Logic Programming with Multiple Context Management Schemes

Joshua S. Hodas

Computer Science Department*
Harvey Mudd College
Claremont, CA 91711
hodas@cs.hmc.edu

Abstract. Two years experience with programming in Linear Logic has shown that while some problems require the full power of linear context management, for many this much control is too much. In such cases a restriction on either weakening or contraction, but not both, is most appropriate. In this article we introduce a refinement of the system proposed by Hodas and Miller in which each of these constraints is independently available. This enables programs to be more succinct, understandable, and efficient.

1 Introduction

Sequential logic programming based on linear logic was first proposed by Hodas and Miller in 1991 [8]. The motivating idea was that the context (database) management provided by traditional languages based on intuitionistic logic—such as Prolog, λProlog [12], N-Prolog [2], and others—was insufficient for many applications. Therefore, a new language was introduced which extended λProlog by using two separate contexts. Clauses in the ordinary, intuitionistic, context continue to be usable as many or as few times as desired; that is, the structural rules of weakening and contraction are available in that context. In contrast, those rules are not available in the bounded, linear, context; the usability of clauses in that context is far more restricted.

In the two years since the system was first proposed, however, it has become apparent that for many purposes the system is too restrictive. In some cases the programmer wants to bar weakening but allow contraction, in others the opposite effect is desired. While both these situations can be simulated in the existing system, the programs that result are not as clear as one would hope, and their execution profiles may be less than ideal. In this paper, therefore, we introduce a further refinement of the Hodas-Miller system in which there are now four separate contexts:

* This material was mostly developed while the author was a student in the Department of Computer Science at the University of Pennsylvania. The work was funded under ONR N00014-88-K-0633, NSF CCR-87-05596, NSF CCR-91-02753, and DARPA N00014-85-K-0018 through the University of Pennsylvania

Linear: Neither contraction nor weakening are available.
Affine: Only weakening is available; clauses may be discarded but not duplicated.
Relevant: Only contraction is available; clauses may be duplicated but not discarded.
Intuitionistic: Both contraction and weakening are available.

2 Logic Programming in Linear Logic

Linear logic was developed in the mid 1980's as a result of Girard's work in the semantics of logic [3]. In this system, the structural rules of contraction and weakening can only be applied to formulas that are marked with the '!' modal. To understand the motivation and effect of this restriction, consider the intuitionistic (and classical) tautology:

$$[(D \supset K) \wedge (D \supset M)] \supset [D \supset (K \wedge M)]$$

which has the following proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{D \longrightarrow D} \quad \dfrac{}{K \longrightarrow K}}{D \supset K, D \longrightarrow K} \supset_L \quad \dfrac{\dfrac{}{D \longrightarrow D} \quad \dfrac{}{M \longrightarrow M}}{D \supset M, D \longrightarrow M} \supset_L}{D \supset K, D \supset M, D, D \longrightarrow K \wedge M} \wedge_R}{D \supset K, D \supset M, D \longrightarrow K \wedge M}\ contract}{D \supset K, D \supset M \longrightarrow D \supset (K \wedge M)} \supset_R}{(D \supset K) \wedge (D \supset M) \longrightarrow D \supset (K \wedge M)} \wedge_L}{\longrightarrow ((D \supset K) \wedge (D \supset M)) \supset (D \supset (K \wedge M))} \supset_R$$

There is nothing disturbing about this proof, until one considers the model:

D := I have a dollar
K := I can buy a pack of Kools
M := I can buy a pack of Marlboros

In which case we have proven that if a dollar is enough to buy a pack of cigarettes, then it is enough to buy two packs. The unlimited availability of the contraction rule amounts to a license to print money. In linear logic this formula is not a tautology. In order for it to be provable it would require the provision of two D formulas in the rightmost implication, as in:

$$[(D \supset K) \wedge (D \supset M)] \supset [(D \wedge D) \supset (K \wedge M)]$$

In order to simplify the presentation of their system, which they called $\mathcal{L}$, Hodas and Miller used a non-standard presentation of the fragment of linear logic on which it is based. The rules of $\mathcal{L}$ are given in Figs. 1 and 2. Rather than using the '!' modal to control the use of contraction and weakening, the sequents in this system have two separate contexts, each consisting of a multiset

of formulas.[2] The structural rules are available in the left hand context but not in the right hand one. Thus the $\mathcal{L}$-sequent:

$$B_1, \ldots, B_n; C_1, \ldots, C_m \longrightarrow D$$

is intended to behave like the linear logic sequent:

$$!B_1, \ldots, !B_n, C_1, \ldots, C_m \longrightarrow D$$

In addition, the structural rules themselves are not explicit, but rather are woven implicitly into the way in which the other rules treat the two contexts. So, for instance, the axioms of the system require that the linear context contain only the formula being matched, while the intuitionistic context's contents are arbitrary. In this way weakening is barred in the linear context, but allowed (and moved to the leaves) in the intuitionistic context.

