Typage

Samuel Mimram

École Polytechnique

Vers des langages typés

Dans le langage IMP, nous avons 3 catégories syntaxiques :
les expressions arithmétiques / les expressions booléennes / les commandes.

Vers des langages typés

Dans le langage IMP, nous avons 3 catégories syntaxiques :
les expressions arithmétiques / les expressions booléennes / les commandes.

Dans un langage plus réaliste, on ne peut pas déterminer le type à partir de la syntaxe : quel est le type d'une variable `x` ?

Vers des langages typés

Comme on n'a plus de catégories syntaxiques, des programmes qui n'ont aucun sens apparaissent :

```
2 + true
```

Vers des langages typés

Comme on n'a plus de catégories syntaxiques, des programmes qui n'ont aucun sens apparaissent :

```
2 + true
```

(on ne veut pas faire de conversions implicites ici)

Vers des langages typés

Comme on n'a plus de catégories syntaxiques, des programmes qui n'ont aucun sens apparaissent :

```
2 + true
```

(on ne veut pas faire de conversions implicites ici)

C'est rarement une « faute directe » :

```
x := true; ...; y := 2 + x
```

On cherche donc

Vers des langages typés

Comme on n'a plus de catégories syntaxiques, des programmes qui n'ont aucun sens apparaissent :

```
2 + true
```

(on ne veut pas faire de conversions implicites ici)

C'est rarement une « faute directe » :

```
x := true; ...; y := 2 + x
```

On cherche donc

- à s'assurer de l'absence d'erreurs à l'exécution,

Vers des langages typés

Comme on n'a plus de catégories syntaxiques, des programmes qui n'ont aucun sens apparaissent :

```
2 + true
```

(on ne veut pas faire de conversions implicites ici)

C'est rarement une « faute directe » :

```
x := true; ...; y := 2 + x
```

On cherche donc

- à s'assurer de l'absence d'erreurs à l'exécution,
- quelle que soit l'exécution (certains paramètres peuvent changer),

Vers des langages typés

Comme on n'a plus de catégories syntaxiques, des programmes qui n'ont aucun sens apparaissent :

```
2 + true
```

(on ne veut pas faire de conversions implicites ici)

C'est rarement une « faute directe » :

```
x := true; ...; y := 2 + x
```

On cherche donc

- à s'assurer de l'absence d'erreurs à l'exécution,
- quelle que soit l'exécution (certains paramètres peuvent changer),
- mais sans avoir à exécuter le programme.

Vers des langages typés

Comme on n'a plus de catégories syntaxiques, des programmes qui n'ont aucun sens apparaissent :

$$\texttt{2 + true}$$

(on ne veut pas faire de conversions implicites ici)

C'est rarement une « faute directe » :

$$x \texttt{ := true; ...; } y \texttt{ := 2} + x$$

On cherche donc

- à s'assurer de l'absence d'erreurs à l'exécution,
- quelle que soit l'exécution (certains paramètres peuvent changer),
- mais sans avoir à exécuter le programme.

Le **typage** va répondre à nos attentes.

Mini-ML

La syntaxe des programmes **mini-ML** est donnée par

$$t \quad ::= \quad \underline{n} \mid \underline{b}$$

Mini-ML

La syntaxe des programmes **mini-ML** est donnée par

$$t \quad ::= \quad \underline{n} \mid \underline{b} \mid \texttt{add}$$

Mini-ML

La syntaxe des programmes **mini-ML** est donnée par

$$
\begin{array}{rcl}
t & ::= & \underline{n} \mid \underline{b} \mid \texttt{add} \\
& \mid & x
\end{array}
$$

Mini-ML

La syntaxe des programmes **mini-ML** est donnée par

$$
\begin{array}{rcl}
t & ::= & \underline{n} \mid \underline{b} \mid \texttt{add} \\
& \mid & x \mid \texttt{fun}\ x \to t
\end{array}
$$

Mini-ML

La syntaxe des programmes **mini-ML** est donnée par

$$
\begin{array}{rcl}
t & ::= & \underline{n} \mid \underline{b} \mid \texttt{add} \\
& \mid & x \mid \texttt{fun}\ x \to t \mid t\ t'
\end{array}
$$

Mini-ML

La syntaxe des programmes **mini-ML** est donnée par

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

Mini-ML

La syntaxe des programmes **mini-ML** est donnée par

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

Mini-ML

La syntaxe des programmes **mini-ML** est donnée par

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

- on dira : *programme*, *expression* ou *terme*.

Mini-ML

La syntaxe des programmes **mini-ML** est donnée par

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

- on dira : *programme*, *expression* ou *terme*.
- on suppose fixé un ensemble dénombrable de variables x, y, etc.

Mini-ML

La syntaxe des programmes **mini-ML** est donnée par

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

- on dira : *programme*, *expression* ou *terme*.
- on suppose fixé un ensemble dénombrable de variables x, y, etc.
- on écrit parfois t + u au lieu de add t u

Mini-ML

La syntaxe des programmes **mini-ML** est donnée par

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

- on dira : *programme*, *expression* ou *terme*.
- on suppose fixé un ensemble dénombrable de variables x, y, etc.
- on écrit parfois t + u au lieu de add t u
- on s'autorise parfois d'autres opérations :
  arithmétique, branchement conditionnel, etc.

Mini-ML

Le langage mini-ML est

- **fonctionnel** (`fun`) : on peut faire des fonctions non-nommées

Mini-ML

Le langage mini-ML est

- **fonctionnel** (`fun`) : on peut faire des fonctions non-nommées
- **pur** : il n'y a pas d'état externe (*x* `:= ...`)

Comportements non spécifiés

Certains programmes mini-ML n'ont clairement pas de sens, car leur résultat n'est pas spécifié :

Comportements non spécifiés

Certains programmes mini-ML n'ont clairement pas de sens, car leur résultat n'est pas spécifié :

- `true + 3`

Comportements non spécifiés

Certains programmes mini-ML n'ont clairement pas de sens, car leur résultat n'est pas spécifié :

- `true + 3`
- `fst 5`

Comportements non spécifiés

Certains programmes mini-ML n'ont clairement pas de sens, car leur résultat n'est pas spécifié :

- `true + 3`
- `fst 5`
- `true 2`

Comportements non spécifiés

Certains programmes mini-ML n'ont clairement pas de sens, car leur résultat n'est pas spécifié :

- `true + 3`
- `fst 5`
- `true 2`
- etc.

Comportements non spécifiés

Certains programmes mini-ML n'ont clairement pas de sens, car leur résultat n'est pas spécifié :

- `true + 3`
- `fst 5`
- `true 2`
- etc.

Encore une fois, ces problèmes peuvent apparaître à l'exécution :

Comportements non spécifiés

Certains programmes mini-ML n'ont clairement pas de sens, car leur résultat n'est pas spécifié :

- `true + 3`
- `fst 5`
- `true 2`
- etc.

Encore une fois, ces problèmes peuvent apparaître à l'exécution :

$$(\texttt{fun}\ x \to \underline{1} + x)\,(\texttt{if}\ \ldots\ \texttt{then}\ \underline{5}\ \texttt{else}\ \texttt{true})$$

Comportements non spécifiés

On pourrait détecter ce genre de situations à l'exécution :

- *vérifications à l'exécution* : lorsqu'on a une fonction appliquée à un argument dont le résultat n'est pas spécifié

Comportements non spécifiés

On pourrait détecter ce genre de situations à l'exécution :

- *vérifications à l'exécution* : lorsqu'on a une fonction appliquée à un argument dont le résultat n'est pas spécifié
- *typage dynamique* : on pourrait spécifier le type d'argument attendu et le vérifier à l'exécution (par exemple `add` attend deux entiers)

Comportements non spécifiés

On pourrait détecter ce genre de situations à l'exécution :

- *vérifications à l'exécution* : lorsqu'on a une fonction appliquée à un argument dont le résultat n'est pas spécifié
- *typage dynamique* : on pourrait spécifier le type d'argument attendu et le vérifier à l'exécution (par exemple `add` attend deux entiers)

Le problème de ce genre d'approche est

- on a besoin d'exécuter le programme
- on a besoin d'écrire des tests en espérant couvrir tous les cas

Première partie I

Typage

Typage

Le **typage** est une méthode

- *statique* : effectuée une fois, généralement pendant la compilation, ne nécessitant pas d'exécuter le programme,

Typage

Le **typage** est une méthode

- *statique* : effectuée une fois, généralement pendant la compilation, ne nécessitant pas d'exécuter le programme,
- *sûre* : permettant de s'assurer qu'il n'y aura jamais d'erreurs dues à des comportements non-spécifiés.

Type

Un **type** est une expression qu'on peut voir comme décrivant un ensemble de valeurs.

Type

Un **type** est une expression qu'on peut voir comme décrivant un ensemble de valeurs.

Dans notre cas, les types sont décrits par la grammaire suivante :

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

Type

Un **type** est une expression qu'on peut voir comme décrivant un ensemble de valeurs.

Dans notre cas, les types sont décrits par la grammaire suivante :

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

Type

Un **type** est une expression qu'on peut voir comme décrivant un ensemble de valeurs.

Dans notre cas, les types sont décrits par la grammaire suivante :

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

Type

Nous allons associer un type à tout programme.

Par exemple

- $\underline{2} + \underline{2}$ a le type

Type

Nous allons associer un type à tout programme.

Par exemple

- 2 + 2 a le type `int`,

Type

Nous allons associer un type à tout programme.

Par exemple

- $\underline{2}$ + $\underline{2}$ a le type `int`,
- `fun` $x \to$ `(`x `+ 1,true)` a le type

Type

Nous allons associer un type à tout programme.

Par exemple

- $\underline{2}$ + $\underline{2}$ a le type `int`,
- `fun` $x \to$ `(`x `+ 1,true)` a le type $\texttt{int} \Rightarrow (\texttt{int} \times \texttt{bool})$,

Type

Nous allons associer un type à tout programme.

Par exemple

- $\underline{2}$ + $\underline{2}$ a le type `int`,
- `fun` $x \to$ `(`x `+ 1,true)` a le type $\texttt{int} \Rightarrow (\texttt{int} \times \texttt{bool})$,
- `fst` $\underline{5}$

Type

Nous allons associer un type à tout programme.

Par exemple

- $\underline{2}$ + $\underline{2}$ a le type `int`,
- `fun` $x \to$ `(`x `+ 1,true)` a le type `int` $\Rightarrow$ `(int` $\times$ `bool)`,
- `fst` $\underline{5}$ n'a pas de type,

Type

Nous allons associer un type à tout programme.

Par exemple

- $\underline{2}$ + $\underline{2}$ a le type `int`,
- `fun` $x \to$ `(`x `+ 1,true)` a le type $\texttt{int} \Rightarrow (\texttt{int} \times \texttt{bool})$,
- `fst` $\underline{5}$ n'a pas de type,
- `fun` $x \to x$ a le type

Type

Nous allons associer un type à tout programme.

Par exemple

- $\underline{2}$ `+` $\underline{2}$ a le type `int`,
- `fun` $x \to$ `(`x `+ 1,true)` a le type `int` $\Rightarrow$ (`int` $\times$ `bool`),
- `fst` $\underline{5}$ n'a pas de type,
- `fun` $x \to x$ a le type $A \Rightarrow A$ pour tout type A,

Type

Nous allons associer un type à tout programme.

Par exemple

- $\underline{2}$ `+` $\underline{2}$ a le type `int`,
- `fun` $x \to$ `(`x `+ 1,true)` a le type `int` $\Rightarrow$ (`int` $\times$ `bool`),
- `fst` $\underline{5}$ n'a pas de type,
- `fun` $x \to x$ a le type $A \Rightarrow A$ pour tout type A,
- `fun` $x \to x$ `+` y a le type

Type

Nous allons associer un type à tout programme.

Par exemple

- $\underline{2}$ `+` $\underline{2}$ a le type `int`,
- `fun` $x \to$ `(`x `+ 1,true)` a le type `int` $\Rightarrow$ (`int` $\times$ `bool`),
- `fst` $\underline{5}$ n'a pas de type,
- `fun` $x \to x$ a le type $A \Rightarrow A$ pour tout type A,
- `fun` $x \to x$ `+` y a le type `int` $\Rightarrow$ `int` si on suppose que y a le type `int`.

Type

Par exemple, en OCaml, on a

```
# 2 + 2;;
- : int = 4
```

Type

Par exemple, en OCaml, on a

```
# 2 + 2;;
- : int = 4

# fun x -> (x + 1, true);;
- : int -> int * bool = <fun>
```

Type

Par exemple, en OCaml, on a

```
# 2 + 2;;
- : int = 4

# fun x -> (x + 1, true);;
- : int -> int * bool = <fun>

# fun x -> x;;
- : 'a -> 'a = <fun>
```

Type

Par exemple, en OCaml, on a

```
# 2 + 2;;
- : int = 4

# fun x -> (x + 1, true);;
- : int -> int * bool = <fun>

# fun x -> x;;
- : 'a -> 'a = <fun>

# 2 + true;;
Error: This expression has type bool but an expression was expected
       of type int
```

Associativité de la flèche

On a le type

$$\texttt{add} \quad : \quad \texttt{int} \Rightarrow \texttt{int} \Rightarrow \texttt{int}$$

Associativité de la flèche

On a le type

$$\texttt{add 1} \quad : \quad \texttt{int} \Rightarrow \texttt{int}$$

Associativité de la flèche

On a le type

$$\texttt{add 1} \quad : \quad \texttt{int} \Rightarrow \texttt{int}$$

Pour cette raison, les flèches sont associatives à droite :

$$\texttt{int} \Rightarrow \texttt{int} \Rightarrow \texttt{int} \quad = \quad \texttt{int} \Rightarrow (\texttt{int} \Rightarrow \texttt{int})$$

Schémas de types

Notons que nos types sont moins expressifs que ceux de OCaml :
pour tout type A, on peut donner le type

$$A \Rightarrow A$$

à l'identité, par exemple

$$\texttt{int} \Rightarrow \texttt{int} \qquad\qquad (\texttt{bool} \times \texttt{int}) \Rightarrow (\texttt{bool} \times \texttt{int})$$

Schémas de types

Notons que nos types sont moins expressifs que ceux de OCaml :
pour tout type A, on peut donner le type

$$A \Rightarrow A$$

à l'identité, par exemple

$$\texttt{int} \Rightarrow \texttt{int} \qquad\qquad (\texttt{bool} \times \texttt{int}) \Rightarrow (\texttt{bool} \times \texttt{int})$$

mais il n'y a pas moyen d'avoir un équivalent de

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

qui signifie que on a le type $A \Rightarrow A$ *pour tout* A.

