LINK: a Proof Environment based on Proof Nets

L. Habert and J.-M. Notin and D. Galmiche

LORIA–Université Henri Poincaré
Campus Scientifique, BP 239
Vandœuvre-lès-Nancy, France

1 Introduction

LINK is a proof environment including based-on proof nets provers for multiplicative linear logics: mixed linear logic, or called now non-commutative logic (MNL) [1], commutative linear logic (MLL) and non-commutative (or cyclic) linear logic (MCyLL). Its main characteristic is the provability analysis through automatic proof nets construction. This notion is a particular graph-theoretic representation of proofs that appears appropriate for proof-search in MLL and MCyLL [4, 5]. It is a powerful alternative to deal with non-permutability and resource management [10, 3] during proof search. In the context of verification of systems properties, such a semantical and graphical representation of proof is useful from a software engineering point of view. It allows to analyze provability (through proof nets) or non-provability (through partial proof structures that are abstract countermodels).
The LINK proof engine is based on a specific algorithm dedicated to MLL [8] that has been adapted to deal with non-commutative connectives (MCyLL) and also with both commutative and non-commutative connectives (MNL). MNL has been studied to define logical characterizations of synchronization mechanisms in concurrent constraint programming [13] and more recently to specify and verify properties about the TCP/IP protocol [9]. This first version of LINK includes efficient and useful proof-search algorithms that are also implementations of new connection methods for these linear logic fragments [4]. Further work, in addition to standard improvements, consists in developing this proof technology for other substructural logics and in designing tools for the analysis of related semantic structures. LINK illustrates that based-on proof nets theorem proving is an appropriate and promising proof-theoretical but also proof-engineering approach of the verification of system properties.

2 Basic principles of LINK

LINK is a proof environment that includes provers for multiplicative linear logics, but each of them is based on the same initial principles presented for MLL in [8] and refined in [5]. Let us summarize its underlying concepts and principles.
Informally, a proof structure for a formula F can be seen as the decomposition tree of F in addition to axiom-links between leaves of the tree, that are dual

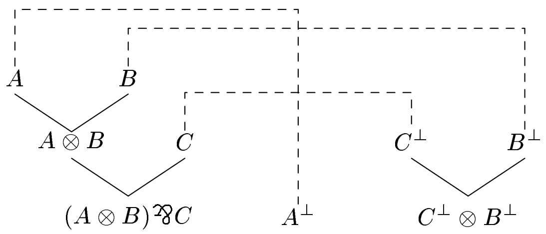

Fig. 1. A MLL proof structure

atomic formulae. A proof net is a proof structure that corresponds to a proof. There exist criteria in order to verify that a given proof structure is a proof net, but our approach is to automatically build a proof net, from the decomposition tree and by an appropriate search of axiom-links. In fact, proof nets construction provides new connection methods for linear logics [4, 11], with automatic generation of connections and parallel construction of sequent proofs. An initial test on a formula F consists in verifying the duality property: for each positive literal A, there is exactly one dual literal $A^{\perp}$.

The kernel algorithm for MLL starts with the decomposition tree of the formula F, that is used as a frame to guide the proof net search. The main idea is to build, step by step, but automatically axiom-links as initial elementary proof nets and to link and extend them in order to have new partial proof nets. Finally, we have a final proof net of F, that "covers" the decomposition tree iff F is provable. Let us consider figure 1 to illustrate some points. The axiom-links $(A, A^{\perp})$ and $(B, B^{\perp})$ are two elementary proof nets. When conclusions of partial proof nets are premises of a so-called 2-link ($\otimes$), they are connected and extended with this link and the result is a new partial proof net. For instance, we can connect the previous axiom-links and extend this proof structure with the $A \otimes B$ link. When conclusions of a partial proof net are premises of a so-called 1-link ($\parr$) then it is extended with this link and the result is a new partial proof net. In our example, if $A \otimes B$ and C are conclusions of the same partial proof net then we extend it with the $(A \otimes B) \parr C$ link. Then we build, step by step, partial proof nets from elementary proof nets (axiom-links) with a final proof net as a goal. Extensions of 2-links ($\otimes$) are treated before extensions of 1-links ($\parr$). More details can be found in [4, 8].

