Typage

Samuel Mimram

École Polytechnique

Première partie I

Réduction

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).

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

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) :

$$\texttt{fix}\,(\texttt{fun}\ x \to x)$$

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) :

$$\texttt{fix}\,(\texttt{fun}\ x \to x)$$

- des erreurs (inacceptable) :

$$\texttt{true + 5}$$

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) :

$$\texttt{fix}\,(\texttt{fun}\ x \to x)$$

- des erreurs (inacceptable) :

$$\texttt{true + 5}$$

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

Réduction

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

Dans ces règles v, v_i désignent toujours des valeurs.

Réduction

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

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.

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
```

Réduction

- application :

Réduction

- application :

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

Réduction

- application :

$$\frac{u \longrightarrow u'}{t\ u \longrightarrow t\ u'}\ (\text{appr}) \qquad \frac{t \longrightarrow t'}{t\ v \longrightarrow t'\ v}\ (\text{appl})$$

Réduction

- application :

$$\frac{u \longrightarrow u'}{t\ u \longrightarrow t\ u'}\ (\text{appr}) \qquad \frac{t \longrightarrow t'}{t\ v \longrightarrow t'\ v}\ (\text{appl})$$

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

Réduction

- application :

$$\frac{u \longrightarrow u'}{t\ u \longrightarrow t\ u'}\ (\text{appr}) \qquad \frac{t \longrightarrow t'}{t\ v \longrightarrow t'\ v}\ (\text{appl})$$

$$\frac{}{(\texttt{fun}\ x\ \to\ t)\ v \longrightarrow t[x \mapsto v]}\ (\text{app}) \qquad \frac{}{\texttt{add}\ \underline{m}\ \underline{n} \longrightarrow \underline{m+n}}\ (\text{add})$$

Réduction

- application :

$$\frac{u \longrightarrow u'}{t\ u \longrightarrow t\ u'}\ (\text{appr}) \qquad \frac{t \longrightarrow t'}{t\ v \longrightarrow t'\ v}\ (\text{appl})$$

$$\frac{}{(\texttt{fun}\ \ x\ \ \rightarrow\ \ t)\ v \longrightarrow t[x \mapsto v]}\ (\text{app}) \qquad \frac{}{\texttt{add}\ \underline{m}\ \underline{n} \longrightarrow \underline{m+n}}\ (\text{add})$$

- paires :

Réduction

- application :

$$\frac{u \longrightarrow u'}{t\ u \longrightarrow t\ u'}\ (\text{appr}) \qquad \frac{t \longrightarrow t'}{t\ v \longrightarrow t'\ v}\ (\text{appl})$$

$$\frac{}{(\texttt{fun}\ x\ \to\ t)\ v \longrightarrow t[x \mapsto v]}\ (\text{app}) \qquad \frac{}{\texttt{add}\ \underline{m}\ \underline{n} \longrightarrow \underline{m+n}}\ (\text{add})$$

- paires :

$$\frac{t \longrightarrow t'}{(t,v) \longrightarrow (t',v)}\ (\text{pairl}) \qquad \frac{u \longrightarrow u'}{(t,u) \longrightarrow (t,u')}\ (\text{pairr})$$

Réduction

- application :

$$\frac{u \longrightarrow u'}{t\ u \longrightarrow t\ u'}\ (\text{appr}) \qquad \frac{t \longrightarrow t'}{t\ v \longrightarrow t'\ v}\ (\text{appl})$$

$$\frac{}{(\texttt{fun}\ x\ \rightarrow\ t)\ v \longrightarrow t[x \mapsto v]}\ (\text{app}) \qquad \frac{}{\texttt{add}\ \underline{m}\ \underline{n} \longrightarrow \underline{m+n}}\ (\text{add})$$

- paires :

$$\frac{t \longrightarrow t'}{(t,v) \longrightarrow (t',v)}\ (\text{pairl}) \qquad \frac{u \longrightarrow u'}{(t,u) \longrightarrow (t,u')}\ (\text{pairr})$$

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

Réduction

- application :

$$\frac{u \longrightarrow u'}{t\ u \longrightarrow t\ u'}\ (\text{appr}) \qquad \frac{t \longrightarrow t'}{t\ v \longrightarrow t'\ v}\ (\text{appl})$$

$$\frac{}{(\texttt{fun}\ x\ \rightarrow\ t)\ v \longrightarrow t[x \mapsto v]}\ (\text{app}) \qquad \frac{}{\texttt{add}\ \underline{m}\ \underline{n} \longrightarrow \underline{m+n}}\ (\text{add})$$

- paires :

$$\frac{t \longrightarrow t'}{(t,v) \longrightarrow (t',v)}\ (\text{pairl}) \qquad \frac{u \longrightarrow u'}{(t,u) \longrightarrow (t,u')}\ (\text{pairr})$$

$$\frac{}{\texttt{fst}\ (v_1,v_2) \longrightarrow v_1}\ (\text{fst}) \qquad \frac{}{\texttt{snd}\ (v_1,v_2) \longrightarrow v_2}\ (\text{snd})$$

Réduction

- opérateur point fixe :

Réduction

- opérateur point fixe : les règles

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

Réduction

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\texttt{fix}\ t \longrightarrow \texttt{fix}\ t'} \qquad \frac{}{\texttt{fix}\ v \longrightarrow v\ (\texttt{fix}\ v)}$$

Réduction

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\texttt{fix}\ t \longrightarrow \texttt{fix}\ t'} \qquad \frac{}{\texttt{fix}\ v \longrightarrow v\ (\texttt{fix}\ v)}$$

  ne conviennent pas,

Réduction

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\texttt{fix}\ t \longrightarrow \texttt{fix}\ t'} \qquad \frac{}{\texttt{fix}\ v \longrightarrow v\ (\texttt{fix}\ v)}$$

ne conviennent pas, car on a toujours la réduction infinie

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

Réduction

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\texttt{fix}\ t \longrightarrow \texttt{fix}\ t'} \qquad \frac{}{\texttt{fix}\ v \longrightarrow v\ (\texttt{fix}\ v)}$$