Typage

Un **contexte**, ou *environnement de typage*, Γ est une liste

$$\Gamma \quad = \quad x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$$

de paires (x_i, A_i) où

- x_i est une variable,
- A_i est un type,

que l'on lit comme « supposons que x_i a le type A_i ».

Typage

Un **jugement de typage**

$$\Gamma \vdash t : A$$

est un triplet constitué de

- un environnement de typage Γ,
- un terme t,
- un type A,

que l'on lit comme « en supposant que les x_i on le type A_i, le terme t a le type A ».

Typage

Un **jugement de typage**

$$\Gamma \vdash t : A$$

est un triplet constitué de

- un environnement de typage Γ,
- un terme t,
- un type A,

que l'on lit comme « en supposant que les x_i on le type A_i, le terme t a le type A ».

Par exemple

$$y : \texttt{int} \vdash \texttt{fun}\ x \to x + y : \texttt{int} \Rightarrow \texttt{int}$$

Typage

Nous allons spécifier les jugements de typage **valides** par des règles de la forme

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \dots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma \vdash t : A}$$

i.e. valide = dérivable en utilisant les règles.

Typage

Nous allons spécifier les jugements de typage **valides** par des règles de la forme

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \dots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma \vdash t : A}$$

i.e. valide = dérivable en utilisant les règles.

Par exemple,

- $\vdash \underline{4} : \texttt{int}$ sera

Typage

Nous allons spécifier les jugements de typage **valides** par des règles de la forme

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \dots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma \vdash t : A}$$

i.e. valide = dérivable en utilisant les règles.

Par exemple,

- $\vdash \underline{4} : \texttt{int}$ sera valide,

Typage

Nous allons spécifier les jugements de typage **valides** par des règles de la forme

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \dots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma \vdash t : A}$$

i.e. valide = dérivable en utilisant les règles.

Par exemple,

- $\vdash \underline{4} : \texttt{int}$ sera valide,
- $\vdash \underline{4} : \texttt{bool}$

Typage

Nous allons spécifier les jugements de typage **valides** par des règles de la forme

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \dots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma \vdash t : A}$$

i.e. valide = dérivable en utilisant les règles.

Par exemple,

- $\vdash \underline{4} : \texttt{int}$ sera valide,
- $\vdash \underline{4} : \texttt{bool}$ ne sera pas valide,

Typage

Nous allons spécifier les jugements de typage **valides** par des règles de la forme

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \dots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma \vdash t : A}$$

i.e. valide = dérivable en utilisant les règles.

Par exemple,

- $\vdash \underline{4} : \texttt{int}$ sera valide,
- $\vdash \underline{4} : \texttt{bool}$ ne sera pas valide,
- $y : \texttt{int} \vdash \texttt{fun}\ x \to x + y : \texttt{int} \Rightarrow \texttt{int}$

Typage

Nous allons spécifier les jugements de typage **valides** par des règles de la forme

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \dots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma \vdash t : A}$$

i.e. valide = dérivable en utilisant les règles.

Par exemple,

- $\vdash \underline{4} : \texttt{int}$ sera valide,
- $\vdash \underline{4} : \texttt{bool}$ ne sera pas valide,
- $y : \texttt{int} \vdash \texttt{fun}\ x \to x + y : \texttt{int} \Rightarrow \texttt{int}$ sera valide,

Typage

Nous allons spécifier les jugements de typage **valides** par des règles de la forme

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \dots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma \vdash t : A}$$

i.e. valide = dérivable en utilisant les règles.

Par exemple,

- $\vdash \underline{4} : \texttt{int}$ sera valide,
- $\vdash \underline{4} : \texttt{bool}$ ne sera pas valide,
- $y : \texttt{int} \vdash \texttt{fun}\ x \to x + y : \texttt{int} \Rightarrow \texttt{int}$ sera valide,
- $y : \texttt{bool} \vdash \texttt{fun}\ x \to x + y : \texttt{int} \Rightarrow \texttt{int}$

Typage

Nous allons spécifier les jugements de typage **valides** par des règles de la forme

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \dots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma \vdash t : A}$$

i.e. valide = dérivable en utilisant les règles.

Par exemple,

- $\vdash \underline{4} : \texttt{int}$ sera valide,
- $\vdash \underline{4} : \texttt{bool}$ ne sera pas valide,
- $y : \texttt{int} \vdash \texttt{fun}\ x \to x + y : \texttt{int} \Rightarrow \texttt{int}$ sera valide,
- $y : \texttt{bool} \vdash \texttt{fun}\ x \to x + y : \texttt{int} \Rightarrow \texttt{int}$ ne sera pas valide,

Typage

Nous allons spécifier les jugements de typage **valides** par des règles de la forme

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \dots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma \vdash t : A}$$

i.e. valide = dérivable en utilisant les règles.

Par exemple,

- $\vdash \underline{4} : \texttt{int}$ sera valide,
- $\vdash \underline{4} : \texttt{bool}$ ne sera pas valide,
- $y : \texttt{int} \vdash \texttt{fun}\ x \to x + y : \texttt{int} \Rightarrow \texttt{int}$ sera valide,
- $y : \texttt{bool} \vdash \texttt{fun}\ x \to x + y : \texttt{int} \Rightarrow \texttt{int}$ ne sera pas valide,
- $\vdash \texttt{fun}\ x \to x + y : \texttt{int} \Rightarrow \texttt{int}$

Typage

Nous allons spécifier les jugements de typage **valides** par des règles de la forme

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \dots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma \vdash t : A}$$

i.e. valide = dérivable en utilisant les règles.

Par exemple,

- $\vdash \underline{4} : \texttt{int}$ sera valide,
- $\vdash \underline{4} : \texttt{bool}$ ne sera pas valide,
- $y : \texttt{int} \vdash \texttt{fun}\ x \to x + y : \texttt{int} \Rightarrow \texttt{int}$ sera valide,
- $y : \texttt{bool} \vdash \texttt{fun}\ x \to x + y : \texttt{int} \Rightarrow \texttt{int}$ ne sera pas valide,
- $\vdash \texttt{fun}\ x \to x + y : \texttt{int} \Rightarrow \texttt{int}$ ne sera pas valide.

Typage

Nous allons spécifier les jugements de typage **valides** par des règles de la forme

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \dots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma \vdash t : A}$$

i.e. valide = dérivable en utilisant les règles.

On dit que t *a le type* A lorsque

$$\vdash t : A$$

est valide.

Typage

Nous allons spécifier les jugements de typage **valides** par des règles de la forme

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \dots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma \vdash t : A}$$

i.e. valide = dérivable en utilisant les règles.

Il y a exactement une règle de typage par construction du langage.

Règles de typage

- entiers :

Règles de typage

- entiers :

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

Règles de typage

- entiers :

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

- booléens :

Règles de typage

- entiers :

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

- booléens :

$$\frac{}{\Gamma \vdash \underline{b} : \texttt{bool}} \ (\text{bool})$$

Règles de typage

- entiers :

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

- booléens :

$$\frac{}{\Gamma \vdash \underline{b} : \texttt{bool}} \text{(bool)}$$

- addition :

Règles de typage

- entiers :

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

- booléens :

$$\frac{}{\Gamma \vdash \underline{b} : \texttt{bool}} \ (\text{bool})$$

- addition :

$$\frac{}{\Gamma \vdash \texttt{add} : \texttt{int} \Rightarrow \texttt{int} \Rightarrow \texttt{int}} \ (\text{add})$$

Règles de typage

- entiers :

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

- booléens :

$$\frac{}{\Gamma \vdash \underline{b} : \texttt{bool}}\ (\text{bool})$$

- addition :

$$\frac{}{\Gamma \vdash \texttt{add} : \texttt{int} \Rightarrow \texttt{int} \Rightarrow \texttt{int}}\ (\text{add})$$

  variante :

$$\frac{}{\Gamma \vdash \texttt{add} : \texttt{int} \times \texttt{int} \Rightarrow \texttt{int}}\ (\text{add})$$

Règles de typage

- variables :

Règles de typage

- variables : si $(x : A) \in \Gamma$ alors

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

Règles de typage

- variables : si $(x : A) \in \Gamma$ alors

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

- fonctions :

Règles de typage

- variables : si $(x : A) \in \Gamma$ alors

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

- fonctions :

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

Règles de typage

- variables : si $(x : A) \in \Gamma$ alors

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

- fonctions :

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

- applications :

Règles de typage

- variables : si $(x : A) \in \Gamma$ alors

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

- fonctions :

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

- applications :

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

Règles de typage

- paires :

Règles de typage

- paires :

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B}{\Gamma \vdash (t, u) : A \times B}\ (\text{pair})$$

Règles de typage

- paires :

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B}{\Gamma \vdash (t, u) : A \times B}\ (\text{pair})$$

- projections :

Règles de typage

- paires :

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B}{\Gamma \vdash (t, u) : A \times B}\ (\text{pair})$$

- projections :

$$\frac{}{\Gamma \vdash \texttt{fst} : A \times B \Rightarrow A}\ (\text{fst}) \qquad \frac{}{\Gamma \vdash \texttt{snd} : A \times B \Rightarrow B}\ (\text{snd})$$

Exemple de typage

$\vdash$ `fun` $x \rightarrow$ `(add (`x`,1),true)` :

Exemple de typage

$$\vdash \texttt{fun}\ x \rightarrow\ (\texttt{add}\ (x,1),\texttt{true}) : \texttt{int} \Rightarrow (\texttt{int} \times \texttt{bool})$$

Exemple de typage

$$\frac{x : \texttt{int} \vdash (\texttt{add}\ (x,1),\texttt{true}) : (\texttt{int} \times \texttt{bool})}{\vdash \texttt{fun}\ x \rightarrow\ (\texttt{add}\ (x,1),\texttt{true}) : \texttt{int} \Rightarrow (\texttt{int} \times \texttt{bool})}\ (\text{fun})$$

Exemple de typage

$$\dfrac{\dfrac{x : \texttt{int} \vdash \texttt{add}\ (x,1) : \texttt{int} \qquad x : \texttt{int} \vdash \texttt{true} : \texttt{bool}}{x : \texttt{int} \vdash (\texttt{add}\ (x,1), \texttt{true}) : (\texttt{int} \times \texttt{bool})}\ (\text{pair})}{\vdash \texttt{fun}\ x \rightarrow (\texttt{add}\ (x,1), \texttt{true}) : \texttt{int} \Rightarrow (\texttt{int} \times \texttt{bool})}\ (\text{fun})$$

Exemple de typage

$$\dfrac{\dfrac{\dfrac{\vdots}{x : \texttt{int} \vdash \texttt{add}\ (x,1) : \texttt{int}}\ \ldots \qquad x : \texttt{int} \vdash \texttt{true} : \texttt{bool}}{x : \texttt{int} \vdash (\texttt{add}\ (x,1), \texttt{true}) : (\texttt{int} \times \texttt{bool})}\ (\text{pair})}{\vdash \texttt{fun}\ x \rightarrow (\texttt{add}\ (x,1), \texttt{true}) : \texttt{int} \Rightarrow (\texttt{int} \times \texttt{bool})}\ (\text{fun})$$

Exemple de typage

$$\dfrac{\dfrac{\dfrac{\vdots}{x : \texttt{int} \vdash \texttt{add}\ (x,1) : \texttt{int}} \cdots \quad \dfrac{}{x : \texttt{int} \vdash \texttt{true} : \texttt{bool}}\ (\text{bool})}{x : \texttt{int} \vdash (\texttt{add}\ (x,1), \texttt{true}) : (\texttt{int} \times \texttt{bool})}\ (\text{pair})}{\vdash \texttt{fun}\ x \rightarrow (\texttt{add}\ (x,1), \texttt{true}) : \texttt{int} \Rightarrow (\texttt{int} \times \texttt{bool})}\ (\text{fun})$$

Exemple de typage

$$x : \texttt{int} \vdash \texttt{add}\ (x, 1) : \texttt{int}$$

Exemple de typage

$$\frac{x : \texttt{int} \vdash \texttt{add} : \texttt{int} \times \texttt{int} \Rightarrow \texttt{int} \qquad x : \texttt{int} \vdash (x, 1) : \texttt{int} \times \texttt{int}}{x : \texttt{int} \vdash \texttt{add}\ (x, 1) : \texttt{int}}\ (\texttt{app})$$

Exemple de typage

$$\dfrac{\dfrac{}{x : \texttt{int} \vdash \texttt{add} : \texttt{int} \times \texttt{int} \Rightarrow \texttt{int}}\,(\text{add}) \qquad x : \texttt{int} \vdash (x, 1) : \texttt{int} \times \texttt{int}}{x : \texttt{int} \vdash \texttt{add}\ (x, 1) : \texttt{int}}\,(\text{app})$$

