Proof search for programming in Intuitionistic Linear Logic* (extended abstract)

D. Galmiche & E. Boudinet

CRIN-CNRS & INRIA Lorraine
Campus Scientifique - B.P. 239
54506 Vandœuvre-les-Nancy Cedex
France
e-mail: {galmiche,boudinet}@loria.fr

1 Introduction

Linear logic (denoted LL) [6] is a powerful and expressive logic with connections to a variety of topics in computer science as logic programming, concurrency or functional programming. From the logical side, LL combines the constructive content of Intuitionistic Logic with the symmetries of Classical Logic and from the computation side, it offers a control on resource management and evaluation order. Concerning functional programming, applications of LL to computation can be seen through the Curry-Howard isomorphism in which propositions are interpreted as types, proofs as programs and proof normalization process as computation. Works have been recently devoted to term assignment for intuitionistic linear logic (ILL) [3, 12] and full LL [1] with proposals of linear lambda calculi having important properties as subject-reduction or substitution property. Having natural deduction and sequent calculus proof systems of ILL (that are proved equivalent), we can investigate the problems of type inference and type safety [12] with the former one, but also apply the *programming with proofs* paradigm through mechanized program extraction from proofs with the latter. In this case, proof search corresponds to program synthesis and proof normalization to computation. The correspondence between ILL and Petri nets [13] or predicate ILL and high-level nets [11] illustrates the interest of proof search methods for proving specifications (and synthesizing programs). Let us mention that works on linear logic programming [9] and on concurrent programming based on *proof construction as computation* involve also specific logical ILL fragments. Thus, the study of proof search methods for automatic or interactive theorem proving is essential for such frameworks. A proposal for automatic proof development consists in defining subclasses of proofs that are complete and tractable as uniform proofs in [9]. But construction based on such proof definition cannot be applied to full ILL or some fragments. Thus we have to propose new normal proofs forms adequate to the logical fragment and to prove they are complete. In the context of program synthesis, it means we need to know how to build one proof or program, but our results about inference movements in a proof allows also to define possible proof transformations and thus another proofs with different computational contents.

*In Proceedings of CADE-12 Workshop on Proof search in type-theoretic languages, edited by D. Galmiche & L. Wallen, Nancy, June 1994

2 The linear logic framework

Intuitionistic Linear Logic is a refinement of intuitionistic logic where formulae must be used exactly once (weakening and contraction rules are removed). Recent works consider the problem of deriving a term assignment system for ILL for both natural deduction and sequent calculus proof systems [1, 3, 12]. We present the sequent calculus SEQ that has two important properties: *subject reduction* (terms reduction is well-typed) and *substitution property* (set of valid deductions closed by substitution). Computation arises from cut-elimination in SEQ and from normalization in natural deduction system. This system SEQ will be adequate for mechanizing proof search.

2.1 The linear term calculus

Let us present the essential of the syntax (see [12] for a complete presentation). Using x, y, z for term variables, and t, u, v for terms, the syntax of linear lambda terms for SEQ is the following: (x) variable, $(let\ x \times y\ be\ t\ in\ u)$ bind x to car and y to cdr of t in u, $(t \times u)$ eager pair (like cons in ML or Lisp), (tu) application, $(\lambda x.t)$ abstraction, $(inr(t))$ determines right branch of case, $(inl(t))$ determines left branch of case, $(case\ t\ of\ inl(x) \Rightarrow u, inr(y) \Rightarrow v$ evaluate t then branch, $< t, u >$ lazy pair, $(let\ < \_, x >\ be\ t\ in\ u)$ bind x to cdr of lazy pair, $(let\ < x, \_ >\ be\ t\ in\ u)$ bind x to car of lazy pair, $(let\ 1\ be\ t\ in\ u)$ evaluate t to 1, then become u, $(store\ u)$ store or delay u, $(discard\ t\ in\ u)$ throw away t, $(read\ store\ x\ as\ t\ in\ u)$ evaluate t to store t', bind x to t', $(copy\ x@y\ as\ t\ in\ u)$ binds x and y to t.
All the $let\ A\ be\ B\ in\ C$ constructs bind variables in A by pattern-matching A against the result of evaluating B, and then evaluate C. The term $store\ u$ is a reusable, or delayed version of u. The *copy* operation inserts multiple copies of a $store\ u$ term, while *discard* completely eliminates a $store\ u$ term. The *read* construct forces evaluation of a $store\ d$ term. The interaction between *read* and *store* is the critical point where the linear calculus determines reduction order. In other terminology, *store* is a wrapper or box which is only opened when the term must be *read*.
The *one-step reduction* rules for this linear lambda calculus are the following:

