LL et paradigmes logiques du calcul
Partie III: Logique linéaire, types et complexité
MPRI

Patrick Baillot

LIPN

CNRS - Université Paris 13

patrick.baillot@lipn.univ-paris13.fr

http://www-lipn.univ-paris13.fr/~baillot/

29/11/2006

Introduction

la LL est née de la décomposition de $\Rightarrow$ en $\multimap$ et ! :

analyse.

un statut logique est ainsi donné à la duplication d'argument

$\longrightarrow$ *synthèse*: modifier les règles du ! pour avoir une discipline plus stricte:
duplication modérée et *via élimination des coupures* caractérisation de classes de complexité.

Introduction (2)

pourquoi faire ...?

- un modèle logique, *implicite*, structuré, de caractérisation de la complexité, alternatif aux machines de Turing, circuits booléns. . . domaine de la *complexité implicite*
- étendre la correspondance de Curry-Howard à Ptime et aux classes de complexité, avec comme bénéfices associés:
  - langage de *programmation* ptime, ou *types* pour la vérification de complexité
  - systèmes pour *preuves* de terminaison ptime
  - sémantique: compréhension mathématique du calcul ptime: catég ories, jeux ?

Quelques objectifs de cette partie du cours

- une application des réseaux de preuves pour l'étude du lambda-calcul,
- calculer avec (des variantes de) la LL,
- illustrer l'utilisation de la LL pour les systèmes de types,
- applications de la LL à la complexité.

Plan

1. Logique linéaire élémentaire (ELL) et complexité élémentairement récursive:
   un premier pas, un système simple . . .
2. Logique linéaire light (LLL) et complexité Ptime:
   la complexité Ptime, via les réseaux de preuves.
3. types light simplifiés (DLAL):
   la complexité Ptime, dans le λ-calcul.
4. types linéaires pour le *calcul en place*.

Préliminaires: machines de Turing, complexité

notations sur les listes:
liste: $\langle a_1, \ldots, a_n \rangle$, concaténation: $l_1 :: l_2$
$|l|$ longueur de la liste.

Definition 1 *Machine de Turing: $\mathcal{M} = (Q, \Sigma, \delta)$, où:*

- *Q ensemble fini d'états, contenant q_i (resp. q_f) textitétat initial (resp. état final),*
- *$\Sigma = \{0, 1, \mathtt{b}\}$ alphabet, alphabet,*
- *$\delta : Q \times \Sigma \to Q \times \Sigma \times \{L, R\}$ fonction de transition.*

Configurations: $config = Q \times \Sigma^\star \times \Sigma^\star$

$(q, l_1, l_2) \in config$

l_1 (resp. l_2) est la partie gauche (resp. droite) de la bande de lecture.
Le symbole lu est le premier élément de l_2.

Préliminaires: suite

À partir de δ on définit: $step : config \to config$
configuration initiale (entrée= $w_0 \in \{0,1\}^\star$): $C_0 = (q_i, \langle\,\rangle, w_0)$

- $\mathcal{M}$ à partir de C_0 *s'arrête* sur $C' = step^n(C_0)$ ssi n est le plus petit entier tel que $step^n(C_0)$ a pour état q_f.
- $g : \{0,1\}^\star \to \{0,1\}^\star$
  $\mathcal{M}$ *représente* g ssi pour tout $w_0 \in \{0,1\}^\star$, $\mathcal{M}$ à partir de C_0 s'arrête sur $C = (q_f, w_1, w_2)$, avec:
  - $w_1 = g(w_0)$,
  - $w_2 = \langle \mathtt{b}, \ldots, \mathtt{b} \rangle$
- $\phi : \mathbb{N} \to \mathbb{N}$
  $\mathcal{M}$ *est de complexité en temps* ϕ ssi pour tout w_0, $\mathcal{M}$ à partir de C_0 s'arrête après au plus $\phi(|w_0|)$ étapes.

Préliminaires: suite

$g : \{0,1\}^\star \to \{0,1\}^\star$
La fonction g est *calculable en temps* ϕ (de complexité ϕ) s'il existe une machine $\mathcal{M}$ de complexité en temps ϕ qui la représente.

$FP = \{$fonctions $\{0,1\}^\star \to \{0,1\}^\star$ calculables en temps polynomial$\}$

$P = \{$fonctions $\{0,1\}^\star \to \{0,1\}$ calculables en temps polynomial$\}$

tour d'exponentielles: $K(d, n) = 2^{2^{\cdots^{2^n}}}$ (hauteur d)

Fonctions de temps *élémentaire* (ou *élémentairement récursives*):

$FElem = \{$fonctions g t.q. $\exists d$ t.q. g est calculable en temps $K(d, .)\}$

ex. de fonction primitive récursive mais non élémentaire: $n \to K(n, 2)$.

Logique linéaire élémentaire (ELL)

- correspond aux fonctions *élémentaires* : temps $2^{2^{\cdots^{2^n}}}$, à hauteur fixée. . . temps
- des applications en réduction optimale: simplification de l'algorithme de Lamping,
- GoI (géométrie de l'interaction)/ algèbre dynamique plus simples que pour LL.

ELL : définition

Pour ELL on garde le même langage de formules que pour LL, mais on va changer les règles d'introduction des modalités !, ?:

- on enlève une règle: déreliction,
- et on remplace la promotion par :

$$\frac{\vdash B_1, \ldots, B_n, A}{\vdash ?B_1, \ldots, ?B_n, !A}\ !\mathrm{E}$$

- la contraction et l'affaiblissement sont inchangés.

exercice: *on considère la règle digging:*

$$\frac{\vdash \Gamma, ??A}{\vdash \Gamma, ?A}\ dig$$

Mq le système avec comme règles exponentielles: ($!E$*, dig, der, contr., affaibl.) est équivalent à LL.*

ELL : quelques propriétés

- les principes $!A \multimap A$ (déreliction) et $!A \multimap !!A$ (digging) ne sont pas démontrables dans ELL;
- contrairement à CLL : un nombre infini de modalités (suites de $!, ?$) à équivalence près;
- étapes d'élimination des coupures: $!E/!E$, $!E/contr.$, $!E/affaibl.$
- traduction ELL $\rightarrow$ LL, compatible avec élimination des coupures (*simulation* de ELL vers LL) (à faire en exercice).

IELL : version intuitionniste

pour étudier l'expressivité de ELL on se restreint à une version intuitionniste: IELL, avec 2nd ordre (à la système F).
le langage de formules est celui de ILL (sans additifs)

$$A, B ::= \alpha \mid A \multimap B \mid A \otimes B \mid !A \mid \forall \alpha.A$$

Ainsi:

- séquents de la forme $\Gamma \vdash A$
- pour avoir plus de souplesse, on peut ajouter l'affaiblissement général : logique *affine* élémentaire (IEAL);
- pour l'intuition calculatoire : on peut voir IELL/IEAL comme un système de types pour le λ-calcul.
- la règle pour ! s'écrit:

$$\frac{B_1, \ldots, B_n \vdash A}{!B_1, \ldots, !B_n \vdash !A}\ !\,\mathrm{E}$$

Lambda-calcul

- lambda-termes:

$$t, u ::= x \mid \lambda x.t \mid (t\ u)$$

notations: $\lambda x_1 x_2.t$ pour $\lambda x_1.\lambda x_2.t$

$(t\ u\ v)$ pour $((t\ u)\ v)$

substitution (du terme u à var. x): $t\{x/u\}$

substitutions simultanées $t\{x/u_1, y/u_2\}$

substitution dans une formule $A\{\alpha/B\}$

$A \multimap B \multimap C$ pour $A \multimap (B \multimap C)$

- β-réduction:
clôture par contexte de:

$$(\lambda x.t)\ u \xrightarrow{1} t\{x/u\}$$

$\rightarrow$ clôture réflexive et transitive de $\xrightarrow{1}$.