Exemple de typage

$$\dfrac{\dfrac{}{x : \texttt{int} \vdash \texttt{add} : \texttt{int} \times \texttt{int} \Rightarrow \texttt{int}}\,(\text{add}) \qquad \dfrac{\dfrac{x : \texttt{int} \vdash x : \texttt{int} \qquad x : \texttt{int} \vdash 1 : \texttt{int}}{x : \texttt{int} \vdash (x, 1) : \texttt{int} \times \texttt{int}}\,(\text{add})}{x : \texttt{int} \vdash \texttt{add}\ (x, 1) : \texttt{int}}\,(\text{app})$$

Exemple de typage

$$\dfrac{\dfrac{}{x : \texttt{int} \vdash \texttt{add} : \texttt{int} \times \texttt{int} \Rightarrow \texttt{int}}\,(\text{add}) \qquad \dfrac{\dfrac{}{x : \texttt{int} \vdash x : \texttt{int}}\,(\text{var}) \qquad x : \texttt{int} \vdash 1 : \texttt{int}}{x : \texttt{int} \vdash (x, 1) : \texttt{int} \times \texttt{int}}\,(\text{add})}{x : \texttt{int} \vdash \texttt{add}\ (x, 1) : \texttt{int}}\,(\text{app})$$

Exemple de typage

$$\dfrac{\dfrac{}{x : \texttt{int} \vdash \texttt{add} : \texttt{int} \times \texttt{int} \Rightarrow \texttt{int}}\,(\text{add}) \qquad \dfrac{\dfrac{}{x : \texttt{int} \vdash x : \texttt{int}}\,(\text{var}) \qquad \dfrac{}{x : \texttt{int} \vdash 1 : \texttt{int}}\,(\text{int})}{x : \texttt{int} \vdash (x, 1) : \texttt{int} \times \texttt{int}}\,(\text{add})}{x : \texttt{int} \vdash \texttt{add}\ (x, 1) : \texttt{int}}\,(\text{app})$$

Exemple de typage

$$\vdash (\texttt{fun}\ f \rightarrow f\ 0)\ (\texttt{fun}\ x \rightarrow x + 1) :$$

Exemple de typage

$$\vdash (\texttt{fun}\ f \rightarrow f\ 0)\ (\texttt{fun}\ x \rightarrow x + 1) : \texttt{int}$$

Exemple de typage

$$\frac{\vdash \texttt{fun}\ f \to f\ 0 : (\texttt{int} \Rightarrow \texttt{int}) \Rightarrow \texttt{int} \qquad \vdash \texttt{fun}\ x \to x + 1 : \texttt{int} \Rightarrow \texttt{int}}{\vdash (\texttt{fun}\ f \to f\ 0)\,(\texttt{fun}\ x \to x + 1) : \texttt{int}}\ ($$

Exemple de typage

$$\dfrac{\dfrac{f : \texttt{int} \Rightarrow \texttt{int} \vdash f\ 0 : \texttt{int}}{\vdash \texttt{fun}\ f \to f\ 0 : (\texttt{int} \Rightarrow \texttt{int}) \Rightarrow \texttt{int}}\ (\text{fun}) \qquad \vdash \texttt{fun}\ x \to x + 1 : \texttt{int} \Rightarrow \texttt{int}}{\vdash (\texttt{fun}\ f \to f\ 0)\ (\texttt{fun}\ x \to x + 1) : \texttt{int}}\ ($$

Exemple de typage

$$\dfrac{\dfrac{\dfrac{}{\Rightarrow \texttt{int} \vdash f : \texttt{int} \Rightarrow \texttt{int}} \qquad \dfrac{}{f : \texttt{int} \Rightarrow \texttt{int} \vdash 0 : \texttt{int}}\ (\text{app})}{f : \texttt{int} \Rightarrow \texttt{int} \vdash f\ 0 : \texttt{int}}\ (\text{fun})}{\vdash \texttt{fun}\ f \to f\ 0 : (\texttt{int} \Rightarrow \texttt{int}) \Rightarrow \texttt{int}} \qquad \dfrac{}{\vdash \texttt{fun}\ x \to x + 1 : \texttt{int} \Rightarrow \texttt{int}}\ ($$

$$\vdash (\texttt{fun}\ f \to f\ 0)\ (\texttt{fun}\ x \to x + 1) : \texttt{int}$$

Exemple de typage

$$\dfrac{\dfrac{\dfrac{}{\texttt{int} \Rightarrow \texttt{int} \vdash f : \texttt{int} \Rightarrow \texttt{int}}\,(\text{var}) \qquad f : \texttt{int} \Rightarrow \texttt{int} \vdash 0 : \texttt{int}}{f : \texttt{int} \Rightarrow \texttt{int} \vdash f\ 0 : \texttt{int}}\,(\text{app})}{\vdash \texttt{fun}\ f \to f\ 0 : (\texttt{int} \Rightarrow \texttt{int}) \Rightarrow \texttt{int}}\,(\text{fun}) \qquad \vdash \texttt{fun}\ x \to x + 1 : \texttt{int} \Rightarrow \texttt{int}}{\vdash (\texttt{fun}\ f \to f\ 0)\ (\texttt{fun}\ x \to x + 1) : \texttt{int}}\,($$

Exemple de typage

$$\dfrac{\dfrac{}{\texttt{int} \Rightarrow \texttt{int} \vdash f : \texttt{int} \Rightarrow \texttt{int}}\,(\text{var}) \qquad \dfrac{}{f : \texttt{int} \Rightarrow \texttt{int} \vdash 0 : \texttt{int}}\,(\text{int})}{\dfrac{f : \texttt{int} \Rightarrow \texttt{int} \vdash f\ 0 : \texttt{int}}{\dfrac{\vdash \texttt{fun}\ f \to f\ 0 : (\texttt{int} \Rightarrow \texttt{int}) \Rightarrow \texttt{int}}{\vdash (\texttt{fun}\ f \to f\ 0)\,(\texttt{fun}\ x \to x + 1) : \texttt{int}}\,(\text{fun})}\,(\text{app}) \qquad \vdash \texttt{fun}\ x \to x + 1 : \texttt{int} \Rightarrow \texttt{int}}\,($$

Exemple de typage

$$\dfrac{\dfrac{\dfrac{}{\Rightarrow \texttt{int} \vdash f : \texttt{int} \Rightarrow \texttt{int}}\,(\text{var}) \qquad \dfrac{}{f : \texttt{int} \Rightarrow \texttt{int} \vdash 0 : \texttt{int}}\,(\text{int})}{f : \texttt{int} \Rightarrow \texttt{int} \vdash f\ 0 : \texttt{int}}\,(\text{app})}{\vdash \texttt{fun}\ f \to f\ 0 : (\texttt{int} \Rightarrow \texttt{int}) \Rightarrow \texttt{int}}\,(\text{fun}) \qquad \dfrac{x : \texttt{int} \vdash x + 1 : \texttt{int}}{\vdash \texttt{fun}\ x \to x + 1 : \texttt{int} \Rightarrow \texttt{int}}\,(\text{f}$$

$$\vdash (\texttt{fun}\ f \to f\ 0)\ (\texttt{fun}\ x \to x + 1) : \texttt{int}$$

Exemple de typage

$$\dfrac{\dfrac{\dfrac{}{\Rightarrow \texttt{int} \vdash f : \texttt{int} \Rightarrow \texttt{int}}\,(\text{var}) \qquad \dfrac{}{f : \texttt{int} \Rightarrow \texttt{int} \vdash 0 : \texttt{int}}\,(\text{int})}{f : \texttt{int} \Rightarrow \texttt{int} \vdash f\ 0 : \texttt{int}}\,(\text{app})}{\vdash \texttt{fun}\ f \to f\ 0 : (\texttt{int} \Rightarrow \texttt{int}) \Rightarrow \texttt{int}}\,(\text{fun}) \qquad \dfrac{\dfrac{\vdots}{x : \texttt{int} \vdash x + 1 : \texttt{int}}}{\vdash \texttt{fun}\ x \to x + 1 : \texttt{int} \Rightarrow \texttt{int}}\,(\text{fu}$$

$$\vdash (\texttt{fun}\ f \to f\ 0)\,(\texttt{fun}\ x \to x + 1) : \texttt{int}$$

Occurrences multiples de variables

En OCaml, le type de

```
fun x -> fun x -> x
```

est

Occurrences multiples de variables

En OCaml, le type de

```
fun x -> fun x -> x
```

est `'a -> 'b -> 'b`.

Occurrences multiples de variables

En OCaml, le type de

```
fun x -> fun x -> x
```

est `'a -> 'b -> 'b`.

Ceci correspond à la dérivation de type suivante :

$$\vdash \texttt{fun}\ x \to \texttt{fun}\ x \to x : A \Rightarrow B \Rightarrow B$$

Occurrences multiples de variables

En OCaml, le type de

```
fun x -> fun x -> x
```

est `'a -> 'b -> 'b`.

Ceci correspond à la dérivation de type suivante :

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

Occurrences multiples de variables

En OCaml, le type de

```
fun x -> fun x -> x
```

est `'a -> 'b -> 'b`.

Ceci correspond à la dérivation de type suivante :

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

Occurrences multiples de variables

En OCaml, le type de

```
fun x -> fun x -> x
```

est `'a -> 'b -> 'b`.

Ceci correspond à la dérivation de type suivante :

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

Occurrences multiples de variables

En OCaml, le type de

```
fun x -> fun x -> x
```

est `'a -> 'b -> 'b`.

Ceci correspond à la dérivation de type suivante :

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

Le type de x dans un contexte Γ correspond toujours à l'occurrence la plus à droite.

Propriétés du typage

Nous allons nous intéresser aux propriétés suivantes du typage :

- *unicité* : le type d'un terme est-il unique ?

Propriétés du typage

Nous allons nous intéresser aux propriétés suivantes du typage :

- *unicité* : le type d'un terme est-il unique ?
- *canonicité* : dans le cas où il n'est pas unique, en a-t-on un « meilleur » ?

Propriétés du typage

Nous allons nous intéresser aux propriétés suivantes du typage :

- *unicité* : le type d'un terme est-il unique ?
- *canonicité* : dans le cas où il n'est pas unique, en a-t-on un « meilleur » ?
- *préservation par réduction* : le type est-il préservé par réduction ?

Propriétés du typage

Nous allons nous intéresser aux propriétés suivantes du typage :

- *unicité* : le type d'un terme est-il unique ?
- *canonicité* : dans le cas où il n'est pas unique, en a-t-on un « meilleur » ?
- *préservation par réduction* : le type est-il préservé par réduction ?
- *progrès* : le typage empêche-t-il les situations non-spécifiées au cours de l'exécution ?

Propriétés du typage

Nous allons nous intéresser aux propriétés suivantes du typage :

- *unicité* : le type d'un terme est-il unique ?
- *canonicité* : dans le cas où il n'est pas unique, en a-t-on un « meilleur » ?
- *préservation par réduction* : le type est-il préservé par réduction ?
- *progrès* : le typage empêche-t-il les situations non-spécifiées au cours de l'exécution ?

D'un point de vue algorithmique, nous allons regarder les problèmes suivants :

Propriétés du typage

Nous allons nous intéresser aux propriétés suivantes du typage :

- *unicité* : le type d'un terme est-il unique ?
- *canonicité* : dans le cas où il n'est pas unique, en a-t-on un « meilleur » ?
- *préservation par réduction* : le type est-il préservé par réduction ?
- *progrès* : le typage empêche-t-il les situations non-spécifiées au cours de l'exécution ?

D'un point de vue algorithmique, nous allons regarder les problèmes suivants :

- *vérification* : on cherche à vérifier qu'un programme t a un type A,

Propriétés du typage

Nous allons nous intéresser aux propriétés suivantes du typage :

- *unicité* : le type d'un terme est-il unique ?
- *canonicité* : dans le cas où il n'est pas unique, en a-t-on un « meilleur » ?
- *préservation par réduction* : le type est-il préservé par réduction ?
- *progrès* : le typage empêche-t-il les situations non-spécifiées au cours de l'exécution ?

D'un point de vue algorithmique, nous allons regarder les problèmes suivants :

- *vérification* : on cherche à vérifier qu'un programme t a un type A,
- *inférence* : on cherche à déterminer un type pour un programme donné.