In linear logic the left hand introduction rules, other than $!_L$ apply only to formulas not marked with '!'. In $\mathcal{L}$ this behavior is mimicked by defining the left hand introduction rules to apply only in the linear context. The *absorb* rule is used to copy a formula from the intuitionistic context to the linear one to make it available to the other left hand rules. The original formula remains in the intuitionistic context, thereby providing some of the behavior of contraction for that context.

Clauses are added to each context by using the corresponding implication operator in goal position. Searching bottom up for a proof of an implication goal leads to an attempt to prove the conclusion of the implication in a setting where the assumption has been added to the appropriate context.

The system $\mathcal{L}$ has several desirable properties. First, the cut-elimination property holds, though the proof is a bit more complex than for intuitionistic logic. Second, uniform proofs, those in which sequents with non-atomic right hand sides are always the conclusion of the right hand rule for the principal logical operator of the right hand side, are complete. Taken together, these facts imply that there is a simple, effective, bottom-up search strategy for finding proofs in the system. This strategy corresponds roughly to SLD-Resolution, and qualifies the system to be called a logic programming language.[3] Another important property is that the proof system of λProlog properly embeds into this one. The proof of a formula that does not include any instances of $\multimap$ can be directly mapped into a corresponding proof in the theory of hereditary Harrop formulas. Thus the $\multimap$ operator extends the behavior of that system. These properties were discussed by Hodas and Miller [8, 9] and proved in full detail in Hodas' dissertation [7].

[2] In the original presentation of $\mathcal{L}$ the intuitionistic context was described as a set rather than a multiset. This assumption eased the proof of certain properties of the system's model theory, but is unnecessary here.

[3] The definition of uniform proofs and the notion that the completeness of such proofs qualifies a logic to be called a logic programming language is due to Miller, et al. [10, 11].

$$\frac{}{\Gamma; B \longrightarrow B}\ \textit{identity} \qquad \frac{}{\Gamma; \Delta \longrightarrow \top}\ \top_R$$

$$\frac{\Gamma, B; \Delta, B \longrightarrow C}{\Gamma, B; \Delta \longrightarrow C}\ \textit{absorb}$$

$$\frac{\Gamma; \Delta, B_i \longrightarrow C}{\Gamma; \Delta, B_1 \,\&\, B_2 \longrightarrow C}\ \&_{L_i} \qquad \frac{\Gamma; \Delta \longrightarrow B \quad \Gamma; \Delta \longrightarrow C}{\Gamma; \Delta \longrightarrow B \,\&\, C}\ \&_R$$

$$\frac{\Gamma; \Delta_1 \longrightarrow B \quad \Gamma; \Delta_2, C \longrightarrow E}{\Gamma; \Delta_1, \Delta_2, B \multimap C \longrightarrow E}\ \multimap_L \qquad \frac{\Gamma; \Delta, B \longrightarrow C}{\Gamma; \Delta \longrightarrow B \multimap C}\ \multimap_R$$

$$\frac{\Gamma; \emptyset \longrightarrow B \quad \Gamma; \Delta, C \longrightarrow E}{\Gamma; \Delta, B \Rightarrow C \longrightarrow E}\ \Rightarrow_L \qquad \frac{\Gamma, B; \Delta \longrightarrow C}{\Gamma; \Delta \longrightarrow B \Rightarrow C}\ \Rightarrow_R$$

$$\frac{\Gamma; \Delta, B[x \mapsto t] \longrightarrow C}{\Gamma; \Delta, \forall x.B \longrightarrow C}\ \forall_L \qquad \frac{\Gamma; \Delta \longrightarrow B[x \mapsto c]}{\Gamma; \Delta \longrightarrow \forall x.B}\ \forall_R$$

provided that c is not free in the lower sequent.

