas modulaire : elle calcule directement le résultat,
- ne permet pas de raisonner sur des programmes avec des évaluations infinies (par exemple, un serveur web).

En particulier, on aimerait pouvoir distinguer entre

- des programmes qui bouclent (acceptable) :

```
fix (fun x → x)
```

- des erreurs (inacceptable) :

```
true + 5
```

Or, dans les deux cas, l'évaluation n'est pas définie.

On définit la relation \longrightarrow de **réduction** entre termes par les règles suivantes.

Dans ces règles v, v_i désignent toujours des valeurs :

$$\begin{aligned} v \quad ::= & \quad \underline{n} \mid \underline{b} \mid \text{add} \mid \text{add } \underline{n} \\ & \mid \text{fun } x \rightarrow t \mid \text{fix} \\ & \mid (v, v') \mid \text{fst} \mid \text{snd} \end{aligned}$$

On est loin d'avoir un choix unique pour ces règles :
on va décrire la **stratégie** que l'on implémente pour réduire.

On est loin d'avoir un choix unique pour ces règles :
on va décrire la **stratégie** que l'on implémente pour réduire.

On va ici prendre une stratégie proche de celle de OCaml.

Réduction

On est loin d'avoir un choix unique pour ces règles :
on va décrire la **stratégie** que l'on implémente pour réduire.

On va ici prendre une stratégie proche de celle de OCaml.

Par exemple, en OCaml, le programme

```
let p = print_string
let _ = (p "A"; (fun x y -> p "B"; x + y)) (p "C"; 2) (p "D"; 3)
```

affiche

Réduction

On est loin d'avoir un choix unique pour ces règles :
on va décrire la **stratégie** que l'on implémente pour réduire.

On va ici prendre une stratégie proche de celle de OCaml.

Par exemple, en OCaml, le programme

```
let p = print_string
let _ = (p "A"; (fun x y -> p "B"; x + y)) (p "C"; 2) (p "D"; 3)
```

affiche

DCAB

- application :

– application :

$$\frac{u \longrightarrow u'}{t u \longrightarrow t u'} \text{ (APPR)}$$

– application :

$$\frac{u \longrightarrow u'}{t u \longrightarrow t u'} \text{ (APPR)}$$

$$\frac{t \longrightarrow t'}{t v \longrightarrow t' v} \text{ (APPL)}$$

– application :

$$\frac{u \longrightarrow u'}{t u \longrightarrow t u'} \text{ (APPR)}$$

$$\frac{t \longrightarrow t'}{t v \longrightarrow t' v} \text{ (APPL)}$$

$$\frac{}{(\text{fun } x \rightarrow t) v \longrightarrow t[x \mapsto v]} \text{ (APP)}$$

Réduction

– application :

$$\frac{u \longrightarrow u'}{t u \longrightarrow t u'} \text{ (APPR)}$$

$$\frac{t \longrightarrow t'}{t v \longrightarrow t' v} \text{ (APPL)}$$

$$\frac{}{(\text{fun } x \rightarrow t) v \longrightarrow t[x \mapsto v]} \text{ (APP)}$$

$$\frac{}{\text{add } \underline{m} \ \underline{n} \longrightarrow \underline{m + n}} \text{ (ADD)}$$

Réduction

– application :

$$\frac{u \longrightarrow u'}{t u \longrightarrow t u'} \text{ (APPR)}$$

$$\frac{t \longrightarrow t'}{t v \longrightarrow t' v} \text{ (APPL)}$$

$$\frac{}{(\text{fun } x \rightarrow t) v \longrightarrow t[x \mapsto v]} \text{ (APP)}$$

$$\frac{}{\text{add } \underline{m} \ \underline{n} \longrightarrow \underline{m + n}} \text{ (ADD)}$$

– paires :

Réduction

– application :

$$\frac{u \longrightarrow u'}{t u \longrightarrow t u'} \text{ (APPR)}$$

$$\frac{t \longrightarrow t'}{t v \longrightarrow t' v} \text{ (APPL)}$$

$$\frac{}{(\text{fun } x \rightarrow t) v \longrightarrow t[x \mapsto v]} \text{ (APP)}$$

$$\frac{}{\text{add } \underline{m} \ \underline{n} \longrightarrow \underline{m + n}} \text{ (ADD)}$$

– paires :

$$\frac{t \longrightarrow t'}{(t, v) \longrightarrow (t', v)} \text{ (PAIRL)}$$

$$\frac{u \longrightarrow u'}{(t, u) \longrightarrow (t, u')} \text{ (PAIRR)}$$

Réduction

– application :

$$\frac{u \longrightarrow u'}{t u \longrightarrow t u'} \text{ (APPR)}$$

$$\frac{t \longrightarrow t'}{t v \longrightarrow t' v} \text{ (APPL)}$$

$$\frac{}{(\text{fun } x \rightarrow t) v \longrightarrow t[x \mapsto v]} \text{ (APP)}$$

$$\frac{}{\text{add } \underline{m} \ \underline{n} \longrightarrow \underline{m + n}} \text{ (ADD)}$$

– paires :

$$\frac{t \longrightarrow t'}{(t, v) \longrightarrow (t', v)} \text{ (PAIRL)}$$

$$\frac{u \longrightarrow u'}{(t, u) \longrightarrow (t, u')} \text{ (PAIRR)}$$

$$\frac{}{\text{fst } (v_1, v_2) \longrightarrow v_1} \text{ (FST)}$$

Réduction

– application :

$$\frac{u \longrightarrow u'}{t u \longrightarrow t u'} \text{ (APPR)}$$

$$\frac{t \longrightarrow t'}{t v \longrightarrow t' v} \text{ (APPL)}$$

$$\frac{}{(\text{fun } x \rightarrow t) v \longrightarrow t[x \mapsto v]} \text{ (APP)}$$

$$\frac{}{\text{add } \underline{m} \ \underline{n} \longrightarrow \underline{m + n}} \text{ (ADD)}$$

– paires :

$$\frac{t \longrightarrow t'}{(t, v) \longrightarrow (t', v)} \text{ (PAIRL)}$$

$$\frac{u \longrightarrow u'}{(t, u) \longrightarrow (t, u')} \text{ (PAIRR)}$$

$$\frac{}{\text{fst } (v_1, v_2) \longrightarrow v_1} \text{ (FST)}$$

$$\frac{}{\text{snd } (v_1, v_2) \longrightarrow v_2} \text{ (SND)}$$

- opérateur point fixe :

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\text{fix } t \longrightarrow \text{fix } t'}$$

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\text{fix } t \longrightarrow \text{fix } t'}$$

$$\frac{}{\text{fix } v \longrightarrow v (\text{fix } v)}$$

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\text{fix } t \longrightarrow \text{fix } t'}$$

$$\frac{}{\text{fix } v \longrightarrow v (\text{fix } v)}$$

ne conviennent pas,

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\text{fix } t \longrightarrow \text{fix } t'}$$

$$\frac{}{\text{fix } v \longrightarrow v (\text{fix } v)}$$

ne conviennent pas, car on a toujours la réduction infinie

$$\text{fix } t \xrightarrow{*}$$

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\text{fix } t \longrightarrow \text{fix } t'}$$

$$\frac{}{\text{fix } v \longrightarrow v (\text{fix } v)}$$

ne conviennent pas, car on a toujours la réduction infinie

$$\text{fix } t \xrightarrow{*} \text{fix } v \longrightarrow$$

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\text{fix } t \longrightarrow \text{fix } t'}$$

$$\frac{}{\text{fix } v \longrightarrow v (\text{fix } v)}$$

ne conviennent pas, car on a toujours la réduction infinie

$$\text{fix } t \xrightarrow{*} \text{fix } v \longrightarrow v (\text{fix } v) \longrightarrow$$

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\text{fix } t \longrightarrow \text{fix } t'}$$

$$\frac{}{\text{fix } v \longrightarrow v (\text{fix } v)}$$

ne conviennent pas, car on a toujours la réduction infinie

$$\text{fix } t \xrightarrow{*} \text{fix } v \longrightarrow v (\text{fix } v) \longrightarrow v (v (\text{fix } v)) \longrightarrow$$

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\text{fix } t \longrightarrow \text{fix } t'}$$

$$\frac{}{\text{fix } v \longrightarrow v (\text{fix } v)}$$

ne conviennent pas, car on a toujours la réduction infinie

$$\text{fix } t \xrightarrow{*} \text{fix } v \longrightarrow v (\text{fix } v) \longrightarrow v (v (\text{fix } v)) \longrightarrow v (v (v (\text{fix } v))) \longrightarrow \dots$$

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\text{fix } t \longrightarrow \text{fix } t'}$$

$$\frac{}{\text{fix } v \longrightarrow v (\text{fix } v)}$$

ne conviennent pas, car on a toujours la réduction infinie

$$\text{fix } t \xrightarrow{*} \text{fix } v \longrightarrow v (\text{fix } v) \longrightarrow v (v (\text{fix } v)) \longrightarrow v (v (v (\text{fix } v))) \longrightarrow \dots$$

on prendra plutôt les règles :

Réduction

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\text{fix } t \longrightarrow \text{fix } t'}$$

$$\frac{}{\text{fix } v \longrightarrow v (\text{fix } v)}$$

ne conviennent pas, car on a toujours la réduction infinie

$$\text{fix } t \xrightarrow{*} \text{fix } v \longrightarrow v (\text{fix } v) \longrightarrow v (v (\text{fix } v)) \longrightarrow v (v (v (\text{fix } v))) \longrightarrow \dots$$

on prendra plutôt les règles :

$$\frac{t \longrightarrow t'}{\text{fix } t \longrightarrow \text{fix } t'}$$

$$\frac{}{\text{fix } (\text{fun } x \rightarrow t) \longrightarrow t[x \mapsto \text{fix } (\text{fun } x \rightarrow t)]}$$

Stratégies de réduction

Il existe plusieurs **stratégies de réduction** :

- en *appel par valeur* : on évalue d'abord les arguments

$$(\text{fun } x \rightarrow x + x) (3 + 2) \longrightarrow (\text{fun } x \rightarrow x + x) 5 \longrightarrow 5 + 5 \longrightarrow 10$$

Stratégies de réduction

Il existe plusieurs **stratégies de réduction** :

- en *appel par valeur* : on évalue d'abord les arguments

$$(\text{fun } x \rightarrow x + x) (3 + 2) \longrightarrow (\text{fun } x \rightarrow x + x) 5 \longrightarrow 5 + 5 \longrightarrow 10$$

- en *appel par nom* : on applique d'abord les fonctions aux arguments

$$(\text{fun } x \rightarrow x + x) (3 + 2) \longrightarrow (3 + 2) + (3 + 2) \longrightarrow 5 + (3 + 2) \longrightarrow 5 + 5 \longrightarrow 10$$

Stratégies de réduction

Il existe plusieurs **stratégies de réduction** :

- en *appel par valeur* : on évalue d'abord les arguments

$$(\text{fun } x \rightarrow x + x) (3 + 2) \longrightarrow (\text{fun } x \rightarrow x + x) 5 \longrightarrow 5 + 5 \longrightarrow 10$$

- en *appel par nom* : on applique d'abord les fonctions aux arguments

$$(\text{fun } x \rightarrow x + x) (3 + 2) \longrightarrow (3 + 2) + (3 + 2) \longrightarrow 5 + (3 + 2) \longrightarrow 5 + 5 \longrightarrow 10$$

- une stratégie *faible* n'évalue pas sous les abstractions, sinon on pourrait faire

$$(\text{fun } x \rightarrow 3 + 2) \longrightarrow (\text{fun } x \rightarrow 5)$$

Formes normales

Un terme est une **forme normale** lorsqu'il ne peut pas se réduire.

Formes normales

Un terme est une **forme normale** lorsqu'il ne peut pas se réduire.

Lemme

Toute valeur est une forme normale.

Démonstration.

Les valeurs sont

$$\begin{aligned} v \quad ::= & \quad \underline{n} \mid \underline{b} \mid \text{add} \mid \text{add } \underline{n} \\ & \mid \text{fun } x \rightarrow t \mid \text{fix} \\ & \mid (v, v') \mid \text{fst} \mid \text{snd} \end{aligned}$$


Formes normales

Un terme est une **forme normale** lorsqu'il ne peut pas se réduire.

Lemme

Toute valeur est une forme normale.

Réciproquement...