Programmes non typables

Il existe des programmes non typables. Par exemple :

```
add true
```

Programmes non typables

Il existe des programmes non typables. Par exemple :

$$\frac{\vdash \texttt{add} : A \Rightarrow B \qquad\qquad \vdash \texttt{true} : A}{\vdash \texttt{add true} : B}\ (\texttt{app})$$

Programmes non typables

Il existe des programmes non typables. Par exemple :

$$\dfrac{\dfrac{}{\vdash \texttt{add} : \texttt{int} \Rightarrow \texttt{int} \Rightarrow \texttt{int}}\,(\texttt{add}) \qquad \vdash \texttt{true} : \texttt{int}}{\vdash \texttt{add true} : \texttt{int} \Rightarrow \texttt{int}}\,(\texttt{app})$$

Programmes non typables

Il existe des programmes non typables. Par exemple :

$$\dfrac{\dfrac{}{\vdash \texttt{add} : \texttt{int} \Rightarrow \texttt{int} \Rightarrow \texttt{int}}\,(\text{add}) \qquad \dfrac{}{\vdash \texttt{true} : \texttt{bool}}\,(\text{bool})}{\vdash \texttt{add true} : \texttt{int} \Rightarrow \texttt{int}}\,(\text{app})$$

Programmes non typables

D'autres exemples de programme non typables :

- `fun` $x \to y$ (dans l'environnement vide)

Programmes non typables

D'autres exemples de programme non typables :

- `fun` $x \to y$ (dans l'environnement vide)
- `fun` $f \to$ `(`f `1,`f `true)`

Programmes non typables

D'autres exemples de programme non typables :

- `fun` $x \to y$ (dans l'environnement vide)
- `fun` $f \to$ `(`f `1,`f `true)`
- `fun` $f \to f\ f$

Unicité du typage

Le typage n'est pas unique :

Unicité du typage

Le typage n'est pas unique :

$$\dfrac{\dfrac{}{x : \texttt{int} \vdash x : \texttt{int}}\ (\text{ax})}{\vdash \texttt{fun}\ x \rightarrow x : \texttt{int} \Rightarrow \texttt{int}}\ (\text{fun})$$

$$\dfrac{\dfrac{}{x : \texttt{bool} \vdash x : \texttt{bool}}\ (\text{ax})}{\vdash \texttt{fun}\ x \rightarrow x : \texttt{bool} \Rightarrow \texttt{bool}}\ (\text{fun})$$

Unicité du typage

Le typage n'est pas unique :

$$\dfrac{\dfrac{}{x : \texttt{int} \vdash x : \texttt{int}}\,(\text{ax})}{\vdash \texttt{fun}\ x \to x : \texttt{int} \Rightarrow \texttt{int}}\,(\text{fun}) \qquad \dfrac{\dfrac{}{x : \texttt{bool} \vdash x : \texttt{bool}}\,(\text{ax})}{\vdash \texttt{fun}\ x \to x : \texttt{bool} \Rightarrow \texttt{bool}}\,(\text{fun})$$

(et de façon plus générale $A \Rightarrow A$ pour tout type A)

Unicité du typage

Le typage n'est pas unique :

$$\dfrac{\dfrac{}{x : \texttt{int} \vdash x : \texttt{int}}\,(\text{ax})}{\vdash \texttt{fun}\ x \to x : \texttt{int} \Rightarrow \texttt{int}}\,(\text{fun}) \qquad \dfrac{\dfrac{}{x : \texttt{bool} \vdash x : \texttt{bool}}\,(\text{ax})}{\vdash \texttt{fun}\ x \to x : \texttt{bool} \Rightarrow \texttt{bool}}\,(\text{fun})$$

(et de façon plus générale $A \Rightarrow A$ pour tout type A)

De même,

$$\dfrac{}{\Gamma \vdash \texttt{fst} : A \times B \Rightarrow A}\,(\text{fst}) \qquad \dfrac{}{\Gamma \vdash \texttt{snd} : A \times B \Rightarrow B}\,(\text{snd})$$

Unicité du typage

Peut-on modifier la syntaxe pour avoir unicité du typage ?

Unicité du typage

Dans l'exemple de l'identité le coupable est la règle

Unicité du typage

Dans l'exemple de l'identité le coupable est la règle

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

on ne sait pas quel est le type A pour x.

Unicité du typage

Dans l'exemple de l'identité le coupable est la règle

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

on ne sait pas quel est le type A pour x.

Ceci suggère de changer la syntaxe :

$$t \quad ::= \quad \ldots \quad | \quad \texttt{fun}\ x\ \texttt{->}\ t \quad | \quad \ldots$$

Unicité du typage

Dans l'exemple de l'identité le coupable est la règle

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

on ne sait pas quel est le type A pour x.

Ceci suggère de changer la syntaxe :

$$t \quad ::= \quad \ldots \quad | \quad \texttt{fun}\ (x : A) \to t \quad | \quad \ldots$$

Unicité du typage

Dans l'exemple de l'identité le coupable est la règle

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

on ne sait pas quel est le type A pour x.

Ceci suggère de changer la syntaxe :

$$t \quad ::= \quad \ldots \quad | \quad \texttt{fun}\ (x : A)\ \to\ t \quad | \quad \ldots$$

et la règle correspondante :

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

Unicité du typage

De même, pour `fst` :

$$\frac{}{\Gamma \vdash \texttt{fst} : A \times B \Rightarrow A}\ (\text{fst})$$

Unicité du typage

De même, pour `fst` :

$$\frac{}{\Gamma \vdash \texttt{fst}_{A,B} : A \times B \Rightarrow A}\ (\text{fst})$$

Unicité du typage

De même, pour `fst` :

$$\frac{}{\Gamma \vdash \texttt{fst}_{A,B} : A \times B \Rightarrow A}\ (\text{fst})$$

Une autre possibilité est de modifier la syntaxe pour imposer à `fst` d'avoir un argument :

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \texttt{fst}(t) : A}\ (\text{fst})$$

Mini-ML avec informations de type

On appelle **Mini-ML avec informations de type** la variante obtenue dont la grammaire est

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

Unicité du typage

Théorème
Dans cette variante de mini-ML, un programme t admet au plus un type.

Démonstration.
Étant donnés Γ et t, on montre qu'il y a un plus un type A tel que $\Gamma \vdash t : A$ par induction sur t. Note : pour chaque forme de t il y a exactement une règle de typage associée.

Unicité du typage

Théorème
Dans cette variante de mini-ML, un programme t admet au plus un type.

Démonstration.
Étant donnés Γ et t, on montre qu'il y a un plus un type A tel que $\Gamma \vdash t : A$ par induction sur t. Note : pour chaque forme de t il y a exactement une règle de typage associée.

- Si $t = \underline{n}$,

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

Unicité du typage

Théorème
Dans cette variante de mini-ML, un programme t admet au plus un type.

Démonstration.
Étant donnés Γ et t, on montre qu'il y a un plus un type A tel que $\Gamma \vdash t : A$ par induction sur t. Note : pour chaque forme de t il y a exactement une règle de typage associée.

- Si $t = x$,

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

ou $(x : A)$ apparaît dans Γ (et c'est l'occurrence la plus à droite de x)

Unicité du typage

Théorème
Dans cette variante de mini-ML, un programme t admet au plus un type.

Démonstration.
Étant donnés Γ et t, on montre qu'il y a un plus un type A tel que $\Gamma \vdash t : A$ par induction sur t. Note : pour chaque forme de t il y a exactement une règle de typage associée.

- Si $t = \texttt{fun}\ (x : A)\ \to\ u$,

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

Unicité du typage

Théorème
Dans cette variante de mini-ML, un programme t admet au plus un type.

Démonstration.
Étant donnés Γ et t, on montre qu'il y a un plus un type A tel que $\Gamma \vdash t : A$ par induction sur t. Note : pour chaque forme de t il y a exactement une règle de typage associée.

- Si $t = u\ v$,

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

Unicité du typage

Théorème
Dans cette variante de mini-ML, un programme t admet au plus un type.

Démonstration.
Étant donnés Γ et t, on montre qu'il y a un plus un type A tel que $\Gamma \vdash t : A$ par induction sur t. Note : pour chaque forme de t il y a exactement une règle de typage associée.

- Les autres cas sont similaires. □

Inférence de type

On remarque la preuve est constructive : étant donné un environnement et un programme on peut construire explicitement le type, i.e. faire de l'**inférence de type**.

Inférence de type

On remarque la preuve est constructive : étant donné un environnement et un programme on peut construire explicitement le type, i.e. faire de l'**inférence de type**.

$\mathit{infer}(\Gamma, t)$:

Inférence de type

On remarque la preuve est constructive : étant donné un environnement et un programme on peut construire explicitement le type, i.e. faire de l'**inférence de type**.

$\mathit{infer}(\Gamma, t)$:

- si $t = \underline{n}$, le type est

Inférence de type

On remarque la preuve est constructive : étant donné un environnement et un programme on peut construire explicitement le type, i.e. faire de l'**inférence de type**.

$\mathit{infer}(\Gamma, t)$:

- si $t = \underline{n}$, le type est `int`,

Inférence de type

On remarque la preuve est constructive : étant donné un environnement et un programme on peut construire explicitement le type, i.e. faire de l'**inférence de type**.

$\mathit{infer}(\Gamma, t)$:

- si $t = \underline{n}$, le type est `int`,
- si $t = x$, le type est

Inférence de type

On remarque la preuve est constructive : étant donné un environnement et un programme on peut construire explicitement le type, i.e. faire de l'**inférence de type**.

$\mathit{infer}(\Gamma, t)$:

- si $t = \underline{n}$, le type est `int`,
- si $t = x$, le type est celui de l'occurrence la plus à droite de x dans Γ (et non typable s'il n'y a pas de telle occurrence),

Inférence de type

On remarque la preuve est constructive : étant donné un environnement et un programme on peut construire explicitement le type, i.e. faire de l'**inférence de type**.

$\mathit{infer}(\Gamma, t)$:

- si $t = \underline{n}$, le type est `int`,
- si $t = x$, le type est celui de l'occurrence la plus à droite de x dans Γ (et non typable s'il n'y a pas de telle occurrence),
- si $t = \texttt{fun}\ (x : A) \to u$, le type est

Inférence de type

On remarque la preuve est constructive : étant donné un environnement et un programme on peut construire explicitement le type, i.e. faire de l'**inférence de type**.

$\mathit{infer}(\Gamma, t)$:

- si $t = \underline{n}$, le type est `int`,
- si $t = x$, le type est celui de l'occurrence la plus à droite de x dans Γ (et non typable s'il n'y a pas de telle occurrence),
- si $t = \texttt{fun}\ (x : A)\ \to\ u$, le type est $A \Rightarrow B$ où $B = \mathit{infer}((\Gamma, x : A), u)$,

Inférence de type

On remarque la preuve est constructive : étant donné un environnement et un programme on peut construire explicitement le type, i.e. faire de l'**inférence de type**.

$\mathit{infer}(\Gamma, t)$:

- si $t = \underline{n}$, le type est `int`,
- si $t = x$, le type est celui de l'occurrence la plus à droite de x dans Γ (et non typable s'il n'y a pas de telle occurrence),
- si $t = \texttt{fun}\ (x : A)\ \rightarrow\ u$, le type est $A \Rightarrow B$ où $B = \mathit{infer}((\Gamma, x : A), u)$,
- si $t = u\ v$,

Inférence de type

On remarque la preuve est constructive : étant donné un environnement et un programme on peut construire explicitement le type, i.e. faire de l'**inférence de type**.

$\mathit{infer}(\Gamma, t)$:

- si $t = \underline{n}$, le type est `int`,
- si $t = x$, le type est celui de l'occurrence la plus à droite de x dans Γ (et non typable s'il n'y a pas de telle occurrence),
- si $t = \texttt{fun}\ (x : A)\ \rightarrow\ u$, le type est $A \Rightarrow B$ où $B = \mathit{infer}((\Gamma, x : A), u)$,
- si $t = u\ v$, c'est typable ssi $\mathit{infer}(\Gamma, u)$ est de la forme $A \Rightarrow B$ avec $A = \mathit{infer}(\Gamma, v)$ et ce type est alors B,

Inférence de type

On remarque la preuve est constructive : étant donné un environnement et un programme on peut construire explicitement le type, i.e. faire de l'**inférence de type**.

$\mathit{infer}(\Gamma, t)$:

- si $t = \underline{n}$, le type est `int`,
- si $t = x$, le type est celui de l'occurrence la plus à droite de x dans Γ (et non typable s'il n'y a pas de telle occurrence),
- si $t = \texttt{fun}\ (x : A)\ \rightarrow\ u$, le type est $A \Rightarrow B$ où $B = \mathit{infer}((\Gamma, x : A), u)$,
- si $t = u\ v$, c'est typable ssi $\mathit{infer}(\Gamma, u)$ est de la forme $A \Rightarrow B$ avec $A = \mathit{infer}(\Gamma, v)$ et ce type est alors B,
- etc.

Sûreté

On cherche maintenant à montrer que le fait d'être typable pour un programme assure l'absence d'erreurs à l'exécution...

Sûreté

On cherche maintenant à montrer que le fait d'être typable pour un programme assure l'absence d'erreurs à l'exécution...

...ce qui nécessite d'abord de formaliser l'exécution des programmes (c'est-à-dire la sémantique à grands et à petits pas).

Sûreté

On cherche maintenant à montrer que le fait d'être typable pour un programme assure l'absence d'erreurs à l'exécution...

...ce qui nécessite d'abord de formaliser l'exécution des programmes (c'est-à-dire la sémantique à grands et à petits pas).

On revient à mini-ML sans indications de types, mais on pourrait sans problème adapter ce qui suit.

Deuxième partie II

Évaluation

Sémantiques opérationnelles

Le langage étant assez différent de IMP, nous devons à nouveau définir les sémantiques à grands et à petits pas.

Commençons par la première, c'est-à-dire l'évaluation.

Valeurs

L'évaluation d'un programme va produire une **valeur**, c'est-à-dire un programme que l'on considère comme un résultat acceptable pour une évaluation.

On rappelle que les programmes sont

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

les **valeurs** sont

$$
v ::=
$$

Valeurs

L'évaluation d'un programme va produire une **valeur**, c'est-à-dire un programme que l'on considère comme un résultat acceptable pour une évaluation.

On rappelle que les programmes sont

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

les **valeurs** sont

$$\begin{array}{rcl} v & ::= & \underline{n} \end{array}$$

Valeurs

L'évaluation d'un programme va produire une **valeur**, c'est-à-dire un programme que l'on considère comme un résultat acceptable pour une évaluation.

On rappelle que les programmes sont

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

les **valeurs** sont

$$\begin{array}{rcl} v & ::= & \underline{n} \mid \underline{b} \end{array}$$

Valeurs

L'évaluation d'un programme va produire une **valeur**, c'est-à-dire un programme que l'on considère comme un résultat acceptable pour une évaluation.

On rappelle que les programmes sont

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

les **valeurs** sont

$$\begin{array}{rcl} v & ::= & \underline{n} \mid \underline{b} \mid \texttt{add} \end{array}$$

Valeurs

L'évaluation d'un programme va produire une **valeur**, c'est-à-dire un programme que l'on considère comme un résultat acceptable pour une évaluation.

