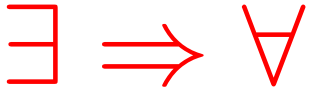
Logique et Calculabilité
INF551



Dr. Stéphane Lengrand,

Stephane.Lengrand@Polytechnique.edu

Cours 6
Le calcul comme une suite de petits pas

I. L'interpréteur

Des programmes qui manipulent d'autres programmes

De tels programmes sont utilisés tous les jours (compilateurs,. . .)

Des programmes qui manipulent d'autres programmes

De tels programmes sont utilisés tous les jours (compilateurs,. . .)

Le problème de l'arrêt :

Il n'y a pas de prog. qui prenne en entrée un programme f et des arguments $p_1, \ldots, p_n$ et dise si f termine en $p_1, \ldots, p_n$

Résultat négatif

Des programmes qui manipulent d'autres programmes

De tels programmes sont utilisés tous les jours (compilateurs,. . .)

Le problème de l'arrêt :

Il n'y a pas de prog. qui prenne en entrée un programme f et des arguments $p_1, \ldots, p_n$ et dise si f termine en $p_1, \ldots, p_n$

Résultat négatif

Résultat positif :

Il existe un prog. qui prend en entrée un programme f et des arguments $p_1, \ldots, p_n$ et renvoie $f(p_1, \ldots, p_n)$ si f termine en $p_1, \ldots, p_n$, et ne s'arrête pas sinon.

Des programmes qui manipulent d'autres programmes

De tels programmes sont utilisés tous les jours (compilateurs,. . .)

Le problème de l'arrêt :

Il n'y a pas de prog. qui prenne en entrée un programme f et des arguments $p_1, \ldots, p_n$ et dise si f termine en $p_1, \ldots, p_n$

Résultat négatif

Résultat positif :

Il existe un prog. qui prend en entrée un programme f et des arguments $p_1, \ldots, p_n$ et renvoie $f(p_1, \ldots, p_n)$ si f termine en $p_1, \ldots, p_n$, et ne s'arrête pas sinon.

C'est l'interpréteur :

G prend 2 arguments : le programme, et la liste d'arguments

La notion d'étape intermédiaire

Construire G demande de décomposer un calcul en une suite de petits pas qui se déroulent dans le temps

La notion d'étape intermédiaire

Construire G demande de décomposer un calcul en une suite de petits pas qui se déroulent dans le temps

On aura besoin de représenter des étapes intermédiaires du calcul

La notion d'étape intermédiaire

Construire G demande de décomposer un calcul en une suite de petits pas qui se déroulent dans le temps

On aura besoin de représenter des étapes intermédiaires du calcul

. . .dans une syntaxe ad hoc

La notion d'étape intermédiaire

Construire G demande de décomposer un calcul en une suite de petits pas qui se déroulent dans le temps

On aura besoin de représenter des étapes intermédiaires du calcul

. . .dans une syntaxe ad hoc i.e. à base d'arbres

La notion d'étape intermédiaire

Construire G demande de décomposer un calcul en une suite de petits pas qui se déroulent dans le temps

On aura besoin de représenter des étapes intermédiaires du calcul

. . .dans une syntaxe ad hoc i.e. à base d'arbres

On veut notamment représenter

"le résultat de l'application d'un programme f à des arguments $p_1, \ldots, p_n$"

avant de l'avoir calculé.

La notion d'étape intermédiaire

Construire G demande de décomposer un calcul en une suite de petits pas qui se déroulent dans le temps

On aura besoin de représenter des étapes intermédiaires du calcul

. . .dans une syntaxe ad hoc i.e. à base d'arbres

On veut notamment représenter

"le résultat de l'application d'un programme f à des arguments $p_1, \ldots, p_n$"

avant de l'avoir calculé.

On va l'exprimer dans cette syntaxe par un terme $App^n(f, p_1, ..., p_n)$

Pourquoi ?

Pourquoi ?

Si $f = \circ_m^n(h, g_1, ...g_m)$:

Pourquoi ?

Si $f = \circ_m^n(h, g_1, ...g_m)$:

$App^n(f, p_1, ..., p_n)$ transformé dans un premier temps en

$$App^m(h, App^n(g_1, p_1, ..., p_n), ..., App^n(g_m, p_1, ..., p_n))$$

Pourquoi ?

Si $f = \circ_m^n(h, g_1, ...g_m)$:

$App^n(f, p_1, ..., p_n)$ transformé dans un premier temps en

$$App^m(h, App^n(g_1, p_1, ..., p_n), ..., App^n(g_m, p_1, ..., p_n))$$

puis on remplace les $App^n(g_i, p_1, ..., p_n)$ par q_i

Pourquoi ?

Si $f = \circ_m^n(h, g_1, ...g_m)$:

$App^n(f, p_1, ..., p_n)$ transformé dans un premier temps en

$$App^m(h, App^n(g_1, p_1, ..., p_n), ..., App^n(g_m, p_1, ..., p_n))$$

puis on remplace les $App^n(g_i, p_1, ..., p_n)$ par q_i

enfin on remplace $App^m(h, q_1, ..., q_m)$ par q

Pourquoi ?

Si $f = \circ_m^n(h, g_1, ...g_m)$:

$App^n(f, p_1, ..., p_n)$ transformé dans un premier temps en

$$App^m(h, App^n(g_1, p_1, ..., p_n), ..., App^n(g_m, p_1, ..., p_n))$$

puis on remplace les $App^n(g_i, p_1, ..., p_n)$ par q_i

enfin on remplace $App^m(h, q_1, ..., q_m)$ par q

D'où le besoin de termes représentant des étapes intermédiaires :

App^m s'applique à des entiers mais aussi à d'autres termes $App^n(...)$ en attente d'être calculés

La syntaxe

Ca tombe bien,

on a déjà une syntaxe arborescente pour représenter les programmes.

La syntaxe

Ca tombe bien,

on a déjà une syntaxe arborescente pour représenter les programmes.

On étend cette syntaxe à une syntaxe de termes $\mathcal{T}$ qui contient au moins :

La syntaxe

Ca tombe bien,

on a déjà une syntaxe arborescente pour représenter les programmes.

On étend cette syntaxe à une syntaxe de termes $\mathcal{T}$ qui contient au moins :

– les constructeurs 0 et S

pour représenter tout entier p par le terme syntaxique $\underline{p} = \overbrace{S(\ldots S}^{p \text{ fois}}\ (0)\ \ldots)$

La syntaxe

Ca tombe bien,

on a déjà une syntaxe arborescente pour représenter les programmes.

On étend cette syntaxe à une syntaxe de termes $\mathcal{T}$ qui contient au moins :

- les constructeurs 0 et S

  pour représenter tout entier p par le terme syntaxique $\underline{p} = \overbrace{S(\ldots S}^{p \text{ fois}}\ (0)\ \ldots)$
- une famille de symboles App^n pour représenter

  “le résultat de l'application de f à des arguments $p_1, \ldots, p_n$”

  par le terme syntaxique $App^n(f, \underline{p_1}, \ldots, \underline{p_n})$
- ...

La syntaxe

Ca tombe bien,

on a déjà une syntaxe arborescente pour représenter les programmes.