IEAL et λ-calcul

$$\frac{}{x{:}A \vdash x{:}A}\ Id \qquad \frac{\Gamma_1 \vdash u{:}A \quad x{:}A, \Gamma_2 \vdash t{:}C}{\Gamma_1, \Gamma_2 \vdash t\{x/u\}{:}C}\ Cut$$

$$\frac{\Gamma_1 \vdash u{:}A_1 \quad x{:}A_2, \Gamma_2 \vdash t{:}C}{\Gamma_1, y{:}A_1 \multimap A_2, \Gamma_2 \vdash t\{x/(y\ u)\}{:}C}\ \multimap l \qquad \frac{x{:}A_1, \Gamma \vdash t{:}A_2}{\Gamma \vdash \lambda x.t{:}A_1 \multimap A_2}\ \multimap r$$

$$\frac{x{:}A\{\alpha/B\}, \Gamma \vdash t{:}C}{x{:}\forall\alpha.A, \Gamma \vdash t{:}C}\ \forall l \qquad \frac{\Gamma \vdash t{:}A}{\Gamma \vdash t{:}\forall\alpha.A}\ \forall r\ (\alpha \text{ non libre dans } \Gamma)$$

$$\frac{\Gamma \vdash t{:}C}{\Delta, \Gamma \vdash t{:}C}\ Weak \qquad \frac{x{:}!A, y{:}!A, \Gamma \vdash t{:}C}{z{:}!A, \Gamma \vdash t\{x/z, y/z\}{:}C}\ Cntr$$

$$\frac{\Gamma \vdash t{:}A}{!\Gamma \vdash t{:}!A}\ !r$$

Dans les règles binaires: contextes (Γ_1 et Γ_2) des deux prémisses ont variables disjointes.

Opération d'oubli: de IEAL à F

application $(.)^- : IEAL \to F$ définie par:

$$(!A)^- = A^-, \quad (A \multimap B)^- = A^- \to B^-, \quad (\forall\alpha.A)^- = \forall\alpha.A^-, \quad \alpha^- = \alpha.$$

Proposition 1 *Si* $\Gamma \vdash_{IEAL} t : A$ *alors* $\Gamma^- \vdash_F t : A^-$.

Si $A^- = T$: A est une *décoration* de T dans IEAL.

Types de données IEAL

- entiers unaires

système F:	ILL:	IEAL:
N^F		N^{IEAL}
$\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha)$	$\forall\alpha.!(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)$	$\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$

Exemple: $\underline{2}$, dans F:

$$\underline{2} = \lambda f^{(\alpha\to\alpha)}.\lambda x^{\alpha}.(f\ (f\ x))\ .$$

- listes binaires

système F:	IEAL:
W^F	W^{IEAL}
$\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha) \to (\alpha \to \alpha)$	$\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$

Exemple: $w = \langle 1, 0, 0\rangle$, dans F:

$$\underline{w} = \lambda s_0^{(\alpha\to\alpha)}.\lambda s_1^{(\alpha\to\alpha)}.\lambda x^{\alpha}.(s_1\ (s_0\ (s_0\ x)))\ .$$

Structures de preuves ELL

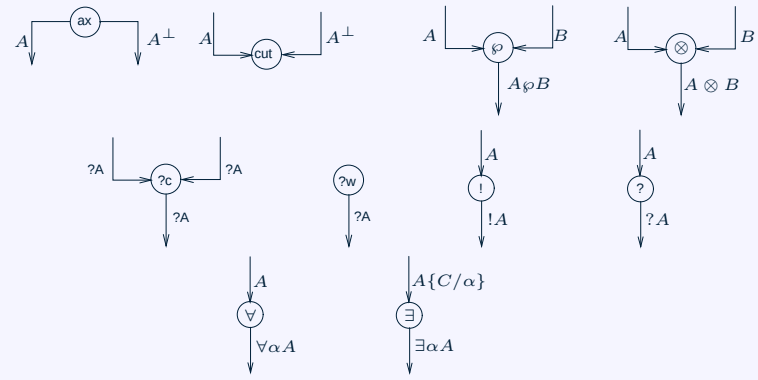
on va représenter les preuves par des réseaux de ELL *classique*

$A \multimap B = A^{\perp} \wp B$

$A^{\perp}$ défini par:

$$(A \otimes B)^{\perp} = A^{\perp} \wp B^{\perp} \quad (A \wp B)^{\perp} = A^{\perp} \otimes B^{\perp}$$
$$(\forall \alpha.A)^{\perp} = \exists \alpha.A^{\perp} \quad (\exists \alpha.A)^{\perp} = \forall \alpha.A^{\perp}$$
$$(!A)^{\perp} = ?A^{\perp} \quad (?A)^{\perp} = !A^{\perp}$$
$$\alpha^{\perp\perp} = \alpha$$

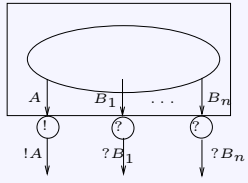
Structures de preuves ELL



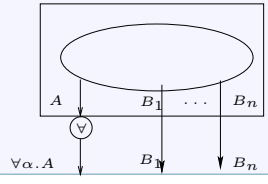
Structures de preuves: boîtes

2 boîtes sont disjointes ou l'une est incluse dans l'autre.

- boîte exponentielle (!-boîte)

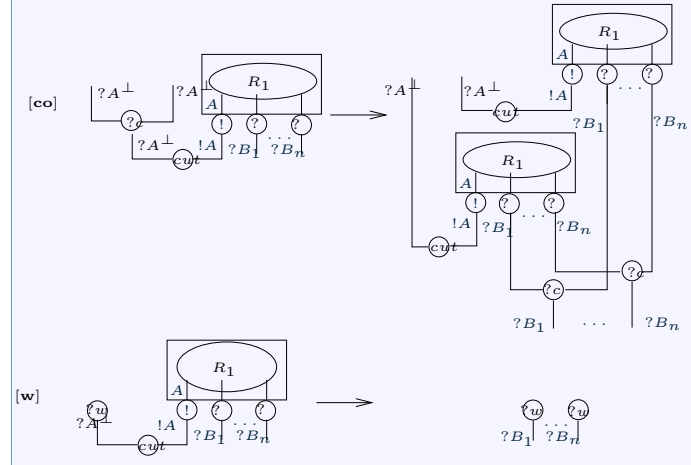


- boîte $\forall$: $\alpha \notin FV(B_i), 1 \leq i \leq n$



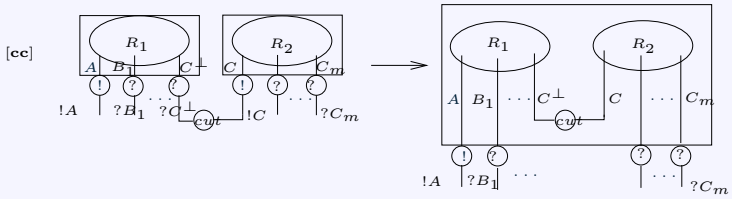
Réduction des réseaux

réduction de coupure contraction ou affaibl.: comme dans réseaux LL



Réduction des réseaux (suite)

réduction de coupure boîte-boîte: fusion



Exemples de termes

addition

$$add = \lambda nmfx.(n\ f)\ (m\ f\ x)$$
$$: N \multimap N \multimap N$$

multiplication

$$mult = \lambda nmf.(n\ (m\ f))$$
$$: N \multimap N \multimap N$$

carré

$$square = \lambda nf.(n\ (n\ f))$$
$$: !N \multimap !N$$

Normalisation des réseaux et réduction des termes

Proposition 2 (simulation ELL-Λ) *Si R est un réseau ELL correspondant à une preuve de conclusion $\Gamma \vdash t : A$ et si $R \to R'$ par réduction, alors R' correspond à une preuve de conclusion $\Gamma \vdash t' : A$, avec $t \to t'$.*