On rappelle que les programmes sont

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

les **valeurs** sont

$$v ::= \underline{n} \mid \underline{b} \mid \texttt{add} \mid \texttt{add}\ \underline{n}$$

Valeurs

L'évaluation d'un programme va produire une **valeur**, c'est-à-dire un programme que l'on considère comme un résultat acceptable pour une évaluation.

On rappelle que les programmes sont

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

les **valeurs** sont

$$
\begin{array}{rcl}
v & ::= & \underline{n} \mid \underline{b} \mid \texttt{add} \mid \texttt{add}\ \underline{n} \\
& \mid & \texttt{fun}\ x \to t
\end{array}
$$

Valeurs

L'évaluation d'un programme va produire une **valeur**, c'est-à-dire un programme que l'on considère comme un résultat acceptable pour une évaluation.

On rappelle que les programmes sont

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

les **valeurs** sont

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

Valeurs

L'évaluation d'un programme va produire une **valeur**, c'est-à-dire un programme que l'on considère comme un résultat acceptable pour une évaluation.

On rappelle que les programmes sont

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

les **valeurs** sont

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

Valeurs

L'évaluation d'un programme va produire une **valeur**, c'est-à-dire un programme que l'on considère comme un résultat acceptable pour une évaluation.

On rappelle que les programmes sont

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

les **valeurs** sont

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

Valeurs

Notons que le corps d'une fonction ne doit pas nécessairement être une valeur.

De façon similaire, en OCaml

```
let f = fun x -> print_string "A"; 3 + 2;;
```

ne provoque pas d'affichage de `A`

Évaluation

On définit la relation d'**évaluation**

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

entre les termes t et les valeurs v par les règles suivantes.

Note : on n'a pas besoin d'un état ici.

Évaluation

- entiers :

Évaluation

- entiers :

$$\frac{}{\underline{n} \longrightarrow\!\!\!\!\!\rightarrow \underline{n}}\ (\mathsf{int})$$

Évaluation

- entiers :

$$\frac{}{\underline{n} \longrightarrow\!\!\!\!\!\rightarrow \underline{n}}\ (\mathsf{int})$$

- booléens :

Évaluation

- entiers :

$$\frac{}{\underline{n} \longrightarrow\!\!\!\!\!\rightarrow \underline{n}}\ (\mathsf{int})$$

- booléens :

$$\frac{}{\underline{b} \longrightarrow\!\!\!\!\!\rightarrow \underline{b}}\ (\mathsf{bool})$$

Évaluation

- entiers :

$$\frac{}{\underline{n} \longrightarrow\!\!\!\!\!\rightarrow \underline{n}}\ (\mathsf{int})$$

- booléens :

$$\frac{}{\underline{b} \longrightarrow\!\!\!\!\!\rightarrow \underline{b}}\ (\mathsf{bool})$$

- addition :

Évaluation

- entiers :

$$\frac{}{\underline{n} \longrightarrow\!\!\!\!\!\rightarrow \underline{n}}\ (\mathsf{int})$$

- booléens :

$$\frac{}{\underline{b} \longrightarrow\!\!\!\!\!\rightarrow \underline{b}}\ (\mathsf{bool})$$

- addition :

$$\frac{t \longrightarrow\!\!\!\!\!\rightarrow \mathsf{add} \qquad u \longrightarrow\!\!\!\!\!\rightarrow \underline{m} \qquad v \longrightarrow\!\!\!\!\!\rightarrow \underline{n}}{t\ u\ v \longrightarrow\!\!\!\!\!\rightarrow \underline{m+n}}\ (\mathsf{sum})$$

Évaluation

- entiers :

$$\frac{}{\underline{n} \longrightarrow\!\!\!\!\!\rightarrow \underline{n}}\ (\mathsf{int})$$

- booléens :

$$\frac{}{\underline{b} \longrightarrow\!\!\!\!\!\rightarrow \underline{b}}\ (\mathsf{bool})$$

- addition :

$$\frac{}{\mathtt{add} \longrightarrow\!\!\!\!\!\rightarrow \mathtt{add}}$$

Évaluation

- entiers :

$$\frac{}{\underline{n} \longrightarrow\!\!\!\!\!\rightarrow \underline{n}}\ (\mathrm{int})$$

- booléens :

$$\frac{}{\underline{b} \longrightarrow\!\!\!\!\!\rightarrow \underline{b}}\ (\mathrm{bool})$$

- addition :

$$\frac{}{\mathtt{add} \longrightarrow\!\!\!\!\!\rightarrow \mathtt{add}} \qquad \frac{t \longrightarrow\!\!\!\!\!\rightarrow \mathtt{add} \qquad u \longrightarrow\!\!\!\!\!\rightarrow \underline{m}}{t\ u \longrightarrow\!\!\!\!\!\rightarrow \mathtt{add}\ \underline{m}}$$

Évaluation

- entiers :

$$\frac{}{\underline{n} \longrightarrow\!\!\!\!\!\rightarrow \underline{n}}\ (\mathrm{int})$$

- booléens :

$$\frac{}{\underline{b} \longrightarrow\!\!\!\!\!\rightarrow \underline{b}}\ (\mathrm{bool})$$

- addition :

$$\frac{}{\mathrm{add} \longrightarrow\!\!\!\!\!\rightarrow \mathrm{add}} \qquad \frac{t \longrightarrow\!\!\!\!\!\rightarrow \mathrm{add} \qquad u \longrightarrow\!\!\!\!\!\rightarrow \underline{m}}{t\ u \longrightarrow\!\!\!\!\!\rightarrow \mathrm{add}\ \underline{m}} \qquad \frac{t \longrightarrow\!\!\!\!\!\rightarrow \mathrm{add}\ \underline{m} \qquad u \longrightarrow\!\!\!\!\!\rightarrow \underline{n}}{t\ u \longrightarrow\!\!\!\!\!\rightarrow \underline{m+n}}$$

Évaluation

- fonctions :

Évaluation

- fonctions :

$$\frac{}{(\texttt{fun}\ x \to t) \longrightarrow\!\!\!\!\!\rightarrow (\texttt{fun}\ x \to t)}\ (\text{fun})$$

Évaluation

- fonctions :

$$\frac{}{(\texttt{fun}\ x\ \to\ t) \longrightarrow\!\!\!\!\!\rightarrow (\texttt{fun}\ x\ \to\ t)}\ (\text{fun})$$

- application :

Par exemple,

$$(\texttt{fun}\ x\ \to\ x\ +\ x)\,(3\ +\ 2) \longrightarrow\!\!\!\!\!\rightarrow 10$$

Évaluation

- fonctions :

$$\frac{}{(\texttt{fun}\ x\ \to\ t) \longrightarrow\!\!\!\!\!\rightarrow (\texttt{fun}\ x\ \to\ t)}\ (\text{fun})$$

- application :

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

où $t'[x \mapsto u']$ est le terme t' où on a remplacé les variables x par u'. Par exemple,

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

Évaluation

- fonctions :

$$\frac{}{(\texttt{fun}\ x\ \to\ t) \longrightarrow\!\!\!\!\!\rightarrow (\texttt{fun}\ x\ \to\ t)}\ (\text{fun})$$

- application :

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

où $t'[x \mapsto u']$ est le terme t' où on a remplacé les variables x par u'. Par exemple,

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

Évaluation

- paires :

Évaluation

- paires :

$$\frac{t \longrightarrow\!\!\!\!\!\rightarrow t' \qquad u \longrightarrow\!\!\!\!\!\rightarrow u'}{(t\,,u) \longrightarrow\!\!\!\!\!\rightarrow (t'\,,u')}\ (\text{pair})$$

Évaluation

- paires :

$$\frac{t \longrightarrow\!\!\!\!\!\rightarrow t' \qquad u \longrightarrow\!\!\!\!\!\rightarrow u'}{(t\,,u) \longrightarrow\!\!\!\!\!\rightarrow (t'\,,u')}\ (\text{pair})$$

- première projection :

Évaluation

- paires :

$$\frac{t \longrightarrow\!\!\!\!\!\rightarrow t' \qquad u \longrightarrow\!\!\!\!\!\rightarrow u'}{(t, u) \longrightarrow\!\!\!\!\!\rightarrow (t', u')}\ (\text{pair})$$

- première projection :

$$\frac{t \longrightarrow\!\!\!\!\!\rightarrow \texttt{fst} \qquad u \longrightarrow\!\!\!\!\!\rightarrow (v_1, v_2)}{t\ u \longrightarrow\!\!\!\!\!\rightarrow v_1}\ (\text{fstp})$$

Évaluation

- paires :

$$\frac{t \longrightarrow\!\!\!\!\!\rightarrow t' \qquad u \longrightarrow\!\!\!\!\!\rightarrow u'}{(t, u) \longrightarrow\!\!\!\!\!\rightarrow (t', u')}\ (\text{pair})$$

- première projection :

$$\frac{}{\texttt{fst} \longrightarrow\!\!\!\!\!\rightarrow \texttt{fst}}\ (\text{fst}) \qquad \frac{t \longrightarrow\!\!\!\!\!\rightarrow \texttt{fst} \qquad u \longrightarrow\!\!\!\!\!\rightarrow (v_1, v_2)}{t\ u \longrightarrow\!\!\!\!\!\rightarrow v_1}\ (\text{fstp})$$

Évaluation

- paires :

$$\frac{t \longrightarrow\!\!\!\!\!\rightarrow t' \qquad u \longrightarrow\!\!\!\!\!\rightarrow u'}{(t, u) \longrightarrow\!\!\!\!\!\rightarrow (t', u')}\ (\text{pair})$$

- première projection :

$$\frac{}{\texttt{fst} \longrightarrow\!\!\!\!\!\rightarrow \texttt{fst}}\ (\text{fst}) \qquad \frac{t \longrightarrow\!\!\!\!\!\rightarrow \texttt{fst} \qquad u \longrightarrow\!\!\!\!\!\rightarrow (v_1, v_2)}{t\ u \longrightarrow\!\!\!\!\!\rightarrow v_1}\ (\text{fstp})$$

- seconde projection : similaire

Déterminisme

Théorème
L'évaluation est déterministe : si $t \longrightarrow\!\!\!\!\!\rightarrow v$ *et* $t \longrightarrow\!\!\!\!\!\rightarrow v'$ *alors* $v = v'$.

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

Déterminisme

Théorème
L'évaluation est déterministe : si $t \longrightarrow\!\!\!\!\!\rightarrow v$ *et* $t \longrightarrow\!\!\!\!\!\rightarrow v'$ *alors* $v = v'$.

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

L'évaluation peut donc être vue comme une fonction partielle qui à un terme associe sa valeur.

L'évaluation n'est pas totale

La fonction d'évaluation n'est *pas* totale.

Par exemple les, termes suivants ne s'évaluent pas en une valeur :

L'évaluation n'est pas totale

La fonction d'évaluation n'est *pas* totale.

Par exemple les, termes suivants ne s'évaluent pas en une valeur :

- `true` $\underline{5}$

L'évaluation n'est pas totale

La fonction d'évaluation n'est *pas* totale.

Par exemple les, termes suivants ne s'évaluent pas en une valeur :

- `true` $\underline{5}$
- `add false`

L'évaluation n'est pas totale

La fonction d'évaluation n'est *pas* totale.

Par exemple les, termes suivants ne s'évaluent pas en une valeur :

- $\texttt{true}\ \underline{5}$
- $\texttt{add}\ \texttt{false}$
- $(\texttt{fun}\ f \to f\ f)\,(\texttt{fun}\ f \to f\ f)$

La substitution

Pour définir l'évaluation, nous avons eu besoin de l'opération

$$t[x \mapsto u]$$

qui représente « le terme t où la variable x a été remplacée par u ».

Cette définition est un peu rapide. Pourquoi ?

Variables liées

Dans une expression de la forme

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

on dit que x est **liée** dans t.

Variables liées

Dans une expression de la forme

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

on dit que x est **liée** dans t.

Cela signifie qu'on pourrait changer le nom de la variable sans changer la signification du terme :

$$\texttt{fun}\ x \to x + x \quad \overset{\alpha}{=} \quad \texttt{fun}\ y \to y + y$$

Variables liées

Dans une expression de la forme

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

on dit que x est **liée** dans t.

Cela signifie qu'on pourrait changer le nom de la variable sans changer la signification du terme :

$$\texttt{fun}\ x \to x + x \quad \overset{\alpha}{=} \quad \texttt{fun}\ y \to y + y$$

cette relation d'équivalence s'appelle l'**α-conversion**

Variables liées

On retrouve couramment ce phénomène en mathématiques :

$$\lim_{x \to \infty} \frac{y}{x} \qquad \int_0^1 tx \, \mathrm{d}t \qquad \sum_{i=0}^{n} x^i$$

Quelles sont les variables liées ?