Formes normales

Un terme est une **forme normale** lorsqu'il ne peut pas se réduire.

Lemme

Toute valeur est une forme normale.

La réciproque n'est pas vraie :

`true + false`

Formes normales

Un terme est une **forme normale** lorsqu'il ne peut pas se réduire.

Lemme

Toute valeur est une forme normale.

La réciproque n'est pas vraie :

`true + false`

Les formes normales qui ne sont pas des valeurs sont précisément les états d'erreur !

Formes normales

Un terme est une **forme normale** lorsqu'il ne peut pas se réduire.

Lemme

Toute valeur est une forme normale.

La réciproque n'est pas vraie :

`true + false`

Les formes normales qui ne sont pas des valeurs sont précisément les états d'erreur !

Nous verrons que le typage permet de les éviter.

Déterminisme de la réduction

Proposition

La réduction est déterministe : si $t \longrightarrow u$ et $t \longrightarrow u'$ alors $u = u'$.

Déterminisme de la réduction

Proposition

La réduction est déterministe : si $t \longrightarrow u$ et $t \longrightarrow u'$ alors $u = u'$.

Démonstration.

Par induction, sur le terme t .

- Si $t = (t_1, t_2)$ alors les deux règles qui s'appliquent sont

$$\frac{t \longrightarrow t'}{(t, v) \longrightarrow (t', v)} \text{ (PAIRL)}$$

$$\frac{u \longrightarrow u'}{(t, u) \longrightarrow (t, u')} \text{ (PAIRR)}$$

Elles ne s'appliquent pas simultanément, car les valeurs sont normales.

Déterminisme de la réduction

Proposition

La réduction est déterministe : si $t \longrightarrow u$ et $t \longrightarrow u'$ alors $u = u'$.

Démonstration.

Par induction, sur le terme t .

- Si $t = (t_1, t_2)$ alors les deux règles qui s'appliquent sont

$$\frac{t \longrightarrow t'}{(t, v) \longrightarrow (t', v)} \text{ (PAIRL)}$$

$$\frac{u \longrightarrow u'}{(t, u) \longrightarrow (t, u')} \text{ (PAIRR)}$$

Elles ne s'appliquent pas simultanément, car les valeurs sont normales.

- Les autres cas sont similaires.

□

Déterminisme de la réduction

Proposition

La réduction est déterministe : si $t \longrightarrow u$ et $t \longrightarrow u'$ alors $u = u'$.

Démonstration.

Par induction, sur le terme t .

- Si $t = (t_1, t_2)$ alors les deux règles qui s'appliquent sont

$$\frac{t \longrightarrow t'}{(t, v) \longrightarrow (t', v)} \text{ (PAIRL)}$$

$$\frac{u \longrightarrow u'}{(t, u) \longrightarrow (t, u')} \text{ (PAIRR)}$$

Elles ne s'appliquent pas simultanément, car les valeurs sont normales.

- Les autres cas sont similaires.

□

Notons qu'une variante non-déterministe aurait été raisonnable.

Équivalence entre les sémantiques

On note $t \xrightarrow{*} u$ lorsqu'il existe une suite de réductions de t à u .

Nous allons montrer que l'évaluation coïncide avec la réduction :

Équivalence entre les sémantiques

On note $t \xrightarrow{*} u$ lorsqu'il existe une suite de réductions de t à u .

Nous allons montrer que l'évaluation coïncide avec la réduction :

Théorème

Pour tout terme t et valeur v , $t \xrightarrow{} v$ ssi $t \longrightarrow v$.*

Équivalence entre les sémantiques

Proposition

Si $t \longrightarrow v$ alors $t \xrightarrow{} v$.*

Équivalence entre les sémantiques

Proposition

Si $t \longrightarrow v$ alors $t \xrightarrow{} v$.*

Démonstration.

Par induction sur la dérivation de $t \longrightarrow v$.

Équivalence entre les sémantiques

Proposition

Si $t \longrightarrow v$ alors $t \xrightarrow{*} v$.

Démonstration.

Par induction sur la dérivation de $t \longrightarrow v$. Si la dernière règle est

$$\frac{t \longrightarrow (\text{fun } x \rightarrow t') \quad u \longrightarrow u' \quad t'[x \mapsto u'] \longrightarrow v}{t u \longrightarrow v} \text{(APP)}$$

Équivalence entre les sémantiques

Proposition

Si $t \longrightarrow v$ alors $t \xrightarrow{*} v$.

Démonstration.

Par induction sur la dérivation de $t \longrightarrow v$. Si la dernière règle est

$$\frac{t \longrightarrow (\text{fun } x \rightarrow t') \quad u \longrightarrow u' \quad t'[x \mapsto u'] \longrightarrow v}{t u \longrightarrow v} \text{ (APP)}$$

Par hypothèse de récurrence, on a

$$t \xrightarrow{*} (\text{fun } x \rightarrow t') \quad u \xrightarrow{*} u' \quad t'[x \mapsto u'] \xrightarrow{*} v$$

Équivalence entre les sémantiques

Proposition

Si $t \longrightarrow v$ alors $t \xrightarrow{*} v$.

Démonstration.

Par induction sur la dérivation de $t \longrightarrow v$. Si la dernière règle est

$$\frac{t \longrightarrow (\text{fun } x \rightarrow t') \quad u \longrightarrow u' \quad t'[x \mapsto u'] \longrightarrow v}{t u \longrightarrow v} \text{ (APP)}$$

Par hypothèse de récurrence, on a

$$t \xrightarrow{*} (\text{fun } x \rightarrow t') \quad u \xrightarrow{*} u' \quad t'[x \mapsto u'] \xrightarrow{*} v$$

et on en déduit la suite de réductions

$$t u \xrightarrow{*} t u' \xrightarrow{*} (\text{fun } x \rightarrow t') u' \longrightarrow t'[x \mapsto u'] \xrightarrow{*} v$$

Équivalence entre les sémantiques

Proposition

Si $t \longrightarrow v$ alors $t \xrightarrow{*} v$.

Démonstration.

Par hypothèse de récurrence, on a

$$t \xrightarrow{*} (\text{fun } x \rightarrow t') \quad u \xrightarrow{*} u' \quad t'[x \mapsto u'] \xrightarrow{*} v$$

et on en déduit la suite de réductions

$$t u \xrightarrow{*} t u' \xrightarrow{*} (\text{fun } x \rightarrow t') u' \longrightarrow t'[x \mapsto u'] \xrightarrow{*} v$$

Pour les deux premières, il faut d'abord montrer les lemmes auxiliaires suivants :

- si $u \xrightarrow{*} u'$ alors $t u \xrightarrow{*} t u'$,
- si $t \xrightarrow{*} t'$ alors $t v \xrightarrow{*} t' v$ (pour v une valeur),

Équivalence entre les sémantiques

Pour la réciproque, nous avons besoin de lemmes :

Équivalence entre les sémantiques

Pour la réciproque, nous avons besoin de lemmes :

Lemme

Pour toute valeur v , on a $v \longrightarrow v$.

Démonstration.

Par induction sur v .



Équivalence entre les sémantiques

Pour la réciproque, nous avons besoin de lemmes :

Lemme

Pour toute valeur v , on a $v \longrightarrow v$.

Démonstration.

Par induction sur v . □

Lemme

Si $t \longrightarrow t'$ et $t' \longrightarrow v$ alors $t \longrightarrow v$.

Démonstration.

Par induction sur la dérivation de $t \longrightarrow t'$. □

Équivalence entre les sémantiques

Pour la réciproque, nous avons besoin de lemmes :

Lemme

Pour toute valeur v , on a $v \longrightarrow v$.

Lemme

Si $t \longrightarrow t'$ et $t' \longrightarrow v$ alors $t \longrightarrow v$.

Proposition

Si $t \xrightarrow{} v$ pour une valeur v alors $t \longrightarrow v$.*

Équivalence entre les sémantiques

Pour la réciproque, nous avons besoin de lemmes :

Lemme

Pour toute valeur v , on a $v \longrightarrow v$.

Lemme

Si $t \longrightarrow t'$ et $t' \longrightarrow v$ alors $t \longrightarrow v$.

Proposition

Si $t \xrightarrow{} v$ pour une valeur v alors $t \longrightarrow v$.*

Démonstration.

Par récurrence sur la longueur de la réduction $t \xrightarrow{*} v$ en utilisant premier lemme pour le cas de base et le second pour le cas de récurrence. □

Équivalence entre les sémantiques

Nous avons montré :

Théorème

Pour v une valeur, on a $t \xrightarrow{} v$ si et seulement si $t \twoheadrightarrow v$.*

Deuxième partie II

Sûreté du typage

Sûreté du typage

Nous allons montrer que les programmes bien typés sont **sûrs** :

il n'y aura jamais d'erreur à l'exécution.

Sûreté du typage

Nous allons montrer que les programmes bien typés sont **sûrs** :

il n'y aura jamais d'erreur à l'exécution.

Plus précisément, on n'arrivera jamais à une situation telle que

`true + false`

durant l'exécution.

Sûreté du typage

Nous allons montrer que les programmes bien typés sont **sûrs** :

il n'y aura jamais d'erreur à l'exécution.

Plus précisément, on n'arrivera jamais à une situation telle que

`true + false`

durant l'exécution.

Théorème

Si t est bien typé et $t \xrightarrow{} u$ avec u normal alors u est une valeur.*

Sûreté du typage

Nous allons montrer que les programmes bien typés sont **sûrs** :

il n'y aura jamais d'erreur à l'exécution.

Plus précisément, on n'arrivera jamais à une situation telle que

`true + false`

durant l'exécution.

Théorème

Un programme bien typé

- soit finit par se réduire en une valeur,*
- soit s'exécute indéfiniment.*

Compatibilité du typage avec la substitution

Le typage est compatible avec la substitution :

Compatibilité du typage avec la substitution

Le typage est compatible avec la substitution :

Lemme

Si $\Gamma, x : A, \Gamma' \vdash t : B$ et $\Gamma, \Gamma' \vdash u : A$ alors $\Gamma, \Gamma' \vdash t[x \mapsto u] : B$.

Compatibilité du typage avec la substitution

Le typage est compatible avec la substitution :

Lemme

Si $\Gamma, x : A, \Gamma' \vdash t : B$ et $\Gamma, \Gamma' \vdash u : A$ alors $\Gamma, \Gamma' \vdash t[x \mapsto u] : B$.

Démonstration.

Par induction sur la dérivation de $\Gamma, x : A, \Gamma' \vdash t : B$.

Compatibilité du typage avec la substitution

Le typage est compatible avec la substitution :

Lemme

Si $\Gamma, x : A, \Gamma' \vdash t : B$ et $\Gamma, \Gamma' \vdash u : A$ alors $\Gamma, \Gamma' \vdash t[x \mapsto u] : B$.

Démonstration.

Par induction sur la dérivation de $\Gamma, x : A, \Gamma' \vdash t : B$.

- Si la dernière règle est

$$\frac{}{\Gamma, x : A, \Gamma' \vdash \underline{n} : \text{int}} \text{(INT)}$$

alors

Compatibilité du typage avec la substitution

Le typage est compatible avec la substitution :

Lemme

Si $\Gamma, x : A, \Gamma' \vdash t : B$ et $\Gamma, \Gamma' \vdash u : A$ alors $\Gamma, \Gamma' \vdash t[x \mapsto u] : B$.

Démonstration.

Par induction sur la dérivation de $\Gamma, x : A, \Gamma' \vdash t : B$.

- Si la dernière règle est

$$\frac{}{\Gamma, x : A, \Gamma' \vdash \underline{n} : \text{int}} \text{(INT)}$$

alors on a immédiatement $\underline{n}[x \mapsto u] = \underline{n}$ et

$$\frac{}{\Gamma, \Gamma' \vdash \underline{n} : \text{int}} \text{(INT)}$$

Compatibilité du typage avec la substitution

Le typage est compatible avec la substitution :

Lemme

Si $\Gamma, x : A, \Gamma' \vdash t : B$ et $\Gamma, \Gamma' \vdash u : A$ alors $\Gamma, \Gamma' \vdash t[x \mapsto u] : B$.

Démonstration.

Par induction sur la dérivation de $\Gamma, x : A, \Gamma' \vdash t : B$.