Proposition 3 *Si R est un réseau ELL correspondant à une preuve de $\Gamma \vdash t : A$ et si R est sans coupure, alors t est sous forme normale.*

Proposition 4 *Si $\Gamma \vdash t : A$, $t \to t'$ et t' est sous forme normale alors $\Gamma \vdash t' : A$.*

Normalisation des réseaux

R réseau ELL. e arête de R.
profondeur de e, $d(e)$: nombre de boîtes exponentielles qui contiennent e.
profondeur de noeud N: idem.
profondeur de R, $d(R)$: maximum des profondeurs de ses arêtes.

taille de R, $|R|$: nombre de noeuds de R
$|R|_i$: nombre de noeuds à profondeur i
$|R|_{i+}$: nombre de noeuds à profondeur $\geq i$

Proposition 5 (Stratification) *La profondeur d'une arête d'un réseau ELL ne change pas après une étape de réduction.*

Noter que ceci n'est pas vrai dans LL: la profondeur peut diminuer (étape déreliction) ou augmenter (étape boîte-boîte).

Borne de complexité sur la réduction

$K(0,n) = n$, $K(k+1,n) = 2^{K(k,n)}$.

Theorem 6 *Si R est un réseau de ELL, alors la réduction de R en sa forme normale peut être faite en un nombre d'étapes inférieur à $K(d+1, |R|)$.*

Remarques:

- hauteur de la tour ne dépend que de la profondeur, et pas de la taille;
- aucune référence aux formules apparaissant sur le réseau;
- cette borne est obtenue en appliquant une stratégie *par niveaux*.

Définitions pour la normalisation

?-noeud: noeud contraction, affaiblissement ou porte auxilliaire de !-boîte.
On dira qu'un ?-noeud dans R est *actif* s'il est *au-dessus* d'un noeud cut.

On considère R sans coupure à prof. $\leq i-1$, et avec uniquement des coupures exponentielles à profondeur i.
On définit $\prec_1$ sur les noeuds cut à prof. i par:

$N \prec_1 N'$ ssi:
il existe une !-boîte à prof. i dont N est au-dessous de la porte principale, et N' est au-dessous d'une porte auxill.

soit $\prec$ la clôture réflexive et transitive de $\prec_1$.

Lemma 7 *la relation $\prec$ est un ordre partiel.*

Stratégie de normalisation par niveaux

convention: pas d'axiome sur formules $!A$, $?A$ (on ajoute des boîtes)

- on procède par tours à profondeurs croissantes, de $i=0$ à $i=d$. à la fin du tour i: le réseau n'a plus de coupures à profondeur $\leq i$
- **tour** i:
  - phase (a): on élimine les coupures multiplicatives/axiomes/quantificateurs à profondeur i,
  - phase (b): on répète l'étape suivante jusqu'à ne plus avoir de coupure exponentielle à prof. i:

    étape: on réduit une coupure maximale pour $\prec$ à prof. i.

Cette stratégie est normalisante.

Calcul de bornes sur nombre d'étapes

dém. du théorème 6.

- tour i, phase (a): $|R|_i$ diminue strictement à chaque étape, donc le nombre d'étapes est majoré par $|R|_i$;
- tour i, phase (b):

  Lemma 8 *une étape de la phase (b) diminue strictement le nombre de ?-noeuds actifs à profondeur i.*

  Lemma 9 *si R est le réseau au début de phase (b), le nombre d'étapes de phase (b) est majoré par $|R|_i$.*

  Fait: à chaque étape de phase (a), $|R|_{(i+1)+}$ est inchangée.
  Fait: à chaque étape de phase (b), $|R|_{(i+1)+}$ est au plus multiplié par 2.
  Notons $R^{(i)}$ le réseau à la fin du tour i.

Représentation de fonctions

notation: $!^k A = !\ldots!A$ (k fois).

Π preuve de $x : N \vdash t : !^l N$.

on dit que π *représente la fonction* $f : \mathbb{N} \to \mathbb{N}$ si:

pour tout entier n la preuve obtenue en coupant $\vdash \underline{n} : N$ avec Π se réduit en $\vdash \underline{n'} : !^l N$, où $n' = f(n)$.

Proposition 10 *S'il existe une dérivation* Π *de* $x : N \vdash_{IELL} t : !^k N$ *alors* t *représente une fonction élémentairement récursive.*

En effet:

soit $n \in \mathbb{N}$ et Π'_n la preuve de $\vdash_{IELL} (t\ \underline{n}) : !^k N$ obtenue par coupure.

Alors $d(\Pi'_n) = max(1, d(\Pi))$ qui ne dépend pas de n.

De plus, $|\Pi'_n| = n + |\Pi| + cste$.

Theorem 11 *les fonctions représentables dans IEAL sont exactement les fonctions élémentairement récursives.*

Itération dans IEAL

on peut définir pour tout A un itérateur $iter_A$:

$$iter_A = \lambda f x n.\ (n\ f\ x)\ : !(A \multimap A) \multimap !A \multimap N \multimap !A$$

alors $(iter_A\ F\ t)\ \underline{n} \to (F\ (F\ \ldots (F\ t)\ldots))$ (n fois)

exemples:

$double : N \multimap N$

$exp = \lambda n.(iter_N\ double\ \underline{1})\ n\ : N \multimap !N$

remarque: exp ne peut pas être itérée: $(iter \exp\ \underline{1})$ non ELL typable.

$coerc_1 = \lambda n.(iter_N\ succ\ \underline{0})\ n : N \multimap !N$

coercition de N vers $!N$. plus généralement: $coerc_i : N \multimap !^i N$, pour $i \in \mathbb{N}$

conséquence: un terme $\vdash t : !^i N \multimap A$ peut être remplacé par $\vdash t' : N \multimap A$ identique extensionnellement:

$$t' = \lambda n.(t\ (coerc_i\ n))$$

Réduire la complexité ?

comment passer de ELL à une complexité polynomiale ?

essayons de limiter la *propagation des contractions*...

idée: on va distinguer dans les réseaux entre

- des boîtes *non* dupliquables,
- des boîtes dupliquables, mais à 1 seule entrée.

pour cela on considère une nouvelle modalité: §.

Logique linéaire light (LLL) [Girard95]

Version intuitionniste: ILLL.