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Si on remplace y par n'importe quelle expression (par ex $t = \sin(\sqrt{2})$), on a

$$f(t) =$$

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Si on remplace y par n'importe quelle expression (par ex $t = \sin(\sqrt{2})$), on a

$$f(t) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto t] =$$

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Si on remplace y par n'importe quelle expression (par ex $t = \sin(\sqrt{2})$), on a

$$f(t) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto t] = \lim_{x \to \infty} \frac{t}{x} =$$

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Si on remplace y par n'importe quelle expression (par ex $t = \sin(\sqrt{2})$), on a

$$f(t) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto t] = \lim_{x \to \infty} \frac{t}{x} = 0$$

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Si on remplace y par n'importe quelle expression (par ex $t = \sin(\sqrt{2})$), on a

$$f(t) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto t] = \lim_{x \to \infty} \frac{t}{x} = 0$$

Mais quid du cas où on remplace par x ?

$$f(x) =$$

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Si on remplace y par n'importe quelle expression (par ex $t = \sin(\sqrt{2})$), on a

$$f(t) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto t] = \lim_{x \to \infty} \frac{t}{x} = 0$$

Mais quid du cas où on remplace par x ?

$$f(x) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto x] =$$

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Si on remplace y par n'importe quelle expression (par ex $t = \sin(\sqrt{2})$), on a

$$f(t) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto t] = \lim_{x \to \infty} \frac{t}{x} = 0$$

Mais quid du cas où on remplace par x ?

$$f(x) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto x] = \lim_{x \to \infty} \frac{x}{x} =$$

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Si on remplace y par n'importe quelle expression (par ex $t = \sin(\sqrt{2})$), on a

$$f(t) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto t] = \lim_{x \to \infty} \frac{t}{x} = 0$$

Mais quid du cas où on remplace par x ?

$$f(x) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto x] = \lim_{x \to \infty} \frac{x}{x} = \lim_{x \to \infty} 1 =$$

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Si on remplace y par n'importe quelle expression (par ex $t = \sin(\sqrt{2})$), on a

$$f(t) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto t] = \lim_{x \to \infty} \frac{t}{x} = 0$$

Mais quid du cas où on remplace par x ?

$$f(x) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto x] = \lim_{x \to \infty} \frac{x}{x} = \lim_{x \to \infty} 1 = 1$$

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Si on remplace y par n'importe quelle expression (par ex $t = \sin(\sqrt{2})$), on a

$$f(t) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto t] = \lim_{x \to \infty} \frac{t}{x} = 0$$

Mais quid du cas où on remplace par x ?

$$f(x) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto x] = \lim_{x \to \infty} \frac{x}{x} = \lim_{x \to \infty} 1 = 1$$

Il faut faire l'hypothèse (implicite) que les variables liées sont **fraîches**, c'est-à-dire qu'elles n'apparaissent pas dans les termes substitués :

$$f(x) =$$

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Si on remplace y par n'importe quelle expression (par ex $t = \sin(\sqrt{2})$), on a

$$f(t) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto t] = \lim_{x \to \infty} \frac{t}{x} = 0$$

Mais quid du cas où on remplace par x ?

$$f(x) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto x] = \lim_{x \to \infty} \frac{x}{x} = \lim_{x \to \infty} 1 = 1$$

Il faut faire l'hypothèse (implicite) que les variables liées sont **fraîches**, c'est-à-dire qu'elles n'apparaissent pas dans les termes substitués :

$$f(x) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto x] =$$

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Si on remplace y par n'importe quelle expression (par ex $t = \sin(\sqrt{2})$), on a

$$f(t) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto t] = \lim_{x \to \infty} \frac{t}{x} = 0$$

Mais quid du cas où on remplace par x ?

$$f(x) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto x] = \lim_{x \to \infty} \frac{x}{x} = \lim_{x \to \infty} 1 = 1$$

Il faut faire l'hypothèse (implicite) que les variables liées sont **fraîches**, c'est-à-dire qu'elles n'apparaissent pas dans les termes substitués :

$$f(x) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto x] = \left(\lim_{z \to \infty} \frac{y}{z} \right) [y \mapsto x] =$$

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Si on remplace y par n'importe quelle expression (par ex $t = \sin(\sqrt{2})$), on a

$$f(t) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto t] = \lim_{x \to \infty} \frac{t}{x} = 0$$

Mais quid du cas où on remplace par x ?

$$f(x) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto x] = \lim_{x \to \infty} \frac{x}{x} = \lim_{x \to \infty} 1 = 1$$

Il faut faire l'hypothèse (implicite) que les variables liées sont **fraîches**, c'est-à-dire qu'elles n'apparaissent pas dans les termes substitués :

$$f(x) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto x] = \left(\lim_{z \to \infty} \frac{y}{z} \right) [y \mapsto x] = \lim_{z \to \infty} \frac{x}{z} =$$

Variables liées et substitution

Il faut faire attention aux variables liées lors de la substitution !

Considérons

$$f(y) = \lim_{x \to \infty} \frac{y}{x}$$

Si on remplace y par n'importe quelle expression (par ex $t = \sin(\sqrt{2})$), on a

$$f(t) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto t] = \lim_{x \to \infty} \frac{t}{x} = 0$$

Mais quid du cas où on remplace par x ?

$$f(x) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto x] = \lim_{x \to \infty} \frac{x}{x} = \lim_{x \to \infty} 1 = 1$$

Il faut faire l'hypothèse (implicite) que les variables liées sont **fraîches**, c'est-à-dire qu'elles n'apparaissent pas dans les termes substitués :

$$f(x) = \left(\lim_{x \to \infty} \frac{y}{x} \right) [y \mapsto x] = \left(\lim_{z \to \infty} \frac{y}{z} \right) [y \mapsto x] = \lim_{z \to \infty} \frac{x}{z} = 0$$

Variables liées et substitution

En mathématiques c'est généralement implicite, mais quand on implémente la substitution, il faut faire explicitement attention à l'α-conversion.

Croyez-le ou non, c'est une source majeure de bugs.

Variables liées et substitution

En mathématiques c'est généralement implicite, mais quand on implémente la substitution, il faut faire explicitement attention à l'α-conversion.

Croyez-le ou non, c'est une source majeure de bugs.

Précisons les définitions de la substitution et de l'α-conversion.

Variables libres

Étant donné un terme t, on note $\mathrm{VL}(t)$ l'ensemble de ses **variables libres** (c'est-à-dire non liées) :

Variables libres

Étant donné un terme t, on note $\mathrm{VL}(t)$ l'ensemble de ses **variables libres** (c'est-à-dire non liées) :

$$\mathrm{VL}(\underline{n}) = \mathrm{VL}(\underline{b}) = \mathrm{VL}(\texttt{add}) = \mathrm{VL}(\texttt{fst}) = \mathrm{VL}(\texttt{snd}) =$$

Variables libres

Étant donné un terme t, on note $\mathrm{VL}(t)$ l'ensemble de ses **variables libres** (c'est-à-dire non liées) :

$$\mathrm{VL}(\underline{n}) = \mathrm{VL}(\underline{b}) = \mathrm{VL}(\texttt{add}) = \mathrm{VL}(\texttt{fst}) = \mathrm{VL}(\texttt{snd}) = \emptyset$$

Variables libres

Étant donné un terme t, on note $\mathrm{VL}(t)$ l'ensemble de ses **variables libres** (c'est-à-dire non liées) :

$$\mathrm{VL}(\underline{n}) = \mathrm{VL}(\underline{b}) = \mathrm{VL}(\texttt{add}) = \mathrm{VL}(\texttt{fst}) = \mathrm{VL}(\texttt{snd}) = \emptyset$$
$$\mathrm{VL}(\texttt{fun } x \to t) =$$

Variables libres

Étant donné un terme t, on note $\mathrm{VL}(t)$ l'ensemble de ses **variables libres** (c'est-à-dire non liées) :

$$\mathrm{VL}(\underline{n}) = \mathrm{VL}(\underline{b}) = \mathrm{VL}(\texttt{add}) = \mathrm{VL}(\texttt{fst}) = \mathrm{VL}(\texttt{snd}) = \emptyset$$
$$\mathrm{VL}(\texttt{fun}\ x \to t) = \mathrm{VL}(t) \setminus \{x\}$$

Variables libres

Étant donné un terme t, on note $\mathrm{VL}(t)$ l'ensemble de ses **variables libres** (c'est-à-dire non liées) :

$$\mathrm{VL}(\underline{n}) = \mathrm{VL}(\underline{b}) = \mathrm{VL}(\texttt{add}) = \mathrm{VL}(\texttt{fst}) = \mathrm{VL}(\texttt{snd}) = \emptyset$$
$$\mathrm{VL}(\texttt{fun}\ x\ \rightarrow\ t) = \mathrm{VL}(t) \setminus \{x\}$$
$$\mathrm{VL}(t\ u) = \mathrm{VL}((t, u)) =$$

Variables libres

Étant donné un terme t, on note $\mathrm{VL}(t)$ l'ensemble de ses **variables libres** (c'est-à-dire non liées) :

$$\mathrm{VL}(\underline{n}) = \mathrm{VL}(\underline{b}) = \mathrm{VL}(\texttt{add}) = \mathrm{VL}(\texttt{fst}) = \mathrm{VL}(\texttt{snd}) = \emptyset$$
$$\mathrm{VL}(\texttt{fun}\ x\ \rightarrow\ t) = \mathrm{VL}(t) \setminus \{x\}$$
$$\mathrm{VL}(t\ u) = \mathrm{VL}((t, u)) = \mathrm{VL}(t) \cup \mathrm{VL}(u)$$

Variables libres

Étant donné un terme t, on note $\mathrm{VL}(t)$ l'ensemble de ses **variables libres** (c'est-à-dire non liées) :

$$\mathrm{VL}(\underline{n}) = \mathrm{VL}(\underline{b}) = \mathrm{VL}(\texttt{add}) = \mathrm{VL}(\texttt{fst}) = \mathrm{VL}(\texttt{snd}) = \emptyset$$
$$\mathrm{VL}(\texttt{fun}\ x\ \texttt{->}\ t) = \mathrm{VL}(t) \setminus \{x\}$$
$$\mathrm{VL}(t\ u) = \mathrm{VL}((t\texttt{,}u)) = \mathrm{VL}(t) \cup \mathrm{VL}(u)$$

Par exemple,

$$\mathrm{VL}(\texttt{fun}\ x\ \texttt{->}\ f\ x) =$$

Variables libres

Étant donné un terme t, on note $\mathrm{VL}(t)$ l'ensemble de ses **variables libres** (c'est-à-dire non liées) :

$$\mathrm{VL}(\underline{n}) = \mathrm{VL}(\underline{b}) = \mathrm{VL}(\texttt{add}) = \mathrm{VL}(\texttt{fst}) = \mathrm{VL}(\texttt{snd}) = \emptyset$$
$$\mathrm{VL}(\texttt{fun}\ x\ \texttt{->}\ t) = \mathrm{VL}(t) \setminus \{x\}$$
$$\mathrm{VL}(t\ u) = \mathrm{VL}((t, u)) = \mathrm{VL}(t) \cup \mathrm{VL}(u)$$

Par exemple,

$$\mathrm{VL}(\texttt{fun}\ x\ \texttt{->}\ f\ x) = \{f\}$$

Variables libres

Étant donné un terme t, on note $\mathrm{VL}(t)$ l'ensemble de ses **variables libres** (c'est-à-dire non liées) :

$$\mathrm{VL}(\underline{n}) = \mathrm{VL}(\underline{b}) = \mathrm{VL}(\texttt{add}) = \mathrm{VL}(\texttt{fst}) = \mathrm{VL}(\texttt{snd}) = \emptyset$$
$$\mathrm{VL}(\texttt{fun}\ x\ \to\ t) = \mathrm{VL}(t) \setminus \{x\}$$
$$\mathrm{VL}(t\ u) = \mathrm{VL}((t, u)) = \mathrm{VL}(t) \cup \mathrm{VL}(u)$$

Par exemple,

$$\mathrm{VL}(\texttt{fun}\ x\ \to\ f\ x) = \{f\} \qquad \mathrm{VL}((\texttt{fun}\ x\ \to\ x)\ x) =$$

Variables libres

Étant donné un terme t, on note $\mathrm{VL}(t)$ l'ensemble de ses **variables libres** (c'est-à-dire non liées) :

$$\mathrm{VL}(\underline{n}) = \mathrm{VL}(\underline{b}) = \mathrm{VL}(\texttt{add}) = \mathrm{VL}(\texttt{fst}) = \mathrm{VL}(\texttt{snd}) = \emptyset$$
$$\mathrm{VL}(\texttt{fun}\ x\ \to\ t) = \mathrm{VL}(t) \setminus \{x\}$$
$$\mathrm{VL}(t\ u) = \mathrm{VL}((t, u)) = \mathrm{VL}(t) \cup \mathrm{VL}(u)$$

Par exemple,

$$\mathrm{VL}(\texttt{fun}\ x\ \to\ f\ x) = \{f\} \qquad \mathrm{VL}((\texttt{fun}\ x\ \to\ x)\ x) = \{x\}$$

Substitution

On définit la **substitution** $t[x \mapsto u]$ par induction sur t par

$$t[x \mapsto u] = \qquad \text{pour } t \in \{\underline{n}, \underline{b}, \texttt{add}, \texttt{fst}, \texttt{snd}\}$$

Substitution

On définit la **substitution** $t[x \mapsto u]$ par induction sur t par

$$t[x \mapsto u] = t \qquad \text{pour } t \in \{\underline{n}, \underline{b}, \texttt{add}, \texttt{fst}, \texttt{snd}\}$$

Substitution

On définit la **substitution** $t[x \mapsto u]$ par induction sur t par

$$t[x \mapsto u] = t \qquad \text{pour } t \in \{\underline{n}, \underline{b}, \texttt{add}, \texttt{fst}, \texttt{snd}\}$$
$$x[x \mapsto u] =$$

Substitution

On définit la **substitution** $t[x \mapsto u]$ par induction sur t par

$$t[x \mapsto u] = t \qquad \text{pour } t \in \{\underline{n}, \underline{b}, \texttt{add}, \texttt{fst}, \texttt{snd}\}$$
$$x[x \mapsto u] = u$$

Substitution

On définit la **substitution** $t[x \mapsto u]$ par induction sur t par

$$
\begin{aligned}
t[x \mapsto u] &= t && \text{pour } t \in \{\underline{n}, \underline{b}, \texttt{add}, \texttt{fst}, \texttt{snd}\} \\
x[x \mapsto u] &= u \\
y[x \mapsto u] &= && \text{pour } x \neq y
\end{aligned}
$$