$$\begin{array}{lll}
\multimap_L & (\lambda x.v)u & \rightarrow_o\ \ v[u/x] \\
\otimes_L & let\ x \times y\ be\ (s \times t)\ in\ u & \rightarrow_o\ \ u[s/x, t/y] \\
\oplus_L & case\ inl(t)\ of\ inl(x) \Rightarrow u, inr(y) \Rightarrow v & \rightarrow_o\ \ u[t/x] \\
\oplus_L & case\ inr(t)\ of\ inl(x) \Rightarrow u, inr(y) \Rightarrow v & \rightarrow_o\ \ v[t/y] \\
\&_L & let\ < x, _ >\ be\ < v, t >\ in\ u & \rightarrow_o\ \ u[v/x] \\
\&_L & let\ < _, y >\ be\ < v, t >\ in\ u & \rightarrow_o\ \ u[t/y] \\
1_L & let\ 1\ be\ 1\ in\ u & \rightarrow_o\ \ u \\
Cong & \dfrac{t \rightarrow_o v}{C[t] \rightarrow_o C[v]} &
\end{array}$$

This reduction relation only allows non-linear reductions (involving the $!W_L$, $!D_L$, and $!C_L$ reduction rules) to apply at the top level of a term, in the empty context. However, the remaining linear reduction steps may be applied anywhere in a term. Thus reduction is neither a congruence with respect to all term formulation rules, nor is it deterministic. This reduction system can be shown to be confluent on untyped terms, even though not all untyped terms have a normal form [12]. The notation $t \rightarrow u$ must be read as t evaluates in any number of steps to u.

$$(I)\ \frac{}{t \rightarrow t} \qquad (Tr)\ \frac{t \rightarrow u \quad u \rightarrow v}{t \rightarrow v} \qquad (Lin)\ \frac{t \rightarrow_o v}{t \rightarrow v} \qquad (!W_L)\ discard\ t\ in\ u \rightarrow u$$

$$(!C_L)\ copy\ x@y\ as\ t\ in\ u \to u[t/x, t/y] \qquad (!D_L)\ \frac{t \to store\ v \quad v \to w}{read\ (store\ x)\ as\ t\ in\ u \to u[w/x]}$$

2.2 The ILL sequent calculus

Let us present now the rules of the SEQ system for ILL.

$$I\ \frac{}{x : A \vdash x : A} \qquad Cut\ \frac{\Sigma \vdash t : A \quad x : A, \Gamma \vdash u : B}{\Sigma, \Gamma \vdash u[t/x] : B}$$

$$\multimap_L\ \frac{\Sigma \vdash t : A \quad \Gamma, x : B \vdash u : C}{\Sigma, \Gamma, f : (A \multimap B) \vdash u[(ft)/x] : C} \qquad \multimap_R\ \frac{\Sigma, x : A \vdash t : B}{\Sigma \vdash \lambda x.t : (A \multimap B)}$$

$$\otimes_L\ \frac{\Sigma, x : A, y : B \vdash t : C}{\Sigma, z : (A \otimes B) \vdash let\ (x \times y)\ be\ z\ in\ t : C} \quad \otimes_R\ \frac{\Sigma \vdash t : A \quad \Gamma \vdash u : B}{\Sigma, \Gamma \vdash (t \times u) : (A \otimes B)}$$

$$\oplus_L\ \frac{\Sigma, x : A \vdash u : C \quad \Sigma, y : B \vdash v : C}{\Sigma, z : (A \oplus B) \vdash case\ z\ of\ inl(x) \Rightarrow u, inr(y) \Rightarrow v : C}$$