On étend cette syntaxe à une syntaxe de termes $\mathcal{T}$ qui contient au moins :

– les constructeurs 0 et S

pour représenter tout entier p par le terme syntaxique $\underline{p} = \overbrace{S(\ldots S}^{p \text{ fois}}(0)\ldots)$

– une famille de symboles App^n pour représenter

"le résultat de l'application de f à des arguments $p_1, \ldots, p_n$"

par le terme syntaxique $App^n(f, \underline{p_1}, \ldots, \underline{p_n})$

– ...

$S(App^2(\pi_2^1, S(0), App^2(\pi_2^1, 0, 0)))$ est un terme

La syntaxe

Ca tombe bien,

on a déjà une syntaxe arborescente pour représenter les programmes.

On étend cette syntaxe à une syntaxe de termes $\mathcal{T}$ qui contient au moins :

- les constructeurs 0 et S

  pour représenter tout entier p par le terme syntaxique $\underline{p} = \overbrace{S(\ldots S}^{p \text{ fois}}(0)\ldots)$
- une famille de symboles App^n pour représenter

  "le résultat de l'application de f à des arguments $p_1, \ldots, p_n$"

  par le terme syntaxique $App^n(f, \underline{p_1}, \ldots, \underline{p_n})$
- ...

$S(App^2(\pi_2^1, S(0), App^2(\pi_2^1, 0, 0)))$ est un terme

On cherchera à le transformer en $S(S(0))$

La syntaxe

Ca tombe bien,

on a déjà une syntaxe arborescente pour représenter les programmes.

On étend cette syntaxe à une syntaxe de termes $\mathcal{T}$ qui contient au moins :

- les constructeurs 0 et S

  pour représenter tout entier p par le terme syntaxique $\underline{p} = \overbrace{S(\ldots S}^{p \text{ fois}}(0)\ldots)$
- une famille de symboles App^n pour représenter

  “le résultat de l'application de f à des arguments $p_1, \ldots, p_n$”

  par le terme syntaxique $App^n(f, \underline{p_1}, \ldots, \underline{p_n})$
- ...

$S(App^2(\pi_2^1, S(0), App^2(\pi_2^1, 0, 0)))$ est un terme

On cherchera à le transformer en $S(S(0))$

Remarque :

$\texttt{parse} : p \mapsto \underline{p}$ et $\texttt{prettyprint} : \underline{p} \mapsto p$ sont calculables

Principe

Pour calculer $G(f, (p_1, \ldots, p_n))$,

on commence avec le terme $App^n(f, \underline{p_1}, \ldots, \underline{p_n})$,

que l'on transforme progressivement en $\underline{f(p_1, \ldots, p_n)}$

Principe

Pour calculer $G(f, (p_1, \ldots, p_n))$,

on commence avec le terme $App^n(f, \underline{p_1}, \ldots, \underline{p_n})$,

que l'on transforme progressivement en $\underline{f(p_1, \ldots, p_n)}$

Plus formellement :

On veut construire une fonction calculable $F : \mathcal{T} \longrightarrow \mathcal{T}$ telle que

$F(App^n(f, \underline{p_1}, \ldots, \underline{p_n})) = \underline{f(p_1, \ldots, p_n)}$

$F(App^n(f, \underline{p_1}, \ldots, \underline{p_n}))$ pas définie si $f(p_1, \ldots, p_n)$ pas définie

Principe

Pour calculer $G(f, (p_1, \ldots, p_n))$,

on commence avec le terme $App^n(f, \underline{p_1}, \ldots, \underline{p_n})$,

que l'on transforme progressivement en $\underline{f(p_1, \ldots, p_n)}$

Plus formellement :

On veut construire une fonction calculable $F : \mathcal{T} \longrightarrow \mathcal{T}$ telle que

$F(App^n(f, \underline{p_1}, \ldots, \underline{p_n})) = \underline{f(p_1, \ldots, p_n)}$

$F(App^n(f, \underline{p_1}, \ldots, \underline{p_n}))$ pas définie si $f(p_1, \ldots, p_n)$ pas définie

Il suffit alors de définir $G(f, (p_1, \ldots, p_n)) =$

$\texttt{prettyprint}(F(App^n(f, \texttt{parse}(p_1), \ldots, \texttt{parse}(p_n))))$

F_1 : une étape de calcul à la racine

Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_n})$

– si $f = \pi_i^n$ on pose

$$F_1(t) = \underline{p_i}$$

F_1 : une étape de calcul à la racine

Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_n})$

– si $f = \pi_i^n$ on pose

$$F_1(t) = \underline{p_i}$$

– si $f = Z^n, Succ, +, \times, \chi_{\leq}$: facile

F_1 : une étape de calcul à la racine

Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_n})$

– si $f = \pi_i^n$ on pose

$$F_1(t) = \underline{p_i}$$

– si $f = Z^n, Succ, +, \times, \chi_{\leq}$: facile

– si $f = \circ_m^n(h, g_1, \ldots, g_m)$, on pose

$$F_1(t) = App^m(h, App^n(g_1, \underline{p_1}, \ldots, \underline{p_n}), \ldots, App^n(g_m, \underline{p_1}, \ldots, \underline{p_n}))$$

F_1 : une étape de calcul à la racine

Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_n})$

– si $f = \pi_i^n$ on pose

$$F_1(t) = \underline{p_i}$$

– si $f = Z^n, Succ, +, \times, \chi_{\leq}$: facile

– si $f = \circ_m^n(h, g_1, \ldots, g_m)$, on pose

$$F_1(t) = App^m(h, App^n(g_1, \underline{p_1}, \ldots, \underline{p_n}), \ldots, App^n(g_m, \underline{p_1}, \ldots, \underline{p_n}))$$

– si $f = \mu^n(g)$, on pose

F_1 : une étape de calcul à la racine

Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_n})$

– si $f = \pi_i^n$ on pose

$$F_1(t) = \underline{p_i}$$

– si $f = Z^n, Succ, +, \times, \chi_{\leq}$: facile

– si $f = \circ_m^n(h, g_1, \ldots, g_m)$, on pose

$$F_1(t) = App^m(h, App^n(g_1, \underline{p_1}, \ldots, \underline{p_n}), \ldots, App^n(g_m, \underline{p_1}, \ldots, \underline{p_n}))$$

– si $f = \mu^n(g)$, on pose

$$F_1(t) = M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, 0)$$

$M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{p})$ = “Le plus petit entier $\geq p$ t.q. $g(\underline{p_1}, \ldots, \underline{p_n}) = 0$”

. . .de nouvelle constructions syntaxiques

Si $t = M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q})$, on pose

$$F_1(t) = \mathit{Ifz}(App^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q}), \underline{q}, M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, S(\underline{q})))$$

. . .de nouvelle constructions syntaxiques

Si $t = M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q})$, on pose

$$F_1(t) = \mathit{Ifz}(App^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q}), \underline{q}, M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, S(\underline{q})))$$

Si $t = \mathit{Ifz}(\underline{p}, u, v)$ alors

– si $\underline{p} = 0$, on pose $F_1(t) = u$

– si $\underline{p} = S(q)$, on pose $F_1(t) = v$

. . .de nouvelle constructions syntaxiques

Si $t = M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q})$, on pose

$$F_1(t) = \mathit{Ifz}(App^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q}), \underline{q}, M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, S(\underline{q})))$$