Langage de formules

$$A, B ::= \alpha \mid A \multimap B \mid A \otimes B \mid !A \mid \S A \mid \forall \alpha.A$$

règles:

$$\frac{B \vdash A}{!B \vdash !A} \qquad \frac{\vdash A}{\vdash !A}$$

$$\frac{\Gamma, \Delta \vdash A}{!\Gamma, \S\Delta \vdash \S A}$$

les autres règles sont inchangées.

- on considérera la version affine : light affine logic (ILAL);
- comme pour IEAL on utilisera ILAL comme un système de types pour le λ-calcul.

ILAL: remarques

- la règle $\S$ peut être vue comme une sorte de déreliction multiple, avec un marqueur $\S$;
- d'un point de vue typage: $!A$ sous-type de $\S A$;
- d'un point de vue sémantique:
  $!$, $\S$ sont des foncteurs, et on a les principes
  $!A \multimap (!A \otimes !A)$
  $!A \multimap \S A \quad \S A \otimes \S B \multimap \S(A \otimes B)$

ILAL et λ-calcul

$$\frac{}{x:A \vdash x:A}\ Id \qquad \frac{\Gamma_1 \vdash u:A \quad x:A, \Gamma_2 \vdash t:C}{\Gamma_1, \Gamma_2 \vdash t\{x/u\}:C}\ Cut$$

$$\frac{\Gamma_1 \vdash u:A_1 \quad x:A_2, \Gamma_2 \vdash t:C}{\Gamma_1, y:A_1 \multimap A_2, \Gamma_2 \vdash t\{x/(y\ u)\}:C}\ \multimap l \qquad \frac{x:A_1, \Gamma \vdash t:A_2}{\Gamma \vdash \lambda x.t:A_1 \multimap A_2}\ \multimap r$$

$$\frac{x:A\{\alpha/B\}, \Gamma \vdash t:C}{x:\forall\alpha.A, \Gamma \vdash t:C}\ \forall l \qquad \frac{\Gamma \vdash t:A}{\Gamma \vdash t:\forall\alpha.A}\ \forall r \ (\alpha \text{ non libre dans } \Gamma)$$

$$\frac{\Gamma \vdash t:C}{\Delta, \Gamma \vdash t:C}\ Weak \qquad \frac{x:!A, y:!A, \Gamma \vdash t:C}{z:!A, \Gamma \vdash t\{x/z, y/z\}:C}\ Cntr$$

$$\frac{x:B \vdash t:A}{x:!B \vdash t:!A}\ !r \qquad \frac{\vdash t:A}{\vdash t:!A}\ !r$$

$$\frac{\Gamma, \Delta \vdash t:A}{!\Gamma, \S\Delta \vdash t:\S A}\ \S r$$

Traduction ILAL -> IEAL

traduction $(.)^o : ILAL \longrightarrow IEAL$:

- $(!A)^o = (\S A)^o = !A^o$,
- $(.)^o$ commute aux autres connecteurs.

Ceci donne une traduction des preuves ILAL vers les preuves IEAL.
En fait il s'agit même d'une simulation.
Bien sûr, toutes les preuves IEAL ne sont pas dans l'image de $(.)^o$.

exercice: mq toute preuve IEAL *sans coupure* est l'image d'une preuve ILAL par $(.)^o$.

Comme pour IEAL, on a une application d'oubli : $(.)^- : ILAL \to F$.

Types de données ILAL

- entiers unaires

IEAL: N^{IEAL} : $\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$,

ILAL: N^{ILAL} : $\forall\alpha.!(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)$.

- listes binaires

IEAL: W^{IEAL} : $\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$,

ILAL W^{ILAL} : $\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)$.

Exemples

addition

$$\begin{array}{lcl} add & = & \lambda nmfx.(n\ f)\ (m\ f\ x) \\ & : & N \multimap N \multimap N \end{array}$$

double

$$\begin{array}{lcl} double & = & \lambda nfx.(n\ f)\ (n\ f\ x) \\ & : & !N \multimap \S N \end{array}$$

Itération dans ILAL

on a dans ILAL comme type pour l'itérateur:

$$iter_A = \lambda fxn.\ (n\ f\ x) : !(A \multimap A) \multimap \S A \multimap N \multimap \S A$$

$(iter_A\ F\ t)\ \underline{n} \to (F\ (F \ldots (F\ t) \ldots))$ (n fois)

exemples:

$add : N \multimap N \multimap N, \quad (add\ \underline{2}) : N \multimap N$

$double' = \lambda n.(iter_N\ (add\ \underline{2})\ \underline{0})\ n\ : N \multimap \S N$

$double'$ ne peut pas être itérée.

similairement:

$m : N \vdash (add\ m) : N \multimap N$, donc $m : !N \vdash (add\ m) : !(N \multimap N)$

la multiplication est obtenue par:

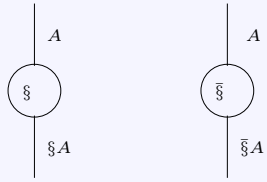
$mult' = \lambda nm.(iter_N\ (add\ m)\ \underline{0})\ n\ : !N \multimap N \multimap \S N$

Structures de preuves LLL

on représente les preuves par des structures de preuves de LLL *classique*.

$$(\S A)^\perp = \bar{\S} A^\perp \quad (\bar{\S} A)^\perp = \S A^\perp$$

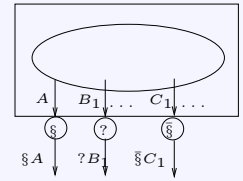
nouveaux noeuds:



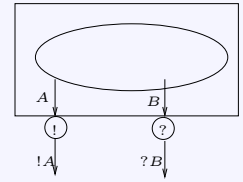
Structures de preuves LLL: boîtes

2 sortes de boîtes exponentielles:

§-boîte:



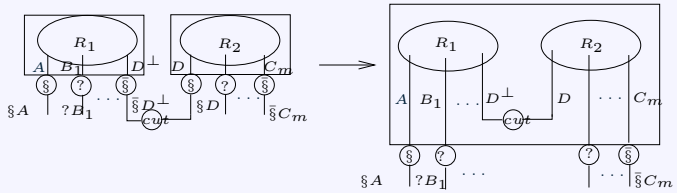
!-boîte:



Réduction des réseaux LLL

comme dans ELL, avec en plus le cas boîte-boîte $\S/\bar{\S}$:

[cc]



Élimination des coupures (LLL)

Proposition 12 (simulation LLL-Λ) *Si R est un réseau LLL correspondant à une preuve de conclusion $\Gamma \vdash t : A$ et si $R \to R'$ par réduction, alors R' correspond à une preuve de conclusion $\Gamma \vdash t' : A$, avec $t \to t'$.*

Borne de complexité sur la réduction

Comme les réseaux ELL, les réseaux LLL vérifient la ppté de stratification.

Theorem 13 *Si R est un réseau de LLL, de profondeur d, alors la réduction de R en sa forme normale peut être faite en un nombre d'étapes en $O((d+1).|R|^{2^{d+1}})$.*

Remarques:

- si la profondeur est fixée, alors le nombre d'étapes est polynomial en $|R|$;
- cette borne est obtenue avec une stratégie *par niveaux*.

