Follia project: “Logical foundations of abstract programming languages”

Soft Linear Logic, Lambda-Calculus and Intersection Types

Simona Ronchi Della Rocca, Marco Gaboardi

Dipartimento di Informatica - Università degli Studi di Torino

Outline

- The Proofs-as-Programs correspondence
  - from proofs to programs
  - from programs to proofs
  - from proofs to programs and back
- Soft Linear Logic
  - a Proofs-as-Programs correspondence for Soft Linear Logic
  - a problem: subject reduction
- Soft Type Assignment System
  - a different Proofs-as-Programs correspondence for Soft Linear Logic
  - complexity for the Soft Type Assignment System
- Comparison and Extension
  - Intersection Types
  - Soft Lambda Calculus

Proofs-as-Programs correspondence

We have mainly two ways to look methodologically at the Proofs-as-Programs correspondence:

- $\Rightarrow$ from proofs to programs, by defining language terms mimicking the logic in hand
- $\Leftarrow$ from programs to proofs, by defining type assignment system suitable for the language in hand

Clearly we have also $\Leftrightarrow$ like in Curry-Howard correspondence.

> So, for reasoning about programs, both the logical and the programming settings seem to be equally suitable. But sometimes it is possible to gain something by crossing the border.

An interesting methodological alternative, is obtained by mapping

$$\text{a proof } \mathfrak{P} \text{ to a program } \mathcal{P}$$

and then look for a map

$$\text{from } \mathcal{P} \text{ to another proof } \mathfrak{P}_2$$

not necessarily in the same setting. This approach seems interesting by considering algorithms instead of programs, since the correspondence gives us a way to translate proofs preserving the algorithmic content.

Soft Linear Logic

In [Laf04] Lafont has introduced soft linear logic (SLL).

> Soft Linear Logic can be seen as a subsystem of Bounded Linear Logic which is powerful enough to encode polynomial time.

Soft Linear Logic has the undoubted merit of hide the polynomials which appear explicitly in Bounded Linear Logic. Lafont has introduced an intuitionistic second order version of SLL named SLL_2 with the following connectives and quantifier: $\multimap$, $\otimes$, $\oplus$, $\&$, $!$, $\forall$. For the moment we consider only the fragment: $\multimap$, $!$, $\forall$

$$\frac{}{A \vdash A}\ (Id) \qquad \frac{\Gamma \vdash A \qquad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}\ (cut)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ (\multimap R) \qquad \frac{\Gamma \vdash A \qquad B, \Delta \vdash C}{A \multimap B, \Gamma, \Delta \vdash C}\ (\multimap L) \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \forall \alpha . A}\ (\forall R)$$

$$\frac{\Gamma, \overbrace{A, \ldots, A}^{n \text{ times}} \vdash C}{\Gamma, !A \vdash C}\ (mpx) \qquad \frac{\Gamma \vdash A}{!\Gamma \vdash !A}\ (sp) \qquad \frac{A[C/\alpha], \Gamma \vdash B}{\forall \alpha . A, \Gamma \vdash B}\ (\forall L)$$

A Proofs-as-Programs for SLL

A “quite standard” way to decor SLL, see for example [MT03], is the following:

$$\frac{}{x : A \vdash_L x : A}\ (Id) \qquad \frac{\Gamma, x : A \vdash_L M : B}{\Gamma \vdash_L \lambda x.M : A \multimap B}\ (\multimap R)$$

$$\frac{\Gamma \vdash_L M : A \qquad x : B, \Delta \vdash_L N : C \qquad \mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Delta) = \emptyset \qquad y \text{ fresh}}{y : A \multimap B, \Gamma, \Delta \vdash_L N[yM/x] : C}\ (\multimap L)$$

$$\frac{\Gamma \vdash_L M : A \qquad \Delta, x : A \vdash_L N : B \qquad \mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Delta) = \emptyset}{\Gamma, \Delta \vdash_L N[M/x] : B}\ (cut)$$

$$\frac{\Gamma, x_1 : A, \ldots, x_n : A \vdash_L M : C}{\Gamma, z : !A \vdash_L M[z/x_1, \cdots, z/x_n] : C}\ (mpx) \qquad \frac{\Gamma \vdash_L M : A}{!\Gamma \vdash_L M : !A}\ (sp)$$

$$\frac{\Gamma \vdash_L M : A}{\Gamma \vdash_L M : \forall \alpha.A}\ (\forall R) \qquad \frac{x : A[C/\alpha], \Gamma \vdash_L M : B}{x : \forall \alpha.A, \Gamma \vdash_L M : B}\ (\forall L)$$

This approach is clearly methodologically in the spirit of “from programs to proofs” ($\Leftarrow$) interpretation of proofs-as-programs correspondence.