Si $t = \mathit{Ifz}(\underline{p}, u, v)$ alors

– si $\underline{p} = 0$, on pose $F_1(t) = u$

– si $\underline{p} = S(q)$, on pose $F_1(t) = v$

Sinon, on pose $F_1(t) = 0$

. . .de nouvelle constructions syntaxiques

Si $t = M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q})$, on pose

$$F_1(t) = \mathit{Ifz}(App^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q}), \underline{q}, M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, S(\underline{q})))$$

Si $t = \mathit{Ifz}(\underline{p}, u, v)$ alors

– si $\underline{p} = 0$, on pose $F_1(t) = u$

– si $\underline{p} = S(q)$, on pose $F_1(t) = v$

Sinon, on pose $F_1(t) = 0$

Définition par cas (même pas récurrence structurelle) : F_1 calculable

F_2 : une étape de calcul

– Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_n})$ on pose $F_2(t) = F_1(t)$

F_2 : une étape de calcul

– Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_n})$ on pose $F_2(t) = F_1(t)$

– Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_{i-1}}, t_i, \ldots, t_n)$ avec $t_i \neq \underline{p}$, on pose

$$F_2(t) = App^n(f, \underline{p_1}, \ldots, \underline{p_{i-1}}, F_2(t_i), t_{i+1}, \ldots, t_n)$$

F_2 : une étape de calcul

– Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_n})$ on pose $F_2(t) = F_1(t)$

– Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_{i-1}}, t_i, \ldots, t_n)$ avec $t_i \neq \underline{p}$, on pose

$$F_2(t) = App^n(f, \underline{p_1}, \ldots, \underline{p_{i-1}}, F_2(t_i), t_{i+1}, \ldots, t_n)$$

– Si $t = M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q})$, on pose $F_2(t) = F_1(t)$

F_2 : une étape de calcul

– Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_n})$ on pose $F_2(t) = F_1(t)$

– Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_{i-1}}, t_i, \ldots, t_n)$ avec $t_i \neq \underline{p}$, on pose

$$F_2(t) = App^n(f, \underline{p_1}, \ldots, \underline{p_{i-1}}, F_2(t_i), t_{i+1}, \ldots, t_n)$$

– Si $t = M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q})$, on pose $F_2(t) = F_1(t)$

– Si $t = \mathit{Ifz}(\underline{p}, v, w)$ on pose $F_2(t) = F_1(t)$

F_2 : une étape de calcul

– Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_n})$ on pose $F_2(t) = F_1(t)$

– Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_{i-1}}, t_i, \ldots, t_n)$ avec $t_i \neq \underline{p}$, on pose

$$F_2(t) = App^n(f, \underline{p_1}, \ldots, \underline{p_{i-1}}, F_2(t_i), t_{i+1}, \ldots, t_n)$$

– Si $t = M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q})$, on pose $F_2(t) = F_1(t)$

– Si $t = \mathit{Ifz}(\underline{p}, v, w)$ on pose $F_2(t) = F_1(t)$

– Si $t = \mathit{Ifz}(u, v, w)$ avec $u \neq \underline{p}$, on pose $F_2(t) = \mathit{Ifz}(F_2(u), v, w)$

F_2 : une étape de calcul

- Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_n})$ on pose $F_2(t) = F_1(t)$
- Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_{i-1}}, t_i, \ldots, t_n)$ avec $t_i \neq \underline{p}$, on pose

$$F_2(t) = App^n(f, \underline{p_1}, \ldots, \underline{p_{i-1}}, F_2(t_i), t_{i+1}, \ldots, t_n)$$

- Si $t = M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q})$, on pose $F_2(t) = F_1(t)$
- Si $t = \mathit{Ifz}(\underline{p}, v, w)$ on pose $F_2(t) = F_1(t)$
- Si $t = \mathit{Ifz}(u, v, w)$ avec $u \neq \underline{p}$, on pose $F_2(t) = \mathit{Ifz}(F_2(u), v, w)$
- Sinon, on pose $F_2(t) = 0$

F_2 : une étape de calcul

– Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_n})$ on pose $F_2(t) = F_1(t)$

– Si $t = App^n(f, \underline{p_1}, \ldots, \underline{p_{i-1}}, t_i, \ldots, t_n)$ avec $t_i \neq \underline{p}$, on pose

$$F_2(t) = App^n(f, \underline{p_1}, \ldots, \underline{p_{i-1}}, F_2(t_i), t_{i+1}, \ldots, t_n)$$

– Si $t = M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q})$, on pose $F_2(t) = F_1(t)$

– Si $t = \mathit{Ifz}(\underline{p}, v, w)$ on pose $F_2(t) = F_1(t)$

– Si $t = \mathit{Ifz}(u, v, w)$ avec $u \neq \underline{p}$, on pose $F_2(t) = \mathit{Ifz}(F_2(u), v, w)$

– Sinon, on pose $F_2(t) = 0$

Définition par récurrence structurelle : F_2 calculable

F_3 : p étapes de calcul

$$
\begin{aligned}
F_3(t, 0) &= t \\
F_3(t, p+1) &= F_3(F_2(t), p)
\end{aligned}
$$

F_3 : p étapes de calcul

$$\begin{aligned} F_3(t,0) &= t \\ F_3(t,p+1) &= F_3(F_2(t),p) \end{aligned}$$

Autrement dit : $F_3(t,p) = F_2^p(t)$

F_3 : p étapes de calcul

$$\begin{aligned} F_3(t,0) &= t \\ F_3(t,p+1) &= F_3(F_2(t),p) \end{aligned}$$

Autrement dit : $F_3(t,p) = F_2^p(t)$

Définition par récurrence : F_3 calculable

F_3 : p étapes de calcul, F_4

$$\begin{aligned} F_3(t,0) &= t \\ F_3(t,p+1) &= F_3(F_2(t),p) \end{aligned}$$

Autrement dit : $F_3(t,p) = F_2^p(t)$

Définition par récurrence : F_3 calculable

F_4 t.q. $F_4(t,n) = 0$ si $F_3(t,n)$ est un arbre qui représente un entier
et $F_4(t,n) = 1$ sinon

F_3 : p étapes de calcul, F_4, F_5

$$\begin{aligned} F_3(t,0) &= t \\ F_3(t,p+1) &= F_3(F_2(t),p) \end{aligned}$$

Autrement dit : $F_3(t,p) = F_2^p(t)$

Définition par récurrence : F_3 calculable

F_4 t.q. $F_4(t,n) = 0$ si $F_3(t,n)$ est un arbre qui représente un entier
et $F_4(t,n) = 1$ sinon

$F_5(t)$: plus petit entier p tel que $F_4(t,p) = 0$

(nombre d'étapes nécessaires pour calculer t)

F_3 : p étapes de calcul, F_4, F_5, l'interpréteur F

$$\begin{aligned} F_3(t,0) &= t \\ F_3(t,p+1) &= F_3(F_2(t),p) \end{aligned}$$

Autrement dit : $F_3(t,p) = F_2^p(t)$

Définition par récurrence : F_3 calculable

F_4 t.q. $F_4(t,n) = 0$ si $F_3(t,n)$ est un arbre qui représente un entier
et $F_4(t,n) = 1$ sinon