- Si la dernière règle est

$$\frac{}{\Gamma, x : A, \Gamma' \vdash \underline{n} : \text{int}} \text{(INT)}$$

alors on a immédiatement $\underline{n}[x \mapsto u] = \underline{n}$ et

$$\frac{}{\Gamma, \Gamma' \vdash \underline{n} : \text{int}} \text{(INT)}$$

Les autres constantes se traitent de façon similaire.

Compatibilité du typage avec la substitution

Le typage est compatible avec la substitution :

Lemme

Si $\Gamma, x : A, \Gamma' \vdash t : B$ et $\Gamma, \Gamma' \vdash u : A$ alors $\Gamma, \Gamma' \vdash t[x \mapsto u] : B$.

Démonstration.

Par induction sur la dérivation de $\Gamma, x : A, \Gamma' \vdash t : B$.

- Si la dernière règle est

$$\frac{}{\Gamma, x : A, \Gamma' \vdash y : A} \text{ (VAR)}$$

avec $x \neq y$ alors

Compatibilité du typage avec la substitution

Le typage est compatible avec la substitution :

Lemme

Si $\Gamma, x : A, \Gamma' \vdash t : B$ et $\Gamma, \Gamma' \vdash u : A$ alors $\Gamma, \Gamma' \vdash t[x \mapsto u] : B$.

Démonstration.

Par induction sur la dérivation de $\Gamma, x : A, \Gamma' \vdash t : B$.

- Si la dernière règle est

$$\frac{}{\Gamma, x : A, \Gamma' \vdash y : A} \text{ (VAR)}$$

avec $x \neq y$ alors on conclut comme dans le cas précédent.

Compatibilité du typage avec la substitution

Le typage est compatible avec la substitution :

Lemme

Si $\Gamma, x : A, \Gamma' \vdash t : B$ et $\Gamma, \Gamma' \vdash u : A$ alors $\Gamma, \Gamma' \vdash t[x \mapsto u] : B$.

Démonstration.

Par induction sur la dérivation de $\Gamma, x : A, \Gamma' \vdash t : B$.

- Si la dernière règle est

$$\frac{}{\Gamma, x : A, \Gamma' \vdash y : A} \text{(VAR)}$$

avec $x \neq y$ alors on conclut comme dans le cas précédent. Si la dernière règle est

$$\frac{}{\Gamma, x : A, \Gamma' \vdash x : A} \text{(VAR)}$$

alors

Compatibilité du typage avec la substitution

Le typage est compatible avec la substitution :

Lemme

Si $\Gamma, x : A, \Gamma' \vdash t : B$ et $\Gamma, \Gamma' \vdash u : A$ alors $\Gamma, \Gamma' \vdash t[x \mapsto u] : B$.

Démonstration.

Par induction sur la dérivation de $\Gamma, x : A, \Gamma' \vdash t : B$.

- Si la dernière règle est

$$\frac{}{\Gamma, x : A, \Gamma' \vdash y : A} \text{ (VAR)}$$

avec $x \neq y$ alors on conclut comme dans le cas précédent. Si la dernière règle est

$$\frac{}{\Gamma, x : A, \Gamma' \vdash x : A} \text{ (VAR)}$$

alors on conclut avec la dérivation de $\Gamma, \Gamma' \vdash u : A$ donnée en hypothèse.

Compatibilité du typage avec la substitution

Le typage est compatible avec la substitution :

Lemme

Si $\Gamma, x : A, \Gamma' \vdash t : B$ et $\Gamma, \Gamma' \vdash u : A$ alors $\Gamma, \Gamma' \vdash t[x \mapsto u] : B$.

Démonstration.

Par induction sur la dérivation de $\Gamma, x : A, \Gamma' \vdash t : B$.

– Si la dernière règle est

$$\frac{\Gamma, x : A, \Gamma' \vdash t : A \Rightarrow B \quad \Gamma, x : A, \Gamma' \vdash t' : A}{\Gamma, x : A, \Gamma' \vdash t t' : B} \text{ (APP)}$$

on conclut par

$$\frac{\frac{\vdots}{\Gamma, \Gamma' \vdash t[x \mapsto u] : A \Rightarrow B} \quad \frac{\vdots}{\Gamma, \Gamma' \vdash t'[x \mapsto u] : A}}{\Gamma, \Gamma' \vdash (t[x \mapsto u]) (t'[x \mapsto u]) : B} \text{ (APP)}$$

Compatibilité du typage avec la substitution

Le typage est compatible avec la substitution :

Lemme

Si $\Gamma, x : A, \Gamma' \vdash t : B$ et $\Gamma, \Gamma' \vdash u : A$ alors $\Gamma, \Gamma' \vdash t[x \mapsto u] : B$.

Démonstration.

Par induction sur la dérivation de $\Gamma, x : A, \Gamma' \vdash t : B$.

- Si la dernière règle est
$$\frac{\Gamma, x : A, \Gamma', y : B \vdash t : C}{\Gamma, x : A, \Gamma' \vdash \text{fun } y \rightarrow t : B \Rightarrow C} \text{ (FUN)}$$

quitte à renommer y , on peut toujours supposer $x \neq y$. On a donc

$$(\text{fun } y \rightarrow t)[x \mapsto u] = \text{fun } y \rightarrow (t[x \mapsto u])$$

⋮

et on conclut

$$\frac{\Gamma, \Gamma', y : B \vdash t[x \mapsto u] : C}{\Gamma, \Gamma' \vdash \text{fun } y \rightarrow t[x \mapsto u] : B \Rightarrow C} \text{ (FUN)}$$

Compatibilité du typage avec la substitution

Le typage est compatible avec la substitution :

Lemme

Si $\Gamma, x : A, \Gamma' \vdash t : B$ et $\Gamma, \Gamma' \vdash u : A$ alors $\Gamma, \Gamma' \vdash t[x \mapsto u] : B$.

Démonstration.

Par induction sur la dérivation de $\Gamma, x : A, \Gamma' \vdash t : B$.

- Les autres cas sont similaires.



Compatibilité du typage avec la substitution

Lemme

Si $\Gamma, x : A, \Gamma' \vdash t : B$ et $\Gamma, \Gamma' \vdash u : A$ alors $\Gamma, \Gamma' \vdash t[x \mapsto u] : B$.

Autrement dit, la règle

$$\frac{\Gamma, x : A, \Gamma' \vdash t : B \quad \Gamma, \Gamma' \vdash u : A}{\Gamma, \Gamma' \vdash t[x \mapsto u] : B}$$

est **admissible** (mais elle n'est pas **dérivable**).

Réduction du sujet

Le typage est préservé par la réduction :

Réduction du sujet

Le typage est préservé par la réduction :

Proposition (Réduction du sujet)

Si $\Gamma \vdash t : A$ est dérivable et $t \longrightarrow t'$ alors $\Gamma \vdash t' : A$ est aussi dérivable.

Démonstration.

On raisonne par induction sur la dérivation de $t \longrightarrow t'$.

Réduction du sujet

Le typage est préservé par la réduction :

Proposition (Réduction du sujet)

Si $\Gamma \vdash t : A$ est dérivable et $t \longrightarrow t'$ alors $\Gamma \vdash t' : A$ est aussi dérivable.

Démonstration.

On raisonne par induction sur la dérivation de $t \longrightarrow t'$.

- Si la dernière règle est
$$\frac{t \longrightarrow t'}{t v \longrightarrow t' v} \text{ (APPL)}$$

alors la dérivation de typage de $t v$ est de la forme

$$\frac{\frac{\vdots}{\Gamma \vdash t : A \Rightarrow B} \quad \frac{\vdots}{\Gamma \vdash v : A}}{\Gamma \vdash t v : B} \text{ (APP)}$$

Réduction du sujet

Le typage est préservé par la réduction :

Proposition (Réduction du sujet)

Si $\Gamma \vdash t : A$ est dérivable et $t \longrightarrow t'$ alors $\Gamma \vdash t' : A$ est aussi dérivable.

Démonstration.

On raisonne par induction sur la dérivation de $t \longrightarrow t'$.

- Si la dernière règle est
$$\frac{t \longrightarrow t'}{t v \longrightarrow t' v} \text{ (APPL)}$$

alors la dérivation de typage de $t v$ est de la forme

$$\frac{\frac{\vdots}{\Gamma \vdash t : A \Rightarrow B} \quad \frac{\vdots}{\Gamma \vdash v : A}}{\Gamma \vdash t v : B} \text{ (APP)} \quad \frac{\frac{\vdots}{\Gamma \vdash t' : A \Rightarrow B} \quad \frac{\vdots}{\Gamma \vdash v : A}}{\Gamma \vdash t' v : B} \text{ (APP)}$$

Réduction du sujet

Le typage est préservé par la réduction :

Proposition (Réduction du sujet)

Si $\Gamma \vdash t : A$ est dérivable et $t \longrightarrow t'$ alors $\Gamma \vdash t' : A$ est aussi dérivable.

Démonstration.

On raisonne par induction sur la dérivation de $t \longrightarrow t'$.

- Si la dernière règle est $\frac{}{(\text{fun } x \rightarrow t) v \longrightarrow t[x \mapsto v]}$ (APP)

alors la dérivation de typage de $(\text{fun } x \rightarrow t) v$ est de la forme

$$\frac{\frac{\frac{\vdots}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \text{fun } x \rightarrow t : A \Rightarrow B} \text{ (FUN)} \quad \frac{\vdots}{\Gamma \vdash v : A}}{\Gamma \vdash (\text{fun } x \rightarrow t) v : B} \text{ (APP)}$$

Réduction du sujet

Le typage est préservé par la réduction :

Proposition (Réduction du sujet)

Si $\Gamma \vdash t : A$ est dérivable et $t \longrightarrow t'$ alors $\Gamma \vdash t' : A$ est aussi dérivable.

Démonstration.

On raisonne par induction sur la dérivation de $t \longrightarrow t'$.

- Si la dernière règle est $\frac{}{(\text{fun } x \rightarrow t) v \longrightarrow t[x \mapsto v]}$ (APP)

alors la dérivation de typage de $(\text{fun } x \rightarrow t) v$ est de la forme

$$\frac{\frac{\frac{\vdots}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \text{fun } x \rightarrow t : A \Rightarrow B} \text{ (FUN)} \quad \frac{\frac{\vdots}{\Gamma \vdash v : A}}{\Gamma \vdash t[x \mapsto v] : B} \text{ (APP)}}{\Gamma \vdash (\text{fun } x \rightarrow t) v : B} \text{ (APP)}$$

Réduction du sujet

Le typage est préservé par la réduction :

Proposition (Réduction du sujet)

Si $\Gamma \vdash t : A$ est dérivable et $t \longrightarrow t'$ alors $\Gamma \vdash t' : A$ est aussi dérivable.

Démonstration.

On raisonne par induction sur la dérivation de $t \longrightarrow t'$.

– etc.



Nous avons vu que toute valeur est normale mais pas le contraire

`true + false`

Nous avons vu que toute valeur est normale mais pas le contraire

`true + false`

Pour les programmes bien typés, la réciproque est vraie :

Nous avons vu que toute valeur est normale mais pas le contraire

`true + false`

Pour les programmes bien typés, la réciproque est vraie :

Proposition (Progrès)

Si $\vdash t : A$ et t est une forme normale alors t est une valeur.

Proposition (Progrès)

Si $\vdash t : A$ et t est une forme normale alors t est une valeur.

Démonstration.

Par récurrence sur la dérivation de $\vdash t : A$.

- La règle (VAR)

Proposition (Progrès)

Si $\vdash t : A$ et t est une forme normale alors t est une valeur.

Démonstration.

Par récurrence sur la dérivation de $\vdash t : A$.

- La règle (VAR) ne peut être appliquée car l'environnement Γ est supposé vide, et t n'est donc pas une variable.

Proposition (Progrès)

Si $\vdash t : A$ et t est une forme normale alors t est une valeur.

Démonstration.

Par récurrence sur la dérivation de $\vdash t : A$.

- Si la dernière règle est

$$\frac{x : A \vdash t : B}{\vdash \text{fun } x \rightarrow t : A \Rightarrow B} \text{ (FUN)}$$

alors

Proposition (Progrès)

Si $\vdash t : A$ et t est une forme normale alors t est une valeur.

Démonstration.

Par récurrence sur la dérivation de $\vdash t : A$.

- Si la dernière règle est

$$\frac{x : A \vdash t : B}{\vdash \text{fun } x \rightarrow t : A \Rightarrow B} \text{ (FUN)}$$

alors on a immédiatement que $\text{fun } x \rightarrow t$ est une valeur.