ne conviennent pas, car on a toujours la réduction infinie

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

Réduction

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\texttt{fix}\ t \longrightarrow \texttt{fix}\ t'} \qquad \frac{}{\texttt{fix}\ v \longrightarrow v\ (\texttt{fix}\ v)}$$

ne conviennent pas, car on a toujours la réduction infinie

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

Réduction

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\texttt{fix}\ t \longrightarrow \texttt{fix}\ t'} \qquad \frac{}{\texttt{fix}\ v \longrightarrow v\ (\texttt{fix}\ v)}$$

ne conviennent pas, car on a toujours la réduction infinie

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

Réduction

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\texttt{fix}\ t \longrightarrow \texttt{fix}\ t'} \qquad \frac{}{\texttt{fix}\ v \longrightarrow v\ (\texttt{fix}\ v)}$$

ne conviennent pas, car on a toujours la réduction infinie

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

Réduction

- opérateur point fixe : les règles

$$\frac{t \longrightarrow t'}{\texttt{fix}\ t \longrightarrow \texttt{fix}\ t'} \qquad \frac{}{\texttt{fix}\ v \longrightarrow v\ (\texttt{fix}\ v)}$$

ne conviennent pas, car on a toujours la réduction infinie

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

on prendra plutôt les règles :

$$\frac{t \longrightarrow t'}{\texttt{fix}\ t \longrightarrow \texttt{fix}\ t'} \qquad \frac{}{\texttt{fix}\ (\texttt{fun}\ x\ \rightarrow\ t) \longrightarrow t[x \mapsto \texttt{fix}\ (\texttt{fun}\ x\ \rightarrow\ t)]}$$

Stratégies de réduction

Il existe plusieurs **stratégies de réduction** :

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

$$(\texttt{fun}\ x \to x + x)\ (3 + 2) \longrightarrow (\texttt{fun}\ x \to x + x)\ 5 \longrightarrow 5 + 5 \longrightarrow 10$$

Stratégies de réduction

Il existe plusieurs **stratégies de réduction** :

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

$$(\texttt{fun}\ x \to x + x)\ (3 + 2) \longrightarrow (\texttt{fun}\ x \to x + x)\ 5 \longrightarrow 5 + 5 \longrightarrow 10$$

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

$$(\texttt{fun}\ x \to x + x)\ (3 + 2) \longrightarrow (3 + 2) + (3 + 2) \longrightarrow 5 + (3 + 2) \longrightarrow 5 + 5 \longrightarrow 10$$

Stratégies de réduction

Il existe plusieurs **stratégies de réduction** :

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

$$(\texttt{fun}\ x \to x + x)\ (3 + 2) \longrightarrow (\texttt{fun}\ x \to x + x)\ 5 \longrightarrow 5 + 5 \longrightarrow 10$$

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

$$(\texttt{fun}\ x \to x + x)\ (3 + 2) \longrightarrow (3 + 2) + (3 + 2) \longrightarrow 5 + (3 + 2) \longrightarrow 5 + 5 \longrightarrow 10$$

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

$$(\texttt{fun}\ x \to 3 + 2) \longrightarrow (\texttt{fun}\ x \to 5)$$

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

$$
\begin{array}{rcl}
v & ::= & \underline{n} \mid \underline{b} \mid \texttt{add} \mid \texttt{add}\,\underline{n} \\
& \mid & \texttt{fun}\ x \to t \mid \texttt{fix} \\
& \mid & (v, v') \mid \texttt{fst} \mid \texttt{snd}
\end{array}
$$

□

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

$$\frac{t \longrightarrow t'}{(t, v) \longrightarrow (t', v)}\ (\text{pairl}) \qquad \frac{u \longrightarrow u'}{(t, u) \longrightarrow (t, u')}\ (\text{pairr})$$

  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

$$\frac{t \longrightarrow t'}{(t, v) \longrightarrow (t', v)}\ (\text{pairl}) \qquad \frac{u \longrightarrow u'}{(t, u) \longrightarrow (t, u')}\ (\text{pairr})$$

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

- Les autres cas sont similaires. □

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 $t \longrightarrow\!\!\!\!\rightarrow v$.

Équivalence entre les sémantiques

Proposition

Si $t \longrightarrow\!\!\!\!\!\rightarrow v$ *alors* $t \overset{*}{\longrightarrow} v$.

Équivalence entre les sémantiques

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

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

Équivalence entre les sémantiques

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

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

$$\frac{t \longrightarrow\!\!\!\!\!\rightarrow (\texttt{fun}\ x \to t') \qquad u \longrightarrow\!\!\!\!\!\rightarrow u' \qquad t'[x \mapsto u'] \longrightarrow\!\!\!\!\!\rightarrow v}{t\ u \longrightarrow\!\!\!\!\!\rightarrow v}\ (\text{app})$$

Équivalence entre les sémantiques

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

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

$$\frac{t \longrightarrow\!\!\!\!\!\rightarrow (\texttt{fun}\ x\ \rightarrow\ t') \qquad u \longrightarrow\!\!\!\!\!\rightarrow u' \qquad t'[x \mapsto u'] \longrightarrow\!\!\!\!\!\rightarrow v}{t\ u \longrightarrow\!\!\!\!\!\rightarrow v}\ (\text{app})$$

Par hypothèse de récurrence, on a

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

Équivalence entre les sémantiques