Some strategies are implemented to efficiently deal with this automatic proof nets construction. For instance, the 1-link waiting strategy considers a waiting list of all conclusions (of partial proof nets) that are premises of a 1-link but not yet in the same partial proof net. It is clear that some automatic choices of axiom-links are wrong choices, w.r.t. provability, and backtracking is necessary at some points. The aims of some strategies is to detect and avoid such choices and to reduce the backtracking. LINK includes strategies to detect failure cases: for instance, when two premises of a 2-link (resp. 1-link) are (resp. cannot be) in the same partial proof net. If we have a sequent with several occurrences of a literal

A (for instance our example where B and C are all replaced by A) the impact of the strategies is significant and the backtracking is then strongly reduced. From our algorithm, we can directly build, in parallel with a proof net, a sequent proof of F, each link extension corresponding to a sequent rule application. Depending on the strategy, we can obtain a sequent proof in different canonical forms (for instance with focusing [2]).

3 The LINK prover

The LINK system is written in the OCaml[1] language. The program and its sources are available at `http://www.loria.fr/~notin/LINK` with information about how to install and use LINK. Thanks to the module system of OCaml, most part of the code is shared between the different provers (for MLL, MCyLL and MNL). Thus, LINK proposes a single command line interface that allows the user to choose the logic, and to enter the formula to prove. Figure 2 shows a typical session of LINK. If the display mode is available, the proof net corresponding to the given formula (if it is provable) is drawn in a graphic window. The user can also choose to have a step by step construction in this window. This mode is very useful to understand the basic principles of the construction of proof nets, as it shows its main stages (creation of axiom-links, extension of subnets by 2- and 1-links).

```
$mnld                          We choose to work in MNL, using dependencies.
((a*b)<c)|a^|(c^@b^)           We enter a first formula.
F = ((((a*b)<c)|a^)|(c^@b^))
Disproved in 0.00 seconds      The formula is not provable.
((a@b)<c)|a^|(c^@b^)           We enter a second formula.
F = ((((a@b)<c)|a^)|(c^@b^))
Proved in 0.00 seconds         This one is provable.
```
(`*` represents $\otimes$, `@` represents $\odot$, `|` represents ⅋ and `<` represents $\lhd$)

Fig. 2. A session example

LINK is composed by four essential modules: *Select* (for the choice of literals), *CC* (Connected Components - to handle subnets already built) and *PW* (Par Waiting - to handle 1-links the premises of which do not belong to the same subnet). The fourth module, *Build*, is common to all provers and contains the procedures that implement the basic proof nets construction algorithm.

The algorithm for MLL is implemented following the basic principles we introduced. Following [5], MCyLL proof nets are MLL proof nets such that there is no crossing between axiom-links. We thus modify the *Select* module to have a choice of literals in concordance with this restriction. MNL is more complicated because it combines both commutative ($\otimes$, ⅋) and non-commutative ($\odot, \lhd$) connectives [1]. For the MNL prover, we now have $\otimes$ and $\odot$ as 2-links and ⅋ and $\lhd$ as 1-links. It means that there is no distinction between commutative and non-commutative connectives at this level, but the interactions between connectives

[1] OCaml is available at `http://caml.inria.fr/ocaml`

during proof search are captured in an appropriate semantic structure, built in parallel with the partial proof nets. For our first MNL algorithm, it consists of a dependency graph on which one has to verify some conditions related to the non-commutative disjunction ($\triangleleft$) [7]. Thus, the CC and PW modules have to be adapted to deal with such dependency graphs associated to subnets. In particular, the procedures that perform the extension of subnets by 1- and 2-links (included in the CC module) include the computation of the dependency set associated to the new subnet.