Proposition (Progrès)

Si $\vdash t : A$ et t est une forme normale alors t est une valeur.

Démonstration.

Par récurrence sur la dérivation de $\vdash t : A$.

- Si la dernière règle est

$$\frac{\frac{\vdots}{\vdash t : A \Rightarrow B} \quad \frac{\vdots}{\vdash u : A}}{\vdash t u : B} \text{ (APP)}$$

Les termes t et u sont normaux (sinon $t u$ ne le serait pas) et donc des valeurs par hypothèse d'induction. En regardant les cas possibles pour t et u valeurs, on en déduit que $t u$ est une valeur (par exemple, t ne peut pas être de la forme $\text{fun } x \rightarrow t'$ sinon $t u$ ne serait pas normal, mais on peut avoir $t = \text{add}$ et $u = \underline{n}$).

Proposition (Progrès)

Si $\vdash t : A$ et t est une forme normale alors t est une valeur.

Démonstration.

Par récurrence sur la dérivation de $\vdash t : A$.

- Les autres cas sont similaires.



Nous pouvons maintenant montrer le théorème de sûreté du typage :

Nous pouvons maintenant montrer le théorème de sûreté du typage :

Théorème (Sûreté)

Si t est un programme typé et $t \xrightarrow{} u$, pour u une forme normale, alors u est une valeur.*

Nous pouvons maintenant montrer le théorème de sûreté du typage :

Théorème (Sûreté)

Si t est un programme typé et $t \xrightarrow{} u$, pour u une forme normale, alors u est une valeur.*

Démonstration.

Supposons $\vdash t : A$.

Nous pouvons maintenant montrer le théorème de sûreté du typage :

Théorème (Sûreté)

Si t est un programme typé et $t \xrightarrow{} u$, pour u une forme normale, alors u est une valeur.*

Démonstration.

Supposons $\vdash t : A$.

Par réduction du sujet on a $\vdash u : A$

Nous pouvons maintenant montrer le théorème de sûreté du typage :

Théorème (Sûreté)

Si t est un programme typé et $t \xrightarrow{} u$, pour u une forme normale, alors u est une valeur.*

Démonstration.

Supposons $\vdash t : A$.

Par réduction du sujet on a $\vdash u : A$

et par progrès on en déduit que u est une valeur. □

Troisième partie III

Types principaux

Non-unicité du typage

Nous avons vu que dans la version de mini-ML *sans indication de types*, certains termes n'ont pas de type canonique :

```
fun x → x
```

a les types

```
int ⇒ int
```

```
bool ⇒ bool
```

```
etc.
```

Non-unicité du typage

Nous avons vu que dans la version de mini-ML *sans indication de types*, certains termes n'ont pas de type canonique :

```
fun x → x
```

a les types

```
int ⇒ int
```

```
bool ⇒ bool
```

```
etc.
```

Nous allons ajouter des variables aux types afin de pouvoir lui donner le type

```
X ⇒ X
```

Non-unicité du typage

Nous avons vu que dans la version de mini-ML *sans indication de types*, certains termes n'ont pas de type canonique :

```
fun x → x
```

a les types

```
int ⇒ int
```

```
bool ⇒ bool
```

```
etc.
```

Nous allons ajouter des variables aux types afin de pouvoir lui donner le type

```
X ⇒ X
```

et montrer que c'est le type le « plus général ».

On définit les **types** par la syntaxe

$$A ::= \text{int} \mid \text{bool} \mid A \Rightarrow A' \mid A \times A'$$

On définit les **types** par la syntaxe

$$A ::= \text{int} \mid \text{bool} \mid A \Rightarrow A' \mid A \times A' \mid X$$

Substitutions

Une **substitution** σ est une fonction des variables dans les types telle que

$$\{X \mid \sigma(X) \neq X\}$$

est fini.

On note parfois

$$\sigma = [X_1 \mapsto A_1, \dots, X_n \mapsto A_n]$$

Substitutions

Une **substitution** σ est une fonction des variables dans les types telle que

$$\{X \mid \sigma(X) \neq X\}$$

est fini.

On note parfois

$$\sigma = [X_1 \mapsto A_1, \dots, X_n \mapsto A_n]$$

On note

$$A[\sigma]$$

le type A où toute variable X a été remplacée par $\sigma(X)$.

On note **id** la **substitution identité** définie par

$$\text{id}(X) = X$$

On note **id** la **substitution identité** définie par

$$\text{id}(X) = X$$

On a

$$A[\text{id}] = A$$

Étant données deux substitutions σ et τ , la **substitution composée** $\tau \circ \sigma$ est définie par

$$\tau \circ \sigma(X) = \sigma(X)[\tau]$$

Étant données deux substitutions σ et τ , la **substitution composée** $\tau \circ \sigma$ est définie par

$$\tau \circ \sigma(X) = \sigma(X)[\tau]$$

On a

$$A[\sigma][\tau] = A[\tau \circ \sigma]$$

On note $A \sqsubseteq B$ lorsqu'il existe une substitution σ telle que $A[\sigma] = B$:

A est **plus général** que B .

(ou B *raffine* A)

Par exemple,

$$X \Rightarrow X \quad \sqsubseteq \quad (\text{int} \Rightarrow Y) \Rightarrow (\text{int} \Rightarrow Y)$$

Ordre de raffinement

On note $A \sqsubseteq B$ lorsqu'il existe une substitution σ telle que $A[\sigma] = B$:

A est **plus général** que B .

(ou B *raffine* A)

Par exemple,

$X \Rightarrow X \quad \sqsubseteq \quad \text{int} \Rightarrow \text{int}$

Ordre de raffinement

On note $A \sqsubseteq B$ lorsqu'il existe une substitution σ telle que $A[\sigma] = B$:

A est plus général que B .

(ou B raffine A)

Par exemple,

$$\begin{array}{ccc} X \Rightarrow X & \sqsubseteq & \text{int} \Rightarrow \text{int} \\ \{\text{id}, \dots\} & \subseteq & \{\text{id}, \text{succ}, \dots\} \end{array}$$

Un **renommage** est une substitution dont toutes les images sont des variables :

$$\sigma(X) = Y$$

mais pas $\sigma(X) = \text{int}$ ou $\sigma(X) = Y \Rightarrow Z$.

Lemme

La relation \sqsubseteq est un pré-ordre sur les types (= cette relation est réflexive et transitive).

Deux types A et B sont équivalents (c'est-à-dire que l'on a $A \sqsubseteq B$ et $B \sqsubseteq A$) si et seulement si il existe un renommage bijectif σ tel que $A[\sigma] = B$.

Démonstration.

Lemme

La relation \sqsubseteq est un pré-ordre sur les types (= cette relation est réflexive et transitive).

Deux types A et B sont équivalents (c'est-à-dire que l'on a $A \sqsubseteq B$ et $B \sqsubseteq A$) si et seulement si il existe un renommage bijectif σ tel que $A[\sigma] = B$.

Démonstration.

- réflexivité : $A \sqsubseteq A$ par

Lemme

La relation \sqsubseteq est un pré-ordre sur les types (= cette relation est réflexive et transitive).

Deux types A et B sont équivalents (c'est-à-dire que l'on a $A \sqsubseteq B$ et $B \sqsubseteq A$) si et seulement si il existe un renommage bijectif σ tel que $A[\sigma] = B$.

Démonstration.

– réflexivité : $A \sqsubseteq A$ par

$$A[\text{id}] = A$$

Lemme

La relation \sqsubseteq est un pré-ordre sur les types (= cette relation est réflexive et transitive).

Deux types A et B sont équivalents (c'est-à-dire que l'on a $A \sqsubseteq B$ et $B \sqsubseteq A$) si et seulement si il existe un renommage bijectif σ tel que $A[\sigma] = B$.

Démonstration.

– réflexivité : $A \sqsubseteq A$ par

$$A[\text{id}] = A$$

– transitivité : si $A \sqsubseteq B$ et $B \sqsubseteq C$,

Lemme

La relation \sqsubseteq est un pré-ordre sur les types (= cette relation est réflexive et transitive).

Deux types A et B sont équivalents (c'est-à-dire que l'on a $A \sqsubseteq B$ et $B \sqsubseteq A$) si et seulement si il existe un renommage bijectif σ tel que $A[\sigma] = B$.

Démonstration.

– réflexivité : $A \sqsubseteq A$ par

$$A[\text{id}] = A$$

– transitivité : si $A \sqsubseteq B$ et $B \sqsubseteq C$, alors $A[\sigma] = B$ et $B[\tau] = C$

Lemme

La relation \sqsubseteq est un pré-ordre sur les types (= cette relation est réflexive et transitive).

Deux types A et B sont équivalents (c'est-à-dire que l'on a $A \sqsubseteq B$ et $B \sqsubseteq A$) si et seulement si il existe un renommage bijectif σ tel que $A[\sigma] = B$.

Démonstration.

- réflexivité : $A \sqsubseteq A$ par

$$A[\text{id}] = A$$

- transitivité : si $A \sqsubseteq B$ et $B \sqsubseteq C$, alors $A[\sigma] = B$ et $B[\tau] = C$ et on conclut par

$$A[\tau \circ \sigma]$$

Lemme

La relation \sqsubseteq est un pré-ordre sur les types (= cette relation est réflexive et transitive).

Deux types A et B sont équivalents (c'est-à-dire que l'on a $A \sqsubseteq B$ et $B \sqsubseteq A$) si et seulement si il existe un renommage bijectif σ tel que $A[\sigma] = B$.

Démonstration.

- réflexivité : $A \sqsubseteq A$ par

$$A[\text{id}] = A$$

- transitivité : si $A \sqsubseteq B$ et $B \sqsubseteq C$, alors $A[\sigma] = B$ et $B[\tau] = C$ et on conclut par

$$A[\tau \circ \sigma]$$

- équivalence :

Lemme

La relation \sqsubseteq est un pré-ordre sur les types (= cette relation est réflexive et transitive).

Deux types A et B sont équivalents (c'est-à-dire que l'on a $A \sqsubseteq B$ et $B \sqsubseteq A$) si et seulement si il existe un renommage bijectif σ tel que $A[\sigma] = B$.

Démonstration.

- réflexivité : $A \sqsubseteq A$ par

$$A[\text{id}] = A$$

- transitivité : si $A \sqsubseteq B$ et $B \sqsubseteq C$, alors $A[\sigma] = B$ et $B[\tau] = C$ et on conclut par

$$A[\tau \circ \sigma]$$

- équivalence : le raffinement fait « grossir » les types. □

Raffinement et typage

Le typage est compatible avec la substitution :

Raffinement et typage

Le typage est compatible avec la substitution :

Proposition

Si $\Gamma \vdash t : A$ alors $\Gamma[\sigma] \vdash t : A[\sigma]$.

Raffinement et typage

Le typage est compatible avec la substitution :

Proposition

Si $\Gamma \vdash t : A$ alors $\Gamma[\sigma] \vdash t : A[\sigma]$.

Démonstration.

Par induction sur la dérivation de $\Gamma \vdash t : A$.

Le typage est compatible avec la substitution :

Proposition

Si $\Gamma \vdash t : A$ alors $\Gamma[\sigma] \vdash t : A[\sigma]$.

Démonstration.

Par induction sur la dérivation de $\Gamma \vdash t : A$.

- Si la dernière règle est

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \text{ (VAR)}$$

alors on a immédiatement

$$\frac{(x : A[\sigma]) \in \Gamma[\sigma]}{\Gamma[\sigma] \vdash x : A[\sigma]} \text{ (VAR)}$$

Raffinement et typage

Le typage est compatible avec la substitution :

Proposition

Si $\Gamma \vdash t : A$ alors $\Gamma[\sigma] \vdash t : A[\sigma]$.

Démonstration.

Par induction sur la dérivation de $\Gamma \vdash t : A$.

– Si la dernière règle est

$$\frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash t u : B} \text{ (APP)}$$

par hypothèse d'induction on a une dérivation de $\Gamma[\sigma] \vdash t : A[\sigma] \Rightarrow B[\sigma]$ et de $\Gamma[\sigma] \vdash u : A[\sigma]$ et on conclut par

$$\frac{\Gamma[\sigma] \vdash t : A[\sigma] \Rightarrow B[\sigma] \quad \Gamma[\sigma] \vdash u : A[\sigma]}{\Gamma[\sigma] \vdash t u : B[\sigma]} \text{ (APP)}$$

Raffinement et typage

Le typage est compatible avec la substitution :

Proposition

Si $\Gamma \vdash t : A$ alors $\Gamma[\sigma] \vdash t : A[\sigma]$.