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

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

$$\frac{t \longrightarrow\!\!\!\!\!\rightarrow (\texttt{fun}\ x\ \rightarrow\ t') \qquad u \longrightarrow\!\!\!\!\!\rightarrow u' \qquad t'[x \mapsto u'] \longrightarrow\!\!\!\!\!\rightarrow v}{t\ u \longrightarrow\!\!\!\!\!\rightarrow v}\ (\text{app})$$

Par hypothèse de récurrence, on a

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

et on en déduit la suite de réductions

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

Équivalence entre les sémantiques

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

Démonstration.
Par hypothèse de récurrence, on a

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

et on en déduit la suite de réductions

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

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 $v \longrightarrow\!\!\!\!\!\rightarrow v$.

Démonstration.
Par induction sur v. □

Équivalence entre les sémantiques

Pour la réciproque, nous avons besoin de lemmes :

Lemme
Pour toute valeur v, on a $v \longrightarrow\!\!\!\!\!\rightarrow v$.

Démonstration.
Par induction sur v. □

Lemme
Si $t \longrightarrow t'$ et $t' \longrightarrow\!\!\!\!\!\rightarrow v$ alors $t \longrightarrow\!\!\!\!\!\rightarrow v$.

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 $v \longrightarrow\!\!\!\!\!\rightarrow v$.

Lemme
Si $t \longrightarrow t'$ et $t' \longrightarrow\!\!\!\!\!\rightarrow v$ alors $t \longrightarrow\!\!\!\!\!\rightarrow v$.

Proposition
Si $t \overset{*}{\longrightarrow} v$ pour une valeur v alors $t \longrightarrow\!\!\!\!\!\rightarrow v$.

Équivalence entre les sémantiques

Pour la réciproque, nous avons besoin de lemmes :

Lemme
Pour toute valeur v, on a $v \longrightarrow\!\!\!\!\!\rightarrow v$.

Lemme
Si $t \longrightarrow t'$ et $t' \longrightarrow\!\!\!\!\!\rightarrow v$ alors $t \longrightarrow\!\!\!\!\!\rightarrow v$.

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

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 $t \longrightarrow\!\!\!\!\!\rightarrow v$.

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

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

  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

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

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

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

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

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

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

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

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

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

on conclut par

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

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
$$\frac{\Gamma, x : A, \Gamma', y : B \vdash t : C}{\Gamma, x : A, \Gamma' \vdash \texttt{fun}\ y\ \to\ t : B \Rightarrow C}\ (\text{fun})$$

  quitte à renommer y, on peut toujours supposer $x \neq y$. On a donc
$$(\texttt{fun}\ y\ \to\ t)[x \mapsto u] = \texttt{fun}\ y\ \to\ (t[x \mapsto u])$$
  et on conclut
$$\frac{\dfrac{\vdots}{\Gamma, \Gamma', y : B \vdash t[x \mapsto u] : C}}{\Gamma, \Gamma' \vdash \texttt{fun}\ y\ \to\ t[x \mapsto u] : B \Rightarrow C}\ (\text{fun})$$

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. □

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
$$\frac{t \longrightarrow t'}{t\ v \longrightarrow t'\ v}\ (\mathsf{appl})$$
alors la dérivation de typage de $t\ v$ est de la forme
$$\frac{\dfrac{\vdots}{\Gamma \vdash t : A \Rightarrow B} \qquad \dfrac{\vdots}{\Gamma \vdash v : A}}{\Gamma \vdash t\ v : B}\ (\mathsf{app})$$

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
$$\frac{\dfrac{\vdots}{\Gamma \vdash t : A \Rightarrow B} \qquad \dfrac{\vdots}{\Gamma \vdash v : A}}{\Gamma \vdash t\ v : B}\ (\text{app}) \qquad \frac{\dfrac{\vdots}{\Gamma \vdash t' : A \Rightarrow B} \qquad \dfrac{\vdots}{\Gamma \vdash v : A}}{\Gamma \vdash t'\ v : B}\ (\text{app})$$

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 $\dfrac{}{(\texttt{fun}\ x\ \to\ t)\ v \longrightarrow t[x \mapsto v]}\ (\text{app})$

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

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \texttt{fun}\ x\ \to\ t : A \Rightarrow B}\ (\text{fun}) \qquad \dfrac{\vdots}{\Gamma \vdash v : A}}{\Gamma \vdash (\texttt{fun}\ x\ \to\ t)\ v : B}\ (\text{app})$$

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 $\dfrac{}{(\texttt{fun}\ x\ \to\ t)\ v \longrightarrow t[x \mapsto v]}\ (\text{app})$

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

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \texttt{fun}\ x\ \to\ t : A \Rightarrow B}\ (\text{fun}) \qquad \dfrac{\vdots}{\Gamma \vdash v : A}}{\Gamma \vdash (\texttt{fun}\ x\ \to\ t)\ v : B}\ (\text{app}) \qquad \dfrac{\vdots}{\Gamma \vdash t[x \mapsto v] : B}$$

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.

□

Progrès

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