$F_5(t)$: plus petit entier p tel que $F_4(t,p) = 0$

(nombre d'étapes nécessaires pour calculer t)

$F(t) = F_3(t, F_5(t))$

F_3 : p étapes de calcul, F_4, F_5, l'interpréteur F

$$\begin{aligned} F_3(t,0) &= t \\ F_3(t,p+1) &= F_3(F_2(t),p) \end{aligned}$$

Autrement dit : $F_3(t,p) = F_2^p(t)$

Définition par récurrence : F_3 calculable

F_4 t.q. $F_4(t,n) = 0$ si $F_3(t,n)$ est un arbre qui représente un entier
et $F_4(t,n) = 1$ sinon

$F_5(t)$: plus petit entier p tel que $F_4(t,p) = 0$

(nombre d'étapes nécessaires pour calculer t)

$F(t) = F_3(t, F_5(t))$ Toutes calculables

F_3 : p étapes de calcul, F_4, F_5, l'interpréteur F

$$\begin{aligned} F_3(t,0) &= t \\ F_3(t,p+1) &= F_3(F_2(t),p) \end{aligned}$$

Autrement dit : $F_3(t,p) = F_2^p(t)$

Définition par récurrence : F_3 calculable

F_4 t.q. $F_4(t,n) = 0$ si $F_3(t,n)$ est un arbre qui représente un entier
et $F_4(t,n) = 1$ sinon

$F_5(t)$: plus petit entier p tel que $F_4(t,p) = 0$

(nombre d'étapes nécessaires pour calculer t)

$F(t) = F_3(t, F_5(t))$ Toutes calculables

Si f pas définie, F_4 jamais nulle, F_5 pas définie, F pas définie

Une généralisation de l'indécidabilité du problème de l'arrêt

Soit A un sous-ensemble décidable de l'ensemble des programmes,

tel que tous les programmes de A terminent toujours

Une généralisation de l'indécidabilité du problème de l'arrêt

Théorème :

Soit A un sous-ensemble décidable de l'ensemble des programmes,

tel que tous les programmes de A terminent toujours

Alors A incomplet : il existe une fonction (unaire) calculable totale qui n'est représentée par aucun programme de A

Preuve

Soit H la fonction calculable définie par :

si t est un programme unaire de A, alors $H(\ulcorner t \urcorner, n) = G(t, n)$

sinon $H(t, n) = 0$

Preuve

Soit H la fonction calculable définie par :

si t est un programme unaire de A, alors $H(\ulcorner t \urcorner, n) = G(t, n)$

sinon $H(t, n) = 0$

H interpréteur pour tous les programmes unaires de A

Preuve

Soit H la fonction calculable définie par :

si t est un programme unaire de A, alors $H(\ulcorner t \urcorner, n) = G(t, n)$

sinon $H(t, n) = 0$

H interpréteur pour tous les programmes unaires de A

H totale

Preuve

Soit H la fonction calculable définie par :

si t est un programme unaire de A, alors $H(\ulcorner t \urcorner, n) = G(t, n)$

sinon $H(t, n) = 0$

H interpréteur pour tous les programmes unaires de A

H totale

On montre que la fonction calculable totale $H'(n) = H(n, n) + 1$

n'est pas dans A

Preuve

Soit H la fonction calculable définie par :

si t est un programme unaire de A, alors $H(\ulcorner t \urcorner, n) = G(t, n)$

sinon $H(t, n) = 0$

H interpréteur pour tous les programmes unaires de A

H totale

On montre que la fonction calculable totale $H'(n) = H(n, n) + 1$

n'est pas dans A

Sinon il y aurait t dans A qui calcule H'

Preuve

Soit H la fonction calculable définie par :

si t est un programme unaire de A, alors $H(\ulcorner t \urcorner, n) = G(t, n)$

sinon $H(t, n) = 0$

H interpréteur pour tous les programmes unaires de A

H totale

On montre que la fonction calculable totale $H'(n) = H(n, n) + 1$

n'est pas dans A

Sinon il y aurait t dans A qui calcule H'

On aurait pour tout n,

$H(\ulcorner t \urcorner, n) = G(t, n) = H'(n) = H(n, n) + 1$

Preuve

Soit H la fonction calculable définie par :

si t est un programme unaire de A, alors $H(\ulcorner t \urcorner, n) = G(t, n)$

sinon $H(t, n) = 0$

H interpréteur pour tous les programmes unaires de A

H totale

On montre que la fonction calculable totale $H'(n) = H(n, n) + 1$

n'est pas dans A

Sinon il y aurait t dans A qui calcule H'

On aurait pour tout n,

$H(\ulcorner t \urcorner, n) = G(t, n) = H'(n) = H(n, n) + 1$

en particulier $H(\ulcorner t \urcorner, \ulcorner t \urcorner) = H(\ulcorner t \urcorner, \ulcorner t \urcorner) + 1$

Corollaire 1 : Indécidabilité du problème de l'arrêt

Supposons $Ar : \mathbb{N}^2 \longrightarrow \mathbb{N}$ décide de l'arrêt

Corollaire 1 : Indécidabilité du problème de l'arrêt

Supposons $Ar : \mathbb{N}^2 \longrightarrow \mathbb{N}$ décide de l'arrêt

Def : Pour tout programme unaire f,

soit f' le programme unaire qui termine toujours défini par

$f'(n) = f(n)$ si $Ar(\ulcorner f \urcorner, n) = 1$ et $f'(n) = 0$ sinon.

Corollaire 1 : Indécidabilité du problème de l'arrêt

Supposons $Ar : \mathbb{N}^2 \longrightarrow \mathbb{N}$ décide de l'arrêt

Def : Pour tout programme unaire f,

soit f' le programme unaire qui termine toujours défini par

$f'(n) = f(n)$ si $Ar(\ulcorner f \urcorner, n) = 1$ et $f'(n) = 0$ sinon.

– L'ensemble des couples (f, f') est décidable

Corollaire 1 : Indécidabilité du problème de l'arrêt

Supposons $Ar : \mathbb{N}^2 \longrightarrow \mathbb{N}$ décide de l'arrêt

Def : Pour tout programme unaire f,

soit f' le programme unaire qui termine toujours défini par

$f'(n) = f(n)$ si $Ar(\ulcorner f \urcorner, n) = 1$ et $f'(n) = 0$ sinon.

– L'ensemble des couples (f, f') est décidable

f est toujours un sous-arbre de f' donc $\ulcorner f \urcorner < \ulcorner f' \urcorner$

Corollaire 1 : Indécidabilité du problème de l'arrêt

Supposons $Ar : \mathbb{N}^2 \longrightarrow \mathbb{N}$ décide de l'arrêt

Def : Pour tout programme unaire f,

soit f' le programme unaire qui termine toujours défini par

$f'(n) = f(n)$ si $Ar(\ulcorner f \urcorner, n) = 1$ et $f'(n) = 0$ sinon.