Problem: subject reduction - 1

For the above system subject reduction is not valid. In fact it could be applied the following counter example due to Ugo Dal Lago. We can derive the following

$$\dfrac{\dfrac{}{s : B \multimap\, !A, w : B \vdash_L (\lambda z.sz)w : !A}\,(cut) \quad \dfrac{\dfrac{}{y : A \multimap A \multimap B, z : A, m : A \vdash_L yzm : B}}{y : A \multimap A \multimap B, x : !A \vdash_L yxx : B}\,(mpx)}{y : A \multimap A \multimap B, s : B \multimap\, !A, w : B \vdash_L y((\lambda z.sz)w)((\lambda z.sz)w) : B}\,(cut)$$

and since $(\lambda z.sz)w \to_\beta sw$ we want a proof with conclusion

$$y : A \multimap A \multimap B, s : B \multimap\, !A, w : B \vdash_L y((\lambda z.sz)w)(sw) : B$$

but such a proof doesn't exist. Instead we have the following proof, where $\tau = A \multimap A \multimap B$

$$\dfrac{\dfrac{\vdots}{s : B \multimap A, w : B \vdash_L (\lambda z.sz)w : A} \quad \dfrac{s' : B \multimap A, w' : B \vdash_L s'w' : A \quad y : \tau, z : A, m : A \vdash_L yzm : B}{y : \tau, z : A, s' : B \multimap A, w' : B \vdash_L yz(s'w') : B}\,(cut)}{\dfrac{y : A \multimap A \multimap B, s : B \multimap A, w : B, s' : B \multimap A, w' : B \vdash_L y((\lambda z.sz)w)(s'w') : B}{y : A \multimap A \multimap B, s :!(B \multimap A), w :!B \vdash_L y((\lambda z.sz)w)(sw)) : B}\,(mpx)}$$

Problem: subject reduction - 2

The problem in the above counter example is that to a term, i.e. $(\lambda z.sz)w$ can be assigned an exponential type $!A$ without assigning an exponential type to the free variables:

$$s : B \multimap !A, w : B \vdash_L (\lambda z.sz)w : !A$$

and this exponential type permits to duplicate the term, without duplicate the context , by means of parallel substitution.

$$y : A \multimap A \multimap B, s : B \multimap !A, w : B \vdash_L y((\lambda z.sz)w)((\lambda z.sz)w) \equiv yxx[(\lambda z.sz)w/x] : B$$

Now the same cannot be done in the case of

$$y : A \multimap A \multimap B, s : B \multimap !A, w : B \vdash_L y((\lambda z.sz)w)(sw) : B$$

because we need two substitutions, in fact

$$y : A \multimap A \multimap B, s : B \multimap !A, w : B \vdash_L y((\lambda z.sz)w)(sw) = yzm[(\lambda z.sz)w/z][sw/m] : B$$

Problem: subject reduction - 3

But to do this, by the constraint on the domains in the cut rule, we need that the two substitutions have different domains i.e. different variables

$$y : A \multimap A \multimap B, s : B \multimap A, w : B, s'x : B \multimap A, w' : B \vdash_L$$
$$y((\lambda z.sz)w)(s'w') \equiv yzm[(\lambda z.sz)w/z][s'w'/m] : B$$

and so the original term could be recovered only by means of the exponential rules

$$y : A \multimap A \multimap B, s :!(B \multimap A), w :!B \vdash_L$$
$$y((\lambda z.sz)w)(sw) \equiv yzm[(\lambda z.sz)w/z][s'w'/m][s/s][s/s'][w/w][w/w'] : B$$

which change the context, in particular the type of the variables. A counter example analogous to the one above is the following:

$$\dfrac{w : B \vdash_L (\lambda z.z)w : B \qquad \dfrac{\dfrac{}{y : A \multimap A \multimap B, z : A, m : A \vdash_L yzm : B}}{y : A \multimap A \multimap B, x :!A \vdash_L yxx : B}\,(mpx)}{y : A \multimap A \multimap B, s : B \multimap !A, w : B \vdash_L y(s(\lambda z.z)w))(s((\lambda z.z)w)) : B}\,(\multimap L)$$

Linear Substitution

The problem of subject reduction we have stressed above is related to some well known facts about linear logic and sequent calculus:

> sequent calculus offer different way to construct a term, such difference could be important in the presence of exponentials since the exponentials impose rigid constraints on contexts i.e. variable management, of these constraints could be keep trace in the terms but we have choose to do not do it.

