Efficient Resource Management for Linear Logic Proof Search

Iliano Cervesato[1], Joshua S. Hodas[2], and Frank Pfenning[1]

[1] Department of Computer Science, Carnegie Mellon University
Pittsburgh, PA 15213-3891, USA
E-mail: {*iliano*|*fp*}@*cs.cmu.edu*

[2] Computer Science Department, Harvey Mudd College
Claremont, CA 91711, USA
E-mail: *hodas@cs.hmc.edu*

August 29, 1997

Abstract

The design of linear logic programming languages and theorem provers opens a number of new implementation challenges not present in more traditional logic languages such as Horn clauses (*Prolog*) and hereditary Harrop formulas (λ*Prolog* and *Elf*). Among these, the problem of efficiently managing the linear context when solving a goal is of crucial importance for the use of these systems in non-trivial applications. This paper studies this problem in the case of *Lolli* [HM94], though its results have application to other systems. We first give a proof-theoretic presentation of the operational semantics of this language as a resolution calculus. We then present a series of resource management systems designed to eliminate the non-determinism in the distribution of linear formulas that undermines the efficiency of a direct implementation of this system.

1 Introduction

Linear logic [Gir87] views logical assumptions as consumable resources. This allows elegant and concise formalizations of a number of problems which are difficult to represent in traditional logics. In particular, many problems centered around the notion of a state that evolves as a computation proceeds fall into this category. Consequently, several logic programming languages based on linear logic have been designed in recent years [AP91, HP91, HM94, Mil96]. Others are the subject of extensive research. Each proposal is accompanied by interesting theoretical results that show its computational relevance, and by numerous examples that prove its practical significance. However, to our knowledge, usable implementations have thus far been released only for *Lolli* [HM94] and *Lygon* [HP91].

Linear logic programming languages offer the implementor new challenges not present in more traditional logic languages such as *Prolog* or λ*Prolog*. Among these, the efficient management of the linear formulas contained in the context is of crucial importance for the use of these languages in non-trivial applications.

Submitted to *Theoretical Computer Science*. **Comments are welcome!**

Managing the clauses of a program is particularly simple in *Prolog*: The only predicates that can modify the program are the extra-logicals `assert` and `retract`, which have global effect [LO87]. In languages admitting implications in goals, λ*Prolog* [Mil91] and *Elf* [Pfe94] for example, the use of scoped assumptions causes the program to grow and contract, but in a simple stack-like fashion. The matter is more complicated in the case of linear logic due to the strict rules placed on the use and reuse of assumptions.

The problem is best exemplified by considering the rule for proving the goal $G_1 \otimes G_2$:

$$\frac{\Delta_1 \longrightarrow G_1 \quad \Delta_2 \longrightarrow G_2}{\Delta_1, \Delta_2 \longrightarrow G_1 \otimes G_2}$$

When the interpreter needs to use this rule during the bottom-up search for a proof (i.e., when solving a goal upward from the root to the leaves), the current context has not already been divided into Δ_1 and Δ_2. The naive choice is to generate all partitions of the assumption set until a pair Δ_1, Δ_2 with the desired properties is found. This non-deterministic behavior is clearly unacceptable since the number of partitions grows exponentially with the number of assumptions in the context. Considering the frequency with which $\otimes$ and other multiplicative connectives occur in practice, an interpreter for a linear logic programming language based on such a generate-and-test algorithm would be usable only for the smallest of toy problems.

In this paper, we begin by reiterating the deterministic solution to this problem presented in the original work on *Lolli* [Hod94, HM94]. We will then concentrate on less apparent issues in context management involving the additive connectives and constants. We do not treat other sources of non-determinism, which can be handled according to standard techniques in a logic programming framework [Pfe92], or that we might want to keep open in a theorem prover and in the implementation of a concurrent linear logic programming language such as *ACL* [KY93]. Similarly, we do not consider orthogonal issues in linear proof search, such as the permutability of inference rules, which have been treated exhaustively elsewhere [GP94, Min93, Tam94]. We will focus our attention on the language *Lolli* [Hod94, HM94], that we used to test the techniques described below [CHMP]. However, our results have already been applied to prototype implementations [HP96, LP97] of a programming language based on Miller's specification logic *Forum* [Mil96]. They should apply equally well to implementations of other linear logic programming languages such as *Lygon* [HP91] and to a concrete realization of the linear logical framework *LLF* [CP96]. It is also possible to adapt our techniques to the development of theorem provers for linear logic, but our main goal has been to remove non-determinism in order to obtain a satisfactory and predictable operational semantics for logic programming. In related work, Harland and Pym [HP97] present a less committed framework for resource management strategies in general linear logic proof search which relies on Boolean constraints. As far as we can see, their framework does not have an immediate operational interpretation or efficient implementation and thus does not directly address our problem. It is possible, however, that our solution could be expressed within their framework.

This paper is organized as follows. Section 2 introduces the fragment of linear logic we consider and presents its semantics as a proof-theoretic resolution system, an uncommitted starting point for an implementation as a logic programming language. In Section 3, we fully expose the context splitting non-determinism exemplified above and give a context management scheme that eliminates it. A more subtle context handling problem, the possibility of weakening the linear context in the presence of additive truth, is pointed out in Section 4 and solved by means of a more refined context management system. Section 5 discusses and solves a remaining problem concerned with the duplication of linear resources when processing an additive conjunction. We discuss further steps towards an efficient implementation of *Lolli* in Section 6. Finally, we summarize our work and compare it with other proposals in the literature in Section 7.

2 Resolution for Linear Hereditary Harrop Formulas

The programming language *Lolli* [Hod94, HM94] is based on the fragment of linear logic freely generated by the operators $\top$, $\&$, $\multimap$, $\supset$ and $\forall$. The connective $\supset$ is called *intuitionistic implication* and is defined as $A \supset B \equiv\; !A \multimap B$. Positive occurrences of $\mathbf{0}$, $\mathbf{1}$, $\oplus$, $\otimes$, $!$, $\exists$ and the syntactic equality among atomic formulas, $a \doteq a'$, are also allowed, as they do not invalidate any essential properties of the language. This extended fragment is called the language of *linear hereditary Harrop formulas* (*LHHF* for short). It is formally defined by the following grammar:

$$\begin{array}{llcl}
\textit{Program formulas:} & D & ::= & a \mid \top \mid D_1 \,\&\, D_2 \mid G \multimap D \mid G \supset D \mid \forall x.D \\
\textit{Goal formulas:} & G & ::= & a \mid \top \mid G_1 \,\&\, G_2 \mid D \multimap G \mid D \supset G \mid \forall x.G \\
 & & \mid & a_1 \doteq a_2 \mid \mathbf{1} \mid \mathbf{0} \mid G_1 \oplus G_2 \mid G_1 \otimes G_2 \mid !G \mid \exists x.G
\end{array}$$

where a, possibly subscripted, stands for the syntactic category of atomic formulas. We do not make any assumption about the structure of the terms embedded in atomic formulas. We write $[t/x]G$ for the capture-free substitution of the term t for the variable x in the goal formula G.

Descriptions of deductive systems treat logical assumptions in a variety of different ways, e.g., as sets, multisets, or sequences of formulas. In our setting it is critical that different occurrences of the same hypothesis can be distinguished, while the order of the assumptions does not matter. This can be achieved by uniquely labelling each assumption and annotating certain inference rules with corresponding labels. We adopt this technique, but drop the labels in the actual presentation for the sake of readability. We will point out a few places in the correctness proofs where the fact that each assumption has a unique label is critical.

A set of uniquely labelled program formulas will be called a *context* and denoted by a Greek letter Γ, Δ, or Ξ, depending on its role in a judgment. We write "$\cdot$" for the empty context and Δ, D for the result of adding D with a new (implicit) label to the context Δ. We overload "," and also write Δ_1, Δ_2 for the disjoint union of two contexts. Other standard operations and predicates on sets will also be used, with the proviso that we consider context difference $\Delta_1 - \Delta_2$ only when $\Delta_2 \subseteq \Delta_1$.

The logic of *LHHF* is conveniently described by sequents of the form:

$$\Gamma; \Delta \Longrightarrow G$$

where Γ and Δ are called the *intuitionistic* and the *linear* contexts respectively, and together constitute the *program*. G is a positive formula called the *goal*. The formulas in the intuitionistic context are treated as if they were implicitly preceded by the modal operator $!$, so that the expression above corresponds to the more traditional linear logic sequent $!\Gamma, \Delta \Longrightarrow G$. This manner of structuring the sequents and the use of $\supset$ retains desirable aspects of the semantics of $!$ (in particular formulas in the intuitionistic context can be used arbitrarily many times), while preventing unwanted behaviors.

Hodas and Miller discuss a proof system, $\boldsymbol{L}$, for *LHHF* based on sequents of this form [HM94]. They also prove the soundness and completeness of $\boldsymbol{L}$ with respect to the usual rules for linear logic restricted to the language of *LHHF*. Most importantly, they prove that *LHHF* possesses the necessary computational properties to be considered an *abstract logic programming language* [MNPS91]. In particular, every provable sequent of $\boldsymbol{L}$ can be transformed into an equivalent proof that consults the program only when the goal formula is atomic (thus proofs are *goal-directed* [MNPS91]), and at that point selects and operates upon a single program formula in order to proceed with the derivation (thus proofs are *focused* [AP91]). Proofs with both properties are called *uniform*. Hodas and Miller capture this behavior in the system $\boldsymbol{L'}$ which eliminates the left-hand rules of the logic in favor of a single rule for *backchaining*.

Residuation

$$\frac{}{a' \gg a \setminus a' \doteq a}\ \textbf{dec-atm} \qquad \frac{}{\top \gg a \setminus \mathbf{0}}\ \textbf{dec-}\top$$

$$\frac{D \gg a \setminus G}{\forall x.D \gg a \setminus \exists x.G}\ \textbf{dec-}\forall \qquad \frac{D \gg a \setminus G'}{G \multimap D \gg a \setminus G' \otimes G}\ \textbf{dec-}\multimap$$

$$\frac{D \gg a \setminus G'}{G \supset D \gg a \setminus G' \otimes !G}\ \textbf{dec-}\supset \qquad \frac{D_1 \gg a \setminus G_1 \quad D_2 \gg a \setminus G_2}{D_1 \,\&\, D_2 \gg a \setminus G_1 \oplus G_2}\ \textbf{dec-\&}$$

Resolution

$$\frac{D \gg a \setminus G \quad \Gamma, D; \Delta \Longrightarrow G}{\Gamma, D; \Delta \Longrightarrow a}\ \textbf{res-atm\_int} \qquad \frac{D \gg a \setminus G \quad \Gamma; \Delta \Longrightarrow G}{\Gamma; \Delta, D \Longrightarrow a}\ \textbf{res-atm\_lin}$$

$$\frac{}{\Gamma; \cdot \Longrightarrow a \doteq a}\ \textbf{res-}\doteq \qquad \frac{}{\Gamma; \cdot \Longrightarrow \mathbf{1}}\ \textbf{res-1}$$

$$\frac{}{\Gamma; \Delta \Longrightarrow \top}\ \textbf{res-}\top \qquad \text{(No rule for } \mathbf{0}\text{)}$$

$$\frac{\Gamma; \Delta \Longrightarrow G_1 \quad \Gamma; \Delta \Longrightarrow G_2}{\Gamma; \Delta \Longrightarrow G_1 \,\&\, G_2}\ \textbf{res-\&} \qquad \frac{\Gamma; \Delta_1 \Longrightarrow G_1 \quad \Gamma; \Delta_2 \Longrightarrow G_2}{\Gamma; \Delta_1, \Delta_2 \Longrightarrow G_1 \otimes G_2}\ \textbf{res-}\otimes$$

$$\frac{\Gamma; \Delta \Longrightarrow G_1}{\Gamma; \Delta \Longrightarrow G_1 \oplus G_2}\ \textbf{res-}\oplus_1 \qquad \frac{\Gamma; \Delta \Longrightarrow G_2}{\Gamma; \Delta \Longrightarrow G_1 \oplus G_2}\ \textbf{res-}\oplus_2$$

$$\frac{\Gamma; \Delta, D \Longrightarrow G}{\Gamma; \Delta \Longrightarrow D \multimap G}\ \textbf{res-}\multimap \qquad \frac{\Gamma, D; \Delta \Longrightarrow G}{\Gamma; \Delta \Longrightarrow D \supset G}\ \textbf{res-}\supset$$

$$\frac{\Gamma; \cdot \Longrightarrow G}{\Gamma; \cdot \Longrightarrow !G}\ \textbf{res-!}$$

$$\frac{\Gamma; \Delta \Longrightarrow [c/x]G}{\Gamma; \Delta \Longrightarrow \forall x.G}\ \textbf{res-}\forall \qquad \frac{\Gamma; \Delta \Longrightarrow [t/x]G}{\Gamma; \Delta \Longrightarrow \exists x.G}\ \textbf{res-}\exists$$

Figure 1: $\boldsymbol{R}$, a Resolution Calculus for *LHHF*.