Stratégie de normalisation

convention: pas d'axiome sur formules $!A$, $?A$ (on ajoute des boîtes)
Stratégie similaire à celle utilisée pour ELL:

- tours à profondeurs croissantes, de $i = 0$ à $i = d$.
- **tour** i:
  - phase (a): on élimine les coupures multiplicatives/axiomes/quantificateurs / § à profondeur i,
  - phase (b): on répète l'étape suivante jusqu'à ne plus avoir de coupure exponentielle à prof. i:

    étape: on réduit une coupure maximale pour $\prec$ à prof. i.

Calcul de bornes

dém. du théorème sur le nombre d'étapes de normalisation.
Soit R un réseau sans coupure à profondeur $< i$ et avec uniquement des coupures $!/?$ à profondeur i.

Definition 2 *Soit N un noeud ! ou § à prof. i dans R. On appelle potentiel de N, $pot(N)$:*

- $pot(N) = 0$ *si N est un noeud § ou ! sans coupure dessous,*
- *sinon* $pot(N) = c + pot(N_1) + \cdots + pot(N_n)$, *où:*
  - c *est le nombre de noeuds contractions de l'arbre coupé contre N,*
  - *les N_i sont les portes principales des boîtes à prof. i dont une porte auxilliaire est au-dessus de la coupure contre N.*

Remarque: $pot(N)$ est bien définie car $\prec$ est un ordre partiel.

Bornes: suite

Lemma 14

$$pot(N) \leq |R|_i - 1.$$

Definition 3 *Soit $B_i(R) = \sum_N (1 + 2.pot(N))$, où la somme porte sur les portes principales N de boîtes exp. à profondeur i.*

Lemma 15

$$B_i(R) \leq 2|R|_i^2.$$

Lemma 16 *$B_i(R)$ décroît strictement à chaque étape de la phase b (réduction de coupure exponentielle).*

Bornes: suite

Soit $t_i(R) = \sum_M (1 + pot(N_M))$, où:

- la somme porte sur les noeuds M à prof. $\geq (i+1)$,
- N_M est la porte principale de la boîte à prof. i contenant M.

Lemma 17 *Si R (resp. R') est le réseau au début (resp. à la fin) du tour i:*

- *$t_i(R)$ n'augmente pas au cours du tour i;*
- $|R'|_{(i+1)+} \leq t_i(R) \leq |R|_{i+}^2$.

Notons $R^{(i)}$ le réseau à la fin du tour i.

Lemma 18 *Pour $0 \leq i \leq (d-1)$ on a:* $|R^{(i)}|_{(i+1)+} \leq |R|^{2^{i+1}}$.

Le nombre d'étapes de la procédure est majoré par:

$$x = 3.|R|_0^2 + 3.|R^{(0)}|_1^2 + 3.|R^{(1)}|_2^2 + \cdots + 3.|R^{(d-1)}|_d^2$$

on obtient: $x \leq 3(d+1)|R|^{2^{d+1}}$.

Bornes: suite

Proposition 19 *Un réseau R de LLL de profondeur d peut être réduit en temps $O((d+1)|R|^{2^{d+2}})$.*

notation: $\S^k A = \S \ldots \S A$ (k fois).

Conséquence:
si $\vdash t : W \multimap \S^k W$ alors t représente une fonction de FP (calculable en temps polynomial).

Coercitions de types

$$iter_A = \lambda fxn.\ (n\ f\ x)\ : !(A \multimap A) \multimap \S A \multimap N \multimap \S A$$

On peut définir une coercition de N vers $\S N$:
$coerc_1 = \lambda n.(iter_N\ succ\ \underline{0})\ : N \multimap \S N$.
Plus généralement: $coerc_{i,j} : N \multimap \S^{i+1}!^j N$, avec $i, j \geq 0$.

De $mult' : !N \multimap N \multimap \S N$, on obtient
$mult'' : N \multimap N \multimap \S^2 N$,
puis:
$square'' : !N \multimap \S^3 N$,

Même chose pour le type W. Ceci nous permet alors de nous ramener à des types de la forme $W \multimap \S^k W$.

Représentation de fonctions

pour le calcul Ptime le choix de la représentation des données est important: listes binaires.

Π preuve de $x : W \vdash t : \S^k W$.
On dit que Π représente la fonction $f : \{0,1\}^\star \to \{0,1\}^\star$ si:
pour tout w de $\{0,1\}^\star$, la preuve obtenue en coupant $\vdash \underline{w} : W$ avec Π se réduit en $\vdash \underline{w'} : \S^k W$, où $w' = f(w)$.

Complexité de ILAL

Theorem 20 *Si $f : \{0,1\}^* \to \{0,1\}^*$ appartient à FP, alors il existe un entier k et une preuve Π de $x : W \vdash t : \S^k W$ qui représente f.*

Corollary 21 *Les fonctions représentables dans ILAL sont exactement les fonctions de FP, c'est-à-dire calculables en temps polynomial sur une machine de Turing.*

Remarques:

- même si un terme t est typable dans ILAL, pour l'exécuter avec la borne de complexité attendue on doit le compiler en un réseau de preuve;
- attention: un terme peut représenter une fonction Ptime et ne pas être typable dans ILAL.

Un système de types light plus simple : DLAL

DLAL: *Dual Light Affine Logic*
langage de types de DLAL :

$$A, B ::= \alpha \mid A \multimap B \mid A \Rightarrow B \mid \S A \mid \forall \alpha.A$$

traduction $(.)^* : DLAL \longrightarrow LAL$:

- $(A \Rightarrow B)^* = !A^* \multimap B^*$,
- $(.)^*$ commute aux autres connecteurs.

Remarquer que les types suivants ne sont pas dans l'image de DLAL :

$$A \multimap !B, \quad \S!A \multimap B, \quad !A\ .$$

Pour typer dans DLAL : jugements mixtes $\Gamma; \Delta \vdash t : C$,
où Δ contexte *affine-linéaire*, Γ *non linéaire*.

DLAL

$$\frac{}{;x:A\vdash x:A}\ (Id) \qquad \frac{\Gamma_1;\Delta_1\vdash u:A \quad \Gamma_2;x:A,\Delta_2\vdash t:C}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2\vdash t\{x/u\}:C}\ (Cut)$$

$$\frac{\Gamma_1;\Delta_1\vdash u:A \quad \Gamma_2;x:B,\Delta_2\vdash t:C}{\Gamma_1,\Gamma_2;y:A\multimap B,\Delta_1,\Delta_2\vdash t\{x/(y\ u)\}:C}\ (\multimap l) \qquad \frac{\Gamma;x:A,\Delta\vdash t:B}{\Gamma;\Delta\vdash \lambda x.t:A\multimap B}\ (\multimap r)$$

$$\frac{;z:D\vdash u:A \quad \Gamma;x:B,\Delta\vdash t:C}{z:D,\Gamma;y:A\Rightarrow B,\Delta\vdash t\{x/(y\ u)\}:C}\ (\Rightarrow l)(*) \qquad \frac{x:A,\Gamma;\Delta\vdash t:B}{\Gamma;\Delta\vdash \lambda x.t:A\Rightarrow B}\ (\Rightarrow r)$$

$$\frac{\Gamma;x:A\{\alpha/B\},\Delta\vdash t:C}{\Gamma;x:\forall\alpha.A,\Delta\vdash t:C}\ (\forall l) \qquad \frac{\Gamma;\Delta\vdash t:A}{\Gamma;\Delta\vdash t:\forall\alpha.A}\ (\forall r),\ \alpha \text{ non libre dans } \Gamma,\Delta$$

$$\frac{\Gamma;\Delta\vdash t:C}{\Sigma,\Gamma;\Pi,\Delta\vdash t:C}\ (Weak) \qquad \frac{x:A,y:A,\Gamma;\Delta\vdash t:C}{z:A,\Gamma;\Delta\vdash t\{x/z,y/z\}:C}\ (Cntr)$$

$$\frac{;\Gamma,x_1:B_1,\ldots,x_n:B_n\vdash t:A}{\Gamma;x_1:\S B_1,\ldots,x_n:\S B_n\vdash t:\S A}\ (\S)$$

(*) $z:D$ peut être absent.