Démonstration.

Par induction sur la dérivation de $\Gamma \vdash t : A$.

– etc.



Corollaire

Si $\vdash t : A$ et $A \sqsubseteq B$ alors $\vdash t : B$.

Corollaire

Si $\vdash t : A$ et $A \sqsubseteq B$ alors $\vdash t : B$.

Un **type principal** pour un terme t est un type A tel que

- $\vdash t : A$,
- pour tout type B tel que $\vdash t : B$, on a $A \sqsubseteq B$.

(= c'est un élément minimal parmi les types de t)

Corollaire

Si $\vdash t : A$ et $A \sqsubseteq B$ alors $\vdash t : B$.

Un **type principal** pour un terme t est un type A tel que

- $\vdash t : A$,
- pour tout type B tel que $\vdash t : B$, on a $B = A[\sigma]$.

Corollaire

Si $\vdash t : A$ et $A \sqsubseteq B$ alors $\vdash t : B$.

Un **type principal** pour un terme t dans Γ est une substitution σ et un type A tels que

- $\Gamma[\sigma] \vdash t : A$,
- si $\Gamma[\tau] \vdash t : B$ alors $\tau = \tau' \circ \sigma$ et $B = A[\tau']$

Corollaire

Si $\vdash t : A$ et $A \sqsubseteq B$ alors $\vdash t : B$.

Un **type principal** pour un terme t dans Γ est une substitution σ et un type A tels que

- $\Gamma[\sigma] \vdash t : A$,
- si $\Gamma[\tau] \vdash t : B$ alors $\tau = \tau' \circ \sigma$ et $B = A[\tau']$

Exemple

Dans $\Gamma = (f : X \Rightarrow (X \Rightarrow X), a : Y \times Y)$ le type principal de $f a$ est

Corollaire

Si $\vdash t : A$ et $A \sqsubseteq B$ alors $\vdash t : B$.

Un **type principal** pour un terme t dans Γ est une substitution σ et un type A tels que

- $\Gamma[\sigma] \vdash t : A$,
- si $\Gamma[\tau] \vdash t : B$ alors $\tau = \tau' \circ \sigma$ et $B = A[\tau']$

Exemple

Dans $\Gamma = (f : X \Rightarrow (X \Rightarrow X), a : Y \times Y)$ le type principal de $f a$ est

- la substitution $\sigma = [X \mapsto Y \times Y]$
- le type $A = (Y \times Y) \Rightarrow (Y \times Y)$

Nous allons décrire un algorithme en deux étapes pour générer le type principal d'un terme :

1. on va générer un système d'équations entre types qui décrit les types possibles,
2. on va trouver la solution la plus générale du système d'équations.

Un système d'équations de types est un ensemble fini

$$E = \{A_1 \stackrel{?}{=} B_1, \dots, A_n \stackrel{?}{=} B_n\}$$

de paires de types.

Un système d'équations de types est un ensemble fini

$$E = \{A_1 \stackrel{?}{=} B_1, \dots, A_n \stackrel{?}{=} B_n\}$$

de paires de types.

Une **solution** ou un **unificateur** de E est une substitution σ telle que $A_i[\sigma] = B_i[\sigma]$ pour tout i .

Système d'équations de type

Notons que si σ est une solution de E alors $\tau \circ \sigma$ l'est aussi :

$$A_i[\sigma] = B_i[\sigma]$$

implique

$$A_i[\tau \circ \sigma] = A_i[\sigma][\tau] = B_i[\sigma][\tau] = A_i[\tau \circ \sigma]$$

Système d'équations de type

Notons que si σ est une solution de E alors $\tau \circ \sigma$ l'est aussi :

$$A_i[\sigma] = B_i[\sigma]$$

implique

$$A_i[\tau \circ \sigma] = A_i[\sigma][\tau] = B_i[\sigma][\tau] = A_i[\tau \circ \sigma]$$

Une solution σ est **la plus générale** lorsque pour tout autre solution τ on a

$$\tau = \tau' \circ \sigma$$

pour un certain τ' .

Systèmes d'équations de type

Par exemple,

$$E = \{(X \Rightarrow X) \stackrel{?}{=} (X \Rightarrow Y), Y \stackrel{?}{=} (Z \Rightarrow Z)\}$$

a pour solution la plus générale

Systèmes d'équations de type

Par exemple,

$$E = \{(X \Rightarrow X) \stackrel{?}{=} (X \Rightarrow Y), Y \stackrel{?}{=} (Z \Rightarrow Z)\}$$

a pour solution la plus générale

$$\sigma = [X \mapsto Z \Rightarrow Z, Y \mapsto Z \Rightarrow Z]$$

Systèmes d'équations de type

Par exemple,

$$E = \{(X \Rightarrow X) \stackrel{?}{=} (X \Rightarrow Y), Y \stackrel{?}{=} (Z \Rightarrow Z)\}$$

a pour solution la plus générale

$$\sigma = [X \mapsto Z \Rightarrow Z, Y \mapsto Z \Rightarrow Z]$$

une autre solution étant

Systèmes d'équations de type

Par exemple,

$$E = \{(X \Rightarrow X) \stackrel{?}{=} (X \Rightarrow Y), Y \stackrel{?}{=} (Z \Rightarrow Z)\}$$

a pour solution la plus générale

$$\sigma = [X \mapsto Z \Rightarrow Z, Y \mapsto Z \Rightarrow Z]$$

une autre solution étant

$$\sigma = [X \mapsto \text{int} \Rightarrow \text{int}, Y \mapsto \text{int} \Rightarrow \text{int}, Z \mapsto \text{int}]$$

Par exemple,

$$E = \{(X \times Y) \stackrel{?}{=} (X \Rightarrow Y)\}$$

Par exemple,

$$E = \{(X \times Y) \stackrel{?}{=} (X \Rightarrow Y)\}$$

n'a pas de solution.

Par exemple,

$$E = \{X \stackrel{?}{=} (X \Rightarrow Y)\}$$

Par exemple,

$$E = \{X \stackrel{?}{=} (X \Rightarrow Y)\}$$

n'a pas de solution.

À un terme t dans un contexte Γ , nous allons associer un type A_t et un système E_t .

L'idée est que A_t va être un type pour t dans Γ sous l'hypothèse que E_t est vérifié.

À un terme t dans un contexte Γ , nous allons associer un type A_t et un système E_t .

L'idée est que A_t va être un type pour t dans Γ sous l'hypothèse que E_t est vérifié.

On procède par induction sur t .

Génération d'équations

À un terme t dans un contexte Γ , nous allons associer un type A_t et un système E_t .

L'idée est que A_t va être un type pour t dans Γ sous l'hypothèse que E_t est vérifié.

On procède par induction sur t .

On note \perp un système d'équations sans solution.

Génération d'équations

- pour un entier n ,

Génération d'équations

– pour un entier n ,

$$A_n = \text{int}$$

$$E_n = \emptyset$$

Génération d'équations

– pour un entier n ,

$$A_n = \text{int}$$

$$E_n = \emptyset$$

Génération d'équations

- pour un entier \underline{n} ,

$$A_{\underline{n}} = \text{int}$$

$$E_{\underline{n}} = \emptyset$$

- pour un booléen \underline{b} ,

Génération d'équations

- pour un entier \underline{n} ,

$$A_{\underline{n}} = \text{int}$$

$$E_{\underline{n}} = \emptyset$$

- pour un booléen \underline{b} ,

$$A_{\underline{b}} = \text{bool}$$

$$E_{\underline{b}} = \emptyset$$

Génération d'équations

- pour un entier \underline{n} ,

$$A_{\underline{n}} = \text{int}$$

$$E_{\underline{n}} = \emptyset$$

- pour un booléen \underline{b} ,

$$A_{\underline{b}} = \text{bool}$$

$$E_{\underline{b}} = \emptyset$$

- pour x avec $(x : A) \in \Gamma$,

Génération d'équations

- pour un entier \underline{n} ,

$$A_{\underline{n}} = \text{int}$$

$$E_{\underline{n}} = \emptyset$$

- pour un booléen \underline{b} ,

$$A_{\underline{b}} = \text{bool}$$

$$E_{\underline{b}} = \emptyset$$

- pour x avec $(x : A) \in \Gamma$,

$$A_x = A$$

$$E_x = \emptyset$$

Génération d'équations

- pour un entier \underline{n} ,

$$A_{\underline{n}} = \text{int}$$

$$E_{\underline{n}} = \emptyset$$

- pour un booléen \underline{b} ,

$$A_{\underline{b}} = \text{bool}$$

$$E_{\underline{b}} = \emptyset$$

- pour x avec $(x : A) \in \Gamma$,

$$A_x = A$$

$$E_x = \emptyset$$

- pour x n'apparaissant pas dans Γ ,

Génération d'équations

- pour un entier \underline{n} ,

$$A_{\underline{n}} = \text{int}$$

$$E_{\underline{n}} = \emptyset$$

- pour un booléen \underline{b} ,

$$A_{\underline{b}} = \text{bool}$$

$$E_{\underline{b}} = \emptyset$$

- pour x avec $(x : A) \in \Gamma$,

$$A_x = A$$

$$E_x = \emptyset$$

- pour x n'apparaissant pas dans Γ ,

$$A_x = ?$$

$$E_x = \perp$$

Génération d'équations

- pour une abstraction `fun x → t`,

Génération d'équations

- pour une abstraction $\text{fun } x \rightarrow t$,

$$A_{\text{fun } x \rightarrow t} = X \Rightarrow A_t$$

$$E_{\text{fun } x \rightarrow t} = E_t$$

avec X une variable fraîche, et A_t et E_t calculés dans le contexte $\Gamma, x : X$,

Génération d'équations

- pour une abstraction `fun x → t`,

$$A_{\text{fun } x \rightarrow t} = X \Rightarrow A_t$$

$$E_{\text{fun } x \rightarrow t} = E_t$$

avec X une variable fraîche, et A_t et E_t calculés dans le contexte $\Gamma, x : X$,

- pour une application `t u`,

Génération d'équations

- pour une abstraction $\text{fun } x \rightarrow t$,

$$A_{\text{fun } x \rightarrow t} = X \Rightarrow A_t$$

$$E_{\text{fun } x \rightarrow t} = E_t$$

avec X une variable fraîche, et A_t et E_t calculés dans le contexte $\Gamma, x : X$,

- pour une application $t u$,

$$A_{t u} = X$$

$$E_{t u} = E_t \cup E_u \cup \{A_t \stackrel{?}{=} (A_u \Rightarrow X)\}$$

où X est une variable fraîche,

Génération d'équations

- pour une abstraction $\text{fun } x \rightarrow t$,

$$A_{\text{fun } x \rightarrow t} = X \Rightarrow A_t$$

$$E_{\text{fun } x \rightarrow t} = E_t$$

avec X une variable fraîche, et A_t et E_t calculés dans le contexte $\Gamma, x : X$,

- pour une application $t u$,

$$A_{t u} = X$$

$$E_{t u} = E_t \cup E_u \cup \{A_t \stackrel{?}{=} (A_u \Rightarrow X)\}$$

où X est une variable fraîche,

- pour une paire (t, u) ,

Génération d'équations

- pour une abstraction $\text{fun } x \rightarrow t$,

$$A_{\text{fun } x \rightarrow t} = X \Rightarrow A_t$$

$$E_{\text{fun } x \rightarrow t} = E_t$$

avec X une variable fraîche, et A_t et E_t calculés dans le contexte $\Gamma, x : X$,

- pour une application $t u$,

$$A_{t u} = X$$

$$E_{t u} = E_t \cup E_u \cup \{A_t \stackrel{?}{=} (A_u \Rightarrow X)\}$$

où X est une variable fraîche,

- pour une paire (t, u) ,

$$A_{(t,u)} = A_t \times A_u$$

$$E_{(t,u)} = E_t \cup E_u$$

Génération d'équations

- pour une abstraction $\text{fun } x \rightarrow t$,

$$A_{\text{fun } x \rightarrow t} = X \Rightarrow A_t$$

$$E_{\text{fun } x \rightarrow t} = E_t$$

avec X une variable fraîche, et A_t et E_t calculés dans le contexte $\Gamma, x : X$,