$$\&_R\ \frac{\Sigma \vdash t : A \quad \Sigma \vdash u : B}{\Sigma \vdash < t, u >: (A \& B)} \quad \oplus_R\ \frac{\Sigma \vdash t : A}{\Sigma \vdash inl(t) : (A \oplus B)} \quad \oplus_R\ \frac{\Sigma \vdash u : B}{\Sigma \vdash inr(u) : (A \oplus B)}$$

$$\&_L\ \frac{\Sigma, x : A \vdash t : C}{\Sigma, z : A \& B \vdash let\ < x, \_ >\ be\ z\ in\ t : C} \quad \&_L\ \frac{\Sigma, y : B \vdash t : C}{\Sigma, z : A \& B \vdash let\ < \_, x >\ be\ z\ in\ t : C}$$

$$!W_L\ \frac{\Sigma \vdash t : A}{\Sigma, z :!B \vdash discard\ z\ in\ t : A} \quad !D_L\ \frac{\Sigma, x : A \vdash t : B}{\Sigma, z :!A \vdash read\ (store\ x)\ as\ z\ in\ t : B}$$

$$!C_L\ \frac{\Sigma, x :!A, y :!A \vdash t : B}{\Sigma, z :!A \vdash copy\ x@y\ as\ z\ in\ t : B} \qquad !S_R\ \frac{!\Sigma \vdash t : A}{!\Sigma \vdash store\ t :!A}$$

$$1_L\ \frac{\Sigma \vdash t : A}{\Sigma, z : 1 \vdash let\ 1\ be\ z\ in\ t : A} \qquad 1_R\ \frac{}{\vdash 1 : 1}$$

This system has the *subject reduction property* [12] (i.e, if a term t has the type A and if t reduce to t' then t' has the type A) and the *substitution property* [12] (set of valid deductions is closed by substitution). Justification of term and rule forms and proofs of properties are in [3, 12].

3 Computing in ILL

Linear logic and its intuitionistic fragment is used in several areas of computer science. Recent work has attempt to find a linear logic basis for optimizations in functional programming language implementations by using linear logic as a type system [1, 12]. Other applications include generalized and concurrent logic programming [2, 9, 10]. From specification point of view, ILL provides a natural and simple encoding of *Petri net reachability* [13] that can be extended to high-level nets [11].

For example, the formula $!((a \otimes c) \multimap b)$ is used to encode a Petri net transition taking tokens from place a and c and adding a token to place b. Similarly, the formula $!((b \otimes d \otimes d) \multimap (c \otimes d))$ represents a transition taking one token from b and two tokens from d and adding one token to c. Thus we encode transitions as reusable linear implications and tokens as atomic propositions. A reachability problem is represented as a sequent $!((a \otimes b) \multimap b), !((b \otimes d \otimes d) \multimap (c \otimes d)), a, c, d, d \vdash c \otimes d$ that is provable in LL if and only if there is a sequence of rule application that transform the token set $\{a,c,d,d\}$ to $\{c,d\}$.

The proof search and construction is fundamental in most of these applications as computation process (logic concurrent programming) or as specification verification (and deduced program synthesis). For solving the previous reachability problem, we have to construct a proof of the sequent, like the following one:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a \vdash a} \quad \dfrac{}{c \vdash c}}{a, c \vdash a \otimes c}\otimes_R \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{b \vdash b} \quad \dfrac{}{d \vdash d}}{d, b \vdash b \otimes d}\otimes_R \quad \dfrac{}{d \vdash d}}{d, d, b \vdash b \otimes d \otimes d}\otimes_R \quad \dfrac{}{c \otimes d \vdash c \otimes d}}{(b \otimes d \otimes d) \multimap (c \otimes d), d, d, b \vdash c \otimes d}\multimap_L}{(a \otimes c) \multimap b, (b \otimes d \otimes d) \multimap (c \otimes d), a, c, d, d \vdash c \otimes d}\multimap_L}{((a \otimes c) \multimap b), !((b \otimes d \otimes d) \multimap (c \otimes d)), a, c, d, d \vdash c \otimes d}\,!D_L}{!((a \otimes c) \multimap b), !((b \otimes d \otimes d) \multimap (c \otimes d)), a, c, d, d \vdash c \otimes d}\,!D_L
$$