Dérivations de type dans DLAL

Proposition 22 *si* $\Gamma;\Delta\vdash_{DLAL} t:A$ *et* $x\in\Delta$*, alors* x *a au plus 1 occurrence dans* t *(*x *linéaire).*

Proposition 23 *si* $\Gamma;\Delta\vdash_{DLAL} t:A$ *alors* $!\Gamma^*,\Delta^*\vdash_{ILAL} t:A^*$*.*
On obtient ainsi une simulation de DLAL dans ILAL.

La *profondeur* d'une dérivation DLAL est la profondeur de la dérivation ILAL correspondante.

Types de données DLAL

- entiers unaires

ILAL: N^{ILAL} : $\forall\alpha.!(\alpha\multimap\alpha)\multimap\S(\alpha\multimap\alpha)$,

DLAL N^{DLAL} : $\forall\alpha.(\alpha\multimap\alpha)\Rightarrow\S(\alpha\multimap\alpha)$.

- listes binaires

ILAL: W^{ILAL} : $\forall\alpha.!(\alpha\multimap\alpha)\multimap!(\alpha\multimap\alpha)\multimap\S(\alpha\multimap\alpha)$,

DLAL W^{DLAL} : $\forall\alpha.(\alpha\multimap\alpha)\Rightarrow(\alpha\multimap\alpha)\Rightarrow\S(\alpha\multimap\alpha)$.

Types dans DLAL

exemples de types:

$$\vdash addition: \quad N\multimap N\multimap N$$
$$\vdash double': \quad N\Rightarrow\S N$$
$$\vdash length: \quad W\multimap N$$

Les fonctions représentables dans DLAL sont données par les types $W\multimap\S^k W$.

Typage de l'itérateur:

$$iter_A=\lambda fxn.\ (n\ f\ x)\ :(A\multimap A)\Rightarrow\S A\multimap N\multimap\S A$$

Exemple: soit $Bool=\forall\alpha.\alpha\multimap\alpha\multimap\alpha$
$parite=(iter_{Bool}\ neg\ true):\ N\multimap\S Bool$

Coercitions dans DLAL

dans ILAL: $coer^{p,q} : N \multimap \S^{p+1}!^q N$

dans DLAL, les règles suivantes sont dérivables:

$$\frac{n : N; \Delta \vdash t : A}{; m : N, \S\Delta \vdash C_1[t] : \S A} \text{ (coerc1)} \qquad \frac{\Gamma; n : \S N, \Delta \vdash t : A}{\Gamma; m : N, \Delta \vdash C_2[t] : A} \text{ (coerc2)}$$

avec C_i des contextes tels que $C_i[t]$ est extensionnellement équivalent à t (représente la même fonction)

exemple:

$mult \quad : \quad N \Rightarrow (N \multimap \S N)$

$mult' \quad : \quad N \multimap \S(N \multimap \S N)$ par (coerc1)

$mult'' \quad : \quad N \multimap N \multimap \S\S N$ par (coerc2)

Proposition 24 *Si $P \in \mathbb{N}[X]$, alors il existe un terme t_P représentant P et un entier k tels que: $\vdash_{DLAL} t_P : N \multimap \S^k N$.*

DLAL: bornes de complexité

Proposition 25 *DLAL satisfait la propriété de subject-reduction pour la β réduction.*

Theorem 26 (borne polynomiale forte) *Si t est typable dans DLAL avec une dérivation de profondeur d, alors toute suite de β-réduits de t est de longueur majorée par $O((d+1).|t|^{2^{d+1}})$.*

Remarques:

- il s'agit ici de β-réduction, et plus de réduction des réseaux;
- cette borne s'applique pour toutes les stratégies de réduction;
- en particulier, si $\vdash t : W \multimap \S^k W$ alors on peut normaliser $(t\ \underline{w})$ en un nombre d'étapes polynomial en $longueur(w)$.

Theorem 27 *Les fonctions représentables par des termes typables dans DLAL sont exactement les fonctions de FP.*

Simulation machines de Turing Ptime dans DLAL (esquisse)

(voir [AspertiRoversi02] pour une démonstration complète, pour ILAL).

$Config$: type de données pour configurations d'une machine de Turing (bande et état).

$\vdash init : W \multimap Config$

$\vdash length : W \multimap N$: longueur d'une liste binaire

étant donnée une machine $\mathcal{M}$ avec polynôme de temps P, on construit alors:

$\vdash step : Config \multimap Config$ représentant 1 pas de la machine

$\vdash t_P : N \multimap \S^k N$

puis, avec $c_0 : Config$ on définit: $s = (iter_{Config}\ step\ c_0)$.

alors on a:

$$; c_0 : \S Config \vdash s : N \multimap \S Config$$

ainsi $(s\ \underline{n})$ calcule la configuration obtenue après n pas de $\mathcal{M}$ à partir de c_0.

Simulation machines de Turing (2/3)

On a:

$; c_0 : \S Config \quad \vdash \quad s : N \multimap \S Config$

$; c_0 : \S^{k+1} Config \quad \vdash \quad s : \S^k N \multimap \S^{k+1} Config$

par ailleurs: $\vdash \quad t_P : N \multimap \S^k N$

d'où:

$; w_1 : W,\ c_0 : \S^{k+1} Config \quad \vdash \quad [s\ (t_P\ (length\ w_1))] : \S^{k+1} Config$

en utilisant $init : W \multimap Config$ on obtient alors un terme s':

$$; w_1 : W,\ w_0 : \S^{k+1} W \quad \vdash \quad s' : \S^{k+1} Config$$

Simulation machines de Turing (3/3)

$$; w_1 : W,\ w_0 : \S^{k+1} W \quad \vdash \quad s' : \S^{k+1} Config$$

avec les coercitions: $; w_1 : W, w_0 : W \vdash s'' : \S^{k+1} Config$

puis par contraction et abstraction:

$$\vdash \quad M : W \Rightarrow \S^{k+2} Config$$

en composant avec $\vdash extract : Config \multimap W$ et coercition on a finalement: $\vdash \quad M' : W \multimap \S^{k+3} W$

Ce terme M' simule notre machine $\mathcal{M}$ de départ.

Conclusion et discussion

- la LL offre une approche logique de la complexité;
- cette approche fait un lien entre
  théorie de la complexité d'une part,
  et programmation fonctionnelle, typage, preuve formelle de l'autre...
- cependant, une limitation pratique :
  toutes les *fonctions* de FP sont représentables, mais certains termes/algorithmes (polynomiaux) courants ne sont pas typables.
  problème: le typage empêche certaines imbrications d'itérations.
  → d'autres systèmes plus flexibles ?