In Figure 1 we present a new *resolution system*, called $\boldsymbol{R}$, for *LHHF*. This system is different from but equivalent to the system $\boldsymbol{L'}$. It is easy to show that the judgment $\Gamma; \Delta \Longrightarrow G$ is provable in $\boldsymbol{R}$ if and only if the sequent $\Gamma; \Delta \longrightarrow G$ is provable in $\boldsymbol{L'}$. In this and all subsequent proof systems, the right introduction rule for universal quantification is assumed to carry the usual proviso that the introduced constant does not appear free in the lower sequent. Similarly, the variable x does not appear free in a, $\forall x.D$ and $\exists x.G$ in rule **dec**-$\forall$.

The rules in the bottom section of Figure 1 describe how to reduce non-atomic goal formulas. They stem from the right introduction rules of linear logic, and are essentially identical to the right rules for $\boldsymbol{L'}$ [HM94]. $\boldsymbol{R}$ differs from $\boldsymbol{L'}$ in the treatment of atomic goal formulas. In order to handle these goals, Hodas and Miller rely on the function $\|\cdot\|$, which converts a formula in the program to a (possibly infinite) set of clauses, each defining a single ground atom. Here, we embed the process of clause selection and elaboration into the proof system itself, giving it a more syntactic and operational flavor.

When the goal formula a is atomic (Figure 1, center), a program formula D is selected from either the intuitionistic context (rule **res-atm_int**) or from the linear context (rule **res-atm_lin**). In either case, the program formula D and atomic goal a are passed to the *residuation* judgment

$$D \gg a \setminus G$$

(Figure 1, top) in order to produce a residual subgoal G. The computation then proceeds by solving G. Note that when **res-atom_lin** is used, D is removed from the context so that it may not be used again.

The residuation judgment has the property that G and D together imply a, that is, solving G is sufficient for a proof of a. In fact, it satisfies the stronger property that from the proof of G and the assumption D we can immediately construct a proof of a by using only left rules of the sequent calculus (or, equivalently, by using only elimination rules in natural deduction).

Moreover, the residuation judgment is completely deterministic: when D and a are given, there *always* exists a *unique* subgoal G such that $D \gg a \setminus G$. This means that all non-determinism in proof search is isolated in the resolution judgment $\Gamma; \Delta \Longrightarrow G$, which leads to an economical and uniform presentation of the various context management systems and their equivalence proofs.

Finally, the residuation judgment $D \gg a \setminus G$ is parametric in the atomic goal a, which means we can use it to compile D to G without knowledge of a.

3 A Resource Consumption Calculus for LHHF

The resolution calculus presented in the last section does not commit to any strategy in order to split the linear context when processing multiplicative goals from the bottom up. The non-determinism involved in this open choice can be computationally harmful unless we devise a sound and complete method to split the linear context deterministically. Let us restate the problem in terms of the proof system just described. The resolution rule for the connective $\otimes$ is as follows:

$$\frac{\Gamma; \Delta_1 \Longrightarrow G_1 \quad \Gamma; \Delta_2 \Longrightarrow G_2}{\Gamma; \underbrace{\Delta_1, \Delta_2}_{\Delta} \Longrightarrow G_1 \otimes G_2}\ \textbf{res-}\otimes$$

In order to construct a proof of the formula $G_1 \otimes G_2$, we need to split the original linear context Δ into Δ_1 and Δ_2 so that G_1 can be solved using the resources in Δ_1 and G_2 can be solved using the resources in Δ_2. Since intuitionistic formulas are reusable, all of Γ is copied to each of the two premisses. Assume that Δ contains n formulas. Then there are 2^n possible splits. In the worst case, finding a workable split (or determining that none exists) will require trying them all.

This problem was given a deterministic solution by Hodas and Miller in [HM94] in what they called the *I/O model* of execution for *Lolli*. We will instead use the name *resource management system* and refer to our presentation of this deduction system as $\boldsymbol{RM_1}$.

The rule above, **res-**$\otimes$, attempts to split the context Δ at a stage when the resources needed to prove the two subgoals G_1 and G_2 are completely unknown. However, if the original goal is to succeed, all resources not used to prove G_1 will be used to solve G_2, and vice versa. The key idea behind the resource consumption model is, therefore, to upgrade the role of goal formulas to be active *resource consumers*. Under this view, we will give one of the subgoals, say G_1, the whole linear context Δ; it will consume part of it and return the remaining portion Δ_2 to be used by G_2. This change in perspective fits well with a common view held in the linear logic community of goals as active processes.

$$\frac{D \gg a \setminus G \quad \Gamma, D; \Delta^I \setminus \Delta^O \Longrightarrow G}{\Gamma, D; \Delta^I \setminus \Delta^O \Longrightarrow a}\,\mathbf{rm_1\text{-}atm\_int} \qquad \frac{D \gg a \setminus G \quad \Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G}{\Gamma; \Delta^I, D \setminus \Delta^O \Longrightarrow a}\,\mathbf{rm_1\text{-}atm\_lin}$$

$$\frac{}{\Gamma; \Delta^I \setminus \Delta^I \Longrightarrow a \doteq a}\,\mathbf{rm_1\text{-}\doteq} \qquad \frac{}{\Gamma; \Delta^I \setminus \Delta^I \Longrightarrow \mathbf{1}}\,\mathbf{rm_1\text{-}1}$$

$$\frac{}{\Gamma; \Delta, \Delta^O \setminus \Delta^O \Longrightarrow \top}\,\mathbf{rm_1\text{-}\top} \qquad \text{(No rule for } \mathbf{0}\text{)}$$

$$\frac{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G_1 \quad \Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G_2}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G_1 \,\&\, G_2}\,\mathbf{rm_1\text{-}\&} \qquad \frac{\Gamma; \Delta^I \setminus \Delta' \Longrightarrow G_1 \quad \Gamma; \Delta' \setminus \Delta^O \Longrightarrow G_2}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G_1 \otimes G_2}\,\mathbf{rm_1\text{-}\otimes}$$

$$\frac{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G_1}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G_1 \oplus G_2}\,\mathbf{rm_1\text{-}\oplus_1} \qquad \frac{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G_2}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G_1 \oplus G_2}\,\mathbf{rm_1\text{-}\oplus_2}$$

$$\frac{\Gamma; \Delta, \Delta^O, D \setminus \Delta^O \Longrightarrow G}{\Gamma; \Delta, \Delta^O \setminus \Delta^O \Longrightarrow D \multimap G}\,\mathbf{rm_1\text{-}\multimap} \qquad \frac{\Gamma, D; \Delta^I \setminus \Delta^O \Longrightarrow G}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow D \supset G}\,\mathbf{rm_1\text{-}\supset}$$

$$\frac{\Gamma; \cdot \setminus \_ \Longrightarrow G}{\Gamma; \Delta^I \setminus \Delta^I \Longrightarrow !G}\,\mathbf{rm_1\text{-}!}$$

$$\frac{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow [c/x]G}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow \forall x.G}\,\mathbf{rm_1\text{-}\forall} \qquad \frac{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow [t/x]G}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow \exists x.G}\,\mathbf{rm_1\text{-}\exists}$$

Figure 2: $\boldsymbol{RM_1}$, a Resource Management System for *LHHF*

This basic idea is formalized in Figure 2 by means of judgments of the form:

$$\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G$$

where Δ^I is the linear part of the context that is given as *input* in order to prove G. In general, G will be just one of the subgoals produced during the derivation of a top-level goal A. The proof of G will consume part of Δ^I and return the portion it did not use as the *output context* Δ^O, that will need to be consumed by some other subgoal derived from A. Clearly the output context for the original overall goal A should be empty. Indeed, the soundness and completeness theorems for resource consumption below states that $\Gamma; \Delta \Longrightarrow G$ is derivable if and only if $\Gamma; \Delta \setminus \cdot \Longrightarrow G$ is derivable, where "$\cdot$" represents the empty context.

In their original paper, Hodas and Miller write this judgment $I\{G\}O$, with G being the goal formula, and I and O being the input and the output contexts respectively [HM94]. The main difference with respect to our judgment is that in their presentation I and O are lists. Each element can be either a linear program formula, an intuitionistic program formula (marked with the tag "!"), or the special constant `del`. This is very close to their original *Prolog* implementation of *LHHF* [CHMP]. Here, Δ^I and Δ^O are instead sets of labelled formulas and the intuitionistic part of the context has been separated out. This is consistent with the resolution judgment presented in Section 2, and permits easier proofs of soundness and completeness. We also make use of the residuation judgment in place of the special predicate pickr which they appeal to. Details of the correctness proofs following Hodas and Miller's formulation can be found in [Hod94, Section 7.1].

When considering the judgment $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G$, we adopt a computational point of view in which the schematic variables Γ, Δ^I and G are given as input to the rules, while Δ^O is returned as an output value from the resolution of the goal. This is consistent with a left-to-right subgoal selection strategy, that we adopt as well. Note, however, that the rules themselves do not commit to this operational interpretation. Rather, they are fully declarative.

We will not discuss the system $\boldsymbol{RM_1}$ in detail since it is isomorphic to the one presented by Hodas and Miller. We will simply point out a few features that will be relevant to the discussion of the refinements we present below.

- The resolution rules for the equality test (rule **res-** $\doteq$) and for the multiplicative unit (rule **res-1**) require an empty linear context, i.e., solving these goals does not consume resources. In $\boldsymbol{RM_1}$ we model this behavior by returning as output the same context these rules received as input. In a similar fashion, the exponential "!" expects its subgoal to be solvable in an empty linear context. Therefore, rule $\mathbf{rm_1}$**-!** passes the empty linear context to its premiss and returns the whole input context as output. In this rule and elsewhere, we write an output context that must be empty due to global invariants as "_". We use "$\cdot$" for the empty context in other circumstances, e.g., when the emptiness of a context needs to be checked to match a rule, or when setting an input context to empty.

- In the resolution system, $\top$ succeeds as a goal in any linear context. We model this behavior by allowing this formula to consume an arbitrary portion Δ of its input context.

- The operational behavior of additive conjunction & requires that we solve both subgoals G_1 and G_2 in the same linear context. This is modelled in $\boldsymbol{RM_1}$ by giving the original input context to both G_1 and G_2, and expecting them to return the same output context, Δ^O, that will be the output context of the compound formula $G_1 \,\&\, G_2$ (rule $\mathbf{rm_1}$**-&**).

- The rule for multiplicative implication ($\mathbf{rm_1}$-$\multimap$) requires some attention. Let Δ^I be the original input context. In order to process this connective, we need to augment Δ^I with the antecedent D of the implication. Let Δ^O be the context returned after solving its consequent G. We can return Δ^O as the output of the proof of $D \multimap G$ only if we are sure that the newly added instance of D does not appear in Δ^O. This is because this D must be consumed during the proof of G. We enforce this constraint by writing Δ^I as Δ, Δ^O and passing Δ, Δ^O, D as the input context to the premiss of the rule. By expecting Δ^O as the output of the whole subproof, we assert that Δ, D represents the portion of the input context that is consumed while proving G. No such complications are needed for the rule dealing with intuitionistic implication since its assumption is added to the intuitionistic context.

We conclude this section with the statements and sketches of the proofs of the soundness and completeness of $\boldsymbol{RM_1}$ with respect to $\boldsymbol{R}$. These results depend on two simple lemmas that we present first and that provide some insight into the behavior of these systems.