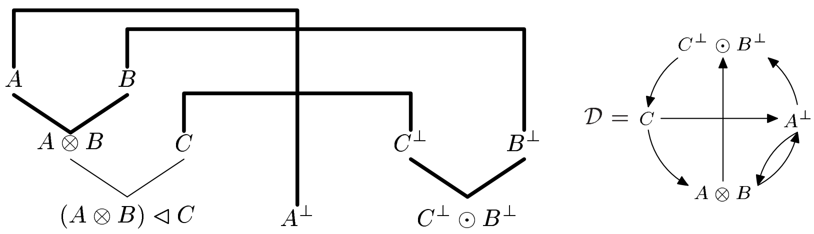

Fig. 3. A step of MNL proof net construction

Figure 3 shows a MNL proof construction (in bold), and the associated dependency set. The algorithm has extended axiom-links by $A \otimes B$ and then by $C^{\perp} \odot B^{\perp}$. Then, it has to check a condition on the dependency graph, related to the $\triangleleft$ connective. When, like here, the condition is not verified then the extension of the $\triangleleft$ link is impossible. As there is no other choice for axiom-links, the algorithm cannot perform any backtracking and the construction fails. In such a case of non-provability, the partial proof net (in bold) and the dependency graph in LINK are displayed. If $A \otimes B$ is replaced by $A \odot B$ then a final proof net is built and the formula is provable. The MLL strategies, extended to MNL, are very efficient to reduce backtracking when many choices are possible for axiom-links construction. LINK also includes two other MNL provers that are based on other semantic structures. One, detailed in [6], with labels that are propagated in proof structures (decomposition tree plus axiom-links already built) during the proof net search. Another, based on a contractibility criterion [12], with a particular graph (a seaweed) and contraction rules applied on the associated structure. The performances of LINK provers on various examples, of provable and non-provable formulae, are good and illustrate the efficiency of the based-on proof net theorem proving, mainly in case of non-provability.

4 Conclusion and perspectives

LINK is a proof environment based on an original and alternative concept, w.r.t. proof-search, namely proof net. It illustrates the actual interest of this concept, completed with semantic structures, for efficient proof search and proof analysis.

It deals with some linear logics that are appropriate to specify particular system properties [9] and for which no prover is implemented, except MLL [11]. LINK provers are efficient, especially for non-provability, and also provide semantic structures that can be analyzed in this case. Further developments of this version will include refinements of strategies, parallel construction of the sequent proof and graphical representations of semantic structures like dependency graphs. Tools for the manipulation and the analysis of these structures will be studied, with a software engineering and users point of view. We aim to use the LINK environment and their further extensions (with additives, quantifiers) for the verification and analysis of system properties [9], focusing on the failure analysis from the graph-theoretical proof representation.

References

1. M. Abrusci and P. Ruet. Non-commutative logic I : the multiplicative fragment. *Annals of Pure and Applied Logic*, 101:29–64, 2000.
2. J.M. Andreoli and R. Maieli. Focusing and proof-nets in linear and non-commutative logic. In *Int. Conf. on Logic for Programming and Automated Reasoning, LPAR'99, LNCS 1705*, pages 320–333, Tbilisi, Georgia, September 1999.
3. I. Cervesato, J. Hodas, and F. Pfenning. Efficient resource management for linear logic proof search. *Theoretical Computer Science*, 232(1-2):133–163, 2000.
4. D. Galmiche. Connection Methods in Linear Logic and Proof nets Construction. *Theoretical Computer Science*, 232(1-2):231–272, 2000.
5. D. Galmiche and B. Martin. Proof nets construction and automated deduction in non-commutative linear logic - extended abstract. *Electronic Notes in Theoretical Computer Science*, 17, 1998.
6. D. Galmiche and J.M. Notin. Proof-search and proof nets in mixed linear logic. *Electronic Notes in Theoretical Computer Science*, 37, 2000.
7. D. Galmiche and J.M. Notin. Calculi with dependency relations for mixed linear logic. In *International Workshop on Logic and Complexity in Computer Science, LCCS'2001*, pages 81–102, Créteil, France, 2001.
8. D. Galmiche and G. Perrier. A procedure for automatic proof nets construction. In *LPAR'92, International Conference on Logic Programming and Automated Reasoning, LNAI 624*, pages 42–53, St. Petersburg, Russia, July 1992.
9. D. Gray, G. Hamilton, J. Power, and D. Sinclair. Specifying and verifying TCP/IP using Mixed Intuitionistic Linear Logic. Technical report, Dublin City Univ., 2001.
10. J. Harland and D. Pym. Resource-distribution via boolean constraints (extended abstract). In *14th Int. Conference on Automated Deduction, CADE-12, LNAI 814*, pages 222–236, Townsville, North Queensland, Australia, July 1997.
11. C. Kreitz, H. Mantel, J. Otten, and S. Schmitt. Connection-based proof construction in linear logic. In *14th Int. Conference on Automated Deduction*, pages 207–221, Townsville, North Queensland, Australia, 1997.
12. V. Mogbil. Quadratic correctness criterion for non commutative logic. In *15th Int. Workshop on Computer Science Logic, CSL 2001, LNCS 2142*, pages 69–83, Paris, France, 2001.
13. P. Ruet and F. Fages. Concurrent constraint programming and non-commutative logic. In *11th Int. Workshop on Computer Science Logic, CSL'97, LNCS 1414*, pages 406–423, Aarhus, Denmark, August 1997.