```
true + false
```

Progrès

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

```
true + false
```

Pour les programmes bien typés, la réciproque est vraie :

Progrès

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.*

Progrès

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)

Progrès

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.

Progrès

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

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

  alors

Progrès

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

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

  alors on a immédiatement que $\texttt{fun}\ x\ \rightarrow\ t$ 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

$$\dfrac{\dfrac{\vdots}{\vdash t : A \Rightarrow B} \qquad \dfrac{\vdots}{\vdash u : A}}{\vdash t\ u : B}\ (\mathsf{app})$$

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 $\texttt{fun}\ x \to t'$ sinon $t\ u$ ne serait pas normal, mais on peut avoir $t = \texttt{add}$ et $u = \underline{n}$).

Progrès

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. □

Sûreté

Nous pouvons maintenant montrer le théorème de sûreté du typage :

Sûreté

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.

Sûreté

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 $\vdash t : A$.

Sûreté

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 $\vdash u : A$

Sûreté

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 :

$$\texttt{fun}\ x \to x$$

a les types

$$\texttt{int} \Rightarrow \texttt{int} \qquad \texttt{bool} \Rightarrow \texttt{bool} \qquad \text{etc.}$$

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 :

$$\texttt{fun}\ x \to x$$

a les types

$$\texttt{int} \Rightarrow \texttt{int} \qquad \texttt{bool} \Rightarrow \texttt{bool} \qquad \text{etc.}$$

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

$$X \Rightarrow X$$

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 :

$$\texttt{fun}\ x \to x$$

a les types

$$\texttt{int} \Rightarrow \texttt{int} \qquad \texttt{bool} \Rightarrow \texttt{bool} \qquad \text{etc.}$$

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

$$X \Rightarrow X$$

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

Types

On définit les **types** par la syntaxe

$$A \quad ::= \quad \texttt{int} \quad | \quad \texttt{bool} \quad | \quad A \Rightarrow A' \quad | \quad A \times A'$$

Types

On définit les **types** par la syntaxe

$$A \quad ::= \quad \texttt{int} \quad | \quad \texttt{bool} \quad | \quad A \Rightarrow A' \quad | \quad A \times A' \quad | \quad X$$

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

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

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

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

On note

$$A[\sigma]$$

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

Substitution identité

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

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

Substitution identité

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

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

On a

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

Composition de substitutions

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

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

Composition de substitutions

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

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

On a

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

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,

$$X \Rightarrow X \qquad \sqsubseteq \qquad (\texttt{int} \Rightarrow Y) \Rightarrow (\texttt{int} \Rightarrow Y)$$

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,

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

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,

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

Renommage

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

Ordre de raffinement

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.

Ordre de raffinement

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

Ordre de raffinement

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

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

Ordre de raffinement

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
$$A[\mathrm{id}] = A$$
- transitivité : si $A \sqsubseteq B$ et $B \sqsubseteq C$,

Ordre de raffinement

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
$$A[\mathrm{id}] = A$$
- transitivité : si $A \sqsubseteq B$ et $B \sqsubseteq C$, alors $A[\sigma] = B$ et $B[\tau] = C$

Ordre de raffinement

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

Ordre de raffinement

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
$$A[\mathrm{id}] = A$$
- 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 $\Gamma \vdash t : A$.

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

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

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

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

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

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

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* $\vdash t : B$.

Raffinement et typage

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)

Raffinement et typage

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 $A = (Y \times Y) \Rightarrow (Y \times Y)$

Inférence de type

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.

Systèmes d'équations de type

Un **système d'équations de types** est un ensemble fini

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

de paires de types.

Systèmes d'équations de type

Un **système d'équations de types** est un ensemble fini

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

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,

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

a pour solution la plus générale

Systèmes d'équations de type

Par exemple,

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

a pour solution la plus générale

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

une autre solution étant

Systèmes d'équations de type

Par exemple,

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

a pour solution la plus générale

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

une autre solution étant