- pour une application $t u$,

$$A_{t u} = X$$

$$E_{t u} = E_t \cup E_u \cup \{A_t \stackrel{?}{=} (A_u \Rightarrow X)\}$$

où X est une variable fraîche,

- pour une paire (t, u) ,

$$A_{(t,u)} = A_t \times A_u$$

$$E_{(t,u)} = E_t \cup E_u$$

- pour fst ,

Génération d'équations

- pour une abstraction $\text{fun } x \rightarrow t$,

$$A_{\text{fun } x \rightarrow t} = X \Rightarrow A_t$$

$$E_{\text{fun } x \rightarrow t} = E_t$$

avec X une variable fraîche, et A_t et E_t calculés dans le contexte $\Gamma, x : X$,

- pour une application $t u$,

$$A_{t u} = X$$

$$E_{t u} = E_t \cup E_u \cup \{A_t \stackrel{?}{=} (A_u \Rightarrow X)\}$$

où X est une variable fraîche,

- pour une paire (t, u) ,

$$A_{(t,u)} = A_t \times A_u$$

$$E_{(t,u)} = E_t \cup E_u$$

- pour fst ,

$$A_{\text{fst}} = (X \times Y) \Rightarrow X$$

$$E_{\text{fst}} = \emptyset$$

avec X et Y fraîches

Génération d'équations

- pour une abstraction `fun x → t`,

$$A_{\text{fun } x \rightarrow t} = X \Rightarrow A_t$$

$$E_{\text{fun } x \rightarrow t} = E_t$$

avec X une variable fraîche, et A_t et E_t calculés dans le contexte $\Gamma, x : X$,

- pour une application `t u`,

$$A_{t\ u} = X$$

$$E_{t\ u} = E_t \cup E_u \cup \{A_t \stackrel{?}{=} (A_u \Rightarrow X)\}$$

où X est une variable fraîche,

- pour une paire `(t, u)`,

$$A_{(t,u)} = A_t \times A_u$$

$$E_{(t,u)} = E_t \cup E_u$$

- pour `fst`,

$$A_{\text{fst}} = (X \times Y) \Rightarrow X$$

$$E_{\text{fst}} = \emptyset$$

avec X et Y fraîches (`snd` et `fix` similaires).

Étant donné t dans Γ ,

Étant donné t dans Γ ,

Proposition (Correction)

Si σ est une solution de E_t alors $\Gamma[\sigma] \vdash t : A_t[\sigma]$.

Étant donné t dans Γ ,

Proposition (Correction)

Si σ est une solution de E_t alors $\Gamma[\sigma] \vdash t : A_t[\sigma]$.

Démonstration.

Par induction sur t .



Étant donné t dans Γ ,

Proposition (Correction)

Si σ est une solution de E_t alors $\Gamma[\sigma] \vdash t : A_t[\sigma]$.

Démonstration.

Par induction sur t .



Proposition (Complétude)

Étant donnés σ et A tels que $\Gamma[\sigma] \vdash t : A$, on a σ solution de E_t et $A = A_t[\sigma]$.

Étant donné t dans Γ ,

Proposition (Correction)

Si σ est une solution de E_t alors $\Gamma[\sigma] \vdash t : A_t[\sigma]$.

Démonstration.

Par induction sur t . □

Proposition (Complétude)

Étant donné σ et A tels que $\Gamma[\sigma] \vdash t : A$, on a σ solution de E_t et $A = A_t[\sigma]$.

Démonstration.

Par induction sur la dérivation de $\Gamma[\sigma] \vdash t : A$. □

Corollaire

Les solutions principales de E_t correspondent aux types principaux de t .

Corollaire

Les solutions principales de E_t correspondent aux types principaux de t .

Comment calculer une solution principale de E_t ?

Nous allons maintenant décrire un algorithme d'**unification** qui calcule la solution la plus générale d'un système d'équations E .

On note le résultat $\text{unify}(E)$.

Nous allons maintenant décrire un algorithme d'**unification** qui calcule la solution la plus générale d'un système d'équations E .

On note le résultat $\text{unify}(E)$.

On le définit *récurivement* de la façon suivante...

`unify(int ?int, E) =`

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$
 $\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) =$

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$
$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\begin{aligned} \text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) &= \text{unify}(E) \\ \text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) &= \text{unify}(E) \\ \text{unify}(X \stackrel{?}{=} X, E) &= \end{aligned}$$

$$\begin{aligned}\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) &= \text{unify}(E) \\ \text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) &= \text{unify}(E) \\ \text{unify}(X \stackrel{?}{=} X, E) &= \text{unify}(E)\end{aligned}$$

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$$

$$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) =$$

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$$

$$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$$

$$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) =$$

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$$

$$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$$

$$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}(X \stackrel{?}{=} A, E) =$$

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$$

$$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}(X \stackrel{?}{=} A, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A]$$

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$$

$$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}(X \stackrel{?}{=} A, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A] \quad \text{si } X \notin \text{VL}(A)$$

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$$

$$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}(X \stackrel{?}{=} A, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A] \quad \text{si } X \notin \text{VL}(A)$$

$$\text{unify}(A \stackrel{?}{=} X, E) =$$

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$$

$$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}(X \stackrel{?}{=} A, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A] \quad \text{si } X \notin \text{VL}(A)$$

$$\text{unify}(A \stackrel{?}{=} X, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A] \quad \text{si } X \notin \text{VL}(A)$$

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$$

$$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}(X \stackrel{?}{=} A, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A] \quad \text{si } X \notin \text{VL}(A)$$

$$\text{unify}(A \stackrel{?}{=} X, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A] \quad \text{si } X \notin \text{VL}(A)$$

$$\text{unify}(\emptyset) =$$

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$$

$$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}(X \stackrel{?}{=} A, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A] \quad \text{si } X \notin \text{VL}(A)$$

$$\text{unify}(A \stackrel{?}{=} X, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A] \quad \text{si } X \notin \text{VL}(A)$$

$$\text{unify}(\emptyset) = \text{id}$$

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$$

$$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}(X \stackrel{?}{=} A, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A] \quad \text{si } X \notin \text{VL}(A)$$

$$\text{unify}(A \stackrel{?}{=} X, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A] \quad \text{si } X \notin \text{VL}(A)$$

$$\text{unify}(\emptyset) = \text{id}$$

Dans tous les autres cas

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$$

$$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}(X \stackrel{?}{=} A, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A] \quad \text{si } X \notin \text{VL}(A)$$

$$\text{unify}(A \stackrel{?}{=} X, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A] \quad \text{si } X \notin \text{VL}(A)$$

$$\text{unify}(\emptyset) = \text{id}$$

Dans tous les autres cas `unify` échoue.

Il n'est pas clair que cette procédure termine sur toute entrée.

Par exemple, la taille de la liste ne décroît pas :

$$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

Nous allons avoir besoin de deux mesures :

- $|E|_{\text{var}}$: le nombre de variable apparaissant dans E ,

Nous allons avoir besoin de deux mesures :

- $|E|_{\text{var}}$: le nombre de variable apparaissant dans E ,
- $|E|_{\text{op}}$: la taille des types dans E

$$|E|_{\text{op}} = \sum_{(A_i \stackrel{?}{=} B_i) \in E} |A_i|_{\text{op}} + |B_i|_{\text{op}}$$

avec

$$|\text{int}|_{\text{op}} = |\text{bool}|_{\text{op}} = |X|_{\text{op}} = 1$$

$$|A \Rightarrow B|_{\text{op}} = |A \times B|_{\text{op}} = 1 + |A|_{\text{op}} + |B|_{\text{op}}$$

Nous allons avoir besoin de deux mesures :

- $|E|_{\text{var}}$: le nombre de variable apparaissant dans E ,
- $|E|_{\text{op}}$: la taille des types dans E

$$|E|_{\text{op}} = \sum_{(A_i \stackrel{?}{=} B_i) \in E} |A_i|_{\text{op}} + |B_i|_{\text{op}}$$

avec

$$|\text{int}|_{\text{op}} = |\text{bool}|_{\text{op}} = |X|_{\text{op}} = 1$$

$$|A \Rightarrow B|_{\text{op}} = |A \times B|_{\text{op}} = 1 + |A|_{\text{op}} + |B|_{\text{op}}$$

La taille est $|E| = (|E|_{\text{var}}, |E|_{\text{op}})$ ordonnée lexicographiquement.

Les appels récursifs décroissent :

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

| $|E|_{\text{var}}$ | $|E|_{\text{op}}$

Les appels récursifs décroissent :

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$ E _{\text{var}}$	$ E _{\text{op}}$
=	<

Les appels récursifs décroissent :

	$ E _{\text{var}}$	$ E _{\text{op}}$
$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$		

Les appels récursifs décroissent :

	$ E _{\text{var}}$	$ E _{\text{op}}$
$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$	=	<
$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$	=	<

Les appels récursifs décroissent :

	$ E _{\text{var}}$	$ E _{\text{op}}$
$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$	=	<
$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$	=	<
$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$		

Les appels récursifs décroissent :

	$ E _{\text{var}}$	$ E _{\text{op}}$
$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$	\leq	$<$

Les appels récursifs décroissent :

	$ E _{\text{var}}$	$ E _{\text{op}}$
$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$	\leq	$<$
$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$		

Les appels récursifs décroissent :

	$ E _{\text{var}}$	$ E _{\text{op}}$
$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$	\leq	$<$
$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$	$=$	$<$

Les appels récursifs décroissent :

	$ E _{\text{var}}$	$ E _{\text{op}}$
$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$	=	<
$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$	=	<
$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$	\leq	<
$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$	=	<
$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$		

Les appels récursifs décroissent :

	$ E _{\text{var}}$	$ E _{\text{op}}$
$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$	=	<
$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$	=	<
$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$	\leq	<
$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$	=	<
$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$	=	<

Les appels récursifs décroissent :

	$ E _{\text{var}}$	$ E _{\text{op}}$
$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$	=	<
$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$	=	<
$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$	\leq	<
$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$	=	<
$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$	=	<
$\text{unify}(X \stackrel{?}{=} A, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A]$		

Les appels récursifs décroissent :

	$ E _{\text{var}}$	$ E _{\text{op}}$
$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$	\leq	$<$
$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$	$=$	$<$
$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$	$=$	$<$
$\text{unify}(X \stackrel{?}{=} A, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A]$	$<$	

Les appels récursifs décroissent :

	$ E _{\text{var}}$	$ E _{\text{op}}$
$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$	=	<
$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$	=	<
$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$	\leq	<
$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$	=	<
$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$	=	<
$\text{unify}(X \stackrel{?}{=} A, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A]$	<	
$\text{unify}(A \stackrel{?}{=} X, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A]$	<	

Les appels récursifs décroissent :

	$ E _{\text{var}}$	$ E _{\text{op}}$
$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$	=	<
$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$	=	<
$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$	\leq	<
$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$	=	<
$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$	=	<
$\text{unify}(X \stackrel{?}{=} A, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A]$	<	
$\text{unify}(A \stackrel{?}{=} X, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A]$	<	

Correction

Montrons que notre algorithme est correct :

Lemme

Pour les équations de la forme

$$\text{unify}(E) = \text{unify}(E')$$

les solutions de E sont les mêmes que celles de E' .

Démonstration.

Par inspection des cas :

$$\text{unify}(\text{int} \stackrel{?}{=} \text{int}, E) = \text{unify}(E)$$

$$\text{unify}(\text{bool} \stackrel{?}{=} \text{bool}, E) = \text{unify}(E)$$

$$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$$

$$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

$$\text{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$$

Lemme

Pour $X \notin \text{VL}(A)$, une substitution σ est une solution de $(X \stackrel{?}{=} A, E)$ ssi on a $\sigma = \tau \circ [X \mapsto A]$, pour τ une solution de $E[X \mapsto A]$ telle que $\tau(X) = X$.

Lemme

Pour $X \notin \text{VL}(A)$, une substitution σ est une solution de $(X \stackrel{?}{=} A, E)$ ssi on a $\sigma = \tau \circ [X \mapsto A]$, pour τ une solution de $E[X \mapsto A]$ telle que $\tau(X) = X$.

Démonstration.