But some questions arise: are we sure to find efficiently a proof ? how to reduce the failure cases during the proof search ? what about the computational content of such a proof i.e., what are the properties of the extracted program *read* $($*store* $k0)$ *as* f *in read* $($*store* $k1)$ *as* g *in let* $(k2 \times k3)$ *be* $k4$ *in* $(k2 \times k3)[(k1(k5 \times (z \times z)))/k4][(k0(x \times y))/k5]$?
For example, in a bottom-up approach, we could start the proof construction by applying the $\otimes_R$ rule to the formula $c \otimes d$, considering a goal-oriented proof search. But, for this example, it leads to a failure. It means that we cannot use the notion of *uniform proof*, issued from works on linear logic programming [8, 9], to prove such ILL sequents. Thus we have to define adequate proof search methods for ILL theorem proving (dedicated here to program synthesis) but also applicable to applications based on *proof-search as computation* paradigm.

4 Proof search in ILL

Recent works have been devoted to proof search methods and to their applications to various topics of computer science as logic (concurrent) programming [2, 7, 10] or concurrency [10, 13]. These various results are based on different fragments of (intuitionistic or not) LL, different proof search approaches (top-down or bottom-up proof construction, goal-oriented proof search)[4, 5, 14] and lead to proposals for designing automated proof search procedures. The main point is to define subclasses of proofs, in the LL fragment, that are complete and tractable. Here, we have adapted recent results [4] on proof normalization for LL to the intuitionistic fragment (with a bottom-up approach) and analyze their impact to efficient proof search and program generation in ILL and its subfragments.
A first step consists in studying the *permutability properties* of SEQ inference rules and then the possible *up* or *down movements* in a proof. Then we are able to define the rules we can move up $T \uparrow = \{\oplus_R, \&_L, \otimes_R, \multimap_L, !D_L, !C_L, !W_L\}$ and the rules we can move down $T \downarrow = \{\otimes_L, 1_L, \&_R, \oplus_L, \multimap_R\}$.

With these results, we can define possible transformations on an ILL proof and thus transform the corresponding linear term. But it is only possible when you have firstly construct a proof and we propose to use these rules sets for defining adequate proof subclasses that are complete (if the sequent is provable then there exists such a proof of it) and tractable.
Let us remind that, for the bottom-up approach, we have at each step of proof search some sources of non-determinism concerning, for instance, the choice of formula (or goal) to prove or the choice of principal formula. The non-determinism due to this last one, is strongly reduced by using the following normal proof form.
The notion of *normal proof* we propose is defined as follows:

Definition 4.1 *A canonical proof, for ILL and for the bottom-up proof direction, is a proof without cuts, where any intermediate conclusion $\Delta \vdash F$ we apply the following order in rule application: the $!S_R$ rule, then a rule of $T \downarrow$, then a rule of $T \uparrow \setminus \{!S_R, !C_L, !D_L, !W_L\}$ (if the formula is not a positive literal). Finally, if $!F$ is the principal formula we have three possible cases: (1) $\Delta \vdash F$ is the conclusion of a $!W_L$ rule and the preceding inferences are weakening or axioms. (2) $\Delta \vdash F$ is the conclusion of a $!C_L$ rule and the preceding inference is an inference of type $!D_L$ introducing one of the active formulas of the inference ending with $\Delta \vdash F$. (3) $\Delta \vdash F$ is the conclusion of an inference of $!D_L$ type and the active formula, if it is not a positive literal, is the principal formula of the preceding conclusion.*

With such definition, we can prove the following result:

Theorem 4.1 *If the sequent $\Delta \vdash F$ is provable in ILL then there exists a canonical proof of it, i.e.,the subclass of canonical proofs is complete.*

Thus, we have a general proof form that we can refine by fixing some free choices of the definition and that depends on proof direction strategy and on logical fragment.