An important invariant of $\boldsymbol{RM_1}$, as well as of the enhanced versions to be introduced, is that, when the judgment $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G$ is derivable, the output context Δ^O is always a subset of the input context Δ^I. This property is formalized in the following lemma. Note that in this and many statements in the sequel, we will abbreviate phrases such as "*if the judgment J is derivable, then ...*" as "*if J, then ...*".

Lemma 3.1 (*Subcontext for* $\boldsymbol{RM_1}$)

If $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G$, *then* $\Delta^O \subseteq \Delta^I$.

Proof.

The proof proceeds by an easy structural induction on a derivation $\mathcal{I}$ of $\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow G$. ☑

We introduced the output context Δ^O of a judgment $\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow G$ as the part of the input context Δ^I *not* consumed in order to prove G. We can indeed write Δ^I as Δ, Δ^O, where Δ corresponds to the portion of the linear context actually used by G. Since Δ^O does not play any active role in the derivation, its actual composition, or its very presence, are unimportant: it can be replaced with any other context Δ' yielding a derivation of G with an isomorphic rule arrangement. This intuition is formalized in the next lemma.

Lemma 3.2 (*Output context replacement for $\boldsymbol{RM_1}$*)

If $\Gamma; \Delta, \Delta^O \backslash \Delta^O \Longrightarrow G$, *then* $\Gamma; \Delta, \Delta' \backslash \Delta' \Longrightarrow G$ *is derivable for every context* Δ'.

Proof.

By induction on the structure of a derivation of $\Gamma; \Delta, \Delta^O \backslash \Delta^O \Longrightarrow G$. ☑

In Sections 4 and 5, we will take advantage of two specific situations: the case where Δ' is some subcontext of Δ^O, possibly $\cdot$, and when Δ' extends Δ^O with additional program formulas. We therefore state the following corollaries.

Corollary 3.3 (*Output context deletion for $\boldsymbol{RM_1}$*)

If $\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow G$, *then* $\Gamma; (\Delta^I - \Delta^O) \backslash \cdot \Longrightarrow G$.

Proof.

Apply the previous lemma with $\Delta' = \Delta^I - \Delta^O$. ☑

Corollary 3.4 (*Output context augmentation for $\boldsymbol{RM_1}$*)

If $\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow G$, *then* $\Gamma; (\Delta^I, \Delta) \backslash (\Delta^O, \Delta) \Longrightarrow G$ *for every context* Δ.

Proof.

Apply the previous lemma with $\Delta' = \Delta^O, \Delta$. ☑

Similar results hold for the context management systems to be introduced in Sections 4 and 5. For the sake of conciseness, their statement will be kept implicit.

The soundness theorem of $\boldsymbol{RM_1}$ with respect to $\boldsymbol{R}$ relates any derivation in the resource management system to a resolution proof. As expected, the resources returned in the output context Δ^O, which are superfluous in order to prove a goal G from a given input context Δ^I, should be elided from Δ^I in order to construct a correct resolution derivation.

Theorem 3.5 (*Soundness of $\boldsymbol{RM_1}$ with respect to $\boldsymbol{R}$*)

If $\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow G$, *then* $\Gamma; (\Delta^I - \Delta^O) \Longrightarrow G$.

Proof.

This relatively simple proof proceeds by induction on the structure of a derivation $\mathcal{I}$ of $\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow G$. Observe that, by the subcontext lemma 3.1, $\Delta^O \subseteq \Delta^I$, and therefore the expression $(\Delta^I - \Delta^O)$ is well-formed. We only show the details of the case in which the derivation $\mathcal{I}$ ends with an application of rule **rm$_1$-atm_lin**. The remaining cases are similar or simpler.

Case rm$_1$-atm_lin: Suppose the given derivation has the form

$$\mathcal{I} = \cfrac{\begin{array}{cc} \mathcal{D} & \mathcal{I}' \\ D \gg a \setminus G' & \Gamma; \Delta_0^I \setminus \Delta^O \Longrightarrow G' \end{array}}{\Gamma; \Delta_0^I, D \setminus \Delta^O \Longrightarrow a} \,\textbf{rm1-atm\_lin}$$

with $\Delta^I = \Delta_0^I, D$ and $G = a$.

By induction hypothesis on $\mathcal{I}'$, there is a derivation $\mathcal{R}'$ of $\Gamma; (\Delta_0^I - \Delta^O) \Longrightarrow G'$. We can then apply rule **res-atm_lin** to $\mathcal{D}$ and $\mathcal{R}'$ to obtain a derivation $\mathcal{R}$ of

$$\Gamma; (\Delta_0^I - \Delta^O), D \Longrightarrow a.$$

Then the equation $(\Delta_0^I - \Delta^O), D = (\Delta_0^I, D) - \Delta^O$ yields the desired result. ☑

We expect the completeness theorem to state that if the judgment $\Gamma; \Delta \Longrightarrow G$ is derivable in $\boldsymbol{R}$, then so is $\Gamma; \Delta \setminus \cdot \Longrightarrow G$ in $\boldsymbol{RM_1}$. However, we need to generalize this result to cope with intermediate judgments, in particular those produced as premisses of rule **res-$\otimes$**. This is achieved in the following theorem, from which the expected property is obtained by choosing Δ^O to be empty. Notice the quantification pattern in this statement, which will recur in similar results in subsequent sections.

Theorem 3.6 (*Completeness of $\boldsymbol{RM_1}$ with respect to $\boldsymbol{R}$*)

If $\Gamma; \Delta \Longrightarrow G$, then, for every context Δ^O, the judgment $\Gamma; \Delta, \Delta^O \setminus \Delta^O \Longrightarrow G$ is derivable.

Proof.

The proof proceeds by induction on the structure of a derivation $\mathcal{R}$ for $\Gamma; \Delta \Longrightarrow G$. We give the details of two cases. The remaining possibilities are handled in a similar or simpler manner.

Case res-1:

$$\mathcal{R} = \cfrac{}{\Gamma; \cdot \Longrightarrow \mathbf{1}} \,\textbf{res-1}$$

with $\Delta = \cdot$ and $G = \mathbf{1}$.

Then, by rule **rm$_1$-1**, for every Δ^O, the judgment $\Gamma; \Delta^O \setminus \Delta^O \Longrightarrow \mathbf{1}$ is derivable. This concludes this case since $(\cdot, \Delta^O) = \Delta^O$.

Case res-$\otimes$:

$$\mathcal{R} = \cfrac{\begin{array}{cc} \mathcal{R}_1 & \mathcal{R}_2 \\ \Gamma; \Delta_1 \Longrightarrow G_1 & \Gamma; \Delta_2 \Longrightarrow G_2 \end{array}}{\Gamma; \Delta_1, \Delta_2 \Longrightarrow G_1 \otimes G_2} \,\textbf{res-}\otimes$$

with $\Delta = \Delta_1, \Delta_2$ and $G = G_1 \otimes G_2$.

By induction hypothesis on $\mathcal{R}_1$, for every context Δ_1^O there is a derivation of $\Gamma; \Delta_1, \Delta_1^O \setminus \Delta_1^O \Longrightarrow G_1$. Similarly, by induction hypothesis on $\mathcal{R}_2$, $\Gamma; \Delta_2, \Delta^O \setminus \Delta^O \Longrightarrow G_2$ for every context Δ^O. In particular, for $\Delta_1^O = \Delta_2, \Delta^O$, there is a derivation of

$$\Gamma; \Delta_1, \Delta_2, \Delta^O \setminus \Delta_2, \Delta^O \Longrightarrow G_1.$$

Therefore, by rule **rm$_1$-$\otimes$**, $\Gamma; \Delta_1, \Delta_2, \Delta^O \setminus \Delta^O \Longrightarrow G_1 \otimes G_2$. ☑

4 Removing Non-Determinism from the Treatment of $\top$

While the resource management policy enforced by system $\boldsymbol{RM_1}$ removes the most serious cause of non-determinism present in the resolution system $\boldsymbol{R}$, it is not yet fully deterministic. This is due to the operational semantics of the logical constant $\top$, as presented in rule $\mathbf{rm_1}$-$\top$:

$$\frac{}{\Gamma; \underbrace{\Delta, \Delta^O}_{\Delta^I} \setminus \Delta^O \Longrightarrow \top}\ \mathbf{rm_1}\text{-}\top$$

This goal is allowed to consume any portion Δ of its input context. If Δ^I contains n formulas, we are left with 2^n possible output contexts Δ^O that might be passed to the remaining computation. An alternative interpretation of this behavior views the branches of a proof tree ending with the resolution of $\top$ as permitting arbitrary weakening on the *linear* context. System $\boldsymbol{RM_1}$ does not address this hidden source of context management non-determinism.

Hodas and Miller initially underestimated the importance of this issue [HM94]. However the subsequent development of sample applications to accompany the first public release of *Lolli* showed this problem to be critical in practice. The solution we describe is adapted from Hodas' dissertation [Hod94, Section 7.4], and was incorporated into that implementation.

Roughly speaking, the idea is that once $\top$ has been encountered as a goal, the remaining subgoals do not need to consume all of their input context since the unused formulas could be "pumped back" to the place in the proof tree where $\top$ was first seen. That is, $\top$ should not actively consume resources on its own; rather, it should give permission to later goals to ignore resources which otherwise would have to be consumed.

We obtain this behavior by adding an extra parameter to the resource management judgments of $\boldsymbol{RM_1}$. We now use sequents of the form:

$$\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_v G$$

where v is a boolean-valued flag (the $\top$-*flag* or *slack indicator*) to be considered as another output argument of the resolution of the goal G. Whenever $v = 0$, the resolution of G uses exactly the resources in $\Delta^I - \Delta^O$. If instead this flag has the value 1, G uses $\Delta^I - \Delta^O$ for sure, but may also absorb part or all of the output context Δ^O. In this case, we say that Δ^O is the *slack* of that branch of the proof tree. When $v = 0$, the computation has no slack. The resulting system, called $\boldsymbol{RM_2}$, is presented in Figure 3.

The main changes with respect to $\boldsymbol{RM_1}$ concern the rules that close the proof trees, and the binary rules. Rules $\mathbf{rm_2}$-$\doteq$ and $\mathbf{rm_2}$-$\mathbf{1}$:

$$\frac{}{\Gamma; \Delta^I \setminus \Delta^I \Longrightarrow_0 a \doteq a}\ \mathbf{rm_2}\text{-}\doteq \qquad\qquad \frac{}{\Gamma; \Delta^I \setminus \Delta^I \Longrightarrow_0 \mathbf{1}}\ \mathbf{rm_2}\text{-}\mathbf{1}$$

both pass their linear context as the output context for the remainder of the computation, since neither can consume any resources. These rules set the $\top$-flag to 0 since no occurrence of $\top$ is encountered during the proof of either $\mathbf{1}$ or the equality test. In contrast, when $\top$ is processed as a goal in rule $\mathbf{rm_2}$-$\top$:

$$\frac{}{\Gamma; \Delta^I \setminus \Delta^I \Longrightarrow_1 \top}\ \mathbf{rm_2}\text{-}\top$$

it passes its input context as output too, but raises the $\top$-flag, indicating that it can be considered to have consumed some of those resources if that proves necessary. The subsequent computation will use this information for context management.