The way we propose to overcome to this problems is by limiting the way of construction of terms to the ones which use only linear substitutions. If we write $M\{N\}$ to mean that N appears at most one time in M and $M\{N/x\}$ for the substitution of the unique (at most) occurrence of x by N then we propose rule which behave as:

$$\frac{\Gamma \vdash M : A \qquad \Delta, x : A \vdash N : C \qquad \mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset \qquad A \text{ linear type}}{\Gamma, \Delta \vdash N\{M/x\} : C}\ (cut)$$

$$\frac{\Gamma \vdash M : B \qquad \Delta, x : A \vdash N : C \qquad \mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset \qquad y \text{ fresh} \qquad A \text{ linear type}}{\Gamma, y : B \multimap A, \Delta \vdash N\{yM/x\} : C}\ (\multimap L)$$

The above side conditions in fact can be internalized and we do this in what follows.

Soft Type Assignment System (STA)

We can define the set $\mathbf{T}$ of Types as:

$$\begin{array}{lcll} A & ::= & a \mid \sigma \multimap A & \text{(Linear Types)} \\ \sigma & ::= & A \mid !\sigma & \end{array}$$

and the Soft Type Assignment System (STA) by the following rules:

$$\frac{}{x : A \vdash_T x : A}\ (Id) \qquad \frac{\Gamma, x : \sigma \vdash_T M : A}{\Gamma \vdash_T \lambda x.M : \sigma \multimap A}\ (\multimap R)$$

$$\frac{\Gamma \vdash_T M : \tau \qquad x : A, \Delta \vdash_T N : \rho \qquad \mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset \qquad y \text{ fresh}}{\Gamma, y : \tau \multimap A, \Delta \vdash_T N[yM/x] : \rho}\ (\multimap L)$$

$$\frac{\Gamma \vdash_T M : A \qquad \Delta, x : A \vdash_T N : \sigma \qquad \mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset}{\Gamma, \Delta \vdash_T N[M/x] : \sigma}\ (cut)$$

$$\frac{\Gamma \vdash_T M : \sigma}{!\Gamma \vdash_T M : !\sigma}\ (!) \qquad \frac{\Gamma, x_1 : \tau, ..., x_n : \tau \vdash_T M : \sigma}{\Gamma, x : !\tau \vdash_T M[x/x_1, ..., x/x_n] : \sigma}\ (m) \qquad \frac{\Gamma \vdash_T M : \sigma}{\Gamma, x : A \vdash_T M : \sigma}\ (weak)$$

Subject Reduction for STA - 1

Theorem 1 (Subject Reduction) $\Gamma \vdash_T M : \mu$ *and* $M \to_\beta M'$ *imply* $\Gamma \vdash_T M' : \mu$

Proof schema. By induction on the derivation, clearly the difficult case is

$$\dfrac{\dfrac{\Gamma, y : \sigma \vdash_T P : B}{\Gamma \vdash_T \lambda y.P : \sigma \multimap B}\ (\multimap R) \qquad \dfrac{\Theta \vdash_T Q : \sigma \qquad \Delta, z : B \vdash_T N\{z\} : \mu}{\Theta, \Delta, x : \sigma \multimap B \vdash_T N\{xQ\} : \mu}\ (\multimap L)}{\Gamma, \Theta, \Delta \vdash_T N[\lambda y.P/x] \equiv N\{(\lambda y.P)Q\} : \mu}\ (cut)$$

because to type the reduced term we need to make a cut like the following

$$\dfrac{\Theta \vdash_T Q : \sigma \qquad \Gamma, y : \sigma \vdash_T P : B}{\Theta, \Gamma \vdash_T P[Q/y] : B}\ (cut)$$

but this cut is not necessarily linear, hence it is not always available in STA, nevertheless it is easy to verify that given $\Theta \vdash_T Q : \sigma \equiv !^q C$ with $q \geq 0$, hence there is a derivation for

$$\Theta' \vdash_T Q : C$$

where Θ can be recovered by Θ' by a sequence of applications of rules (!) and $(weak)$.

