Coercitions de types

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

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:

$$\begin{array}{lcll} mult & : & N \Rightarrow (N \multimap \S N) & \\ mult' & : & N \multimap \S(N \multimap \S N) & \text{par (coerc1)} \\ mult'' & : & N \multimap N \multimap \S\S N & \text{par (coerc2)} \end{array}$$

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:

$$\begin{array}{ll} ; c_0 : \S Config & \vdash \quad s : N \multimap \S Config \\ ; c_0 : \S^{k+1} Config & \vdash \quad s : \S^k N \multimap \S^{k+1} Config \end{array}$$

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:
$double : L(A) \multimap \S L(A) \qquad insert : L(A) \multimap \S(A \multimap L(A))$
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:

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

et

$u = \lambda l^{\mathrm{L}(A)}.l$

$t = \lambda d^{\Diamond}.\lambda a^A.\lambda f^B.\lambda l'^{\mathrm{L}(A)}.\mathtt{cons}_A\ d\ a\ (f\ l')\ : \Diamond \multimap A \multimap B \multimap B$

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

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:

$F = \mathtt{iter}_B^{\mathrm{L}(A)}\ t\ \mathtt{nil}_B$

avec

$t = \lambda d^{\Diamond}.\lambda a^A.\lambda l'^{\mathrm{L}(B)}.(\mathtt{cons}_A\ d\ (f\ a))\ l'\ : \Diamond \multimap A \multimap \mathrm{L}(B) \multimap \mathrm{L}(B)$

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:

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

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

par

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

et

$u = \lambda d^{\Diamond}.\lambda a^A.\mathtt{cons}_A\ d\ a\ \mathtt{nil}_A$

$t = \lambda d^{\Diamond}.\lambda a^A.\lambda f^B.\lambda d'^{\Diamond}.\lambda a'^A.$

$\mathtt{let}\ comp\ a\ a'\ \mathtt{be}\ a_1 \otimes a_2\ \mathtt{in}\ \mathtt{cons}_A\ d\ a_1\ (f\ d'\ a_2)$

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:

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

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 . . .

[Laf04] Y. Lafont. Soft linear logic and polynomial time. Theoretical 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)