$$\frac{D \gg a \setminus G \quad \Gamma, D; \Delta^I \setminus \Delta^O \Longrightarrow_v G}{\Gamma, D; \Delta^I \setminus \Delta^O \Longrightarrow_v a}\ \textbf{rm}_\mathbf{2}\textbf{-atm\_int} \qquad \frac{D \gg a \setminus G \quad \Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_v G}{\Gamma; \Delta^I, D \setminus \Delta^O \Longrightarrow_v a}\ \textbf{rm}_\mathbf{2}\textbf{-atm\_lin}$$

$$\frac{}{\Gamma; \Delta \setminus \Delta \Longrightarrow_0 a \doteq a}\ \textbf{rm}_\mathbf{2}\textbf{-}\doteq \qquad \frac{}{\Gamma; \Delta \setminus \Delta \Longrightarrow_0 \mathbf{1}}\ \textbf{rm}_\mathbf{2}\textbf{-1}$$

$$\frac{}{\Gamma; \Delta \setminus \Delta \Longrightarrow_1 \top}\ \textbf{rm}_\mathbf{2}\textbf{-}\top \qquad \text{(No rule for } \mathbf{0}\text{)}$$

$$\frac{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_1 \quad \Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_2}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_1 \,\&\, G_2}\ \textbf{rm}_\mathbf{2}\textbf{-}\&_\mathbf{00}$$

$$\frac{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_1 \quad \Gamma; \Delta^I \setminus \Delta_2, \Delta^O \Longrightarrow_1 G_2}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_1 \,\&\, G_2}\ \textbf{rm}_\mathbf{2}\textbf{-}\&_\mathbf{01}$$

$$\frac{\Gamma; \Delta^I \setminus \Delta_1, \Delta^O \Longrightarrow_1 G_1 \quad \Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_2}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_1 \,\&\, G_2}\ \textbf{rm}_\mathbf{2}\textbf{-}\&_\mathbf{10}$$

$$\frac{\Gamma; \Delta^I \setminus \Delta_1^O \Longrightarrow_1 G_1 \quad \Gamma; \Delta^I \setminus \Delta_2^O \Longrightarrow_1 G_2}{\Gamma; \Delta^I \setminus \Delta_1^O \cap \Delta_2^O \Longrightarrow_1 G_1 \,\&\, G_2}\ \textbf{rm}_\mathbf{2}\textbf{-}\&_\mathbf{11}$$

$$\frac{\Gamma; \Delta^I \setminus \Delta' \Longrightarrow_v G_1 \quad \Gamma; \Delta' \setminus \Delta^O \Longrightarrow_w G_2}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_{v \vee w} G_1 \otimes G_2}\ \textbf{rm}_\mathbf{2}\textbf{-}\otimes_\mathbf{vw}$$

$$\frac{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_v G_1}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_v G_1 \oplus G_2}\ \textbf{rm}_\mathbf{2}\textbf{-}\oplus_\mathbf{1} \qquad \frac{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_v G_2}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_v G_1 \oplus G_2}\ \textbf{rm}_\mathbf{2}\textbf{-}\oplus_\mathbf{2}$$

$$\frac{\Gamma; \Delta^O, \Delta, D \setminus \Delta^O \Longrightarrow_0 G}{\Gamma; \Delta^O, \Delta \setminus \Delta^O \Longrightarrow_0 D \multimap G}\ \textbf{rm}_\mathbf{2}\textbf{-}\multimap_\mathbf{0} \qquad \frac{\Gamma; \Delta^I, D \setminus \Delta^O \Longrightarrow_1 G}{\Gamma; \Delta^I \setminus \Delta^I \cap \Delta^O \Longrightarrow_1 D \multimap G}\ \textbf{rm}_\mathbf{2}\textbf{-}\multimap_\mathbf{1}$$

$$\frac{\Gamma, D; \Delta^I \setminus \Delta^O \Longrightarrow_v G}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_v D \supset G}\ \textbf{rm}_\mathbf{2}\textbf{-}\supset \qquad \frac{\Gamma; \cdot \setminus \_ \Longrightarrow_v G}{\Gamma; \Delta \setminus \Delta \Longrightarrow_0 !G}\ \textbf{rm}_\mathbf{2}\textbf{-!}$$

$$\frac{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_v [c/x]G}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_v \forall x.G}\ \textbf{rm}_\mathbf{2}\textbf{-}\forall \qquad \frac{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_v [t/x]G}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_v \exists x.G}\ \textbf{rm}_\mathbf{2}\textbf{-}\exists$$

Figure 3: $\boldsymbol{RM_2}$, an Improved Resource Management System for *LHHF*

Rule $\mathbf{rm_1}$**-&** is split into four rules in $\boldsymbol{RM_2}$. Each rule handles one possible combination of $\top$-flags returned by the two premisses. If no $\top$ was encountered while solving either G_1 or G_2 (rule $\mathbf{rm_2}$**-**$\&_\mathbf{00}$), then the context is managed as in the previous system and the $\top$-flag for the proof of the compound goal $G_1 \,\&\, G_2$ is set to 0.

When exactly one of the two premisses sets the slack indicator, then the behavior of the rule is determined by the other premiss. Consider for example the case where the left premiss sets the

$\top$-flag (the other case, rule $\mathbf{rm_2\text{-}\&_{01}}$, is symmetrical). We have the following rule:

$$\frac{\Gamma; \Delta^I \backslash \Delta_1, \Delta^O \Longrightarrow_1 G_1 \quad \Gamma; \Delta^I \backslash \Delta^O \Longrightarrow_0 G_2}{\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow_0 G_1 \,\&\, G_2}\ \mathbf{rm_2\text{-}\&_{10}}$$

Let Δ be the portion of the context used while proving G_2 (clearly, $\Delta^I = \Delta, \Delta^O$). Since both G_1 and G_2 must consume the same portion of the context, the proof of G_1 can use part of Δ but no formula from Δ^O. However, it does not need to consume explicitly all the formulas in Δ, because, unlike G_2, its slack indicator is set. We can therefore write the output context of G_1 as Δ_1, Δ^O, where Δ_1 is the actual slack of this branch of the proof tree, and is some subcontext of Δ. The $\top$-flag for the proof of $G_1 \,\&\, G_2$ is set to 0: since both premisses must consume the same resources and G_2 cannot take up slack, the composed goal cannot have any slack. For the same reason, the output context of $G_1 \,\&\, G_2$ is Δ^O.

In the final case, if both premisses return their $\top$-flag set to 1, both subgoals allow arbitrary slack. Therefore, we set the $\top$-flag for the proof of the compound formula, since in this case any excess resources can be "pumped back" to both premisses. The output context for the compound goal is the intersection of the output contexts returned by each of the premisses: since both branches must end up having consumed the same resources, only what is not used in either branch can be forwarded. This yields the following rule:

$$\frac{\Gamma; \Delta^I \backslash \Delta_1^O \Longrightarrow_1 G_1 \quad \Gamma; \Delta^I \backslash \Delta_2^O \Longrightarrow_1 G_2}{\Gamma; \Delta^I \backslash \Delta_1^O \cap \Delta_2^O \Longrightarrow_1 G_1 \,\&\, G_2}\ \mathbf{rm_2\text{-}\&_{11}}$$

Slack handling in the rule for multiplicative conjunction is quite simple since resources are allowed to flow freely from one premiss to the other. We set the $\top$-flag if either subgoal allows slack, indicated by $v \vee w$. The overall output context is the linear context returned after proving the right premiss. It will be convenient to distinguish four rules for this connective, corresponding to the four possible slack indicator combinations in the premisses.

Finally, the rule for ! resets the $\top$-flag regardless of whether $\top$ has been encountered while solving its subgoal or not. Since the output context must coincide with the input context in this rule, there is no place for any slack.

We will now formalize the correspondence between $\boldsymbol{RM_2}$ and $\boldsymbol{RM_1}$ (and indirectly to $\boldsymbol{R}$). As with the basic resource management scheme, the output context of an $\boldsymbol{RM_2}$ is always a subset of its input context.

Lemma 4.1 (*Subcontext for* $\boldsymbol{RM_2}$)

If $\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow_v G$, *then* $\Delta^O \subseteq \Delta^I$.

Proof.

By induction on the structure of a derivation for $\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow_v G$. ☑

The output context replacement property applies also to $\boldsymbol{RM_2}$, as expressed by the following lemma.

Lemma 4.2 (*Output context replacement for* $\boldsymbol{RM_2}$)

If $\Gamma; \Delta, \Delta^O \backslash \Delta^O \Longrightarrow_v G$, *then* $\Gamma; \Delta, \Delta' \backslash \Delta' \Longrightarrow_v G$ *is derivable for every context* Δ'.

Proof.

By induction on the structure of a derivation for $\Gamma; \Delta, \Delta^O \setminus \Delta^O \Longrightarrow_v G$. ☑

While in the previous lemmas the two different judgments that participate in $\boldsymbol{RM_2}$ behaved uniformly, we must treat the two possible values of the $\top$-flag separately in the following soundness result. Indeed, whenever slack is not permitted, every $\boldsymbol{RM_2}$ judgment is provable in $\boldsymbol{RM_1}$. However, when the $\top$-flag is set to 1, any portion of the output context can be "pumped back" to some occurrence of $\top$. These facts are formally expressed in the following theorem.

Theorem 4.3 (*Soundness of $\boldsymbol{RM_2}$ with respect to $\boldsymbol{RM_1}$*)

i. If $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_0 G$, *then* $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G$.

ii. If $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_1 G$, *then* $\Gamma; \Delta^I \setminus \Delta \Longrightarrow G$ *for every context* $\Delta \subseteq \Delta^O$.

Proof.

The proof proceeds by mutual induction on the structure of a derivation $\mathcal{I}$ of $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_v G$ for the two parts of the statement. We will consider one representative situation. The remaining cases are similar or simpler.

Case $\mathbf{rm_2}$-$\otimes_{\mathbf{10}}$:

$$\mathcal{I} = \dfrac{\begin{array}{cc} \mathcal{I}_1 & \mathcal{I}_2 \\ \Gamma; \Delta^I \setminus \Delta' \Longrightarrow_1 G_1 & \Gamma; \Delta' \setminus \Delta^O \Longrightarrow_0 G_2 \end{array}}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_1 G_1 \otimes G_2}\ \mathbf{rm_2\text{-}\otimes_{10}}$$

with $G = G_1 \otimes G_2$.

By induction hypothesis (*ii*) on $\mathcal{I}_1$, there is a derivation of $\Gamma; \Delta^I \setminus \Delta^* \Longrightarrow G_1$ for every context $\Delta^* \subseteq \Delta'$. By the subcontext lemma 4.1 on $\mathcal{I}_2$, $\Delta^O \subseteq \Delta'$, and therefore $\Delta' = \tilde{\Delta}, \Delta^O$ for some context $\tilde{\Delta}$. Therefore, for any $\Delta \subseteq \Delta^O$, there is a derivation $\mathcal{I}_1'$ of

$$\Gamma; \Delta^I \setminus \tilde{\Delta}, \Delta \Longrightarrow G_1.$$

By induction hypothesis (*i*) on $\mathcal{I}_2$, $\Gamma; \Delta' \setminus \Delta^O \Longrightarrow G_2$. By the output context replacement lemma 3.2, there is a derivation $\mathcal{I}_2'$ of

$$\Gamma; \tilde{\Delta}, \Delta \setminus \Delta \Longrightarrow G_2.$$

It suffices now to apply rule $\mathbf{rm_1}$-$\otimes$ to $\mathcal{I}_1'$ and $\mathcal{I}_2'$ to obtain the desired result. ☑

As an immediate consequence of this result, we have that $\boldsymbol{RM_1}$ admits weakening on the linear context whenever it corresponds to an $\boldsymbol{RM_2}$ derivation that allows slack. This is formalized as the following corollary.

Corollary 4.4 (*Weakening in $\boldsymbol{RM_2}$ in the presence of slack*)

If $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_1 G$, *then* $\Gamma; \Delta^I, \Delta \setminus \Delta^O \Longrightarrow G$ *for any context* Δ.

Proof.

By the replacement lemma 4.2, we have that $\Gamma; \Delta^I, \Delta \setminus \Delta^O, \Delta \Longrightarrow_1 G$ is derivable for every context Δ. By the above soundness theorem, $\Gamma; \Delta^I, \Delta \setminus \Delta' \Longrightarrow G$ for every $\Delta' \subseteq \Delta^O, \Delta$. The desired derivation is obtained by choosing Δ' to be Δ^O. ☑

Notice that a similar property does not hold in $\boldsymbol{RM_2}$ since all slack is collected in the output context.

The completeness of $\boldsymbol{RM_2}$ with respect to $\boldsymbol{RM_1}$ is expressed by the following theorem. Depending on whether the given derivation of the judgment $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G$ mentions certain occurrences of rule $\mathbf{rm_1}$-$\top$, the slack indicator of the corresponding $\boldsymbol{RM_2}$ judgment will be set to 0 or 1. Notice that, in the latter case, Δ^O will in general be a subcontext of the produced output context.

Theorem 4.5 (*Completeness of $\boldsymbol{RM_2}$ with respect to $\boldsymbol{RM_1}$*)

If $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G$, *then*

- *either* $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_0 G$,
- *or* $\Gamma; \Delta^I \setminus \Delta, \Delta^O \Longrightarrow_1 G$ *for some* $\Delta \subseteq \Delta^I - \Delta^O$.