Exemple

on considère 2 exemples de programmes définis par itérations successives:
double / exponentiation, puis insertion / tri:

$double : L(A) \to L(A)$

$$\begin{cases} double(\texttt{nil}) & = & \texttt{nil} \\ double(a :: l) & = & a :: a :: double(l) \end{cases}$$

$exp : L(A) \to L(A)$

$$\begin{cases} exp(\texttt{nil}) & = & [a_0] \\ exp(a :: l) & = & double(exp(l)) \end{cases}$$

$insert : L(A) \to A \to L(A)$

$$\begin{cases} insert(\texttt{nil}, a) & = & [a] \\ insert(a' :: l', a) & = & \texttt{if } a' \leq a \texttt{ then} \\ & & \quad a' :: (insert(l', a)) \\ & & \texttt{else } a :: (insert(l', a')) \end{cases}$$

$sort : L(A) \to L(A)$

$$\begin{cases} sort(\texttt{nil}) & = & \texttt{nil} \\ sort(a :: l) & = & insert(a, sort(l)) \end{cases}$$

cependant: exp n'est *pas* de temps polynomial, mais $sort$ l'est ...

dans ILAL, le codage *naturel* donne:
$double : L(A) \multimap \S L(A)$ $\quad insert : L(A) \multimap \S(A \multimap L(A))$
et exp, $sort$ non ILAL-typables.

→ ILAL exclue les 2 cas. comment les distinguer ?

Types linéaires pour le calcul *en place*

on va considérer un autre système de types pour le temps polynomial, moins général mais incluant plus d'algorithmes courants.

idée:
en général, l'itération d'une fonction f polynomiale en temps ne donne pas une fonction polynomiale en temps (ex: itération de la fonction $double$)
→ on va contrôler aussi l'espace de calcul des fonctions, et ne permettre l'itération que de fonctions calculant *en place* (n'utilisant que l'espace de l'argument)

Types linéaires pour le calcul *en place*

on considère un système de types linéaires pour le calcul *non-size-increasing* (calcul en place) [Hofmann03], qu'on va appeler NSI.

langage de types

$$A, B ::= \Diamond \mid \mathtt{Bool} \mid A \multimap B \mid A \otimes B \mid \mathtt{L}(A)$$

langage de termes:

$$t, u \quad ::= \quad x \mid c \mid \lambda x.t \mid (t\ u) \mid t \otimes u \mid \mathtt{let}\ t\ \mathtt{be}\ x \otimes y\ \mathtt{in}\ u \mid \mathtt{if} \quad \mid \mathtt{iter}_B^{\mathtt{L}(A)}\ t\ u$$

où c sont des constructeurs:

$$\mathtt{true},\ \mathtt{false},\ \mathtt{nil}_A,\ \mathtt{cons}_A,$$

remarque: intuition pour type $\Diamond$: pointeur vers mémoire libre.

Règles de typage pour NSI

$$\frac{}{x:A \vdash x:A} \qquad \frac{\Gamma \vdash t:C}{\Delta, \Gamma \vdash t:C}$$

$$\frac{\Gamma_1 \vdash t:A \multimap B \quad \Gamma_2 \vdash u:A}{\Gamma_1, \Gamma_2 \vdash (t\ u):B} \qquad \frac{x:A, \Gamma \vdash t:B}{\Gamma \vdash \lambda x.t:A \multimap B}$$

$$\frac{\Gamma_1 \vdash t:A \otimes B \quad x:A, y:B, \Gamma_2 \vdash u:C}{\Gamma_1, \Gamma_2 \vdash \mathtt{let}\ t\ \mathtt{be}\ x \otimes y\ \mathtt{in}\ u:C} \qquad \frac{\Gamma_1 \vdash t:A \quad \Gamma_2 \vdash u:B}{\Gamma_1, \Gamma_2 \vdash t \otimes u:A \otimes B}$$

$$\frac{\vdash t:\Diamond \multimap A \multimap B \multimap B \quad \vdash u:B}{\vdash \mathtt{iter}_B^{\mathtt{L}(A)}\ t\ u : \mathtt{L}(A) \multimap B}$$

$$\frac{\vdash \mathtt{true}:\mathtt{Bool}}{\vdash \mathtt{cons}_A:\Diamond \multimap A \multimap \mathtt{L}(A) \multimap \mathtt{L}(A)} \qquad \frac{\vdash \mathtt{false}:\mathtt{Bool}}{\vdash \mathtt{nil}_A:\mathtt{L}(A)}$$

$$\frac{}{\vdash if:\mathtt{Bool} \multimap A \multimap A \multimap A}$$

condition pour les règles binaires: Γ_1 et Γ_2 n'ont pas de variable en commun.

Typage: remarques

notations: $|t|$ taille du terme t;
$FV(t)$: occurrences de variables libres de t;
$|FV(t)|$: nombre d'occurrences de variables libres.

- Remarquer qu'il n'y a pas de contraction et:

  Lemma 28 *Si $x_1 : A_1, \ldots, x_n : A_n \vdash_{NSI} t : B$, alors x_i a au plus une occurrence dans t et $FV(t) \subseteq \{x_1, \ldots, x_n\}$.*

- Noter que le typage NSI impose que l'itérateur $\mathtt{iter}_B^{\mathtt{L}(A)}$ ait ses (2 premiers) arguments clos.

Listes

Représentation de la liste $\langle a_1, \ldots, a_n \rangle$ de type $\mathtt{L}(A)$:

$$l = (\mathtt{cons}_A\ d_1\ a_1\ (\mathtt{cons}_A\ d_2\ a_2\ \ldots (\mathtt{cons}_A\ d_n\ a_n\ \mathtt{nil}_A)\ldots))$$

où chaque d_i est une variable libre de type $\Diamond$.
On a:

$$d_1 : \Diamond, \ldots, d_n : \Diamond \vdash_{NSI} l : \mathtt{L}(A)$$

Pour représenter les listes binaires on utilise le type $\mathtt{L}(\mathtt{Bool})$.

Réduction

$$
\begin{array}{rcl}
(\lambda x.t)u & \xrightarrow{1} & t\{x/u\} \\
(\mathtt{iter}_B^{\mathrm{L}(A)}\ t\ u)\ \mathtt{nil}_A & \xrightarrow{1} & u \\
(\mathtt{iter}_B^{\mathrm{L}(A)}\ t\ u)\ (\mathtt{cons}_A\ d\ a\ l) & \xrightarrow{1} & t\ d\ a\ (\mathtt{iter}_B^{\mathrm{L}(A)}\ t\ u\ l) \\
\mathtt{let}\ t_1 \otimes t_2\ \mathtt{be}\ x \otimes y\ \mathtt{in}\ u & \xrightarrow{1} & u\{x/t_1, y/t_2\} \\
\mathtt{if\ true}\ u\ v & \xrightarrow{1} & u \\
\mathtt{if\ false}\ u\ v & \xrightarrow{1} & v
\end{array}
$$

clôture de $\xrightarrow{1}$ par tous contextes, sauf abstractions λx et $\mathtt{let}\ t_1 \otimes t_2\ \mathtt{be}\ x \otimes y\ \mathtt{in}\ (.)$ (ie on ne réduit pas sous λ et sous in : réduction *paresseuse*).

Proposition 29 (*Subject reduction*) *si* $\Gamma \vdash_{NSI} t : A$ *et* $t \xrightarrow{1} t'$, *alors* $\Gamma \vdash_{NSI} t' : A$.