Let us come back to the example on petri net specification. To solve the reachability problem, we know now that we can search a normal proof of the SEQ sequent $f :!(a \otimes c \multimap b), g : !(b \otimes d \otimes d \multimap c \otimes d), x : a, y : c, z : d, z : d \vdash U : c \otimes d$ where U is a metavariable that will be instanciated by the linear term corresponding to a proof.
Thus, the following proof (without terms) is normal

$$\dfrac{\dfrac{\otimes_R\,\dfrac{a \vdash a \quad c \vdash c}{a, c \vdash a \otimes c} \qquad \dfrac{\dfrac{\dfrac{b \vdash b \quad \dfrac{d \vdash d \quad d \vdash d}{d, d \vdash d \otimes d}\otimes_R}{d, d, b \vdash b \otimes d \otimes d} \qquad \dfrac{\dfrac{c \vdash c \quad d \vdash d}{c, d \vdash c \otimes d}\otimes_R}{c \otimes d \vdash c \otimes d}\otimes_L}{d, d, b, b \otimes d \otimes d \multimap c \otimes d \vdash c \otimes d}\multimap_L}{!(b \otimes d \otimes d \multimap c \otimes d), d, d, b \vdash c \otimes d}!D_L}{!(b \otimes d \otimes d \multimap c \otimes d), a, c, d, d, a \otimes c \multimap b \vdash c \otimes d}\multimap_L}{!(a \otimes c \multimap b), !(b \otimes d \otimes d \multimap c \otimes d), a, c, d, d \vdash c \otimes d}!D_L$$

and the corresponding linear term is: $U \equiv read\ (store\ k0)\ as\ f\ in\ read\ (store\ k1)\ as\ g\ in\ let\ (k2 \times k3)\ be\ k4\ in\ (k2 \times k3)[(k1(k5 \times (z \times z)))/k4][(k0(x \times y))/k5]$.

Let us note that the first proof (section 3) is not in normal form because after applying $!D_L$, the active formula is not the principal formula of the preceding conclusion. If we compare the linear terms in both cases, we can see that they are identical. In this case, the inference permutation has no effect on the generated term. But, let us consider another normal proof of the petri net specification

$$\dfrac{\dfrac{\dfrac{\dfrac{a \vdash a \quad c \vdash c}{a, c \vdash a \otimes c}\otimes_R \qquad \dfrac{b \vdash b \quad \dfrac{d \vdash d \quad d \vdash d}{d, d \vdash d \otimes d}\otimes_R}{d, d, b \vdash b \otimes d \otimes d}\otimes_R}{a, c, d, d, a \otimes c \multimap b \vdash b \otimes d \otimes d}\multimap_L}{!(a \otimes c \multimap b), a, c, d, d \vdash b \otimes d \otimes d}!D_L \qquad \dfrac{\dfrac{c \vdash c \quad d \vdash d}{c, d \vdash c \otimes d}\otimes_R}{c \otimes d \vdash c \otimes d}\otimes_L}{\dfrac{!(a \otimes c \multimap b), a, c, d, d, b \otimes d \otimes d \multimap c \otimes d \vdash c \otimes d}{!(a \otimes c \multimap b), !(b \otimes d \otimes d \multimap c \otimes d), a, c, d, d \vdash c \otimes d}!D_L}\multimap_L$$

The corresponding linear is: $U \equiv \mathit{read}\ (\mathit{store}\ k0)\ \mathit{as}\ g\ \mathit{in}\ \mathit{let}\ (k1 \times k2)\ \mathit{be}\ k3\ \mathit{in}\ (k3 \times k2)[(k0\mathit{read}\ (\mathit{store}\ k4)\ \mathit{as}\ f\ \mathit{in}\ (k5 \times (z \times z))[(k4(x \times y))/k5])/k3]$, that is different from the previous ones. The two terms without substitutions are respectively:
$\mathit{read}\ (\mathit{store}\ k_0)\ \mathit{as}\ f\ \mathit{in}\ \mathit{read}\ (\mathit{store}\ k_1)\ \mathit{as}\ g\ \mathit{in}\ \mathit{let}\ (k_2 \times k_3)\ \mathit{be}\ ((k_1(k_0(x \times y))) \times (z \times z)))\ \mathit{in}\ (k_2 \times k_3)$ and $\mathit{read}\ (\mathit{store}\ k_0)\ \mathit{as}\ g\ \mathit{in}\ \mathit{let}\ (k_1 \times k_2)\ \mathit{be}\ (k_0\ \mathit{read}\ (\mathit{store}\ k_4)\ \mathit{as}\ f\ \mathit{in}\ ((k_4(x \times y)) \times (z \times z)))\ \mathit{in}\ (k_1 \times k_2)$. If we suppose that $f \to \mathit{store}\ v$ and $g \to \mathit{store}\ u$, by application of the $!D_L$ reduction rule we obtain in both case the same term : $\mathit{let}\ (k_1 \times k_2)\ \mathit{be}\ (u\ ((w\ (x \times y)) \times (z \times z)))\ \mathit{in}\ (k_1 \times k_2)$.