Proof.

The proof proceeds by induction on the structure of a derivation $\mathcal{I}$ of $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G$. We show the details of the most complex case.

Case $\mathbf{rm_1}$-&:

$$\mathcal{I} = \cfrac{\begin{matrix}\mathcal{I}_1 \\ \Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G_1\end{matrix} \quad \begin{matrix}\mathcal{I}_2 \\ \Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G_2\end{matrix}}{\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow G_1 \,\&\, G_2}\ \mathbf{rm_1}\text{-}\&$$

with $G = G_1 \,\&\, G_2$.

By induction hypothesis on $\mathcal{I}_i$ either $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_i$ or $\Gamma; \Delta^I \setminus \Delta_i, \Delta^O \Longrightarrow_1 G_i$ for some $\Delta_i \subseteq \Delta^I - \Delta^O$, for $i = 1, 2$. Since there are two possibilities for each of the two subderivations $\mathcal{I}_i$, we must consider four possible $\top$-flag combinations:

Subcase (0,0): Then, the induction hypothesis has produced derivations of $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_1$ and $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_2$. Combining them by means of rule $\mathbf{rm_2}$-$\&_{\mathbf{00}}$ yields the desired result.

Subcase (0,1): We have derivations of $\Gamma; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_1$ and $\Gamma; \Delta^I \setminus \Delta_2, \Delta^O \Longrightarrow_1 G_2$ for some $\Delta_2 \subseteq \Delta^I - \Delta^O$. It suffices then to apply rule $\mathbf{rm_2}$-$\&_{\mathbf{01}}$.

Subcase (1,0): We proceed in a symmetric way.

Subcase (1,1): We know that $\Gamma; \Delta^I \setminus \Delta_1, \Delta^O \Longrightarrow_1 G_1$ for some $\Delta_1 \subseteq \Delta^I - \Delta^O$ and $\Gamma; \Delta^I \setminus \Delta_2, \Delta^O \Longrightarrow_1 G_2$ for some $\Delta_2 \subseteq \Delta^I - \Delta^O$. An application of rule $\mathbf{rm_2}$-$\&_{\mathbf{11}}$ yields a derivation of

$$\Gamma; \Delta^I \setminus (\Delta_1, \Delta^O) \cap (\Delta_2, \Delta^O) \Longrightarrow_1 G_1 \,\&\, G_2,$$

which is the desired result if we take Δ to be $\Delta_1 \cap \Delta_2$ and we observe that $(\Delta_1, \Delta^O) \cap (\Delta_2, \Delta^O) = \Delta, \Delta^O$. ☑

It is important to note that $\boldsymbol{RM_1}$ and $\boldsymbol{RM_2}$ improve the efficiency of the resolution system $\boldsymbol{R}$ in two different ways. The proofs obtainable in $\boldsymbol{RM_1}$ are in one to one correspondence with the derivations we could achieve with $\boldsymbol{R}$. $\boldsymbol{RM_1}$ improves the efficiency of proof-search by pruning from the search space branches corresponding to unsuccessful splits of the linear context. In contrast, the system $\boldsymbol{RM_2}$ actually collapses some proofs by identifying successful derivations that differ only by the distribution of unused assumptions among various occurrences of $\top$. For example, consider an attempt to solve the goal $a \multimap b \multimap c \multimap (\top \otimes \top)$ in the empty context. There are eight distinct proofs in $\boldsymbol{RM_1}$ corresponding to the different ways of dividing the consumption of the context (a, b, c) between the two occurrences of $\top$. This is summarized in the following schematic derivation:

$$\dfrac{\dfrac{\dfrac{}{\cdot; a, b, c \backslash \Delta \Longrightarrow \top}\,\mathbf{rm_1}\text{-}\top \qquad \dfrac{}{\cdot; \Delta \backslash \cdot \Longrightarrow \top}\,\mathbf{rm_1}\text{-}\top}{\cdot; a, b, c \backslash \cdot \Longrightarrow \top \otimes \top}\,\mathbf{rm_1}\text{-}\otimes}{\cdot; \cdot \backslash \cdot \Longrightarrow a \multimap b \multimap c \multimap (\top \otimes \top)}\,\mathbf{rm_1}\text{-}\multimap \ \textit{(3 times)}$$

where Δ can be any of the 8 subcontexts of (a, b, c). On the other hand, there is only one proof of $a \multimap b \multimap c \multimap (\top \otimes \top)$ in $\boldsymbol{RM_2}$:

$$\dfrac{\dfrac{\dfrac{}{\cdot; a, b, c \backslash a, b, c \Longrightarrow_1 \top}\,\mathbf{rm_2}\text{-}\top \qquad \dfrac{}{\cdot; a, b, c \backslash a, b, c \Longrightarrow_1 \top}\,\mathbf{rm_2}\text{-}\top}{\cdot; a, b, c \backslash a, b, c \Longrightarrow_1 \top \otimes \top}\,\mathbf{rm_2}\text{-}\otimes_{11}}{\cdot; \cdot \backslash \cdot \Longrightarrow_1 a \multimap b \multimap c \multimap (\top \otimes \top)}\,\mathbf{rm_2}\text{-}\multimap_1 \ \textit{(3 times)}$$

5 Improving the Treatment of Additive Conjunction

The system $\boldsymbol{RM_2}$ presented in the last section achieves determinism in context management to the extent that no arbitrary context splitting choices remain. Nevertheless, a close examination of the rules reveals that some serious efficiency and completeness problems still remain. In particular, the rules concerning & are unsatisfactory. The problem is already present in $\boldsymbol{RM_1}$, where we had the following rule:

$$\dfrac{\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow G_1 \quad \Gamma; \Delta^I \backslash \Delta^O \Longrightarrow G_2}{\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow G_1 \,\&\, G_2}\,\mathbf{rm_1}\text{-}\&$$

Assuming a sequential execution for the two premisses, this rule requires that we first solve G_1 obtaining, say, an output context Δ_1^O. Then G_2 will be proved and return the output context Δ_2^O. At this point, and only at this point, we check that Δ_1^O and Δ_2^O are equal.

Even though this test can be done efficiently (for example by having a bit vector where each position records whether the corresponding resource has been used), we may end up rejecting many pairs of proofs before finding a pair that consumes the same set of resources. At best this is inefficient. At worst, when a proof of G_2 proceeds down a divergent path that it might avoid with better pruning, it leads to added incompleteness in the system. Further, in a language with a notion of side-effect (such as screen output), an avoidable failed proof may nevertheless produce a recordable effect.

An example, written using *Lolli*'s concrete syntax, will help illustrating this point. *Lolli* allows mixing intuitionistic and linear clauses in a program. It distinguishes the latter with the keyword `LINEAR`. In the programs below, we will make use of the logical constant $\mathbf{1}$, written `true`, and of three connectives, $\multimap$, & and $\otimes$, written `:-` (with the arguments reversed), `&` and `,` (comma), respectively. Program clauses are terminated with a dot (`.`). We rely on the extra-logical operator `write`, which outputs the string it is given as an argument.

Consider the following example:

```
test :- (a & b), c.
```

```
LINEAR c.
a.
b :- c, write "Some Output".     % Fails, but prints
```

In a left-to-right execution model, the goal '`?- test.`' is solved by first proving `a` (without consuming any linear resources), then attempting to prove `b`. The clause for this goal is selected and its body attempted. The linear resource `c` is consumed, the message is printed, and `b` succeeds. At this point, the resources consumed while solving `a` and `b` are compared and the conjunction fails since the latter conjunct used `c` while the former did not. This causes the failure of the original query. Clearly, it would be preferable for the attempt to solve `b` to fail as soon as `c` is accessed, so that the message is never printed.

Even in a *Prolog*-based implementation [HM94] (one is included in [CHMP]), where the constraint on the output contexts is enforced by unification rather than by an after-the-fact check, the same problem occurs if we replace the body of the rule for `b` with '`c, write "Some Output", true`'. The problem is that an "output constraint" on the result of the search is not as strong as a priori constraining the input. This is because intermediate rules (those dealing with $\top$ in particular), prohibit the propagation of constraints on the output all the way to the input.

In order to more quickly recognize those failures caused by the second goal incorrectly accessing resources unused by the first, we could modify the rule $\mathbf{rm_1}$-$\mathbf{\&}$ as follows:

$$\frac{\Gamma; \Delta^I \backslash\, \Delta^O \Longrightarrow G_1 \quad \Gamma; \Delta^I - \Delta^O \backslash \cdot \Longrightarrow G_2}{\Gamma; \Delta^I \backslash\, \Delta^O \Longrightarrow G_1 \,\&\, G_2}\ \mathbf{rm_1}\text{-}\&'$$

In this rule, we give G_2 exactly the portion of the linear context that it can use and expect the empty context as an output. In this way, the resources not consumed by G_1 are inaccessible to G_2; this achieves our purposes.

This change will not, however, help the system to detect early enough failures caused by the second conjunct failing to consume resources that the first conjunct does use. To see how this becomes an issue, consider another *Lolli* program:

```
test :- (a, c) & b.
LINEAR a.
LINEAR c.
b :- c & (write "Some Output", c).     % Fails, but prints
```

If we execute the query '`?- test.`' the system will first solve the goal to the left of the additive conjunction by consuming `a` and then `c`. At this point it will attempt to prove `b`. Since the left conjunct has used all of the resources in the input context, `b` can and must use them all as well (so there is no new restriction added by the change to the rule for & we just described). The rule for `b` is selected, and its left conjunct is solved using just `c`. At this point, since the right conjunct can only use `c` but the overall proof of `b` was supposed to use both `a` and `c`, we know enough to fail. Unfortunately, the system will not recognize this situation and will print the message. The resource `c` will then be consumed and the proof of `b` will succeed, having consumed `c`. Only when checking that all the resources passed to `b` have been used, will the system finally recognize the failure, and cause the original query to fail.

In order to obtain the desired behavior, we modify the form of our judgment to include three input contexts on the left of the arrow:

$$\Gamma; \Xi; \Delta^I \backslash\, \Delta^O \Longrightarrow_v G$$

$$\frac{D \gg a \setminus G \quad (\Gamma, D); \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v G}{(\Gamma, D); \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v a} \text{rm}_3\text{-atm\_int}$$

$$\frac{D \gg a \setminus G \quad \Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v G}{\Gamma; \Xi; (\Delta^I, D) \setminus \Delta^O \Longrightarrow_v a} \text{rm}_3\text{-atm\_lax} \qquad \frac{D \gg a \setminus G \quad \Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v G}{\Gamma; (\Xi, D); \Delta^I \setminus \Delta^O \Longrightarrow_v a} \text{rm}_3\text{-atm\_strict}$$

$$\frac{}{\Gamma; \cdot; \Delta^I \setminus \Delta^I \Longrightarrow_0 a \doteq a} \text{rm}_3\text{-}\doteq \qquad \frac{}{\Gamma; \cdot; \Delta^I \setminus \Delta^I \Longrightarrow_0 \mathbf{1}} \text{rm}_3\text{-}\mathbf{1}$$

$$\frac{}{\Gamma; \Xi; \Delta^I \setminus \Delta^I \Longrightarrow_1 \top} \text{rm}_3\text{-}\top \qquad \text{(No rule for } \mathbf{0}\text{)}$$

$$\frac{\Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_1 \quad \Gamma; (\Xi, \Delta^I - \Delta^O); \cdot \setminus \_ \Longrightarrow_v G_2}{\Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_1 \,\&\, G_2} \text{rm}_3\text{-}\&_{0v}$$

$$\frac{\Gamma; \Xi; \Delta^I \setminus \Delta' \Longrightarrow_1 G_1 \quad \Gamma; (\Xi, \Delta^I - \Delta'); \Delta' \setminus \Delta^O \Longrightarrow_v G_2}{\Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v G_1 \,\&\, G_2} \text{rm}_3\text{-}\&1v$$

$$\frac{\Gamma; \cdot; (\Xi, \Delta^I) \setminus \Delta' \Longrightarrow_0 G_1 \quad \Gamma; (\Xi \cap \Delta'); (\Delta^I \cap \Delta') \setminus \Delta^O \Longrightarrow_v G_2}{\Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v G_1 \otimes G_2} \text{rm}_3\text{-}\otimes_{0v}$$

$$\frac{\Gamma; \cdot; (\Xi, \Delta^I) \setminus \Delta' \Longrightarrow_1 G_1 \quad \Gamma; \cdot; \Delta' \setminus \Delta^O \Longrightarrow_v G_2}{\Gamma; \Xi; \Delta^I \setminus \Delta^I \cap \Delta^O \Longrightarrow_1 G_1 \otimes G_2} \text{rm}_3\text{-}\otimes_{1v}$$

$$\frac{\Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v G_1}{\Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v G_1 \oplus G_2} \text{rm}_3\text{-}\oplus_1 \qquad \frac{\Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v G_2}{\Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v G_1 \oplus G_2} \text{rm}_3\text{-}\oplus_2$$

$$\frac{\Gamma; (\Xi, D); \Delta^I \setminus \Delta^O \Longrightarrow_v G}{\Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v D \multimap G} \text{rm}_3\text{-}\multimap \qquad \frac{(\Gamma, D); \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v G}{\Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v D \supset G} \text{rm}_3\text{-}\supset$$

$$\frac{\Gamma; \cdot; \cdot \setminus \_ \Longrightarrow_v G}{\Gamma; \cdot; \Delta^I \setminus \Delta^I \Longrightarrow_0 !G} \text{rm}_3\text{-}!$$

$$\frac{\Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v [c/x]G}{\Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v \forall x.G} \text{rm}_3\text{-}\forall \qquad \frac{\Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v [t/x]G}{\Gamma; \Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v \exists x.G} \text{rm}_3\text{-}\exists$$

Figure 4: $\boldsymbol{RM_3}$, a Further Improved Resource Management System for *LHHF*.