Subject Reduction for STA - 2

At the same time we have that if P has n free occurrences of y, there is a derivation for

$$\Gamma, y_1 : C, ..., y_n : C \vdash_T P\{y_1/y, ..., y_n/y\} : B$$

and there are n applications of rules (Id) of the shape:

$$y_i : C \vdash_T y_i : C$$

Hence we can replace each one of these rules by:

$$\frac{y_i : C \vdash_T y_i : C \qquad \Theta'_i \vdash_T Q_i : C}{\Theta'_i \vdash_T Q_i : C}\ (cut)$$

where $Q_1, ..., Q_n$ are n disjoint copies of Q. Finally we can now recover

$$\Theta, \Gamma \vdash_T P[Q/y] : B$$

by a sequence of rules (m) collapsing all the involved Q_i's into Q.

Some consideration

- We conjecture that:

$$\Gamma \vdash_L M : A \iff \Gamma \vdash_T M : A'$$

  this fact could justify our approach, in fact our approach seems more interesting if we look to lambda-terms not as programs but as algorithms, then the above fact give us in the two systems different proofs with the same algorithmic content.

- The proof of subject reduction mimic the cut elimination process, this is not surprising, but a finer analysis permits us to note some interesting facts:
  - In our system cut elimination process is no longer definable by means of local rewriting rules, instead is necessary a global rewriting of the proofs.
  - In the rewriting all new generated cuts are on the rule (Id), so for these cuts the constraint on the linear types is trivially preserved.
  - The global rewriting rules on the proofs is in one to one correspondence with β-reduction on the terms.

- It is not surprising that in our system the rules for exponentials has lost their status:
  - The rule (!) is no longer really used inside the proofs.
  - The rule (m) could be considered as a way to safe space in writing algorithms.

Complexity in SLL

Consider a derivation Π for

$$\Gamma \vdash A$$

we have that the cut elimination process for Π take at most

$$\mathcal{O}(s^{d+2}) \text{ steps}$$

where s is the size of Π and the degree d is the maximum nesting of (!) rules in Π. Looking at the proof, an interesting point is that the bound on principal reduction (symmetric and axiom reduction,i.e. without commutative reduction) for the derivation Π given by adapting theorem 2 of [Laf04] is

$$s \times n^d$$

where s is the size of Π, n is the maximal rank of a multiplexor in Π and d is the degree of Π.

Complexity in STA

If we remove term information by derivation in STA for

$$\Gamma \vdash_T M : A$$

we obtain a derivation in SLL for

$$\Gamma \vdash A$$

for which we have Lafont bound:

$$s \times n^d$$

In our system a reduction step generates directly a number $\leq n$ of axiom-cuts, hence it perform in one step a number ≥ 1 of Lafont-step. This means that the Lafont's complexity limit holds trivially.
On the terms, a reduction step corresponds to a number $\geq 1, \leq m$ of substitutions of a single occurrence of a variable. So

$$s \times n^d$$

is a superior limit to the number of occurrence-substitutions necessary for reducing the term.

Polynomial Time Completeness

We can doubt that since our system STA is a subsystem of SLL completeness could fail, but it seems that this is not the case.
The types of data structures in SLL could be kept as the types of the same data structures in STA. Consider for example church numerals, in SLL we can give them the following type:

$$\mathbf{N} = \forall\alpha.!(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$$

clearly this is a linear type and we can encode church numerals with this type also in STA. Furthermore if we consider the other connectives $\&, \oplus$ and $\mathbf{1}$ of SLL we can encode Turing Machines with the type:

$$M = \forall\alpha.!((\alpha \multimap \alpha)\&(\alpha \multimap \alpha)) \multimap F_k \otimes F_3 \otimes ((\alpha \multimap \alpha) \otimes (\alpha \multimap \alpha))$$

where $F_k = \overbrace{\mathbf{1} \oplus \mathbf{1} \oplus \cdots \mathbf{1}}^{k \text{ times}}$. Clearly also M is linear and if we define the other connectives in STA then we can consider it the type of Turing Machine. An interesting question is:

Which consequence our approach has on the encoding of algorithm?