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

Systèmes d'équations de type

Par exemple,

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

Systèmes d'équations de type

Par exemple,

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

n'a pas de solution.

Systèmes d'équations de type

Par exemple,

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

Systèmes d'équations de type

Par exemple,

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

n'a pas de solution.

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.

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.

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 $\bot$ un système d'équations sans solution.

Génération d'équations

- pour un entier n,

Génération d'équations

- pour un entier $\underline{n}$,

$$A_{\underline{n}} = \texttt{int} \qquad E_{\underline{n}} = \emptyset$$

Génération d'équations

- pour un entier $\underline{n}$,

$$A_{\underline{n}} = \texttt{int} \qquad E_{\underline{n}} = \emptyset$$

Génération d'équations

- pour un entier $\underline{n}$,

$$A_{\underline{n}} = \texttt{int} \qquad E_{\underline{n}} = \emptyset$$

- pour un booléen $\underline{b}$,

Génération d'équations

- pour un entier $\underline{n}$,

$$A_{\underline{n}} = \texttt{int} \qquad E_{\underline{n}} = \emptyset$$

- pour un booléen $\underline{b}$,

$$A_{\underline{b}} = \texttt{bool} \qquad E_{\underline{b}} = \emptyset$$

Génération d'équations

- pour un entier $\underline{n}$,

$$A_{\underline{n}} = \texttt{int} \qquad E_{\underline{n}} = \emptyset$$

- pour un booléen $\underline{b}$,

$$A_{\underline{b}} = \texttt{bool} \qquad E_{\underline{b}} = \emptyset$$

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

Génération d'équations

- pour un entier $\underline{n}$,

$$A_{\underline{n}} = \texttt{int} \qquad E_{\underline{n}} = \emptyset$$

- pour un booléen $\underline{b}$,

$$A_{\underline{b}} = \texttt{bool} \qquad E_{\underline{b}} = \emptyset$$

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

$$A_x = A \qquad E_x = \emptyset$$

Génération d'équations

- pour un entier $\underline{n}$,

$$A_{\underline{n}} = \texttt{int} \qquad E_{\underline{n}} = \emptyset$$

- pour un booléen $\underline{b}$,

$$A_{\underline{b}} = \texttt{bool} \qquad E_{\underline{b}} = \emptyset$$

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

$$A_x = A \qquad E_x = \emptyset$$

- pour x n'apparaissant pas dans Γ,

Génération d'équations

- pour un entier $\underline{n}$,

$$A_{\underline{n}} = \texttt{int} \qquad E_{\underline{n}} = \emptyset$$

- pour un booléen $\underline{b}$,

$$A_{\underline{b}} = \texttt{bool} \qquad E_{\underline{b}} = \emptyset$$

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

$$A_x = A \qquad E_x = \emptyset$$

- pour x n'apparaissant pas dans Γ,

$$A_x = ? \qquad E_x = \bot$$

Génération d'équations

- pour une abstraction $\texttt{fun}\ x \to t$,

Génération d'équations

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

$$A_{\texttt{fun}\ x\ \rightarrow\ t} = X \Rightarrow A_t \qquad\qquad E_{\texttt{fun}\ x\ \rightarrow\ t} = E_t$$

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

Génération d'équations

- pour une abstraction $\texttt{fun}\ x\ \texttt{->}\ t$,

$$A_{\texttt{fun}\ x\ \texttt{->}\ t} = X \Rightarrow A_t \qquad\qquad E_{\texttt{fun}\ x\ \texttt{->}\ t} = E_t$$

  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 $\texttt{fun}\ x \to t$,

$$A_{\texttt{fun}\ x \to t} = X \Rightarrow A_t \qquad\qquad E_{\texttt{fun}\ x \to t} = E_t$$

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

- pour une application $t\ u$,

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

  où X est une variable fraîche,

Génération d'équations

- pour une abstraction $\texttt{fun}\ x \to t$,

$$A_{\texttt{fun}\ x \to t} = X \Rightarrow A_t \qquad\qquad E_{\texttt{fun}\ x \to t} = E_t$$

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

- pour une application $t\ u$,

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

  où X est une variable fraîche,

- pour une paire (t, u),

Génération d'équations

- pour une abstraction $\texttt{fun}\ x \to t$,

$$A_{\texttt{fun}\ x \to t} = X \Rightarrow A_t \qquad\qquad E_{\texttt{fun}\ x \to t} = E_t$$

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

- pour une application $t\ u$,

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

  où X est une variable fraîche,

- pour une paire (t, u),

$$A_{(t,u)} = A_t \times A_u \qquad\qquad E_{(t,u)} = E_t \cup E_u$$

Génération d'équations

- pour une abstraction $\texttt{fun}\ x \to t$,

$$A_{\texttt{fun}\ x \to t} = X \Rightarrow A_t \qquad\qquad E_{\texttt{fun}\ x \to t} = E_t$$

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

- pour une application $t\ u$,

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

  où X est une variable fraîche,

- pour une paire (t, u),

$$A_{(t,u)} = A_t \times A_u \qquad\qquad E_{(t,u)} = E_t \cup E_u$$

- pour $\texttt{fst}$,

Génération d'équations

- pour une abstraction $\texttt{fun}\ x \to t$,

$$A_{\texttt{fun}\ x \to t} = X \Rightarrow A_t \qquad\qquad E_{\texttt{fun}\ x \to t} = E_t$$

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

- pour une application $t\ u$,

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

  où X est une variable fraîche,

- pour une paire (t, u),

$$A_{(t,u)} = A_t \times A_u \qquad\qquad E_{(t,u)} = E_t \cup E_u$$

- pour $\texttt{fst}$,

$$A_{\texttt{fst}} = (X \times Y) \Rightarrow X \qquad\qquad E_{\texttt{fst}} = \emptyset$$

  avec X et Y fraîches

Génération d'équations

- pour une abstraction $\texttt{fun}\ x \to t$,

$$A_{\texttt{fun}\ x \to t} = X \Rightarrow A_t \qquad\qquad E_{\texttt{fun}\ x \to t} = E_t$$

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

- pour une application $t\ u$,

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

  où X est une variable fraîche,

- pour une paire (t, u),

$$A_{(t,u)} = A_t \times A_u \qquad\qquad E_{(t,u)} = E_t \cup E_u$$

- pour $\texttt{fst}$,

$$A_{\texttt{fst}} = (X \times Y) \Rightarrow X \qquad\qquad E_{\texttt{fst}} = \emptyset$$

  avec X et Y fraîches ($\texttt{snd}$ et $\texttt{fix}$ similaires).

Génération d'équations

Étant donné t dans Γ,

Génération d'équations

Étant donné t dans Γ,

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

Génération d'équations

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. □

Génération d'équations