In this judgment (which also features the slack indicator v of $\boldsymbol{RM_2}$) the input linear context is logically divided into two parts: the *strict* context Ξ that must be entirely consumed during the resolution of the goal G, and the *lax* context Δ^I whose contents might be consumed while solving G. Thus the strict context Ξ will be managed like the linear context in the system $\boldsymbol{R}$; only the lax context Δ^I may transmit unused resources to the output Δ^O, as in $\boldsymbol{RM_2}$. The rules defining the semantics of this judgment are represented in Figure 4; together with the rules for residuation, they constitute the system $\boldsymbol{RM_3}$. We will now briefly describe the principal characteristics of this system.

First, since we have split the linear context, we need to provide two separate rules for accessing a linear formula when the goal is atomic (rules **rm$_3$-atm_lax** and **rm$_3$-atm_strict**). The rules for the equality judgment and for proving the goal **1** are straightforward (rules **rm$_3$-** $\doteq$ and **rm$_3$-1**): since neither is allowed to consume any resources, the strict context (which contains resources that must be consumed) must be empty; the lax context is passed over unmodified as output. In contrast, the rule for $\top$ deletes whatever portion of the strict context it is provided with, and forwards as output its lax context, while setting the $\top$-flag to indicate that the output is now slack (rule **rm$_3$-**$\top$).

The rules for & are more complicated. In order to solve the goal $G_1 \,\&\, G_2$ with respect to the linear context $\Xi; \Delta^I$, we first solve G_1 in $\Xi; \Delta^I$, obtaining the output context Δ' (remember, Ξ must be entirely consumed in each of the two conjuncts). Two different courses of action are now possible, depending on the value of the slack indicator:

1. If this flag was not set (rules **rm$_3$-&$_{\mathbf{0v}}$**), the output is fixed to be $\Delta' = \Delta^O$. Moreover, G_2 must consume everything that has been used by G_1, i.e., Ξ as well as $\Delta^I - \Delta'$. These two components are packaged together into the strict context of the judgment for G_2. Since this goal is not allowed to consume any other resources, it is given an empty lax context.

2. If the resolution of G_1 has encountered an occurrence of $\top$ and slack is admitted (rules **rm$_3$-&$_{\mathbf{1v}}$**), G_2 must still consume every resource used by G_1 (i.e. $\Xi, \Delta^I - \Delta'$), but is also allowed to access the resources not used by this goal (Δ'), therefore, we supply this as the lax context for the proof of G_2. The output context and slack indicator for this second premiss then provide the corresponding values for the lower sequent.

When solving a goal of the form $G_1 \otimes G_2$, the strict context Ξ must be consumed by either G_1 or G_2. Since the first of these subgoals may use an arbitrary part of Ξ as well as some portion of the lax context Δ^I, we put both Ξ and Δ^I in the lax context of G_1 and leave the strict context empty. As with &, how to solve G_2 depends on the value of the $\top$-flag.

1. If no slack is allowed (rules **rm$_3$-**$\otimes_{\mathbf{0v}}$), G_2 must consume whatever portion of the original strict context G_1 did not use, and may consume some formulas in Δ^I that were not already consumed by G_1. Therefore, we restore the remainder of Ξ and Δ^I to the strict and lax contexts of the judgment for G_2, respectively. To do this we take the intersection of these contexts with the output context Δ' of G_1.

   The importance of requiring assumptions to have unique labels is particularly apparent in this rule. Indeed, assume we drop this requirement and try to prove the goal $a \otimes a$ in a situation where both the strict and the lax context contain only the (unlabelled) assumption a. In order to do so, both copies of a are packaged in the lax context of the leftmost premiss of these rules. A derivation $\mathcal{I}_1$ of

   $$\cdot; \cdot; (a, a) \backslash\, a \Longrightarrow_0 a$$

   is easily found. We need now to intersect the resulting output context (a) with the original strict and lax contexts (both a) in order to assemble the rightmost premiss. Without labels to distinguish the occurrences of the context formula a we started with, both intersections evaluate to a and a derivation $\mathcal{I}_2$ of

   $$\cdot; a; a \backslash\, a \Longrightarrow_0 a$$

   is incorrectly produced. These two derivations can then be combined by means of rule **rm$_3$-**$\otimes_{\mathbf{00}}$: a single linear assumption a has been consumed in order to prove the goal $a \otimes a$.

Unique labelling prevents this erroneous behavior by distinguishing which occurrence of a is returned in the output context of $\mathcal{I}_1$. Then, intersection will cause either the strict or the lax context of the rightmost premiss to be empty. Therefore, both assumptions a will be used to prove $a \otimes a$. Notice that the overall sequent $\cdot; a; a \backslash \cdot \Longrightarrow_0 a \otimes a$ has two derivations since the leftmost premiss can consume either the strict or the lax assumption a in order to prove the subgoal a, leaving the other assumption for the second subgoal.

2. If the slack indicator was set by the proof of G_1 (rules $\mathbf{rm_3}$-$\otimes_{\mathbf{1v}}$), all the strict resources in the original Ξ can be "pumped back" to G_1 in the case that G_2 does not use them. Therefore we call this goal with an empty strict context and the output context of G_1, Δ', as its lax context. We must be careful, however, not to return strict resources from Ξ as part of the output context of $G_1 \otimes G_2$, since they are presumed to have been used by the slack consumer in G_1. Therefore we intersect the context returned by G_2 with the original lax input context Δ^I of the composed goal.

The rule dealing with linear implication takes advantage of the strict context to simplify the task of managing the new assumption (rules $\mathbf{rm_3}$-$\multimap$). Since D must be used while proving G, it is simply put into the strict context of this subgoal. The rules for $\supset$, $!$, $\oplus$, and the quantifiers display no interesting new features.

We conclude this section by proving the correspondence between $\boldsymbol{RM_2}$ and $\boldsymbol{RM_3}$. We first present an adaptation of the subcontext lemma already encountered in the previous two context management systems. Notice that the output context may not mention any formula occurring in the strict context.

Lemma 5.1 (*Subcontext for* $\boldsymbol{RM_3}$)

If $\Gamma; \Xi; \Delta^I \backslash \Delta^O \Longrightarrow_v G$, *then* $\Delta^O \subseteq \Delta^I$.

Proof.

By induction on the structure of a derivation for $\Gamma; \Xi; \Delta^I \backslash \Delta^O \Longrightarrow_v G$. ☑

Next, we need to prove a version of the output context replacement lemma specifically tailored for $\boldsymbol{RM_3}$. Notice that this property holds only relative to the lax context. This is due to the fact that, by the above subcontext lemma, no assumption in the strict context is ever passed as output.

Lemma 5.2 (*Output context replacement for* $\boldsymbol{RM_3}$)

If $\Gamma; \Xi; \Delta, \Delta^O \backslash \Delta^O \Longrightarrow_v G$, *then* $\Gamma; \Xi; \Delta, \Delta' \backslash \Delta' \Longrightarrow_v G$ *is derivable for every context* Δ'.

Proof.

By induction on the structure of a derivation of $\Gamma; \Xi; \Delta, \Delta^O \backslash \Delta^O \Longrightarrow_v G$. ☑

The soundness theorem below maps derivations in $\boldsymbol{RM_3}$ to $\boldsymbol{RM_2}$ proofs by collapsing the strict and lax context of that system into the single input context of an $\boldsymbol{RM_2}$ sequent. Care must be taken in the presence of slack since, due to the form of rule $\mathbf{rm_2}$-$\top$, part of the strict context might need to be returned as output in that system.

Theorem 5.3 (*Soundness of* $\boldsymbol{RM_3}$ *with respect to* $\boldsymbol{RM_2}$)

i. If $\Gamma; \Xi; \Delta^I \backslash \Delta^O \Longrightarrow_0 G$, *then* $\Gamma; (\Xi, \Delta^I) \backslash \Delta^O \Longrightarrow_0 G$.

ii. If $\Gamma;\Xi;\Delta^I\backslash\,\Delta^O \Longrightarrow_1 G$*, then* $\Gamma;(\Xi,\Delta^I)\backslash\,(\Delta^O,\Xi') \Longrightarrow_1 G$ *for some context* $\Xi' \subseteq \Xi$*.*

Proof.

The proof is conducted by mutual induction on the structure of a derivation $\mathcal{I}$ of $\Gamma;\Xi;\Delta^I\backslash\,\Delta^O \Longrightarrow_v G$ for the two parts of the theorem. We develop three representative cases.

Case rm$_3$-$\top$:

$$\mathcal{I} = \frac{}{\Gamma;\Xi;\Delta^I\backslash\,\Delta^I \Longrightarrow_1 \top}\,\mathbf{rm_3\text{-}\top}$$

with $\Delta^O = \Delta^I$ and $G = \top$.

Then, we can take Ξ as Ξ' and apply rule **rm$_2$-$\top$** to obtain the desired derivation of $\Gamma;(\Xi,\Delta^I)\backslash\,(\Xi,\Delta^I) \Longrightarrow_1 \top$.

Case rm$_3$-&$_{01}$:

$$\mathcal{I} = \frac{\begin{array}{cc} \mathcal{I}_1 & \mathcal{I}_2 \\ \Gamma;\Xi;\Delta^I\backslash\,\Delta^O \Longrightarrow_0 G_1 & \Gamma;(\Xi,\Delta^I-\Delta^O);\cdot\backslash\,\cdot \Longrightarrow_1 G_2 \end{array}}{\Gamma;\Xi;\Delta^I\backslash\,\Delta^O \Longrightarrow_0 G_1 \,\&\, G_2}\,\mathbf{rm_3\text{-}\&_{01}}$$

with $G = G_1 \,\&\, G_2$.

By induction hypothesis on $\mathcal{I}_1$ and $\mathcal{I}_2$, there are derivations of $\Gamma;(\Xi,\Delta^I)\backslash\,\Delta^O \Longrightarrow_0 G_1$ and $\Gamma;(\Xi,\Delta^I-\Delta^O)\backslash\,\Xi' \Longrightarrow_1 G_2$ for $\Xi' \subseteq \Xi,\Delta^I-\Delta^O$. By the replacement lemma 4.2, we can transform the latter derivation into a proof of $\Gamma;(\Xi,\Delta^I)\backslash\,(\Xi',\Delta^O) \Longrightarrow_1 G_2$. It suffices then to apply rule **rm$_2$-&$_{01}$** to obtain the desired result.