Substitution

On définit la **substitution** $t[x \mapsto u]$ par induction sur t par

$$t[x \mapsto u] = t \qquad \text{pour } t \in \{\underline{n}, \underline{b}, \texttt{add}, \texttt{fst}, \texttt{snd}\}$$
$$x[x \mapsto u] = u$$
$$y[x \mapsto u] = y \qquad \text{pour } x \neq y$$

Substitution

On définit la **substitution** $t[x \mapsto u]$ par induction sur t par

$$t[x \mapsto u] = t \qquad \text{pour } t \in \{\underline{n}, \underline{b}, \texttt{add}, \texttt{fst}, \texttt{snd}\}$$

$$x[x \mapsto u] = u$$

$$y[x \mapsto u] = y \qquad \text{pour } x \neq y$$

$$(\texttt{fun } y \to t)[x \mapsto u] =$$

Substitution

On définit la **substitution** $t[x \mapsto u]$ par induction sur t par

$$
\begin{aligned}
t[x \mapsto u] &= t && \text{pour } t \in \{\underline{n}, \underline{b}, \texttt{add}, \texttt{fst}, \texttt{snd}\} \\
x[x \mapsto u] &= u \\
y[x \mapsto u] &= y && \text{pour } x \neq y \\
(\texttt{fun } y \to t)[x \mapsto u] &= \texttt{fun } y \to (t[x \mapsto u])
\end{aligned}
$$

Substitution

On définit la **substitution** $t[x \mapsto u]$ par induction sur t par

$$
\begin{array}{rll}
t[x \mapsto u] = t & & \text{pour } t \in \{\underline{n}, \underline{b}, \texttt{add}, \texttt{fst}, \texttt{snd}\} \\
x[x \mapsto u] = u & & \\
y[x \mapsto u] = y & & \text{pour } x \neq y \\
(\texttt{fun } y \to t)[x \mapsto u] = \texttt{fun } y \to (t[x \mapsto u]) & & \text{si } x \neq y \text{ et } y \notin \mathsf{VL}(u)
\end{array}
$$

Substitution

On définit la **substitution** $t[x \mapsto u]$ par induction sur t par

$$
\begin{aligned}
t[x \mapsto u] &= t && \text{pour } t \in \{\underline{n}, \underline{b}, \texttt{add}, \texttt{fst}, \texttt{snd}\} \\
x[x \mapsto u] &= u \\
y[x \mapsto u] &= y && \text{pour } x \neq y \\
(\texttt{fun } y \to t)[x \mapsto u] &= \texttt{fun } y \to (t[x \mapsto u]) && \text{si } x \neq y \text{ et } y \notin \mathsf{VL}(u) \\
(t\ t')[x \mapsto u] &= &&
\end{aligned}
$$

Substitution

On définit la **substitution** $t[x \mapsto u]$ par induction sur t par

$$
\begin{array}{rll}
t[x \mapsto u] = t & & \text{pour } t \in \{\underline{n}, \underline{b}, \texttt{add}, \texttt{fst}, \texttt{snd}\} \\
x[x \mapsto u] = u & & \\
y[x \mapsto u] = y & & \text{pour } x \neq y \\
(\texttt{fun } y \to t)[x \mapsto u] = \texttt{fun } y \to (t[x \mapsto u]) & & \text{si } x \neq y \text{ et } y \notin \mathsf{VL}(u) \\
(t\ t')[x \mapsto u] = (t[x \mapsto u])\ (t'[x \mapsto u]) & &
\end{array}
$$

Substitution

On définit la **substitution** $t[x \mapsto u]$ par induction sur t par

$$
\begin{aligned}
t[x \mapsto u] &= t && \text{pour } t \in \{\underline{n}, \underline{b}, \texttt{add}, \texttt{fst}, \texttt{snd}\} \\
x[x \mapsto u] &= u \\
y[x \mapsto u] &= y && \text{pour } x \neq y \\
(\texttt{fun } y \to t)[x \mapsto u] &= \texttt{fun } y \to (t[x \mapsto u]) && \text{si } x \neq y \text{ et } y \notin \mathsf{VL}(u) \\
(t\ t')[x \mapsto u] &= (t[x \mapsto u])\ (t'[x \mapsto u]) \\
(t, t')[x \mapsto u] &=
\end{aligned}
$$

Substitution

On définit la **substitution** $t[x \mapsto u]$ par induction sur t par

$$
\begin{aligned}
t[x \mapsto u] &= t && \text{pour } t \in \{\underline{n}, \underline{b}, \texttt{add}, \texttt{fst}, \texttt{snd}\} \\
x[x \mapsto u] &= u \\
y[x \mapsto u] &= y && \text{pour } x \neq y \\
(\texttt{fun } y \to t)[x \mapsto u] &= \texttt{fun } y \to (t[x \mapsto u]) && \text{si } x \neq y \text{ et } y \notin \mathsf{VL}(u) \\
(t\ t')[x \mapsto u] &= (t[x \mapsto u])\ (t'[x \mapsto u]) \\
(t, t')[x \mapsto u] &= (t[x \mapsto u], t'[x \mapsto u])
\end{aligned}
$$

Substitution

On définit la **substitution** $t[x \mapsto u]$ par induction sur t par

$$
\begin{array}{rll}
t[x \mapsto u] = t & & \text{pour } t \in \{\underline{n}, \underline{b}, \texttt{add}, \texttt{fst}, \texttt{snd}\} \\
x[x \mapsto u] = u & & \\
y[x \mapsto u] = y & & \text{pour } x \neq y \\
(\texttt{fun } y \to t)[x \mapsto u] = \texttt{fun } y \to (t[x \mapsto u]) & & \text{si } x \neq y \text{ et } y \notin \mathsf{VL}(u) \\
(t\ t')[x \mapsto u] = (t[x \mapsto u])\ (t'[x \mapsto u]) & & \\
(t, t')[x \mapsto u] = (t[x \mapsto u], t'[x \mapsto u]) & &
\end{array}
$$

Notons que a priori cette opération n'est que partiellement définie.

α-convertibilité

La relation d'**α-convertibilité** $\overset{\alpha}{=}$ sur les termes est la plus petite congruence telle que

$$\texttt{fun}\ x \to t \quad \overset{\alpha}{=} \quad \texttt{fun}\ y \to (t[x \mapsto y])$$

α-convertibilité

La relation d'**α-convertibilité** $\overset{\alpha}{=}$ sur les termes est la plus petite congruence telle que

$$\texttt{fun}\ x \to t \quad \overset{\alpha}{=} \quad \texttt{fun}\ y \to (t[x \mapsto y])$$

Par **congruence**, on entend ici qu'elle « passe au contexte », c'est-à-dire

α-convertibilité

La relation d'**α-convertibilité** $\overset{\alpha}{=}$ sur les termes est la plus petite congruence telle que

$$\texttt{fun}\ x \to t \quad \overset{\alpha}{=} \quad \texttt{fun}\ y \to (t[x \mapsto y])$$

Par **congruence**, on entend ici qu'elle « passe au contexte », c'est-à-dire

$$\frac{t \overset{\alpha}{=} t'}{\texttt{fun}\ x \to t \overset{\alpha}{=} \texttt{fun}\ x \to t'}$$

α-convertibilité

La relation d'**α-convertibilité** $\overset{\alpha}{=}$ sur les termes est la plus petite congruence telle que

$$\texttt{fun}\ x\ \to\ t \quad \overset{\alpha}{=} \quad \texttt{fun}\ y\ \to\ (t[x \mapsto y])$$

Par **congruence**, on entend ici qu'elle « passe au contexte », c'est-à-dire

$$\frac{t \overset{\alpha}{=} t'}{\texttt{fun}\ x\ \to\ t \overset{\alpha}{=} \texttt{fun}\ x\ \to\ t'} \qquad \frac{t \overset{\alpha}{=} t' \qquad u \overset{\alpha}{=} u'}{t\ u \overset{\alpha}{=} t'\ u'}$$

α-convertibilité

La relation d'**α-convertibilité** $\overset{\alpha}{=}$ sur les termes est la plus petite congruence telle que

$$\texttt{fun}\ x \to t \quad \overset{\alpha}{=} \quad \texttt{fun}\ y \to (t[x \mapsto y])$$

Par **congruence**, on entend ici qu'elle « passe au contexte », c'est-à-dire

$$\frac{t \overset{\alpha}{=} t'}{\texttt{fun}\ x \to t \overset{\alpha}{=} \texttt{fun}\ x \to t'} \qquad \frac{t \overset{\alpha}{=} t' \qquad u \overset{\alpha}{=} u'}{t\ u \overset{\alpha}{=} t'\ u'}$$

$$\frac{t \overset{\alpha}{=} t' \qquad u \overset{\alpha}{=} u'}{(t, u) \overset{\alpha}{=} (t', u')}$$

α-convertibilité

À partir de maintenant, on considérera toujours les termes modulo la relation d'α-conversion.

α-convertibilité

À partir de maintenant, on considérera toujours les termes modulo la relation d'α-conversion.

Proposition
La substitution $t[x \mapsto u]$ est toujours définie sur les termes modulo α-conversion.

α-convertibilité

À partir de maintenant, on considérera toujours les termes modulo la relation d'α-conversion.

Proposition
La substitution $t[x \mapsto u]$ est toujours définie sur les termes modulo α-conversion.

Par exemple,

$$(\texttt{fun}\ x \to x + y)[y \mapsto x + 1]$$

α-convertibilité

À partir de maintenant, on considérera toujours les termes modulo la relation d'α-conversion.

Proposition
La substitution $t[x \mapsto u]$ est toujours définie sur les termes modulo α-conversion.

Par exemple,

$$(\texttt{fun}\ x \to x + y)[y \mapsto x + 1] \overset{\alpha}{=} (\texttt{fun}\ z \to z + y)[y \mapsto x + 1]$$

α-convertibilité

À partir de maintenant, on considérera toujours les termes modulo la relation d'α-conversion.

Proposition
La substitution $t[x \mapsto u]$ est toujours définie sur les termes modulo α-conversion.

Par exemple,

$$\begin{aligned}(\texttt{fun } x \to x + y)[y \mapsto x + 1] &\overset{\alpha}{=} (\texttt{fun } z \to z + y)[y \mapsto x + 1] \\ &= \texttt{fun } z \to z + (x + 1)\end{aligned}$$

Troisième partie III

Opérateurs de point fixe

Extensions du langage

Le langage décrit précédemment est très minimal, on peut penser à plusieurs ajouts :

- on peut ajouter des fonctions classiques :
  - opérations booléennes (conjonction, disjonction, négation, etc.)
  - opérations arithmétique (soustraction, produit, etc.)
- on peut ajouter d'autres types de données et les fonctions associées
  - chaînes de caractères (concaténation, etc.)
  - flottants (opérations arithmétiques)
  - etc.

Branchement conditionnel

Par exemple, pour ajouter le **branchement conditionnel**

- on étend la syntaxe des programmes par

$$t \quad ::= \quad \ldots \quad | \quad \texttt{if } t \texttt{ then } t' \texttt{ else } t''$$

Branchement conditionnel

Par exemple, pour ajouter le **branchement conditionnel**

- on étend la syntaxe des programmes par

$$t \quad ::= \quad \ldots \quad | \quad \texttt{if } t \texttt{ then } t' \texttt{ else } t''$$

- on ajoute la règle de typage

Branchement conditionnel

Par exemple, pour ajouter le **branchement conditionnel**

- on étend la syntaxe des programmes par

$$t \quad ::= \quad \ldots \quad | \quad \texttt{if } t \texttt{ then } t' \texttt{ else } t''$$

- on ajoute la règle de typage

$$\frac{\Gamma \vdash t : \texttt{bool} \qquad \Gamma \vdash u : A \qquad \Gamma \vdash v : A}{\Gamma \vdash \texttt{if } t \texttt{ then } u \texttt{ else } v : A} \text{ (if)}$$

Branchement conditionnel

Par exemple, pour ajouter le **branchement conditionnel**

- on étend la syntaxe des programmes par

$$t \quad ::= \quad \ldots \quad | \quad \texttt{if } t \texttt{ then } t' \texttt{ else } t''$$

- on ajoute la règle de typage

$$\frac{\Gamma \vdash t : \texttt{bool} \qquad \Gamma \vdash u : A \qquad \Gamma \vdash v : A}{\Gamma \vdash \texttt{if } t \texttt{ then } u \texttt{ else } v : A} \text{ (if)}$$

- on ajoute les règles d'évaluation

Branchement conditionnel

Par exemple, pour ajouter le **branchement conditionnel**

- on étend la syntaxe des programmes par

$$t \quad ::= \quad \ldots \quad | \quad \texttt{if}\ t\ \texttt{then}\ t'\ \texttt{else}\ t''$$

- on ajoute la règle de typage

$$\frac{\Gamma \vdash t : \texttt{bool} \qquad \Gamma \vdash u : A \qquad \Gamma \vdash v : A}{\Gamma \vdash \texttt{if}\ t\ \texttt{then}\ u\ \texttt{else}\ v : A}\ (\text{if})$$

- on ajoute les règles d'évaluation

$$\frac{t \longrightarrow\!\!\!\!\rightarrow \texttt{true} \qquad u \longrightarrow\!\!\!\!\rightarrow u'}{\texttt{if}\ t\ \texttt{then}\ u\ \texttt{else}\ v \longrightarrow\!\!\!\!\rightarrow u'} \qquad \frac{t \longrightarrow\!\!\!\!\rightarrow \texttt{false} \qquad v \longrightarrow\!\!\!\!\rightarrow v'}{\texttt{if}\ t\ \texttt{then}\ u\ \texttt{else}\ v \longrightarrow\!\!\!\!\rightarrow v'}$$

Turing complétude

Le langage tels que nous l'avons décrit n'est pas Turing complet :

- toutes les fonctions typables s'évaluent en une valeur
  (c'est hautement non-trivial, voir le cours de λ-calcul)
- donc il existe des fonctions calculables qui ne peuvent pas être implémentées
  (par un argument de diagonalisation)

Turing complétude

Le langage tels que nous l'avons décrit n'est pas Turing complet :

- toutes les fonctions typables s'évaluent en une valeur
  (c'est hautement non-trivial, voir le cours de λ-calcul)
- donc il existe des fonctions calculables qui ne peuvent pas être implémentées
  (par un argument de diagonalisation)

Quelle opération nous manque-t-il ?