– L'ensemble des couples (f, f') est décidable

f est toujours un sous-arbre de f' donc $\ulcorner f \urcorner < \ulcorner f' \urcorner$

donc l'ensemble A des programmes de la forme f' est décidable

Corollaire 1 : Indécidabilité du problème de l'arrêt

Supposons $Ar : \mathbb{N}^2 \longrightarrow \mathbb{N}$ décide de l'arrêt

Def : Pour tout programme unaire f,

soit f' le programme unaire qui termine toujours défini par

$f'(n) = f(n)$ si $Ar(\ulcorner f \urcorner, n) = 1$ et $f'(n) = 0$ sinon.

– L'ensemble des couples (f, f') est décidable

f est toujours un sous-arbre de f' donc $\ulcorner f \urcorner < \ulcorner f' \urcorner$

donc l'ensemble A des programmes de la forme f' est décidable

– Pour tout programme unaire f qui termine toujours,

f' et f calculent la même fonction.

Corollaire 1 : Indécidabilité du problème de l'arrêt

Supposons $Ar : \mathbb{N}^2 \longrightarrow \mathbb{N}$ décide de l'arrêt

Def : Pour tout programme unaire f,

soit f' le programme unaire qui termine toujours défini par

$f'(n) = f(n)$ si $Ar(\ulcorner f \urcorner, n) = 1$ et $f'(n) = 0$ sinon.

– L'ensemble des couples (f, f') est décidable

f est toujours un sous-arbre de f' donc $\ulcorner f \urcorner < \ulcorner f' \urcorner$

donc l'ensemble A des programmes de la forme f' est décidable

– Pour tout programme unaire f qui termine toujours,

f' et f calculent la même fonction.

Toute fonction unaire calculable totale est représentable par un programme de A.

Corollaire 1 : Indécidabilité du problème de l'arrêt

Supposons $Ar : \mathbb{N}^2 \longrightarrow \mathbb{N}$ décide de l'arrêt

Def : Pour tout programme unaire f,

soit f' le programme unaire qui termine toujours défini par

$f'(n) = f(n)$ si $Ar(\ulcorner f \urcorner, n) = 1$ et $f'(n) = 0$ sinon.

– L'ensemble des couples (f, f') est décidable

f est toujours un sous-arbre de f' donc $\ulcorner f \urcorner < \ulcorner f' \urcorner$

donc l'ensemble A des programmes de la forme f' est décidable

– Pour tout programme unaire f qui termine toujours,

f' et f calculent la même fonction.

Toute fonction unaire calculable totale est représentable par un programme de A.

Contradiction avec le théorème précédent.

Autres corollaires

Corollaire 2 : Existence d'une fonction calculable totale qui n'est pas primitive récursive

Autres corollaires

Corollaire 2 : Existence d'une fonction calculable totale qui n'est pas primitive récursive

Preuve : Soit A l'ensemble des programmes écrits sans la minimisation.

Autres corollaires

Corollaire 2 : Existence d'une fonction calculable totale qui n'est pas primitive récursive

Preuve : Soit A l'ensemble des programmes écrits sans la minimisation.

Corollaire 3 : Un langage de programmation dont tous les programmes terminent est toujours incomplet : il ne contient pas son interpréteur

(e.g. langage impératif formé de la déclaration de variable, de l'affectation, de la séquence, du test et de la boucle `for`)

II. Une suite de petits pas

Une “nouvelle” conception du calcul

– Un langage pour les programmes f

– Une extension de ce langage pour représenter les termes $App^n(f, \underline{p_1}, \ldots, \underline{p_n})$

– Une notion de petit pas de calcul

que l’on itère

Une “nouvelle” conception du calcul

– Un langage pour les programmes f

– Une extension de ce langage pour représenter les termes $App^n(f, \underline{p_1}, \ldots, \underline{p_n})$

– Une notion de petit pas de calcul

que l’on itère

$\Longrightarrow$ définit une nouvelle notion de calculabilité

Une “nouvelle” conception du calcul

– Un langage pour les programmes f

– Une extension de ce langage pour représenter les termes $App^n(f, \underline{p_1}, \ldots, \underline{p_n})$

– Une notion de petit pas de calcul

que l’on itère

$\Longrightarrow$ définit une nouvelle notion de calculabilité

Def :

Une fonction est représentable s’il existe un programme f qui la calcule

Une “nouvelle” conception du calcul

– Un langage pour les programmes f

– Une extension de ce langage pour représenter les termes $App^n(f, \underline{p_1}, \ldots, \underline{p_n})$

– Une notion de petit pas de calcul (calculable au sens usuel)

que l’on itère

$\Longrightarrow$ définit une nouvelle notion de calculabilité

a priori plus faible que la notion standard

Def :

Une fonction est représentable s’il existe un programme f qui la calcule

Toutes les fonctions représentables sont calculables au sens standard

Une “nouvelle” conception du calcul

– Un langage pour les programmes f

– Une extension de ce langage pour représenter les termes $App^n(f, \underline{p_1}, \ldots, \underline{p_n})$

– Une notion de petit pas de calcul (calculable au sens usuel)

que l’on itère

$\Longrightarrow$ définit une nouvelle notion de calculabilité

a priori plus faible que la notion standard

Def :

Une fonction est représentable s’il existe un programme f qui la calcule

Toutes les fonctions représentables sont calculables au sens standard

Def : Si toutes les fonctions calculables sont représentables, on dit que le calcul est Turing-complet

Une “nouvelle” conception du calcul

Tous les langages de programmation entrent dans ce cadre (Caml, C, Java, ...) :

sémantique opérationnelle à petits pas

Une “nouvelle” conception du calcul

Tous les langages de programmation entrent dans ce cadre (Caml, C, Java, ...) : sémantique opérationnelle à petits pas

Mais utile d’avoir des langages plus dépouillés

Algorithmes plus difficiles à exprimer, mais plus faciles à étudier (terminaison, invariants, complexité, . . .)

Une “nouvelle” conception du calcul

Tous les langages de programmation entrent dans ce cadre (Caml, C, Java, ...) : sémantique opérationnelle à petits pas

Mais utile d’avoir des langages plus dépouillés

Algorithmes plus difficiles à exprimer, mais plus faciles à étudier (terminaison, invariants, complexité, . . .)

Trois langages : réécriture, lambda-calcul, machines de Turing

III. La réécriture

Les bases

Termes : termes d'un langage sans lieurs

Les bases

Termes : termes d'un langage sans lieurs

Petit pas : ensemble de règles de réécriture

Exemple :

$$x + 0 \longrightarrow x$$

N'importe quel terme de la forme $t + 0$ se réécrit en t

Les bases

Termes : termes d'un langage sans lieurs

Petit pas : ensemble de règles de réécriture

Exemple :

$$x + 0 \longrightarrow x$$

N'importe quel terme de la forme $t + 0$ se réécrit en t

Quid de $(5 + 0) + 4$?