Case rm$_3$-$\otimes_{10}$:

$$\mathcal{I} = \frac{\begin{array}{cc} \mathcal{I}_1 & \mathcal{I}_2 \\ \Gamma;\cdot;(\Xi,\Delta^I)\backslash\,\Delta' \Longrightarrow_1 G_1 & \Gamma;\cdot;\Delta'\backslash\,\Delta^O_1 \Longrightarrow_0 G_2 \end{array}}{\Gamma;\Xi;\Delta^I\backslash\,\Delta^I\cap\Delta^O_1 \Longrightarrow_1 G_1 \otimes G_2}\,\mathbf{rm_3\text{-}\otimes_{10}}$$

with $\Delta^O = \Delta^I \cap \Delta^O_1$ and $G = G_1 \otimes G_2$.

By induction hypothesis on $\mathcal{I}_1$ and $\mathcal{I}_2$, the sequents $\Gamma;(\Xi,\Delta^I)\backslash\,\Delta' \Longrightarrow_1 G_1$ and $\Gamma;\Delta'\backslash\,\Delta^O_1 \Longrightarrow_0 G_2$ are derivable (in the first case, since we start from an empty strict context, the only possibility for Ξ' is $\cdot$). We can therefore apply rule **rm$_2$-$\otimes_{10}$** and obtain a derivation of

$$\Gamma;(\Xi,\Delta^I)\backslash\,\Delta^O_1 \Longrightarrow_1 G_1 \otimes G_2.$$

By the subcontext lemma 4.1, $\Delta^O_1 \subseteq \Xi,\Delta^I$. Therefore, $\Delta^O_1 = \Delta^O_1\cap(\Xi,\Delta^I) = (\Delta^O_1\cap\Xi),(\Delta^O_1\cap\Delta^I)$. This serves our purpose since $\Xi' = \Delta^O_1\cap\Xi \subseteq \Xi$. ☑

We have been unable to obtain a direct inductive proof of the completeness of ***RM*$_3$** with respect to ***RM*$_2$**. Instead, we define an intermediate resource management system between ***RM*$_2$** and ***RM*$_3$**. The system ***RM*$'_2$** differs from ***RM*$_2$** by the replacement of rules **rm$_2$-&$_{00}$** and **rm$_2$-&$_{10}$** with the following two rules:

$$\frac{\Gamma;\Delta^I\backslash\,\Delta^O \Longrightarrow_0 G_1 \quad \Gamma;\Delta^I-\Delta^O\backslash\,\cdot \Longrightarrow_0 G_2}{\Gamma;\Delta^I\backslash\,\Delta^O \Longrightarrow_0 G_1 \,\&\, G_2}\,\mathbf{rm'_2\text{-}\&_{00}}$$

and

$$\frac{\Gamma; \Delta^I \backslash \Delta_1, \Delta^O \Longrightarrow_1 G_1 \quad \Gamma; \Delta^I - \Delta^O \backslash \cdot \Longrightarrow_0 G_2}{\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow_0 G_1 \,\&\, G_2}\ \mathbf{rm_2'\text{-}\&_{10}}$$

which borrow ideas from the rule $\mathbf{rm_1\text{-}\&'}$ we discussed above. Modifying rules $\mathbf{rm_2\text{-}\&_{01}}$ and $\mathbf{rm_2\text{-}\&_{11}}$ in a similar way is counterproductive. Every proof in $\boldsymbol{RM_2}$ can be transformed into an almost isomorphic derivation in $\boldsymbol{RM_2'}$, as formalized by the following lemma.

Lemma 5.4 (*Completeness of $\boldsymbol{RM_2'}$ with respect to $\boldsymbol{RM_2}$*)

For every derivation $\mathcal{I}$ of $\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow_v G$ in $\boldsymbol{RM_2}$ there is a derivation of the same judgment in $\boldsymbol{RM_2'}$.

Proof.

The proof proceeds by an easy induction on the structure of $\mathcal{I}$, with applications of the output context replacement lemma 4.2 in correspondence of rules $\mathbf{rm_2\text{-}\&_{00}}$ and $\mathbf{rm_2\text{-}\&_{10}}$. ☑

We are now in a position to prove the completeness of $\boldsymbol{RM_3}$ with respect to $\boldsymbol{RM_2}$, via $\boldsymbol{RM_2'}$.

Theorem 5.5 (*Completeness of $\boldsymbol{RM_3}$ with respect to $\boldsymbol{RM_2}$*)

i. If $\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow_0 G$, then $\Gamma; \Xi; (\Delta^I - \Xi) \backslash \Delta^O \Longrightarrow_0 G$ for every $\Xi \subseteq \Delta^I - \Delta^O$.

ii. If $\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow_1 G$, then $\Gamma; \Xi; (\Delta^I - \Xi) \backslash (\Delta^O - (\Xi \cap \Delta^O)) \Longrightarrow_1 G$ for every $\Xi \subseteq \Delta^I$.

Proof.

By Lemma 5.4, we can translate the premisses of this statement into $\boldsymbol{RM_2'}$ derivations. We proceed then by mutual induction on the structure of a derivation $\mathcal{I}$ for $\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow_v G$ relative to the rules of this formalism.

We sketch this proof, the most complex in the paper, by presenting the details of three of the hardest cases.

Case $\mathbf{rm_2'\text{-}\&_{01}}$:

$$\mathcal{I} = \frac{\overset{\mathcal{I}_1}{\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow_0 G_1} \quad \overset{\mathcal{I}_2}{\Gamma; \Delta^I \backslash \Delta^O, \Delta_2 \Longrightarrow_1 G_2}}{\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow_0 G_1 \,\&\, G_2}\ \mathbf{rm_2'\text{-}\&_{01}}$$

with $G = G_1 \,\&\, G_2$.

By induction hypothesis on $\mathcal{I}_1$ and $\mathcal{I}_2$, there are derivations $\mathcal{I}_1'$ and $\mathcal{I}_2'$ of $\Gamma; \Xi'; (\Delta^I - \Xi') \backslash \Delta^O \Longrightarrow_0 G_1$ for every $\Xi' \subseteq \Delta^I - \Delta^O$ and

$$\Gamma; \Xi''; (\Delta^I - \Xi'') \backslash (\Delta^O, \Delta_2) - (\Xi'' \cap (\Delta^O, \Delta_2)) \Longrightarrow_1 G_2$$

for every $\Xi'' \subseteq \Delta^I$.

We set Ξ' to be Ξ, and take $\Xi'' = (\Xi, (\Delta^I - \Xi)) - \Delta^O = \Delta^I - \Delta^O$. We can then reduce the expressions appearing in the last judgment:

$$\Delta^I - \Xi'' \;=\; \Delta^I - (\Delta^I - \Delta^O) \;=\; \Delta^O,$$

and

$$
\begin{array}{rcl}
(\Delta^O, \Delta_2) - (\Xi'' \cap (\Delta^O, \Delta_2)) & = & (\Delta^O, \Delta_2) - ((\Delta^I - \Delta^O) \cap (\Delta^O, \Delta_2)) \\
& = & (\Delta^O, \Delta_2) - ((\Delta^I \cap (\Delta^O, \Delta_2)) - (\Delta^O \cap (\Delta^O, \Delta_2))).
\end{array}
$$

Now, since by the subcontext lemma 4.1 $\Delta^O, \Delta_2 \subseteq \Delta^I$, we have that $\Delta^I \cap (\Delta^O, \Delta_2) = \Delta^O, \Delta_2$. Moreover, $\Delta^O \cap (\Delta^O, \Delta_2) = \Delta^O$. Therefore the above expression reduces to $(\Delta^O, \Delta_2) - ((\Delta^O, \Delta_2) - \Delta^O)$, i.e., Δ^O.

The endsequent of $\mathcal{I}_2'$ can therefore be rewritten as $\Gamma; (\Xi, (\Delta^I - \Xi)) - \Delta^O; \Delta^O \backslash \Delta^O \Longrightarrow_1 G_2$. We can now apply the replacement lemma 5.2 and obtain a derivation $\mathcal{I}_2''$ of the judgment $\Gamma; (\Xi, (\Delta^I - \Xi)) - \Delta^O; \cdot \backslash \cdot \Longrightarrow_1 G_2$. We can then use rule $\mathbf{rm_3\text{-}\&_{01}}$ to $\mathcal{I}_1'$ and $\mathcal{I}_2''$ in order to obtain the desired result.

Case $\mathbf{rm_2'\text{-}\otimes_{01}}$:

$$
\mathcal{I} = \cfrac{\begin{array}{cc} \mathcal{I}_1 & \mathcal{I}_2 \\ \Gamma; \Delta^I \backslash \Delta' \Longrightarrow_0 G_1 & \Gamma; \Delta' \backslash \Delta^O \Longrightarrow_1 G_2 \end{array}}{\Gamma; \Delta^I \backslash \Delta^O \Longrightarrow_1 G_1 \otimes G_2}\ \mathbf{rm_2'\text{-}\otimes_{01}}
$$

with $G = G_1 \otimes G_2$.

By induction hypothesis, the sequents $\Gamma; \Xi'; (\Delta^I - \Xi') \backslash \Delta' \Longrightarrow_0 G_1$ for every $\Xi' \subseteq \Delta^I - \Delta'$, and $\Gamma; \Xi''; (\Delta' - \Xi'') \backslash (\Delta^O - (\Delta^O \cap \Xi'')) \Longrightarrow_1 G_2$ for every $\Xi'' \subseteq \Delta'$ are derivable. In particular, for $\Xi' = \cdot$, we obtain $\Gamma; \cdot; \Delta^I \backslash \Delta' \Longrightarrow_0 G_1$.

If we set $\Xi'' = \Delta' \cap \Xi$ for an arbitrary context $\Xi \subseteq \Delta^I$ (note that this acceptable since $\Xi'' \subseteq \Delta'$), the second judgment above rewrites to

$$
\Gamma; (\Delta' \cap \Xi); (\Delta' - (\Delta' \cap \Xi)) \backslash (\Delta^O - (\Delta^O \cap \Delta' \cap \Xi)) \Longrightarrow_1 G_2.
$$

Let us rewrite the various expressions appearing in it. $\Delta' - (\Delta' \cap \Xi) = (\Delta' \cap \Delta^I) - (\Delta' \cap \Xi) = \Delta' \cap (\Delta^I - \Xi)$, since $\Delta' \subseteq \Delta^I$ by the subcontext lemma 4.1. On the other hand, since by the same lemma $\Delta^O \subseteq \Delta'$, we have that $\Delta^O - (\Delta^O \cap \Delta' \cap \Xi) = \Delta^O - (\Delta^O \cap \Xi)$. After carrying out these simplifications, our judgment rewrites to

$$
\Gamma; (\Delta' \cap \Xi); (\Delta' \cap (\Delta^I - \Xi)) \backslash (\Delta^O - (\Delta^O \cap \Xi)) \Longrightarrow_1 G_2.
$$

We are now in the position of applying rule $\mathbf{rm_3\text{-}\otimes_{10}}$, obtaining the expected $\Gamma; \Xi; (\Delta^I - \Xi) \backslash (\Delta^O - (\Delta^O \cap \Xi)) \Longrightarrow_1 G_2$.

In this part of the proof, we relied on the equality $(\Delta_1 \cap \Delta_2) - (\Delta_1 \cap \Delta_3) = \Delta_1 \cap (\Delta_2 - \Delta_3)$, which holds only if every element in these contexts has exactly one occurrence. Otherwise, we have the following counterexample, where $\Delta_1 = (a, a)$, $\Delta_2 = (a, a, a)$ and $\Delta_3 = (a)$.

$$
\begin{array}{lclcll}
((a,a) \cap (a,a,a)) - ((a,a) \cap (a)) & = & (a,a) - (a) & = & (a) & \neq \\
(a,a) \cap ((a,a,a) - (a)) & = & (a,a) \cap (a,a) & = & (a,a) &
\end{array}
$$

The unique labelling of context assumptions ensures that no such situation can arise, and therefore entitles us to make use of the above equivalence.

Case $\mathbf{rm_2'\text{-}\multimap_1}$:

$$
\mathcal{I} = \cfrac{\begin{array}{c} \mathcal{I}_1 \\ \Gamma; \Delta^I, D \backslash \Delta_1^O \Longrightarrow_1 G_1 \end{array}}{\Gamma; \Delta^I \backslash (\Delta^I \cap \Delta_1^O) \Longrightarrow_1 D \multimap G_1}\ \mathbf{rm_2'\text{-}\multimap_1}
$$

with $\Delta^O = \Delta^I \cap \Delta_1^O$ and $G = D \multimap G_1$.