Turing complétude

Le langage tels que nous l'avons décrit n'est pas Turing complet :

- toutes les fonctions typables s'évaluent en une valeur
  (c'est hautement non-trivial, voir le cours de λ-calcul)
- donc il existe des fonctions calculables qui ne peuvent pas être implémentées
  (par un argument de diagonalisation)

Quelle opération nous manque-t-il ?

Le `while`, ou un équivalent adapté aux langages fonctionnels.

Opérateurs de point fixe

En maths, un **point fixe** d'une fonction $f : A \to A$ est

Opérateurs de point fixe

En maths, un **point fixe** d'une fonction $f : A \to A$ est un élément $a \in A$ tel que

$$f(a) = a$$

Opérateurs de point fixe

En maths, un **point fixe** d'une fonction $f : A \to A$ est un élément $a \in A$ tel que

$$f(a) = a$$

Dans la plupart des langages de programmation fonctionnels,

Opérateurs de point fixe

En maths, un **point fixe** d'une fonction $f : A \to A$ est un élément $a \in A$ tel que

$$f(a) = a$$

Dans la plupart des langages de programmation fonctionnels,

- toutes les fonctions $f : A \Rightarrow A$ ont un point fixe, i.e. un élément $a : A$ tel que

  $$f\ a = a$$

  (l'égalité dit ici que les deux s'évaluent pareil)

Opérateurs de point fixe

En maths, un **point fixe** d'une fonction $f : A \to A$ est un élément $a \in A$ tel que

$$f(a) = a$$

Dans la plupart des langages de programmation fonctionnels,

- toutes les fonctions $f : A \Rightarrow A$ ont un point fixe, i.e. un élément $a : A$ tel que

  $$f\ a = a$$

  (l'égalité dit ici que les deux s'évaluent pareil)
- ce point fixe peut être calculé par un opérateur

  `fix`

Opérateurs de point fixe

En maths, un **point fixe** d'une fonction $f : A \to A$ est un élément $a \in A$ tel que

$$f(a) = a$$

Dans la plupart des langages de programmation fonctionnels,

- toutes les fonctions $f : A \Rightarrow A$ ont un point fixe, i.e. un élément $a : A$ tel que

  $$f\ a = a$$

  (l'égalité dit ici que les deux s'évaluent pareil)
- ce point fixe peut être calculé par un opérateur

  $$\texttt{fix} : (A \Rightarrow A) \Rightarrow A$$

Opérateurs de point fixe

En maths, un **point fixe** d'une fonction $f : A \to A$ est un élément $a \in A$ tel que

$$f(a) = a$$

Dans la plupart des langages de programmation fonctionnels,

- toutes les fonctions $f : A \Rightarrow A$ ont un point fixe, i.e. un élément $a : A$ tel que

  $$f\ a = a$$

  (l'égalité dit ici que les deux s'évaluent pareil)
- ce point fixe peut être calculé par un opérateur

  $$\texttt{fix} : (A \Rightarrow A) \Rightarrow A$$

- celui-ci suffit pour implémenter la récursion

Opérateurs de point fixe

En OCaml, on peut programmer un opérateur de point fixe par

Opérateurs de point fixe

En OCaml, on peut programmer un opérateur de point fixe par

```
let rec fix f   = f (fix f)
```

Opérateurs de point fixe

En OCaml, on peut programmer un opérateur de point fixe par

```
let rec fix f   = f (fix f)
```

La factorielle peut être programmée par

Opérateurs de point fixe

En OCaml, on peut programmer un opérateur de point fixe par

```
let rec fix f   = f (fix f)
```

La factorielle peut être programmée par

```
let rec fact   n =
  if n = 0 then 1 else n * fact (n - 1)
```

Opérateurs de point fixe

En OCaml, on peut programmer un opérateur de point fixe par

```
let rec fix f   = f (fix f)
```

La factorielle peut être programmée par

```
let fact_fun f n =
  if n = 0 then 1 else n * f    (n - 1)
```

Opérateurs de point fixe

En OCaml, on peut programmer un opérateur de point fixe par

```
let rec fix f   = f (fix f)
```

La factorielle peut être programmée par

```
let fact_fun f n =
  if n = 0 then 1 else n * f    (n - 1)
```

puis

```
let fact = fix fact_fun
```

Opérateurs de point fixe

En OCaml, on peut programmer un opérateur de point fixe par

```
let rec fix f   = f (fix f)
```

La factorielle peut être programmée par

```
let fact_fun f n =
  if n = 0 then 1 else n * f    (n - 1)
```

puis

```
let fact = fix fact_fun
```

Problème :

```
Stack overflow during evaluation (looping recursion?).
```

Opérateurs de point fixe

En OCaml, on peut programmer un opérateur de point fixe par

```
let rec fix f x = f (fix f) x
```

La factorielle peut être programmée par

```
let fact_fun f n =
  if n = 0 then 1 else n * f    (n - 1)
```

puis

```
let fact = fix fact_fun
```

Problème résolu :

```
# fact 5;;
- : int = 120
```

Opérateur de point fixe en mini-ML

On peut ajouter ces opérateurs de point fixe en mini-ML :

- on étend la syntaxe des programmes :

$$t \quad ::= \quad \ldots \quad | \quad \texttt{fix}$$

Opérateur de point fixe en mini-ML

On peut ajouter ces opérateurs de point fixe en mini-ML :

- on étend la syntaxe des programmes :

$$t \quad ::= \quad \ldots \quad | \quad \texttt{fix}$$

- on étend les valeurs :

$$v \quad ::= \quad \ldots \quad | \quad \texttt{fix}$$

Opérateur de point fixe en mini-ML

On peut ajouter ces opérateurs de point fixe en mini-ML :

- on étend la syntaxe des programmes :

$$t \quad ::= \quad \ldots \quad | \quad \texttt{fix}$$

- on étend les valeurs :

$$v \quad ::= \quad \ldots \quad | \quad \texttt{fix}$$

- on ajoute la règle de typage

$$\frac{}{\Gamma \vdash \texttt{fix} : (A \Rightarrow A) \Rightarrow A}\ (\text{fix})$$

Opérateur de point fixe en mini-ML

On peut ajouter ces opérateurs de point fixe en mini-ML :

- on étend la syntaxe des programmes :

$$t \quad ::= \quad \ldots \quad | \quad \texttt{fix}$$

- on étend les valeurs :

$$v \quad ::= \quad \ldots \quad | \quad \texttt{fix}$$

- on ajoute la règle de typage

$$\frac{}{\Gamma \vdash \texttt{fix} : (A \Rightarrow A) \Rightarrow A}\ (\text{fix})$$

- on étend l'évaluation...

Opérateur de point fixe en mini-ML

On peut ajouter ces opérateurs de point fixe en mini-ML avec annotations de type :

- on étend la syntaxe des programmes :

$$t \quad ::= \quad \ldots \quad | \quad \texttt{fix}(t)$$

- on étend les valeurs :

$$v \quad ::= \quad \ldots$$

- on ajoute la règle de typage

$$\frac{\Gamma \vdash t : A \Rightarrow A}{\Gamma \vdash \texttt{fix}\ t : A}\ (\text{fix})$$

- on étend l'évaluation...

Opérateur de point fixe en mini-ML

La règle d'évaluation

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

Opérateur de point fixe en mini-ML

La règle d'évaluation

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

ne convient pas, car on va avoir

$$\texttt{fix}\ t \longrightarrow\!\!\!\!\!\rightarrow v$$

Opérateur de point fixe en mini-ML

La règle d'évaluation

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

ne convient pas, car on va avoir

$$t \longrightarrow\!\!\!\!\!\rightarrow t'$$

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

Opérateur de point fixe en mini-ML

La règle d'évaluation

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

ne convient pas, car on va avoir

$$t \longrightarrow\!\!\!\!\!\rightarrow t'$$

$$\dfrac{\dfrac{}{t' \longrightarrow\!\!\!\!\!\rightarrow \texttt{fun}\ x \to t''} \qquad \texttt{fix}\ t' \longrightarrow\!\!\!\!\!\rightarrow v \qquad \dfrac{\vdots}{t''[x \mapsto v'] \longrightarrow\!\!\!\!\!\rightarrow v}}{t'\ (\texttt{fix}\ t') \longrightarrow\!\!\!\!\!\rightarrow v}$$

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

Opérateur de point fixe en mini-ML

La règle d'évaluation

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

ne convient pas, car on va avoir

$$t \longrightarrow\!\!\!\!\!\rightarrow\ t'$$

$$\dfrac{\dfrac{}{t' \longrightarrow\!\!\!\!\!\rightarrow\ \texttt{fun}\ x\ \rightarrow\ t''} \qquad \dfrac{\dfrac{}{t' \longrightarrow\!\!\!\!\!\rightarrow\ t'} \qquad t'\ (\texttt{fix}\ t') \longrightarrow\!\!\!\!\!\rightarrow\ v}{\texttt{fix}\ t' \longrightarrow\!\!\!\!\!\rightarrow\ v} \qquad \dfrac{\vdots}{t''[x \mapsto v'] \longrightarrow\!\!\!\!\!\rightarrow\ v}}{\dfrac{t'\ (\texttt{fix}\ t') \longrightarrow\!\!\!\!\!\rightarrow\ v}{\texttt{fix}\ t \longrightarrow\!\!\!\!\!\rightarrow\ v}}$$

Opérateur de point fixe en mini-ML

La règle d'évaluation

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

ne convient pas, car on va avoir

$$\dfrac{\dfrac{}{t' \longrightarrow\!\!\!\!\!\rightarrow\ \texttt{fun}\ x\ \rightarrow\ t''} \qquad \dfrac{\dfrac{}{t' \longrightarrow\!\!\!\!\!\rightarrow\ t'} \qquad \dfrac{t \longrightarrow\!\!\!\!\!\rightarrow\ t'}{\vdots}}{\texttt{fix}\ t' \longrightarrow\!\!\!\!\!\rightarrow\ v} \qquad \dfrac{\vdots}{t''[x \mapsto v'] \longrightarrow\!\!\!\!\!\rightarrow\ v}}{t'\ (\texttt{fix}\ t') \longrightarrow\!\!\!\!\!\rightarrow\ v}$$

$$\frac{}{\texttt{fix}\ t \longrightarrow\!\!\!\!\!\rightarrow\ v}$$

Opérateur de point fixe en mini-ML

La bonne règle est

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

car elle force l'évaluation du point fixe à être paresseuse.

Opérateur de point fixe en mini-ML

On peut maintenant implémenter la factorielle :

$$\texttt{fix}\,(\texttt{fun}\ f \to \texttt{fun}\ n \to \texttt{if}\ n = 0\ \texttt{then}\ 1\ \texttt{else}\ n \times (f\,(n-1)))$$

Opérateur de point fixe en mini-ML

On peut maintenant implémenter la factorielle :

$$\texttt{fix}\,(\texttt{fun}\ f \to \texttt{fun}\ n \to \texttt{if}\ n = 0\ \texttt{then}\ 1\ \texttt{else}\ n \times (f\ (n-1)))$$

On note F pour

$$\texttt{if}\ n = 0\ \texttt{then}\ 1\ \texttt{else}\ n \times (f\ (n-1))$$

Opérateur de point fixe en mini-ML

L'évaluation de la factorielle termine :

$$\frac{\begin{array}{c}(\texttt{fun}\ f \to \texttt{fun}\ n \to F) \longrightarrow\!\!\!\!\rightarrow (\texttt{fun}\ f \to \texttt{fun}\ n \to F) \\ (\texttt{fun}\ n \to F)[f \mapsto \texttt{fix}(\texttt{fun}\ f \to \texttt{fun}\ n \to F)] \longrightarrow\!\!\!\!\rightarrow (\texttt{fun}\ n \to F[f \mapsto \texttt{fix}(\ldots)])\end{array}}{\texttt{fix}(\texttt{fun}\ f \to \texttt{fun}\ n \to F) \longrightarrow\!\!\!\!\rightarrow (\texttt{fun}\ n \to F[f \mapsto \texttt{fix}(\ldots)])}\ (\text{fix})$$

Opérateur de point fixe en mini-ML

Il existe maintenant des programmes typables dont l'évaluation n'est pas définie :

Opérateur de point fixe en mini-ML

Il existe maintenant des programmes typables dont l'évaluation n'est pas définie :

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

Opérateur de point fixe en mini-ML

Il existe maintenant des programmes typables dont l'évaluation n'est pas définie :

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

qui est typable par

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

Opérateur de point fixe en mini-ML

Il existe maintenant des programmes typables dont l'évaluation n'est pas définie :

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

qui est typable par

$$\frac{\vdash \texttt{fun}\ x \to x : A \Rightarrow A}{\vdash \texttt{fix}\,(\texttt{fun}\ x \to x) : A}\ (\text{fix})$$

Opérateur de point fixe en mini-ML

Il existe maintenant des programmes typables dont l'évaluation n'est pas définie :

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

qui est typable par

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

Opérateur de point fixe en mini-ML

Il existe maintenant des programmes typables dont l'évaluation n'est pas définie :

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

qui est typable par

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

La semaine prochaine

À la prochaine séance :

- nous allons montrer la propriété de sûreté du typage
- nous allons étendre le typage au cas sans annotations