Quelques définitions

Règle de réécriture :

couple de termes (l, r), noté $l \longrightarrow r$, avec $fv(r) \subseteq fv(l)$

Quelques définitions

Règle de réécriture :

couple de termes (l, r), noté $l \longrightarrow r$, avec $fv(r) \subseteq fv(l)$

Une étape de réduction à la racine :

$t \longrightarrow u$ s'il existe $l \longrightarrow r$ et σ t.q. $\sigma l = t$ et $\sigma r = u$

Quelques définitions

Règle de réécriture :

couple de termes (l, r), noté $l \longrightarrow r$, avec $fv(r) \subseteq fv(l)$

Une étape de réduction à la racine :

$t \longrightarrow u$ s'il existe $l \longrightarrow r$ et σ t.q. $\sigma l = t$ et $\sigma r = u$

Radical : terme réductible par la relation $\longrightarrow$: σl

Quelques définitions

Règle de réécriture :

couple de termes (l, r), noté $l \longrightarrow r$, avec $fv(r) \subseteq fv(l)$

Une étape de réduction à la racine :

$t \longrightarrow u$ s'il existe $l \longrightarrow r$ et σ t.q. $\sigma l = t$ et $\sigma r = u$

Radical : terme réductible par la relation $\longrightarrow$: σl

Une étape de réduction : relation inductivement définie par

– si $t \longrightarrow u$, alors $t \rhd u$

– si $t \rhd u$, alors $f(t_1, ..., t_{i-1}, t, t_{i+1}, ..., t_n) \rhd f(t_1, ..., t_{i-1}, u, t_{i+1}, ..., t_n)$

Une étape, des étapes

réduction $\triangleright^*$ fermeture réflexive-transitive de $\triangleright$, inductivement définie par

– $t \triangleright^* t$

– si $t \triangleright t'$ et $t' \triangleright^* t''$, alors $t \triangleright^* t''$

Une étape, des étapes

réduction $\triangleright^*$ fermeture réflexive-transitive de $\triangleright$, inductivement définie par

– $t \triangleright^* t$

– si $t \triangleright t'$ et $t' \triangleright^* t''$, alors $t \triangleright^* t''$

Dérivation de t à t' :

suite finie $t_0, \ldots, t_n$, telle que

– $t_0 = t$

– $t_n = t'$

– et pour tout $i \leq n - 1$, $t_i \triangleright t_{i+1}$

Un exemple

$$x + 0 \longrightarrow x$$

$$x + S(y) \longrightarrow S(x + y)$$

Un exemple

$$x + 0 \longrightarrow x$$

$$x + S(y) \longrightarrow S(x + y)$$

$S(S(0)) + S(S(0))$?

Un exemple

$$x + 0 \longrightarrow x$$

$$x + S(y) \longrightarrow S(x + y)$$

$S(S(0)) + S(S(0))$?

$\triangleright^*$ $S(S(S(S(0))))$

Existence d'un résultat ?

Terme irréductible : ne peut pas être réduit par la relation $\triangleright$

(aucun de ses sous-termes n'est un radical)

Existence d'un résultat ?

Terme irréductible : ne peut pas être réduit par la relation $\triangleright$

(aucun de ses sous-termes n'est un radical)

Le terme t termine s'il existe t' irréductible t.q. $t \triangleright^* t'$

Existence d'un résultat ?

Terme irréductible : ne peut pas être réduit par la relation $\triangleright$

(aucun de ses sous-termes n'est un radical)

Le terme t termine s'il existe t' irréductible t.q. $t \triangleright^* t'$

Un exemple :

$$f(x) \longrightarrow a$$

$$\omega \longrightarrow \omega$$

Existence d'un résultat ?

Terme irréductible : ne peut pas être réduit par la relation $\triangleright$

(aucun de ses sous-termes n'est un radical)

Le terme t termine s'il existe t' irréductible t.q. $t \triangleright^* t'$

Un exemple :

$$f(x) \longrightarrow a$$

$$\omega \longrightarrow \omega$$

ω ?

Existence d'un résultat ?

Terme irréductible : ne peut pas être réduit par la relation $\triangleright$

(aucun de ses sous-termes n'est un radical)

Le terme t termine s'il existe t' irréductible t.q. $t \triangleright^* t'$

Un exemple :

$$f(x) \longrightarrow a$$

$$\omega \longrightarrow \omega$$

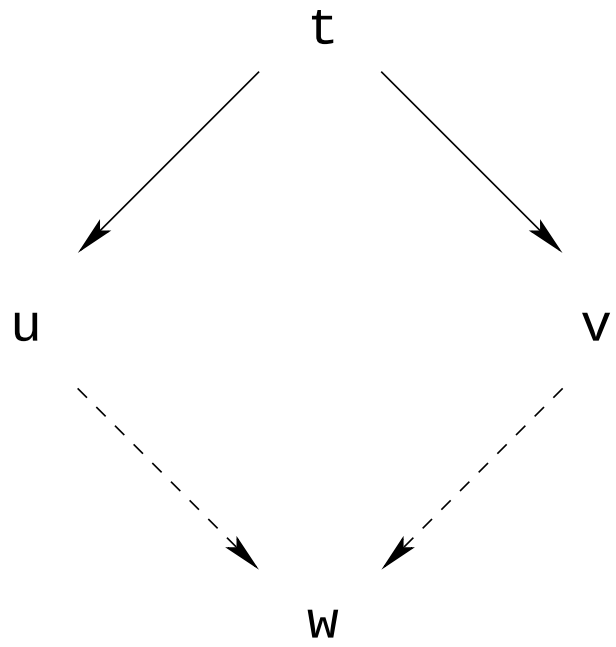
ω ?

$f(\omega)$?

Et l'unicité ?

Definition: $\triangleright$ est confluent :

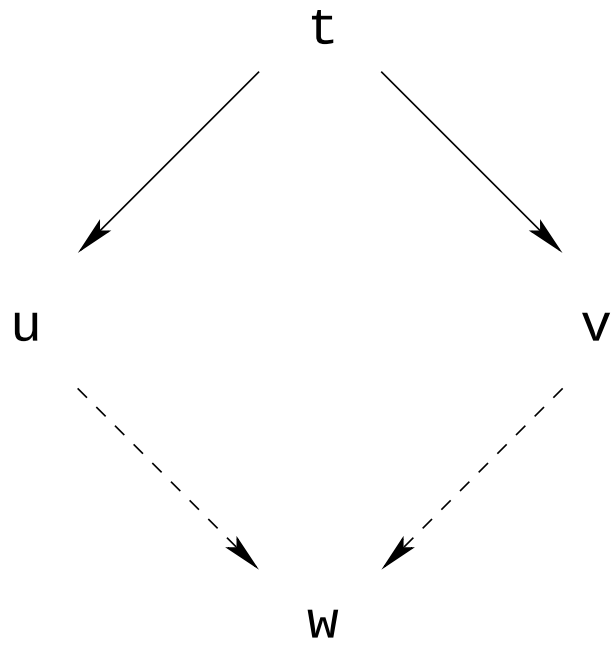
si $t \ \triangleright^* \ u$ et $t \ \triangleright^* \ v$, il existe w t.q. $u \ \triangleright^* \ w$ et $v \ \triangleright^* \ w$



Et l'unicité ?