By induction hypothesis, there is a derivation of the judgment

$$\Gamma; \Xi'; ((\Delta^I, D) - \Xi') \backslash (\Delta_1^O - (\Delta_1^O \cap \Xi')) \Longrightarrow_1 G_1$$

for every $\Xi' \subseteq \Delta^I, D$. Let us take an arbitrary context $\Xi \subseteq \Delta^I$ and consider $\Xi' = \Xi, D$. Then, we have the following instance of the above judgment:

$$\Gamma; \Xi, D; ((\Delta^I, D) - (\Xi, D)) \backslash (\Delta_1^O - (\Delta_1^O \cap (\Xi, D))) \Longrightarrow_1 G_1.$$

Observe that $(\Delta^I, D) - (\Xi, D) = \Delta^I - \Xi$.

By applying rule $\mathbf{rm_3}$- $\multimap$, we obtain a derivation of the judgment

$$\Gamma; \Xi; (\Delta^I - \Xi) \backslash (\Delta_1^O - (\Delta_1^O \cap (\Xi, D))) \Longrightarrow_1 D \multimap G_1.$$

In order to show that the output context in this judgment coincides with the expected $(\Delta^I \cap \Delta_1^O) - (\Delta^I \cap \Delta_1^O \cap \Xi)$, we must distinguish two cases:

$\boldsymbol{D \in \Delta_1^O}$**:** Let $\Delta_1^O = \Delta_2^O, D$. Then, $(\Delta^I \cap \Delta_1^O) - (\Delta^I \cap \Delta_1^O \cap \Xi) = \Delta_2^O - (\Delta_2^O \cap \Xi) = (\Delta_2^O, D) - ((\Delta_2^O \cap \Xi), D) = \Delta_1^O - (\Delta_1^O \cap (\Xi, D))$.

$\boldsymbol{D \notin \Delta_1^O}$**:** Then, $(\Delta^I \cap \Delta_1^O) - (\Delta^I \cap \Delta_1^O \cap \Xi) = \Delta_1^O - (\Delta_1^O \cap \Xi) = \Delta_1^O - ((\Delta_1^O \cap \Xi), (\Delta_1^O \cap D)) = \Delta_1^O - (\Delta_1^O \cap (\Xi, D))$.

In both cases, we have taken implicit advantage of several simple facts about sets and relied upon our unique labelling assumption. ☑

6 Implementation Issues

The system $\boldsymbol{RM_3}$ provides a satisfactory solution to all the resource management problems we discussed in the previous sections. Unfortunately, it does so at a rather high price since most of its rules involve complex operations on the context (exhaustive tests on the status of one of the contexts, shuffling formulas from the strict to the non-strict context or vice versa, etc.).

This situation is complicated by the fact that, in an actual implementation, the order in which clauses occur in the program and the order in which new assumptions are added to it during execution must be preserved so that the programmer can predict in which sequence clauses are tried when solving atomic goals. Thus we store the intuitionistic, strict and non-strict assumptions in a common data structure, differentiating the role of each formula by means of a tag. Further, when a formula is consumed, it is generally more efficient to mark it as such (rather than actually delete it) in order to facilitate backtracking. In this type of implementation, each time we perform a test to check, for instance, if the strict context is empty, we have to visit all the formulas present in all contexts. Similar costs are incurred when we perform operations like taking the intersection of two contexts.

We have achieved a substantial improvement in performance by maintaining additional information about the program, in particular the number of formulas present in the strict and non-strict contexts. The implementation of the operations that manipulate the context are in charge of maintaining the correct value of these counters. In particular, each time an output context is produced, we must make available the number of formulas it contains. Then, checking the emptiness of the strict context, for example, reduces to an inexpensive arithmetic comparison. This approach also benefits the implementation of the context operations themselves by limiting the portion of the context they need to examine: for example, if the context contains s strict assumptions, a routine implementing rule $\mathbf{rm_3}$-$\top$ can return as soon as it has encountered s strict resources. This can produce significant speed-ups since a typical *Lolli* context consists of a large body of intuitionistic program clauses loaded from a file followed by assumptions made at run-time (which should therefore be accessed first).

The rules for handling the tensor still perform a relatively expensive operation, since they must move the contents of the strict context into the non-strict context unless the former is initially empty. We can eliminate this overhead for nested occurrences of $\otimes$ by requiring this connective to be parsed as a left associative operator. In this way, the leftmost occurrence of $\otimes$ will undergo the shuffling process. But, since all inner occurrences appear in the left conjunct (G_1 in rules $\mathbf{rm_3}$-$\otimes_{\mathbf{vw}}$), they will be proved with an empty strict context, avoiding any additional shuffling.

The techniques presented in this section have been applied into an enhanced version of the *ML* interpreter for *Lolli*. A series of implementations realizing each of the ideas discussed in this paper is available by anonymous ftp (see [CHMP] for details). The declarative nature of the rules makes these same ideas applicable to implementations based on other programming paradigms. In particular, the original *Prolog* prototype (also available in [CHMP]) for *Lolli* [HM94] can be easily adapted to take advantage of these observations.

7 Conclusions and Related Work

The issue of efficient context management has proved to be crucial for the use of linear logic programming languages in non-trivial applications. In this paper, we have presented a general technique that not only eliminates sources of non-determinism deriving from naive context management, but also permits early recognition of certain failure situations. We have implemented these ideas in the interpreter for a new release of the language *Lolli* [CHMP]. Tests showed a general improvement in performance and, in some examples, arbitrary speed-ups. We also achieved convergence for some previously non-terminating programs. The determinism also simplifies the programmer's task: Despite the apparent complexity of $\boldsymbol{RM_3}$ it is relatively straightforward to predict the operational behavior of programs and avoid inefficient generate-and-test situations.

To our knowledge, the only other authors who have been concerned with the issue of efficiency in context management for linear logic programming languages are the designers of *Lygon*. In their first publication on this subject [HW94], they build on the work of Hodas and Miller and independently develop a system with the characteristics of Hodas' efficient handling of $\top$. They do not, however, present a notion equivalent to our strict context, and make no mention of techniques akin to our linear formula counters to reduce the overhead at the implementation level. They recently proposed a new and promising approach to context management based on the idea of maintaining boolean constraints in order to specify how linear resources should be distributed [HP97]. We are currently investigating a related method as a way to achieve a practical implementation of linear higher-order unification [CP97].

Our analysis was motivated primarily by the goal of building an efficient interpreter [CHMP], but should also be applicable to the design of compilers which will ultimately be necessary for the execution of large programs. We expect that compilation techniques developed for *Prolog* [LO87] and λ*Prolog* [Kwo94, NJK95] may be combined with our methods.

The results described in the paper can be applied to other programming languages based on linear logic. Hodas and Polakow have extended the system $\boldsymbol{RM_3}$ to Miller's specification logic *Forum* [Mil96] and have based a prototype implementation on it [HP96]. López and Pimentel have designed another implementation of *Forum* on a system similar to $\boldsymbol{RM_3}$ but currently without a slack indicator [LP97]. These techniques should extend just as easily to implementations of

Lygon [HP91, HW94] and other linear languages.

Acknowledgments

We would like to thank Vladimir Alexiev, James Harland, Dale Miller, Jeffrey Polakow and Roberto Virga, as well as the anonymous reviewers of the 1996 International Workshop on Extensions of Logic Programming (ELP'96), for their valuable comments on early versions of this paper.

References

[AP91] Jean-Marc Andreoli and Remo Pareschi. Linear objects: Logical processes with built-in inheritance. *New Generation Computing*, 9:445–473, 1991.

[CHMP] Iliano Cervesato, Joshua S. Hodas, Dale Miller, and Frank Pfenning. Some implementations of the linear logic programming language Lolli. Available on the World Wide Web at address `ftp://ftp.cs.cmu.edu/user/iliano/misc/lolli.tar.gz`.

[CP96] Iliano Cervesato and Frank Pfenning. A linear logical framework. In E. Clarke, editor, *Proceedings of the Eleventh Annual Symposium on Logic in Computer Science — LICS'96*, pages 264–275, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press. This work also appeared as Preprint 1834 of the Department of Mathematics of Technical University of Darmstadt, Germany.

[CP97] Iliano Cervesato and Frank Pfenning. Linear higher-order pre-unification. In G. Winskel, editor, *Twelfth Annual Symposium on Logic in Computer Science — LICS'97*, pages 422–433, Warsaw, Poland, 29 June – 2 July 1997. IEEE Computer Society Press.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[GP94] Didier Galmiche and Guy Perrier. Foundations of proof search strategies design in linear logic. In *Symposium on Logical Foundations of Computer Science*, pages 101–113, St. Petersburg, Russia, 1994. Springer-Verlag LNCS 813. Also available as Technical Report CRIN 94-R-112 from the Centre di Recherche en Informatique de Nancy.

[HM94] Joshua S. Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994. Extended abstract in the Proceedings of the Sixth Annual Symposium on Logic in Computer Science, Amsterdam, July 15–18, 1991.

[Hod94] Joshua S. Hodas. *Logic Programming in Intuitionistic Linear Logic: Theory, Design and Implementation*. PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 1994.

[HP91] James Harland and David Pym. The uniform proof-theoretic foundation of linear logic programming. In V. Saraswat and K. Ueda, editors, *Proceedings of the International Logic Programming Symposium*, pages 304–318, San Diego, California, October 1991.

[HP96] Joshua S. Hodas and Jeffrey Polakow. Forum as a logic programming language: Preliminary results and observations. In M. Okada, editor, *Proceedings of the Linear Logic '96 Meeting*, Tokyo, Japan, 1996. Elsevier Electronic Notes in Theoretical Computer Science, volume 3.

[HP97] James Harland and David Pym. Resource distribution via boolean constraints. In W. McCune, editor, *Proceedings of the Fourteenth International Conference on Automated Deduction — CADE-14*, Townsville, Australia, July 1997. To appear.

[HW94] James Harland and Michael Winikoff. Deterministic resource management for the linear logic programming language Lygon. Technical Report TR 94/23, Melbourne University, Department of Computer Science, 1994.

[Kwo94] Keehang Kwon. *Towards a Verified Abstract Machine for a Logic Programming Language with a Notion of Scope.* PhD thesis, Department of Computer Science, Duke University, December 1994. Available as Technical Report CS-1994-36.

[KY93] Naoki Kobayashi and Akinori Yonezawa. ACL — A concurrent linear logic programming paradigm. In D. Miller, editor, *Proceedings of the 1993 International Logic Programming Symposium*, pages 279–294, Vancouver, Canada, October 1993. MIT Press.

[LO87] Timothy G. Lindholm and Richard A. O'Keefe. Efficient implementation of a defensible semantics for dynamic Prolog code. In J.L. Lassez, editor, *Proceedings of the Fourth International Conference on Logic Programming — ICLP'87*, pages 21–39, Melbourne, Australia, 1987. MIT Press.

[LP97] Pablo López and Ernesto Pimentel. A lazy splitting system for Forum. In M. Falaschi, M. Navarro, and A. Policriti, editors, *Proceedings of the Joint Conference on Declarative Programming — APPIA-GULP-PRODE'97*, pages 247–258, Grado, Italy, 1997.

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[Mil96] Dale Miller. A multiple-conclusion specification logic. *Theoretical Computer Science*, 165(1):201–232, 1996.

[Min93] Grigori Mints. Resolution calculus for the first order linear logic. *Journal of Logic, Language and Information*, 2(1):59–83, 1993.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[NJK95] Gopalan Nadathur, Bharat Jayaraman, and Keehang Kwon. Scoping constructs in logic programming: Implementation problems and their solution. *Journal of Logic Programming*, 25(2):119–161, 1995.

[Pfe92] Frank Pfenning. Computation and deduction. Unpublished lecture notes, 277 pp. Revised May 1994, April 1996, May 1992.

[Pfe94] Frank Pfenning. Elf: A meta-language for deductive systems. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, pages 811–815, Nancy, France, June 1994. Springer-Verlag LNAI 814. System abstract.

[Tam94] T. Tammet. Proof strategies in linear logic. *Journal of Automated Reasoning*, 12:273–304, 1994. Also available as Programming Methodology Group Report 70, Chalmers University, 1993.