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 $\Gamma[\sigma] \vdash t : A$, il existe σ' telle que $\Gamma[\sigma] = \Gamma[\sigma']$ et $A = A_t[\sigma']$.

Génération d'équations

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 $\Gamma[\sigma] \vdash t : A$, il existe σ' telle que $\Gamma[\sigma] = \Gamma[\sigma']$ et $A = A_t[\sigma']$.

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

Génération d'équations

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 ?

Unification

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 $\text{unify}(E)$.

Unification

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 *récursivement* de la façon suivante...

Unification

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

Unification

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

Unification

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

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

Unification

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

Unification

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

Unification

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

Unification

$$\begin{aligned}
\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(X \stackrel{?}{=} X, E) &= \mathsf{unify}(E) \\
\mathsf{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) &=
\end{aligned}$$

Unification

$$
\begin{aligned}
\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(X \stackrel{?}{=} X, E) &= \mathsf{unify}(E) \\
\mathsf{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)
\end{aligned}
$$

Unification

$$
\begin{aligned}
\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(X \stackrel{?}{=} X, E) &= \mathsf{unify}(E) \\
\mathsf{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) &=
\end{aligned}
$$

Unification

$$
\begin{aligned}
\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(X \stackrel{?}{=} X, E) &= \mathsf{unify}(E) \\
\mathsf{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)
\end{aligned}
$$

Unification

$$
\begin{aligned}
\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(X \stackrel{?}{=} X, E) &= \mathsf{unify}(E) \\
\mathsf{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}(X \stackrel{?}{=} A, E) &=
\end{aligned}
$$

Unification

$$
\begin{aligned}
\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(X \stackrel{?}{=} X, E) &= \mathsf{unify}(E) \\
\mathsf{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}(X \stackrel{?}{=} A, E) &= \mathsf{unify}(E[X \mapsto A]) \circ [X \mapsto A]
\end{aligned}
$$

Unification

$$
\begin{aligned}
\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(X \stackrel{?}{=} X, E) &= \mathsf{unify}(E) \\
\mathsf{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}(X \stackrel{?}{=} A, E) &= \mathsf{unify}(E[X \mapsto A]) \circ [X \mapsto A] \qquad \text{si } X \notin \mathsf{VL}(A)
\end{aligned}
$$

Unification

$$
\begin{aligned}
\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(X \stackrel{?}{=} X, E) &= \mathsf{unify}(E) \\
\mathsf{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}(X \stackrel{?}{=} A, E) &= \mathsf{unify}(E[X \mapsto A]) \circ [X \mapsto A] \qquad \text{si } X \notin \mathsf{VL}(A) \\
\mathsf{unify}(A \stackrel{?}{=} X, E) &=
\end{aligned}
$$

Unification

$$
\begin{aligned}
\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(X \stackrel{?}{=} X, E) &= \mathsf{unify}(E) \\
\mathsf{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}(X \stackrel{?}{=} A, E) &= \mathsf{unify}(E[X \mapsto A]) \circ [X \mapsto A] \qquad \text{si } X \notin \mathsf{VL}(A) \\
\mathsf{unify}(A \stackrel{?}{=} X, E) &= \mathsf{unify}(E[X \mapsto A]) \circ [X \mapsto A] \qquad \text{si } X \notin \mathsf{VL}(A)
\end{aligned}
$$

Unification

$$
\begin{aligned}
\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(X \stackrel{?}{=} X, E) &= \mathsf{unify}(E) \\
\mathsf{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}(X \stackrel{?}{=} A, E) &= \mathsf{unify}(E[X \mapsto A]) \circ [X \mapsto A] \qquad \text{si } X \notin \mathsf{VL}(A) \\
\mathsf{unify}(A \stackrel{?}{=} X, E) &= \mathsf{unify}(E[X \mapsto A]) \circ [X \mapsto A] \qquad \text{si } X \notin \mathsf{VL}(A) \\
\mathsf{unify}(\emptyset) &=
\end{aligned}
$$

Unification

$$
\begin{aligned}
\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(X \stackrel{?}{=} X, E) &= \mathsf{unify}(E) \\
\mathsf{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}(X \stackrel{?}{=} A, E) &= \mathsf{unify}(E[X \mapsto A]) \circ [X \mapsto A] \qquad \text{si } X \notin \mathsf{VL}(A) \\
\mathsf{unify}(A \stackrel{?}{=} X, E) &= \mathsf{unify}(E[X \mapsto A]) \circ [X \mapsto A] \qquad \text{si } X \notin \mathsf{VL}(A) \\
\mathsf{unify}(\emptyset) &= \mathsf{id}
\end{aligned}
$$

Unification

$$
\begin{aligned}
\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(X \stackrel{?}{=} X, E) &= \mathsf{unify}(E) \\
\mathsf{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}(X \stackrel{?}{=} A, E) &= \mathsf{unify}(E[X \mapsto A]) \circ [X \mapsto A] \qquad \text{si } X \notin \mathsf{VL}(A) \\
\mathsf{unify}(A \stackrel{?}{=} X, E) &= \mathsf{unify}(E[X \mapsto A]) \circ [X \mapsto A] \qquad \text{si } X \notin \mathsf{VL}(A) \\
\mathsf{unify}(\emptyset) &= \mathsf{id}
\end{aligned}
$$

Dans tous les autres cas

Unification

$$
\begin{aligned}
\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) &= \mathsf{unify}(E) \\
\mathsf{unify}(X \stackrel{?}{=} X, E) &= \mathsf{unify}(E) \\
\mathsf{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}((A \times B) \stackrel{?}{=} (A' \times B'), E) &= \mathsf{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E) \\
\mathsf{unify}(X \stackrel{?}{=} A, E) &= \mathsf{unify}(E[X \mapsto A]) \circ [X \mapsto A] \qquad \text{si } X \notin \mathsf{VL}(A) \\
\mathsf{unify}(A \stackrel{?}{=} X, E) &= \mathsf{unify}(E[X \mapsto A]) \circ [X \mapsto A] \qquad \text{si } X \notin \mathsf{VL}(A) \\
\mathsf{unify}(\emptyset) &= \mathsf{id}
\end{aligned}
$$

Dans tous les autres cas unify échoue.

Terminaison

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

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

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

Terminaison

Nous allons avoir besoin de deux mesures :

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

Terminaison

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

$$|\texttt{int}|_{\text{op}} = |\texttt{bool}|_{\text{op}} = |X|_{\text{op}} = 1$$