On a la suite d'équivalences :

(i) σ est une solution $(X \stackrel{?}{=} A, E)$,

Lemme

Pour $X \notin \text{VL}(A)$, une substitution σ est une solution de $(X \stackrel{?}{=} A, E)$ ssi on a $\sigma = \tau \circ [X \mapsto A]$, pour τ une solution de $E[X \mapsto A]$ telle que $\tau(X) = X$.

Démonstration.

On a la suite d'équivalences :

- (i) σ est une solution $(X \stackrel{?}{=} A, E)$,
- (ii) $\sigma(X) = A[\sigma]$ et σ est une solution de E ,

Lemme

Pour $X \notin \text{VL}(A)$, une substitution σ est une solution de $(X \stackrel{?}{=} A, E)$ ssi on a $\sigma = \tau \circ [X \mapsto A]$, pour τ une solution de $E[X \mapsto A]$ telle que $\tau(X) = X$.

Démonstration.

On a la suite d'équivalences :

- (i) σ est une solution $(X \stackrel{?}{=} A, E)$,
- (ii) $\sigma(X) = A[\sigma]$ et σ est une solution de E ,
- (iii) $\sigma(X) = A[\sigma]$ et σ est une solution de $E[X \mapsto A]$,

Lemme

Pour $X \notin \text{VL}(A)$, une substitution σ est une solution de $(X \stackrel{?}{=} A, E)$ ssi on a $\sigma = \tau \circ [X \mapsto A]$, pour τ une solution de $E[X \mapsto A]$ telle que $\tau(X) = X$.

Démonstration.

On a la suite d'équivalences :

- (i) σ est une solution $(X \stackrel{?}{=} A, E)$,
- (ii) $\sigma(X) = A[\sigma]$ et σ est une solution de E ,
- (iii) $\sigma(X) = A[\sigma]$ et σ est une solution de $E[X \mapsto A]$,
- (iv) σ est de la forme $\tau \circ [X \mapsto A]$ pour τ une solution de $E[X \mapsto A]$. □

Lemme

*La solution la plus générale du système \emptyset est **id**.*

Lemme

*La solution la plus générale du système \emptyset est **id**.*

En combinant les lemmes précédents :

Théorème

*Étant donné un système E , si **unify**(E) est défini alors c'est une solution la plus générale de E .*

Lemme

*La solution la plus générale du système \emptyset est **id**.*

En combinant les lemmes précédents :

Théorème

*Étant donné un système E , si **unify**(E) est défini alors c'est une solution la plus générale de E .*

Démonstration.

Essentiellement : le cas de base produit la solution la plus générale et les autres préservent les solutions. □

Proposition (Complétude)

Si l'algorithme `unify(E)` échoue alors E n'admet pas de solution.

Démonstration.

Les cas d'échecs sont de deux natures.

- Il y a une équation $A \stackrel{?}{=} B$ avec A et B sont de forme différente (par exemple $A = \text{int}$ et $B = \text{bool}$, ou $A = \text{int}$ et $B = B_1 \Rightarrow B_2$, ou $A = A_1 \Rightarrow A_2$ et $B = B_1 \times B_2$) : le système n'admet pas de solution, puisqu'une solution σ devrait vérifier $A[\sigma] = B[\sigma]$, ce qui est impossible.

Proposition (Complétude)

Si l'algorithme `unify(E)` échoue alors E n'admet pas de solution.

Démonstration.

Les cas d'échecs sont de deux natures.

- Il a une équation $X \stackrel{?}{=} A$ avec $X \in \text{VL}(A)$ et $A \neq X$. Une solution σ vérifie $\sigma(X) = A[\sigma]$, et donc

$$|\sigma(X)| = |A[\sigma]| > |\sigma(X)|$$

Impossible. □

En pratique, la règle

$$\text{unify}(X \stackrel{?}{=} A, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A]$$

est coûteuse : il faut parcourir toutes les équations de E pour remplacer X par A ...

En pratique, la règle

$$\text{unify}(X \stackrel{?}{=} A, E) = \text{unify}(E[X \mapsto A]) \circ [X \mapsto A]$$

est coûteuse : il faut parcourir toutes les équations de E pour remplacer X par A ...

Plutôt que de remplacer X , on peut la faire pointer vers une cellule mémoire que l'on va modifier.

Implémentation

Ceci suggère d'implémenter les références par

```
type t =  
  | TInt  
  | TArr of t * t  
  | TVar of t option ref  
  ...
```

Implémentation

Ceci suggère d'implémenter les références par

```
type t =  
  | TInt  
  | TArr of t * t  
  | TVar of t option ref  
  ...
```

L'idée étant que

```
let r = ref None  
let x = TVar r
```

désigne une variable libre, où on suppose que *toutes* les occurrences de **x** vont pointer vers la même cellule mémoire **r**.

Implémentation

L'idée étant que

```
let r = ref None
```

```
let x = TVar r
```

désigne une variable libre, où on suppose que *toutes* les occurrences de **x** vont pointer vers la même cellule mémoire **r**.

On peut alors substituer la variable par **a** :

```
r := Some a
```

Implémentation

L'idée étant que

```
let r = ref None
```

```
let x = TVar r
```

désigne une variable libre, où on suppose que *toutes* les occurrences de **x** vont pointer vers la même cellule mémoire **r**.

On peut alors substituer la variable par **a** :

```
r := Some a
```

Il faudra donc considérer

```
TVar { contents = Some a }
```

comme équivalent à **a** (avec une indirection).

Implémentation

L'idée étant que

```
let r = ref None
```

```
let x = TVar r
```

désigne une variable libre, où on suppose que *toutes* les occurrences de **x** vont pointer vers la même cellule mémoire **r**.

On peut alors substituer la variable par **a** :

```
r := Some a
```

Il faudra donc considérer

```
TVar { contents = Some a }
```

comme équivalent à **a** (avec une indirection).

Notons qu'on peut utiliser l'égalité physique `==` pour comparer les variables de type !

Complexité de l'unification

L'unificateur d'un problème de taille n peut être de taille 2^n .

Par exemple,

$$\{((((A \times X_1) \times X_2) \times X_3) \times X_4 \stackrel{?}{=} X_4 \times (X_3 \times (X_2 \times (X_1 \times A))))\}$$

a pour solution

Complexité de l'unification

L'unificateur d'un problème de taille n peut être de taille 2^n .

Par exemple,

$$\{((((A \times X_1) \times X_2) \times X_3) \times X_4 \stackrel{?}{=} X_4 \times (X_3 \times (X_2 \times (X_1 \times A))))\}$$

a pour solution

- $X_1 \mapsto A$
- $X_2 \mapsto A \times A$
- $X_3 \mapsto (A \times A) \times (A \times A)$
- $X_4 \mapsto ((A \times A) \times (A \times A)) \times ((A \times A) \times (A \times A))$

Complexité de l'unification

L'unificateur d'un problème de taille n peut être de taille 2^n .

Par exemple,

$$\{((((A \times X_1) \times X_2) \times X_3) \times X_4 \stackrel{?}{=} X_4 \times (X_3 \times (X_2 \times (X_1 \times A))))\}$$

a pour solution

- $X_1 \mapsto A$
- $X_2 \mapsto A \times A$
- $X_3 \mapsto (A \times A) \times (A \times A)$
- $X_4 \mapsto ((A \times A) \times (A \times A)) \times ((A \times A) \times (A \times A))$

Pour éviter cela, il existe des variantes sur des termes avec partage d'expressions...

Quatrième partie IV

Polymorphisme

Les déclarations `let` peuvent être introduites comme du sucre syntaxique :

$$\text{let } x = t \text{ in } u \quad =$$

Les déclarations `let` peuvent être introduites comme du sucre syntaxique :

$$\text{let } x = t \text{ in } u \quad = \quad (\text{fun } x \rightarrow u) t$$

(mais on peut les introduire explicitement)

Monomorphisme

L'une des limitations du système actuel est que les types sont **monomorphes** : on ne peut utiliser un terme qu'avec un seul type.

Le programme suivant est typable :

```
((fun x → x) 5, (fun x → x) true)
```

Monomorphisme

L'une des limitations du système actuel est que les types sont **monomorphes** : on ne peut utiliser un terme qu'avec un seul type.

Le programme suivant est typable :

```
((fun x → x) 5, (fun x → x) true)
```

mais pas le programme

```
let f = (fun x → x) in (f 5, f true)
```

Monomorphisme

L'une des limitations du système actuel est que les types sont **monomorphes** : on ne peut utiliser un terme qu'avec un seul type.

Le programme suivant est typable :

```
((fun x → x) 5, (fun x → x) true)
```

mais pas le programme

```
let f = (fun x → x) in (f 5, f true)
```

qui équivaut à

```
(fun f → (f 5, f true)) (fun x → x)
```

Les langages ML réalistes sont **polymorphes** : on peut donner des schémas de type

$$\forall X_1 \dots \forall X_n. A$$

Les langages ML réalistes sont **polymorphes** : on peut donner des schémas de type

$$\forall X_1 \dots \forall X_n. A$$

Par exemple, l'identité aura le type

$$\forall X. (X \Rightarrow X)$$

ce qui est généralement noté

$$'a \rightarrow 'a$$

La discipline de typage est la suivante.

Par exemple :

```
let f = fun x -> x in (f true , f 1)
```

Généralisation

La discipline de typage est la suivante.

- Lorsqu'on fait une déclaration `let`, le type inféré est **généralisé** en quantifiant implicitement sur les variables de types libres.

Par exemple :

```
let f = fun x -> x in (f true , f 1)
```

La discipline de typage est la suivante.

- Lorsqu'on fait une déclaration `let`, le type inféré est **généralisé** en quantifiant implicitement sur les variables de types libres.
- Lorsqu'on utilise une variable, on **instancie** les variables quantifiées en les remplaçant par des variables fraîches.

Par exemple :

```
let f = fun x -> x in (f true , f 1)
```

Généralisation

La discipline de typage est la suivante.

- Lorsqu'on fait une déclaration `let`, le type inféré est **généralisé** en quantifiant implicitement sur les variables de types libres.
- Lorsqu'on utilise une variable, on **instancie** les variables quantifiées en les remplaçant par des variables fraîches.

Par exemple :

```
let f = fun x -> x in (f true , f 1)
```

↑ $X \Rightarrow X$

Généralisation

La discipline de typage est la suivante.

- Lorsqu'on fait une déclaration `let`, le type inféré est **généralisé** en quantifiant implicitement sur les variables de types libres.
- Lorsqu'on utilise une variable, on **instancie** les variables quantifiées en les remplaçant par des variables fraîches.

Par exemple :

```
let f = fun x -> x in (f true , f 1)
```

$\uparrow \forall X.X \Rightarrow X \uparrow X \Rightarrow X$

Généralisation

La discipline de typage est la suivante.

- Lorsqu'on fait une déclaration **let**, le type inféré est **généralisé** en quantifiant implicitement sur les variables de types libres.
- Lorsqu'on utilise une variable, on **instancie** les variables quantifiées en les remplaçant par des variables fraîches.

Par exemple :

```
let f = fun x -> x in (f true , f 1)
```

$\uparrow \forall X.X \Rightarrow X \uparrow X \Rightarrow X \uparrow Y \Rightarrow Y \uparrow Z \Rightarrow Z$

Généralisation

La discipline de typage est la suivante.

- Lorsqu'on fait une déclaration `let`, le type inféré est **généralisé** en quantifiant implicitement sur les variables de types libres.
- Lorsqu'on utilise une variable, on **instancie** les variables quantifiées en les remplaçant par des variables fraîches.

Par exemple :

```
let f = fun x -> x in (f true , f 1)
```

$\uparrow \forall X.X \Rightarrow X \uparrow X \Rightarrow X \uparrow Y \Rightarrow Y \uparrow Z \Rightarrow Z$

Notons qu'on ne généralise pas les abstractions : on ne peut pas typer

```
(fun f -> (f true , f 1)) (fun x -> x)
```

Schémas de types

Un schéma de type est une expression de la forme

$$\forall X_1 \dots \forall X_n. A$$

où A est un type (sans quantification).

Schémas de types

Un schéma de type est une expression de la forme

$$\forall X_1 \dots \forall X_n. A$$

où A est un type (sans quantification).

Les variables X_i sont alors liées dans A (on peut les renommer).

Un schéma de type est une expression de la forme

$$\forall X_1. \dots \forall X_n. A$$

où A est un type (sans quantification).

Les variables X_i sont alors liées dans A (on peut les renommer).