Definition: $\triangleright$ est confluent :

si $t \ \triangleright^* \ u$ et $t \ \triangleright^* \ v$, il existe w t.q. $u \ \triangleright^* \ w$ et $v \ \triangleright^* \ w$



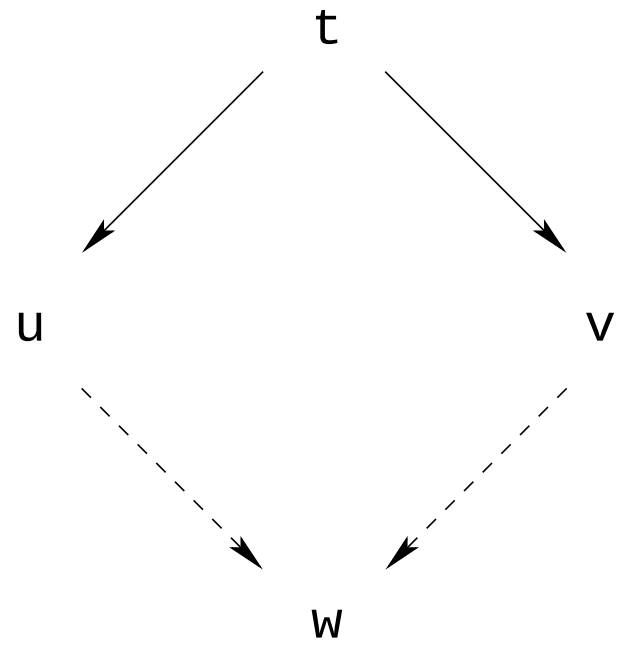
Unicité de la forme réduite si u et v sont réduits $u = w = v$

Deux chemins ne peuvent mener à des formes réduites différentes

Et l'unicité ?

Definition: $\triangleright$ est confluent :

si $t \ \triangleright^* \ u$ et $t \ \triangleright^* \ v$, il existe w t.q. $u \ \triangleright^* \ w$ et $v \ \triangleright^* \ w$



Unicité de la forme réduite si u et v sont réduits $u = w = v$

Deux chemins ne peuvent mener à des formes réduites différentes

Mais tous les chemins ne se valent pas pour autant : $f(\omega)$

L'orthogonalité

Une condition suffisante (et décidable) pour la confluence :

L'orthogonalité

Une condition suffisante (et décidable) pour la confluence :

– Si dans toute règle $l \longrightarrow r$

  chaque variable x a au plus une occurrence dans l,

– Si pour toutes règles $l \longrightarrow r$, $l' \longrightarrow r'$

  et pour tout l'' sous-terme de l' distinct d'une variable,

  aucunes substitutions σ et τ ne satisfont $\sigma l = \tau l''$

L'orthogonalité

Une condition suffisante (et décidable) pour la confluence :

– Si dans toute règle $l \longrightarrow r$

  chaque variable x a au plus une occurrence dans l,

– Si pour toutes règles $l \longrightarrow r$, $l' \longrightarrow r'$

  et pour tout l'' sous-terme de l' distinct d'une variable,

  aucunes substitutions σ et τ ne satisfont $\sigma l = \tau l''$

Alors $\rhd^*$ est confluent

Un exemple

$$g(h(x)) \longrightarrow a$$

$$f(g(x)) \longrightarrow b$$

Orthogonal ?

$f(g(h(c)))$?

Un autre exemple

$$x - x \longrightarrow 0$$

$$S(x) - x \longrightarrow 1$$

$$\infty \longrightarrow S(\infty)$$

Orthogonal ?

$\infty - \infty$?

IV. La représentation des fonctions

Des entiers, des fonctions

Deux symboles 0, S, ($\underline{p} = S(S(...(S(0))...))$)

D'autres symboles F

Des règles de réécriture $\mathcal{R}$

Les $S(S(...(S(0))...))$ irréductibles

Des entiers, des fonctions

Deux symboles 0, S, ($\underline{p} = S(S(...(S(0))...))$)

D'autres symboles F

Des règles de réécriture $\mathcal{R}$

Les $S(S(...(S(0))...))$ irréductibles

Def :

$\mathcal{R}, F$ représente f

– si $f(p_1, ..., p_n) = q$, alors $F(\underline{p_1}, ..., \underline{p_n}) \rhd^* \underline{q}$,

– si f pas définie en $p_1, ..., p_n$, alors $F(\underline{p_1}, ..., \underline{p_n})$ ne termine pas

Une suite de petits pas ?

Non, car mille manières de réduire $F(\underline{p_1}, ..., \underline{p_n})$

Une suite de petits pas ?

Non, car mille manières de réduire $F(\underline{p_1}, ..., \underline{p_n})$

Stratégie : sélectionne 1 radical à réduire lorsqu'il y a le choix.

Une suite de petits pas ?

Non, car mille manières de réduire $F(\underline{p_1}, ..., \underline{p_n})$

Stratégie : sélectionne 1 radical à réduire lorsqu'il y a le choix.

Exemple : la réduction en appel par nom

Une suite de petits pas ?

Non, car mille manières de réduire $F(\underline{p_1}, ..., \underline{p_n})$

Stratégie : sélectionne 1 radical à réduire lorsqu'il y a le choix.

Exemple : la réduction en appel par nom

Remarque : système orthogonal $\Rightarrow$ un seul résultat possible

mais certaines stratégies peuvent terminer d'autres non

L'appel par nom

Une étape de réduction en appel par nom inductivement déf. par

– si $t \longrightarrow t'$, alors $t \succ t'$

– si $f(t_1, ..., t_n)$ n'est pas un radical, si $t_1, ..., t_{i-1}$ sont irréductibles, et si $t_i \succ t'_i$, alors $f(t_1, ..., t_{i-1}, t_i, t_{i+1}, ..., t_n) \succ f(t_1, ..., t_{i-1}, t'_i, t_{i+1}, ..., t_n)$

L'appel par nom

Une étape de réduction en appel par nom inductivement déf. par

– si $t \longrightarrow t'$, alors $t \succ t'$

– si $f(t_1, ..., t_n)$ n'est pas un radical, si $t_1, ..., t_{i-1}$ sont irréductibles, et si $t_i \succ t'_i$, alors $f(t_1, ..., t_{i-1}, t_i, t_{i+1}, ..., t_n) \succ f(t_1, ..., t_{i-1}, t'_i, t_{i+1}, ..., t_n)$

Def :

$\mathcal{R}, F$ représente f en appel par nom

– si $f(p_1, ..., p_n) = q$, alors $F(\underline{p_1}, ..., \underline{p_n}) \succ^* \underline{q}$

– si f n'est pas définie en $p_1, ..., p_n$, alors $F(\underline{p_1}, ..., \underline{p_n})$ ne termine pas en appel par nom

V. La représentation des fonctions calculables

But du jeu

On veut montrer que la réécriture est Turing-complète :

Toute fonction calculable peut être représentée

(en général et en appel par nom)

But du jeu

On veut montrer que la réécriture est Turing-complète :

Toute fonction calculable peut être représentée

(en général et en appel par nom)

f partielle calculable : on construit un ens. de règles de réé. t.q.

– Si $f(p_1, ..., p_n) = q$, alors $F(\underline{p_1}, ..., \underline{p_n})$ se réduit en $\underline{q}$ en appel par nom (et *a fortiori*, en général)
– Si f n'est pas définie en $p_1, ..., p_n$, alors $F(\underline{p_1}, ..., \underline{p_n})$ ne termine pas (et *a fortiori*, ne termine pas en appel par nom)

Plutôt facile

si $f = \pi_i^n$

$$F(x_1, ..., x_n) \longrightarrow x_i$$

si $f = Z^n$

$$F(x_1, ..., x_n) \longrightarrow 0$$

si $f = Succ$

$$F(x) \longrightarrow S(x)$$

si $f = h \circ g$

$$F(x) \longrightarrow H(G(x))$$

Mais un peu trop naïf

Attention lorsque l'on jette des termes à la poubelle !