$$|A \Rightarrow B|_{\text{op}} = |A \times B|_{\text{op}} = 1 + |A|_{\text{op}} + |B|_{\text{op}}$$

Terminaison

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

$$|\texttt{int}|_{\text{op}} = |\texttt{bool}|_{\text{op}} = |X|_{\text{op}} = 1$$
$$|A \Rightarrow B|_{\text{op}} = |A \times B|_{\text{op}} = 1 + |A|_{\text{op}} + |B|_{\text{op}}$$

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

Terminaison

Les appels récursifs décroissent :

$$\frac{}{\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) = \mathsf{unify}(E)} \quad \Big| E |_{\mathrm{var}} \; \Big| E |_{\mathrm{op}}$$

Terminaison

Les appels récursifs décroissent :

	$\lvert E\rvert_{\text{var}}$	$\lvert E\rvert_{\text{op}}$
$\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) = \mathsf{unify}(E)$	$=$	$<$

Terminaison

Les appels récursifs décroissent :

	$\lvert E\rvert_{\text{var}}$	$\lvert E\rvert_{\text{op}}$
$\mathsf{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) = \mathsf{unify}(E)$	$=$	$<$
$\mathsf{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) = \mathsf{unify}(E)$		

Terminaison

Les appels récursifs décroissent :

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

Terminaison

Les appels récursifs décroissent :

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

Terminaison

Les appels récursifs décroissent :

	$\lvert E\rvert_{\text{var}}$	$\lvert E\rvert_{\text{op}}$
$\text{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$	$\leqslant$	$<$

Terminaison

Les appels récursifs décroissent :

	$\lvert E\rvert_{\text{var}}$	$\lvert E\rvert_{\text{op}}$
$\text{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$	$\leqslant$	$<$
$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$		

Terminaison

Les appels récursifs décroissent :

	$\lvert E\rvert_{\text{var}}$	$\lvert E\rvert_{\text{op}}$
$\text{unify}(\texttt{int} \stackrel{?}{=} \texttt{int}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(\texttt{bool} \stackrel{?}{=} \texttt{bool}, E) = \text{unify}(E)$	$=$	$<$
$\text{unify}(X \stackrel{?}{=} X, E) = \text{unify}(E)$	$\leqslant$	$<$
$\text{unify}((A \Rightarrow B) \stackrel{?}{=} (A' \Rightarrow B'), E) = \text{unify}(A \stackrel{?}{=} A', B \stackrel{?}{=} B', E)$	$=$	$<$

Terminaison

Les appels récursifs décroissent :

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

Terminaison

Les appels récursifs décroissent :

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

Terminaison

Les appels récursifs décroissent :

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

Terminaison

Les appels récursifs décroissent :

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

Terminaison

Les appels récursifs décroissent :

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

Terminaison

Les appels récursifs décroissent :

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

Correction

Montrons que notre algorithme est correct :

Lemme
Pour les équations de la forme

$$\mathsf{unify}(E) \quad = \quad \mathsf{unify}(E')$$

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

Démonstration.
Par inspection des cas :

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

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

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

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

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

Correction

Lemme

Pour $X \notin \mathrm{VL}(A)$, *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* $\tau(X) = X$.

Correction

Lemme

Pour $X \notin \mathrm{VL}(A)$, 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 $(X \stackrel{?}{=} A, E)$,

Correction

Lemme

Pour $X \notin \mathrm{VL}(A)$, 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,

Correction

Lemme

Pour $X \notin \mathrm{VL}(A)$, 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 $X \notin \mathrm{VL}(A)$, 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 $E[X \mapsto A]$. □

Correction

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

Correction

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*.*

Correction

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. □

Complétude

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 $A = \texttt{int}$ et $B = \texttt{bool}$, ou $A = \texttt{int}$ 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.

Complétude

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 $X \in \mathrm{VL}(A)$ et $A \neq X$. Une solution σ vérifie $\sigma(X) = A[\sigma]$, et donc

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

  Impossible. □

Implémentation

En pratique, la règle

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

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

Implémentation

En pratique, la règle

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

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 `a` (avec une indirection).

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

Let

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

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

Let

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

$$\texttt{let } x \texttt{ = } t \texttt{ in } u \quad = \quad (\texttt{fun } x \to u)\, t$$

(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 type.

Le programme suivant est typable :

$$\texttt{((fun } x \texttt{ -> } x\texttt{) 5,(fun } x \texttt{ -> } x\texttt{) true)}$$

Monomorphisme

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

Le programme suivant est typable :

$$\texttt{((fun } x \texttt{ -> } x\texttt{) 5,(fun } x \texttt{ -> } x\texttt{) true)}$$

mais pas le programme

$$\texttt{let } f \texttt{ = (fun } x \texttt{ -> } x\texttt{) in (}f\texttt{ 5,}f\texttt{ true)}$$

Monomorphisme

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

Le programme suivant est typable :

$$\texttt{((fun } x \texttt{ -> } x\texttt{) 5,(fun } x \texttt{ -> } x\texttt{) true)}$$

mais pas le programme

$$\texttt{let } f \texttt{ = (fun } x \texttt{ -> } x\texttt{) in (}f\texttt{ 5,}f\texttt{ true)}$$

qui équivaut à

$$\texttt{(fun } f \texttt{ -> (}f\texttt{ 5,}f\texttt{ true)) (fun } x \texttt{ -> } x\texttt{)}$$

Polymorphisme

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

$$\forall X_1 \ldots \forall X_n . A$$

Polymorphisme

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

$$\forall X_1 \ldots \forall X_n.A$$

Par exemple, l'identité aura le type

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

ce qui est généralement noté