B est une instance de $\forall X_1. \dots \forall X_n. A$, ce que l'on note

$$\forall X_1. \dots \forall X_n. A \preceq B$$

lorsqu'il existe des types A_1, \dots, A_n tels que

$$B = A[X_1 \mapsto A_1, \dots, X_n \mapsto A_n]$$

Schémas de types

Un schéma de type est une expression de la forme

$$\forall X_1 \dots \forall X_n. A$$

où A est un type (sans quantification).

Les variables X_i sont alors liées dans A (on peut les renommer).

La **généralisation** d'un type A vis-à-vis de Γ est le schéma

$$\forall_{\Gamma} A = \forall X_1 \dots \forall X_n. A$$

où $VL(A) \setminus VL(\Gamma) = \{X_1, \dots, X_n\}$.

Règles de typage polymorphes

On considère des contextes

$$\Gamma = x_1 : A_1, \dots, x_n : A_n$$

où les A_i sont des schémas de types.

Les règles de typage sont les suivantes.

Règles de typage polymorphes

- variables :

Règles de typage polymorphes

– variables :

$$\frac{(x : A) \in \Gamma \quad A \preceq B}{\Gamma \vdash x : B} \text{ (VAR)}$$

Règles de typage polymorphes

– variables :

$$\frac{(x : A) \in \Gamma \quad A \preceq B}{\Gamma \vdash x : B} \text{ (VAR)}$$

– fonctions :

Règles de typage polymorphes

– variables :

$$\frac{(x : A) \in \Gamma \quad A \preceq B}{\Gamma \vdash x : B} \text{ (VAR)}$$

– fonctions :

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \text{fun } x \rightarrow t : A \Rightarrow B} \text{ (FUN)}$$

Règles de typage polymorphes

– variables :

$$\frac{(x : A) \in \Gamma \quad A \preceq B}{\Gamma \vdash x : B} \text{ (VAR)}$$

– fonctions :

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \text{fun } x \rightarrow t : A \Rightarrow B} \text{ (FUN)}$$

– applications :

Règles de typage polymorphes

– variables :

$$\frac{(x : A) \in \Gamma \quad A \preceq B}{\Gamma \vdash x : B} \text{ (VAR)}$$

– fonctions :

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \text{fun } x \rightarrow t : A \Rightarrow B} \text{ (FUN)}$$

– applications :

$$\frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash t u : B} \text{ (APP)}$$

Règles de typage polymorphes

– variables :

$$\frac{(x : A) \in \Gamma \quad A \preceq B}{\Gamma \vdash x : B} \text{ (VAR)}$$

– fonctions :

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \text{fun } x \rightarrow t : A \Rightarrow B} \text{ (FUN)}$$

– applications :

$$\frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash t u : B} \text{ (APP)}$$

– let :

Règles de typage polymorphes

– variables :

$$\frac{(x : A) \in \Gamma \quad A \preceq B}{\Gamma \vdash x : B} \text{ (VAR)}$$

– fonctions :

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \text{fun } x \rightarrow t : A \Rightarrow B} \text{ (FUN)}$$

– applications :

$$\frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash t u : B} \text{ (APP)}$$

– let :

$$\frac{\Gamma \vdash t : A \quad \Gamma, x : \forall_{\Gamma} A \vdash u : B}{\Gamma \vdash \text{let } x = t \text{ in } u : B} \text{ (LET)}$$

```
⊢ let  $f = \text{fun } x \rightarrow x$  in ( $f$  true,  $f$  1)
```

$$\vdash \text{fun } x \rightarrow x : X \Rightarrow X$$
$$f : \forall X. X \Rightarrow X \vdash (f \text{ true}, f 1) : \text{bool} \times \text{int}$$

$$\vdash \text{let } f = \text{fun } x \rightarrow x \text{ in } (f \text{ true}, f 1)$$

(LET)

$$\frac{x : X \vdash x : X}{\vdash \text{fun } x \rightarrow x : X \Rightarrow X} \text{ (FUN)}$$

$$\frac{f : \forall X. X \Rightarrow X \vdash (f \text{ true}, f 1) : \text{bool} \times \text{int}}{\vdash \text{let } f = \text{fun } x \rightarrow x \text{ in } (f \text{ true}, f 1)} \text{ (LET)}$$

$$\frac{\frac{}{x : X \vdash x : X} \text{(VAR)}}{\vdash \text{fun } x \rightarrow x : X \Rightarrow X} \text{(FUN)}$$

$$\frac{f : \forall X. X \Rightarrow X \vdash (f \text{ true}, f 1) : \text{bool} \times \text{int}}{\vdash \text{let } f = \text{fun } x \rightarrow x \text{ in } (f \text{ true}, f 1)} \text{(LET)}$$

Un exemple

$$\frac{\frac{}{x : X \vdash x : X} \text{(VAR)}}{\vdash \text{fun } x \rightarrow x : X \Rightarrow X} \text{(FUN)}$$

$$f : \forall X. X \Rightarrow X \vdash f \text{ true} : \text{bool}$$

$$\frac{\frac{f : \forall X. X \Rightarrow X \vdash f \text{ true} : \text{bool} \quad f : \forall X. X \Rightarrow X \vdash f \text{ 1} : \text{int}}{f : \forall X. X \Rightarrow X \vdash (f \text{ true}, f \text{ 1}) : \text{bool} \times \text{int}} \text{(PAIR)}}{\vdash \text{let } f = \text{fun } x \rightarrow x \text{ in } (f \text{ true}, f \text{ 1})} \text{(LET)}$$

Un exemple

$$\frac{}{x : X \vdash x : X} \text{ (VAR)}$$
$$\frac{}{\vdash \text{fun } x \rightarrow x : X \Rightarrow X} \text{ (FUN)}$$

$$\frac{f : \forall X. X \Rightarrow X \vdash f : \text{bool} \Rightarrow \text{bool} \quad \dots \vdash \text{true} : \text{bool}}{f : \forall X. X \Rightarrow X \vdash f \text{ true} : \text{bool}} \text{ (APP)}$$

$$f : \forall X. X \Rightarrow X \vdash f \ 1 : \text{int}$$

$$f : \forall X. X \Rightarrow X \vdash (f \ \text{true}, f \ 1) : \text{bool} \times \text{int}$$

$$\vdash \text{let } f = \text{fun } x \rightarrow x \text{ in } (f \ \text{true}, f \ 1)$$

(PAIR)

(LET)

Un exemple

$$\frac{\frac{\frac{}{x : X \vdash x : X} \text{(VAR)}}{\vdash \text{fun } x \rightarrow x : X \Rightarrow X} \text{(FUN)}}{\frac{}{f : \forall X.X \Rightarrow X \vdash f : \text{bool} \Rightarrow \text{bool}} \text{(VAR)} \quad \dots \vdash \text{true} : \text{bool}}{f : \forall X.X \Rightarrow X \vdash f \text{ true} : \text{bool}} \text{(APP)}$$
$$\frac{f : \forall X.X \Rightarrow X \vdash f \ 1 : \text{int}}{f : \forall X.X \Rightarrow X \vdash (f \ \text{true}, f \ 1) : \text{bool} \times \text{int}} \text{(PAIR)}$$
$$\frac{}{\vdash \text{let } f = \text{fun } x \rightarrow x \text{ in } (f \ \text{true}, f \ 1)} \text{(LET)}$$

Un exemple

$$\frac{\frac{\frac{}{x : X \vdash x : X} \text{(VAR)}}{\vdash \text{fun } x \rightarrow x : X \Rightarrow X} \text{(FUN)}}{\frac{\frac{}{f : \forall X. X \Rightarrow X \vdash f : \text{bool} \Rightarrow \text{bool}} \text{(VAR)} \quad \frac{}{\dots \vdash \text{true} : \text{bool}} \text{(BOOL)}}{\vdash f \text{ true} : \text{bool}} \text{(APP)}}{\frac{\frac{}{f : \forall X. X \Rightarrow X \vdash f 1 : \text{int}} \text{(PAIR)}}{\vdash f : \forall X. X \Rightarrow X \vdash (f \text{ true}, f 1) : \text{bool} \times \text{int}} \text{(LET)}}{\vdash \text{let } f = \text{fun } x \rightarrow x \text{ in } (f \text{ true}, f 1)}$$

Un exemple

$$\frac{\frac{\frac{}{x : X \vdash x : X} \text{(VAR)}}{\vdash \text{fun } x \rightarrow x : X \Rightarrow X} \text{(FUN)}}{\frac{}{f : \forall X. X \Rightarrow X \vdash f : \text{bool} \Rightarrow \text{bool}} \text{(VAR)} \quad \frac{}{\dots \vdash \text{true} : \text{bool}} \text{(BOOL)}}{\vdash f \text{ true} : \text{bool}} \text{(APP)}$$

⋮

$$\frac{\frac{}{f : \forall X. X \Rightarrow X \vdash f \text{ 1} : \text{int}} \text{(PAIR)}}{\vdash f : \forall X. X \Rightarrow X \vdash (f \text{ true}, f \text{ 1}) : \text{bool} \times \text{int}} \text{(LET)}$$

Dans la règle

$$\frac{\Gamma \vdash t : A \quad \Gamma, x : \forall_{\Gamma} A \vdash u : B}{\Gamma \vdash \text{let } x = t \text{ in } u : B} \text{ (LET)}$$

on ne généralise que les variables qui ne sont pas dans Γ .

Sinon, on aurait

$$\vdash \text{fun } x \rightarrow \text{let } y = x \text{ in } y : X \Rightarrow Y$$

Généralisation

Dans la règle

$$\frac{\Gamma \vdash t : A \quad \Gamma, x : \forall_{\Gamma} A \vdash u : B}{\Gamma \vdash \text{let } x = t \text{ in } u : B} \text{ (LET)}$$

on ne généralise que les variables qui ne sont pas dans Γ .

Sinon, on aurait

$$\frac{x : X \vdash \text{let } y = x \text{ in } y : Y}{\vdash \text{fun } x \rightarrow \text{let } y = x \text{ in } y : X \Rightarrow Y} \text{ (FUN)}$$

Généralisation

Dans la règle

$$\frac{\Gamma \vdash t : A \quad \Gamma, x : \forall_{\Gamma} A \vdash u : B}{\Gamma \vdash \text{let } x = t \text{ in } u : B} \text{ (LET)}$$

on ne généralise que les variables qui ne sont pas dans Γ .

Sinon, on aurait

$$\frac{x : X \vdash x : X \quad x : X, y : \forall X.X \vdash y : Y}{x : X \vdash \text{let } y = x \text{ in } y : Y} \text{ (LET)}$$
$$\frac{x : X \vdash \text{let } y = x \text{ in } y : Y}{\vdash \text{fun } x \rightarrow \text{let } y = x \text{ in } y : X \Rightarrow Y} \text{ (FUN)}$$

Généralisation

Dans la règle

$$\frac{\Gamma \vdash t : A \quad \Gamma, x : \forall \Gamma A \vdash u : B}{\Gamma \vdash \text{let } x = t \text{ in } u : B} \text{ (LET)}$$

on ne généralise que les variables qui ne sont pas dans Γ .

Sinon, on aurait

$$\frac{\frac{\frac{}{x : X \vdash x : X} \text{ (VAR)} \quad x : X, y : \forall X.X \vdash y : Y}{x : X \vdash \text{let } y = x \text{ in } y : Y} \text{ (LET)}}{\vdash \text{fun } x \rightarrow \text{let } y = x \text{ in } y : X \Rightarrow Y} \text{ (FUN)}$$

Généralisation

Dans la règle

$$\frac{\Gamma \vdash t : A \quad \Gamma, x : \forall \Gamma A \vdash u : B}{\Gamma \vdash \text{let } x = t \text{ in } u : B} \text{ (LET)}$$

on ne généralise que les variables qui ne sont pas dans Γ .

Sinon, on aurait

$$\frac{\frac{\frac{}{x : X \vdash x : X} \text{ (VAR)} \quad \frac{}{x : X, y : \forall X.X \vdash y : Y} \text{ (VAR)}}{x : X \vdash \text{let } y = x \text{ in } y : Y} \text{ (LET)}}{\vdash \text{fun } x \rightarrow \text{let } y = x \text{ in } y : X \Rightarrow Y} \text{ (FUN)}$$

Le cours et les TD sont disponibles sur



<https://agreg.mimram.fr/>