Parfois on veut s'assurer qu'ils sont bien évalués (de la forme $\underline{p}$)

Mais un peu trop naïf

Attention lorsque l'on jette des termes à la poubelle !

Parfois on veut s'assurer qu'ils sont bien évalués (de la forme $\underline{p}$)

Exemple :

g pas définie en 4, $h = Z^1$, $f = h \circ g$

$$H(x) \longrightarrow 0$$

$$F(x) \longrightarrow H(G(x))$$

Mais un peu trop naïf

Attention lorsque l'on jette des termes à la poubelle !

Parfois on veut s'assurer qu'ils sont bien évalués (de la forme $\underline{p}$)

Exemple :

g pas définie en 4, $h = Z^1$, $f = h \circ g$

$$H(x) \longrightarrow 0$$

$$F(x) \longrightarrow H(G(x))$$

f pas définie en 4 mais

$$F(\underline{4}) \succ H(G(\underline{4})) \succ 0$$

Passer en appel par valeur ?

(i.e. on ne réduit un radical que quand ses sous-termes sont irréductibles)

Passer en appel par valeur ?

(i.e. on ne réduit un radical que quand ses sous-termes sont irréductibles)

Hélas le système interprétant $f = \mu^n(g)$ ne terminerait pas ! Rappel :

$$F_1(M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q})) =$$

$$\mathit{Ifz}(App^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q}), \underline{q}, M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, S(\underline{q})))$$

Ne surtout pas commencer par calculer $M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, S(\underline{q}))$!

Passer en appel par valeur ?

(i.e. on ne réduit un radical que quand ses sous-termes sont irréductibles)

Hélas le système interprétant $f = \mu^n(g)$ ne terminerait pas ! Rappel :

$$F_1(M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q})) =$$

$$\mathit{Ifz}(App^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, \underline{q}), \underline{q}, M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, S(\underline{q})))$$

Ne surtout pas commencer par calculer $M^{n+1}(g, \underline{p_1}, \ldots, \underline{p_n}, S(\underline{q}))$!

Solution : si $h = Z^1$ on pose $H(x) \longrightarrow 0\&x$

$a\&b$: a pourvu que b termine (sur un entier)

$$x\&0 \longrightarrow x$$

$$x\&S(y) \longrightarrow x\&y$$

Génération du système de réecriture

$$x \& 0 \longrightarrow x$$

$$x \& S(y) \longrightarrow x \& y$$

Génération du système de réecriture

$$x \& 0 \longrightarrow x$$

$$x \& S(y) \longrightarrow x \& y$$

$$\mathit{Ifz}(0, y, z) \longrightarrow y$$

$$\mathit{Ifz}(S(x), y, z) \longrightarrow z \& x$$

Génération du système de réecriture

$$x\&0 \longrightarrow x$$

$$x\&S(y) \longrightarrow x\&y$$

$$\mathit{Ifz}(0, y, z) \longrightarrow y$$

$$\mathit{Ifz}(S(x), y, z) \longrightarrow z\&x$$

si $f = \pi_i^n$

$$F(x_1, ..., x_n) \longrightarrow ((((x_i\&x_1)\&...\&x_{i-1})\&x_{i+1})\&...\&x_n)$$

Génération du système de réecriture

$$x\&0 \longrightarrow x$$

$$x\&S(y) \longrightarrow x\&y$$

$$\mathit{Ifz}(0, y, z) \longrightarrow y$$

$$\mathit{Ifz}(S(x), y, z) \longrightarrow z\&x$$

si $f = \pi_i^n$

$$F(x_1, ..., x_n) \longrightarrow ((((x_i\&x_1)\&...\&x_{i-1})\&x_{i+1})\&...\&x_n)$$

si $f = Z^n$

$$F(x_1, ..., x_n) \longrightarrow ((0\&x_1)\&...\&x_n)$$

Génération du système de réecriture

$$x\&0 \longrightarrow x$$

$$x\&S(y) \longrightarrow x\&y$$

$$\mathit{Ifz}(0, y, z) \longrightarrow y$$

$$\mathit{Ifz}(S(x), y, z) \longrightarrow z\&x$$

si $f = \pi_i^n$

$$F(x_1, ..., x_n) \longrightarrow ((((x_i\&x_1)\&...\&x_{i-1})\&x_{i+1})\&...\&x_n)$$

si $f = Z^n$

$$F(x_1, ..., x_n) \longrightarrow ((0\&x_1)\&...\&x_n)$$

si $f = S$

$$F(x) \longrightarrow S(x)$$

Génération du système de réecriture

$$x\&0 \longrightarrow x$$

$$x\&S(y) \longrightarrow x\&y$$

$$\mathit{Ifz}(0, y, z) \longrightarrow y$$

$$\mathit{Ifz}(S(x), y, z) \longrightarrow z\&x$$

si $f = \pi_i^n$

$$F(x_1, ..., x_n) \longrightarrow ((((x_i\&x_1)\&...\&x_{i-1})\&x_{i+1})\&...\&x_n)$$

si $f = Z^n$

$$F(x_1, ..., x_n) \longrightarrow ((0\&x_1)\&...\&x_n)$$

si $f = S$

$$F(x) \longrightarrow S(x)$$

si $f = +$ déjà vu, $\times$, $\chi_{\leq}$, facile

Génération du système de réecriture

si $f = \circ_m^n(h, g_1, \ldots, g_m)$, règles de $h, g_1, \ldots, g_m$ +

$$F(x_1, \ldots, x_n) \longrightarrow H(G_1(x_1, \ldots, x_n), \ldots, G_m(x_1, \ldots, x_n))$$

Génération du système de réecriture

si $f = \circ_m^n(h, g_1, \ldots, g_m)$, règles de $h, g_1, \ldots, g_m$ +

$$F(x_1, \ldots, x_n) \longrightarrow H(G_1(x_1, \ldots, x_n), \ldots, G_m(x_1, \ldots, x_n))$$

si $f = \mu^n(g)$ règles de g +

$$F(x_1, ..., x_n) \longrightarrow F'(x_1, ..., x_n, 0)$$

$$F'(x_1, ..., x_n, y) \longrightarrow \mathit{Ifz}(G(x_1, ..., x_n, y), y, F'(x_1, ..., x_n, S(y)))$$

Le théorème

Si $f(p_1, ..., p_n) = q$, alors $F(\underline{p_1}, ..., \underline{p_n}) \succ^* \underline{q}$

Le théorème

Si $f(p_1, ..., p_n) = q$, alors $F(\underline{p_1}, ..., \underline{p_n}) \succ^* \underline{q}$

Si f n'est pas définie en $p_1, \ldots, p_n$ et $F(\underline{p_1}, \ldots, \underline{p_n}) \rhd^* t$

alors t n'est pas irréductible

(toujours un sous-terme qui ne termine pas)

La suite

En PC : la confluence

La prochaine fois : le lambda-calcul et les machines de Turing

Questions?