```
'a -> 'a
```

Généralisation

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)
```

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)
```

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)
```

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)
```

$\forall X.X \Rightarrow X$ $\quad X \Rightarrow X$

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)
```

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

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)
```

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

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

Schémas de types

Un **schéma de type** est une expression de la forme

$$\forall X_1 \ldots \forall X_n . A$$

où A est un type (sans quantification).

Schémas de types

Un **schéma de type** est une expression de la forme

$$\forall X_1 \ldots \forall X_n . A$$

où A est un type (sans quantification).
Les variables X_i sont alors liées dans A (on peut les renommer).

Schémas de types

Un **schéma de type** est une expression de la forme

$$\forall X_1 \ldots . \forall X_n . A$$

où A est un type (sans quantification).
Les variables X_i sont alors liées dans A (on peut les renommer).

A est une **instance** de $\forall X_1 \ldots . \forall X_n . B$, ce qu'on note

$$A \quad \preceq \quad \forall X_1 \ldots . \forall X_n . B$$

lorsqu'il existe des types A_1, ..., A_n tels que

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

Schémas de types

Un **schéma de type** est une expression de la forme

$$\forall X_1 \ldots \forall X_n . A$$

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

$$\forall_\Gamma A \quad = \quad \forall X_1 \ldots \forall X_n . B$$

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

Règles de typage polymorphes

On considère des contextes

$$\Gamma \quad = \quad x_1 : A_1, \ldots, x_n : A_n$$

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 :

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

Règles de typage polymorphes

- variables :

$$\frac{(x : B) \in \Gamma \qquad A \preceq B}{\Gamma \vdash x : A}\,(\text{var})$$

- fonctions :

Règles de typage polymorphes

- variables :

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

- fonctions :

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

Règles de typage polymorphes

- variables :

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

- fonctions :

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

- applications :

Règles de typage polymorphes

- variables :

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

- fonctions :

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \texttt{fun}\ x\ \texttt{->}\ t : A \Rightarrow B}\ (\text{fun})$$

- applications :

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

Règles de typage polymorphes

- variables :

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

- fonctions :

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

- applications :

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

- let :

Règles de typage polymorphes

- variables :

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

- fonctions :

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \texttt{fun}\ x\ \texttt{->}\ t : A \Rightarrow B}\ (\text{fun})$$

- applications :

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

- let :

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

Un exemple

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

Un exemple

$$\vdash \texttt{fun}\ x \to x : X \Rightarrow X$$

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

Un exemple

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

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

Un exemple

$$\dfrac{\dfrac{}{x : X \vdash x : X}\,(\text{var})}{\vdash \texttt{fun}\ x \to x : X \Rightarrow X}\,(\text{fun})$$

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

Un exemple

$$\dfrac{\dfrac{}{x : X \vdash x : X}\,(\text{var})}{\vdash \texttt{fun}\ x \to x : X \Rightarrow X}\,(\text{fun})$$

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

$$\dfrac{\dfrac{f : \forall X.X \Rightarrow X \vdash f\ 1 : \texttt{int}}{f : \forall X.X \Rightarrow X \vdash (f\ \texttt{true}, f\ 1) : \texttt{bool} \times \texttt{int}}\,(\text{pair})}{\vdash \texttt{let}\ f = \texttt{fun}\ x \to x\ \texttt{in}\ (f\ \texttt{true}, f\ 1)}\,(\text{let})$$

Un exemple

$$\dfrac{\dfrac{\dfrac{}{x : X \vdash x : X}\,(\text{var})}{\vdash \texttt{fun}\ x \to x : X \Rightarrow X}\,(\text{fun}) \qquad \ldots \vdash \texttt{true} : \texttt{bool}}{f : \forall X.X \Rightarrow X \vdash f\ \texttt{true}}\,(\text{app})$$

$$\dfrac{\dfrac{f : \forall X.X \Rightarrow X \vdash f\ 1 : \texttt{int}}{f : \forall X.X \Rightarrow X \vdash (f\ \texttt{true}, f\ 1) : \texttt{bool} \times \texttt{int}}\,(\text{pair})}{\vdash \texttt{let}\ f\ =\ \texttt{fun}\ x \to x\ \texttt{in}\ (f\ \texttt{true}, f\ 1)}\,(\text{let})$$

Un exemple

$$\dfrac{\dfrac{\dfrac{}{x : X \vdash x : X}\,(\text{var})}{\vdash \texttt{fun}\ x\ \to\ x : X \Rightarrow X}\,(\text{fun}) \qquad \dfrac{}{f : \forall X.X \Rightarrow X \vdash f : \texttt{bool} \Rightarrow \texttt{bool}}\,(\text{var}) \qquad \ldots \vdash \texttt{true} : \texttt{bool}}{f : \forall X.X \Rightarrow X \vdash f\ \texttt{true}}\,(\text{app})$$

$$\dfrac{\dfrac{f : \forall X.X \Rightarrow X \vdash f\ 1 : \texttt{int}}{f : \forall X.X \Rightarrow X \vdash (f\ \texttt{true}, f\ 1) : \texttt{bool} \times \texttt{int}}\,(\text{pair})}{\vdash \texttt{let}\ f\ =\ \texttt{fun}\ x\ \to\ x\ \texttt{in}\ (f\ \texttt{true}, f\ 1)}\,(\text{let})$$

Un exemple

$$\dfrac{\dfrac{\dfrac{}{x : X \vdash x : X}\,(\text{var})}{\vdash \texttt{fun}\ x\ \to\ x : X \Rightarrow X}\,(\text{fun}) \qquad \dfrac{}{\ldots \vdash \texttt{true} : \texttt{bool}}\,(\text{bool})}{f : \forall X.X \Rightarrow X \vdash f\ \texttt{true}}\,(\text{app})$$

$$\dfrac{\dfrac{f : \forall X.X \Rightarrow X \vdash f\ 1 : \texttt{int}}{f : \forall X.X \Rightarrow X \vdash (f\ \texttt{true}, f\ 1) : \texttt{bool} \times \texttt{int}}\,(\text{pair})}{\vdash \texttt{let}\ f\ =\ \texttt{fun}\ x\ \to\ x\ \texttt{in}\ (f\ \texttt{true}, f\ 1)}\,(\text{let})$$

Un exemple

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : X \vdash x : X}\,(\text{var})}{\vdash \texttt{fun}\ x\ \to\ x : X \Rightarrow X}\,(\text{fun}) \qquad \dfrac{}{\ldots \vdash \texttt{true} : \texttt{bool}}\,(\text{bool})}{f : \forall X.X \Rightarrow X \vdash f\ \texttt{true}}\,(\text{app})}{\dfrac{\dfrac{\vdots}{f : \forall X.X \Rightarrow X \vdash f\ 1 : \texttt{int}}}{f : \forall X.X \Rightarrow X \vdash (f\ \texttt{true}, f\ 1) : \texttt{bool} \times \texttt{int}}\,(\text{pair})}{\vdash \texttt{let}\ f\ =\ \texttt{fun}\ x\ \to\ x\ \texttt{in}\ (f\ \texttt{true}, f\ 1)}\,(\text{let})$$

Généralisation

Dans la règle

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

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

Sinon, on aurait

$$\vdash \texttt{fun } x \to \texttt{let } y \texttt{ = } x \texttt{ in } y : X \Rightarrow Y$$

Généralisation

Dans la règle

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

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

Sinon, on aurait

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

Généralisation

Dans la règle

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

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

Sinon, on aurait

$$\cfrac{\cfrac{x : X \vdash x : X \qquad x : X, y : \forall X.X \vdash y : Y}{x : X \vdash \texttt{let } y \texttt{ = } x \texttt{ in } y : Y}\ (\text{let})}{\vdash \texttt{fun } x \to \texttt{let } y \texttt{ = } x \texttt{ in } y : X \Rightarrow Y}\ (\text{fun})$$

Généralisation

Dans la règle

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

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

Sinon, on aurait

$$\dfrac{\dfrac{\dfrac{}{x : X \vdash x : X}\ (\text{var}) \qquad x : X, y : \forall X.X \vdash y : Y}{x : X \vdash \texttt{let } y \texttt{ = } x \texttt{ in } y : Y}\ (\text{let})}{\vdash \texttt{fun } x \to \texttt{let } y \texttt{ = } x \texttt{ in } y : X \Rightarrow Y}\ (\text{fun})$$

Généralisation

Dans la règle

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

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

Sinon, on aurait

$$\dfrac{\dfrac{\dfrac{}{x : X \vdash x : X}\ (\text{var}) \qquad \dfrac{}{x : X, y : \forall X.X \vdash y : Y}\ (\text{var})}{x : X \vdash \texttt{let } y \texttt{ = } x \texttt{ in } y : Y}\ (\text{let})}{\vdash \texttt{fun } x \to \texttt{let } y \texttt{ = } x \texttt{ in } y : X \Rightarrow Y}\ (\text{fun})$$