Intersection Types

In a previous work we have shown that we can add a multiplicative intersection connective $\cap_{\otimes}$ and an additive intersection connective $\cap_{\&}$ in a linear setting in the following way:

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash M : B}{\Gamma \vdash M : A \cap_{\&} B}\ (\cap_{\&} R) \qquad \frac{\Gamma, x : A \vdash M : C}{\Gamma, x : A \cap_{\&} B \vdash M : C}\ (\cap_{\&} L)$$

$$\frac{\Gamma \vdash M : A \quad \Delta \vdash M : B}{\Gamma \uplus \Delta \vdash M : A \cap_{\otimes} B}\ (\cap_{\otimes} R) \qquad \frac{\Gamma, x : A, y : B \vdash M : C}{\Gamma, z : A \cap_{\otimes} B \vdash M[z/x, z/y] : C}\ (\cap_{\otimes} L)$$

an important fact is that multiplicative intersection connective contains contraction modulo the equivalence

$$!A \cong !A \cap_{\otimes} !A$$

So we can think to extends typable terms in SLL by

substituting exponentials with multiplicative intersection

An interesting fact is that STA is related with the well known strict intersection types system.

Soft Linear Lambda Calculus

In [BM04] it was introduced a language based on Soft Linear Logic. Either if the complexity consideration are given on the untyped language, we could simplify its introduction by using the following rules:

$$\frac{}{x : A \vdash_P x : A}\ (Id) \qquad \frac{\Gamma, x : A \vdash_P M : B}{\Gamma \vdash_P \lambda x.M : A \multimap B}\ (\multimap R) \qquad \frac{\Gamma \vdash_P M : A}{\Gamma \vdash_P M : \forall \alpha.A}\ (\forall R)$$

$$\frac{\Gamma \vdash_P M : A \qquad x : B, \Delta \vdash_P N : C \qquad y \text{ fresh}}{y : A \multimap B, \Gamma, \Delta \vdash_P N[yM/x] : C}\ (\multimap L) \qquad \frac{x : A[C/\alpha], \Gamma \vdash_P M : B}{x : \forall \alpha.A, \Gamma \vdash_P M : B}\ (\forall L)$$

$$\frac{\Gamma \vdash_P M : A \qquad \Delta, x : A \vdash_P N :}{\Gamma, \Delta \vdash_P N[M/x] : B}\ (cut) \qquad \frac{x_1 : A_1, \ldots, x_n : A_n \vdash_P M : A}{y_1 :!A_1, \cdots, y_n :!A_n \vdash_P \text{let } \vec{y} \text{ be } !\vec{x} \text{ in } !M :!A}\ (sp)$$

$$\frac{\Gamma, x_1 : A, \ldots, x_n : A \vdash_P M : C}{\Gamma, z :!A \vdash_P \text{let } z \text{ be } !x \text{ in } M[z/x_1, \cdots, z/x_n] : C}\ (mpx)$$

For this systems the problem with subject reduction don't arise since it keep trace of exponentials also in the term.

This approach of Baillot and Mogbil is methodologically in the spirit of "from proofs to programs" $(\Rightarrow)$ interpretation of proofs-as-programs correspondence.

Future works

- Show that polynomial time Turing machine can be encoded in STA:
  - Could we follow the approach of Lafont in [Laf04] or the one of Mairson and Terui in [MT03]?
  - Which consequence our approach has on the encoding of algorithm?
- Investigate the extension of STA by mean of Intersection Types:
  - Could the addition of intersection types keep we above polynomial time?
  - How can we encode in an uniform way data structure with intersection types?
- Generalize the approach in order to assign pure lambda calculus to other logics:
  - Could the same approach be useful for Light Linear Logic?
  - Could it be interesting to extend the approach to assign a pure lambda calculus language to linear logic?

References

[BM04] Patrick Baillot and Virgile Mogbil. Soft lambda-calculus: A language for polynomial time computation. In *FoSSaCS*, pages 27–41, 2004.

[Laf04] Yves Lafont. Soft linear logic and polynomial time. *TCS: Theoretical Computer Science*, 318, 2004.

[MT03] Harry G. Mairson and Kazushige Terui. On the computational complexity of cut-elimination in linear logic. In *ICTCS: Italian Conference on Theoretical Computer Science*. LNCS, 2003.