Regarding to proof transformations, our results gives us the possibility, by inference movements of rules, to transform a proof (canonical or not) in an another proof (canonical or not). Thus we can study program transformation through proof transformation, having in mind the efficiency of the generated linear terms. A difference between two terms can effectively appear regarding to the number of reduction steps. If we look at the two normals proofs, the difference is that in the first one we choose as principal formula for $\multimap_L$ the formula $(a \otimes c) \multimap b$ and in the second $(b \otimes d \otimes d) \multimap (c \otimes d)$. But in this last case, b is an hypothesis of an linear implication which not appears as atom in the sequent premises like a, c, d, d. In fact, we start working without b and get b latter and thus the difference leads in the choice of the principal formula.
From the results on inference movements and on normal proof definition in ILL, we can design interactive or automatic theorem provers and specification verifiers (with linear term synthesis). We can use them also for other applications based on *proof construction as computation* paradigm.

5 Concluding remarks

Normal proof forms for ILL (more general than *uniform proof* used in linear logic programming)) allow to reduce some non-determinism points in the proof search and to automate proof construction. Moreover, we can use the results for proof and linear term transformations and extend them, for example to predicate ILL for proving properties on high-level nets [11].

References

[1] S. Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111(1-2):3–58, 1993.

[2] J.M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[3] N. Benton, G. Bierman, V. de Paiva, and M. Hyland. A term calculus for intuitionistic linear logic. In *Int. Conference on Typed Lambda Calculi and Applications, LNCS 664*, pages 75–90, Utrecht, The Netherlands, March 1993.

[4] D. Galmiche and G. Perrier. On proof normalisation in linear logic. Research report 93-R-059, CRIN-CNRS, 1993. To appear in Theoretical Computer Science.

[5] D. Galmiche and G. Perrier. Foundations of proof search strategies design in linear logic. In *Logic at St Petersburg '94, Symposium on Logical Foundations of Computer Science, LNCS 813*, pages 101–113, St Petersburg, Russia, July 1994.

[6] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[7] J. Harland. A proof-theoretic analysis of goal-directed provability. *Journal of Logic and Computation*, 4(1):69–88, 1994.

[8] J. Harland and D. Pym. On resolution in fragments of classical linear logic. In *LPAR'92, International Conference on Logic Programming and Automated Reasoning, LNAI 624*, pages 30–41, St. Petersburg, Russia, July 1992.

[9] J.S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. In *6th IEEE Symposium on Logic in Computer Science*, pages 32–42, Amsterdam, The Netherlands, July 1991.

[10] N. Kobayashi and A. Yonezawa. ACL - a concurrent linear logic programming paradigm. In *Int. Symposium on Logic Programming*, pages 279–294, Vancouver, October 1993.

[11] J. Lilius. High-level nets and linear logic. In *13th Int. Conference on Applications and Theory of Petri Nets, LNCS 616*, pages 310–327, Sheffield, UK, 1992.

[12] P. Lincoln and J. Mitchell. Operational aspects of linear lambda calculus. In *7th IEEE Symposium on Logic in Computer Science*, pages 235–246, Santa-Cruz, California, June 1992.

[13] J. Meseguer and N. Marti-Oliet. From Petri nets to Linear Logic. *Math. Struct. in Comp. Science*, 1:69–101, 1991.

[14] T. Tammet. Proof search strategies in linear logic. Programming Methodology Group Report 70, Chalmers University Group, University of Göteborg, 1993.