Exemples de programmes

Ex 1: concaténation de listes: défini par:

$$
\begin{array}{rcl}
append & = & \mathtt{iter}_B^{\mathrm{L}(A)}\ t\ u \text{ avec } B = \mathrm{L}(A) \multimap \mathrm{L}(A) \\
\text{et} & & \\
u & = & \lambda l^{\mathrm{L}(A)}.l \\
t & = & \lambda d^{\Diamond}.\lambda a^{A}.\lambda f^{B}.\lambda l'^{\mathrm{L}(A)}.\mathtt{cons}_A\ d\ a\ (f\ l') \ : \Diamond \multimap A \multimap B \multimap B \\
 & \vdash & append : \mathrm{L}(A) \multimap \mathrm{L}(A) \multimap \mathrm{L}(A)
\end{array}
$$

Ex 2: Supposons qu'on ait un terme clos $\vdash f : A \multimap B$;
on définit alors $\vdash F : \mathrm{L}(A) \multimap \mathrm{L}(B)$ ("$mapf$") par:

$$
\begin{array}{rcl}
F & = & \mathtt{iter}_B^{\mathrm{L}(A)}\ t\ \mathtt{nil}_B \\
\text{avec} & & \\
t & = & \lambda d^{\Diamond}.\lambda a^{A}.\lambda l'^{\mathrm{L}(B)}.(\mathtt{cons}_A\ d\ (f\ a))\ l' \ : \Diamond \multimap A \multimap \mathrm{L}(B) \multimap \mathrm{L}(B)
\end{array}
$$

Exemples (suite)

Ex 3: tri par insertion
supp. que A représente un ens. totalemt ordonné et qu'on ait
$comp : A \multimap A \multimap A \otimes A$ tq:
$comp\ a_1\ a_2 \to a_1 \otimes a_2$ si $a_1 \leq a_2$, $comp\ a_1\ a_2 \to a_2 \otimes a_1$ si $a_2 \leq a_1$.
On définit:

$$
\begin{array}{rcl}
insert & : & \mathrm{L}(A) \multimap \Diamond \multimap A \multimap \mathrm{L}(A) \\
sort & : & \mathrm{L}(A) \multimap \mathrm{L}(A)
\end{array}
$$

par

$$
insert = \mathtt{iter}_B^{\mathrm{L}(A)}\ t^{\Diamond \multimap A \multimap B \multimap B}\ u^B \text{ avec } B = \Diamond \multimap A \multimap \mathrm{L}(A)
$$

et

$$
\begin{array}{rcl}
u & = & \lambda d^{\Diamond}.\lambda a^{A}.\mathtt{cons}_A\ d\ a\ \mathtt{nil}_A \\
t & = & \lambda d^{\Diamond}.\lambda a^{A}.\lambda f^{B}.\lambda d'^{\Diamond}.\lambda a'^{A}. \\
 & & \quad \mathtt{let}\ comp\ a\ a'\ \mathtt{be}\ a_1 \otimes a_2\ \mathtt{in}\ \mathtt{cons}_A\ d\ a_1\ (f\ d'\ a_2)
\end{array}
$$

Exemples (suite)

(**Ex 3:** tri par insertion, suite)
On a: $\vdash insert : \mathrm{L}(A) \multimap \Diamond \multimap A \multimap \mathrm{L}(A)$
Soit $insert' = \lambda d^{\Diamond}.\lambda a^{A}.\lambda l^{\mathrm{L}(A)}.(insert\ l\ d\ a)$

$$\vdash insert' : \Diamond \multimap A \multimap \mathrm{L}(A) \multimap \mathrm{L}(A)$$

On pose:

$$sort = \mathtt{iter}_{\mathrm{L}(A)}^{\mathrm{L}(A)}\ insert'\ \mathtt{nil}_{\mathrm{L}(A)}$$

d'où:

$$\vdash sort : \mathrm{L}(A) \multimap \mathrm{L}(A).$$

Propriétés

Lemma 30 *si $\Gamma \vdash t : \Diamond$, alors t a au moins une variable libre ($\Gamma \neq \emptyset$).*

Proposition 31 *Si $\vdash t : \mathtt{L}(A) \multimap \mathtt{L}(A)$, alors t représente une fonction f telle que: si $f(l) = l'$ alors $longueur(l') \leq longueur(l)$.*

Borne sur réduction

A chaque terme t on associe un polynôme P_t, avec en particulier:

$$P_{(t\ u)} = P_t + P_u,$$
$$P_t = 0 \quad \text{si } t = x \text{ ou } c.$$

Theorem 32 *Si t est typé et $t \to t'$ en N étapes, alors:*

$$N \leq P_t(|FV(t)|) + |t|.$$

Corollary 33 *Si $\vdash t : \mathtt{L}(\mathtt{Bool}) \multimap \mathtt{L}(\mathtt{Bool})$, alors il existe un polynôme P tel que:*
pour tout $l : \mathtt{L}(\mathtt{Bool})$, toute réduction de $(t\ l)$ a au plus $P(longueur(l))$ étapes.

Expressivité

toutes les fonctions de FP ne sont pas représentables dans ce système (contrairement à LLL), mais on a:

Theorem 34 *soit $f : \{0,1\}^\star \to \{0,1\}^\star$ une fonction calculable par une machine de Turing en espace linéaire et en temps polynomial, et telle que pour tout x , $longueur(f(x)) \leq longueur(x)$, alors:*
il existe un terme t tel que $\vdash t : \mathtt{L}(\mathtt{Bool}) \multimap \mathtt{L}(\mathtt{Bool})$ qui représente f.

Conclusion

- le système de types linéaires NSI pour le calcul non-size-increasing donne un système plus flexible pour caractériser certains programmes de temps polynomial,
- cependant contrairement aux systèmes ELL, LLL il n'offre pas de correspondance avec les preuves, ni certains traits comme: la duplication (contraction), le polymorphisme . . .

Computer Science, 318(1–2):163–180, 2004.

[Ter01] K. Terui. Light Affine Lambda-calculus and polytime strong normalization. In Proceedings of LICS'01, pages 209–220, 2001.

[Ter04] K. Terui. Light affine set theory: a naive set theory of polynomial time. in Studia Logica, Vol. 77, pp. 9-40, 2004

References

[AS02] K. Aehlig, H.Schwichtenberg. A syntactical analysis of non-size-increasing polynomial time computation. In ACM Transactions on Computational Logic 3(3): 383-401 (2002).

[AR02] A. Asperti and L. Roversi. Intuitionistic light affine logic. ACM Transactions on Computational Logic, 3(1):1–39, 2002.

[BM04] P. Baillot and V. Mogbil. Soft lambda-calculus: a language for polynomial time computation. In Proceedings of FOSSACS'04, LNCS, Springer.

[BT04b] P. Baillot and K. Terui. Light types for polynomial time computation in lambda-calculus. In Proceedings of LICS'04, pages 266–275, 2004.

[DJ03] V. Danos and J.-B. Joinet. Linear logic & elementary time. 183(1):123–137, 2003. Information and Computation.

[Gir98] J.-Y. Girard. Light linear logic. Information and Computation, 143:175–204, 1998.

[GSS92] J.-Y. Girard, A. Scedrov, and P. Scott. Bounded linear logic: A modular approach to polynomial time computability. Theoretical Computer Science, 97:1–66, 1992.

[H03] M. Hofmann. Linear types and non-size-increasing polynomial time computation. Information and Computation 183(1): 57-85 (2003)