$$\frac{\Gamma_1; \Delta_1 \longrightarrow B \quad \Gamma_2; \Delta_2, B \longrightarrow C}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \longrightarrow C}\ \textit{cut} \qquad \frac{\Gamma_1; \emptyset \longrightarrow B \quad \Gamma_2, B; \Delta \longrightarrow C}{\Gamma_1, \Gamma_2; \Delta \longrightarrow C}\ \textit{cut}_I$$

Fig. 1. $\mathcal{L}$: A proof system for the connectives $\top$, &, $\multimap$, $\Rightarrow$, and $\forall$.

$$\frac{}{\Gamma; \emptyset \longrightarrow \mathbf{1}}\ \mathbf{1}_R$$

$$\frac{\Gamma; \emptyset \longrightarrow C}{\Gamma; \emptyset \longrightarrow\, !C}\ !R \qquad \frac{\Gamma; \Delta_1 \longrightarrow B_1 \quad \Gamma; \Delta_2 \longrightarrow B_2}{\Gamma; \Delta_1, \Delta_2 \longrightarrow B_1 \otimes B_2}\ \otimes R$$

$$\frac{\Gamma; \Delta \longrightarrow B[t/x]}{\Gamma; \Delta \longrightarrow \exists x.B}\ \exists_R \qquad \frac{\Gamma; \Delta \longrightarrow B_i}{\Gamma; \Delta \longrightarrow B_1 \oplus B_2}\ \oplus_{R_i}$$

Fig. 2. Additional operator rules for $\mathcal{L}$

A programming language, Lolli, which directly implements the logic of this system has been developed and distributed. The behavior of the various operators, and their relation to the operators of λProlog has heavily influenced the syntax of Lolli [5, 9]:

- Clauses, unless marked with the keyword **LINEAR**, are assumed to be loaded into the intuitionistic context.
- The implication operators $\multimap$ and $\Rightarrow$ are represented by '-o' and '=>', respectively, in goals, and by ':-' and '<=' in clauses.
- The two conjunctions, & and $\otimes$ are represented by '&' and ',', respectively.

- The atomic truth formulas $\top$, and $\mathbf{1}$ are written '`erase`', and '`true`', respectively.
- The standard quantifier assumptions are made. Explicit quantifiers are written as in '`forall x\ (foo x)`'.

3 Motivations for a New System

The system described by Hodas and Miller succeeded in many ways at meeting the goals of its designers. It is an attractive system for implementing a variety of programs in which the management of clausal resources during execution is of interest [6, 7, 8, 9].

However, in many cases the system has proven to be less than ideal. Consider one of the simplest motivating examples for the system: the simulation of a toggle switch. The state of a named switch is represented by a clause for the binary relation *state*(*name*, *value*) which is stored in the linear context. A program to manage the switch might be given by:

```
init Name State G :- state Name State -o G.
set Name NewState G :- state Name State,
                       state Name NewState -o G.

toggle Name G :- state Name off, state Name on -o G.
toggle Name G :- state Name on, state Name off -o G.

setting Name State :- state Name State.
```

Unfortunately, when there is more than one switch, the system behaves in unexpected ways. Consider the following interaction:[4]

```
?- init s1 on top.
?- toggle s1 top.
?- setting s1 S.
S <- off
yes
?- init s2 on top.
?- setting s1 S.
no
```

The problem is that once the second switch is initialized there are two formulas in the linear context. Any future goal must use both of these formulas if its proof is to succeed. Thus, in order to check the state of one switch, the state of the other must also be accessed:

[4] The goal `top` used in this interaction is a built in predicate used to re-invoke the read-prove-print loop at that point in a proof.

```
?- setting s1 S1, setting s2 S2.
S1 <- off
S2 <- on
yes
```

If we are not actually interested in the setting of the second switch at the moment, we can direct the interpreter to ignore it (and any other formulas in the linear context) by including a $\mathbb{1}$ in the query, in the form of the **erase** command, as in:

```
?- setting s1 S, erase.
S <- off
yes
```

This behavior could be added to the definition of the **setting** predicate, but that is somewhat confusing and leads to other complications. The problem is that the formula used to store the state of the switch cannot be weakened or contracted, while the programmer really wants to restrict only contraction. This is the form of control provided in affine logic. It seems that such a constraint should be indicated at the point where the clauses for the predicate are assumed, rather than at the point where the predicate is called.

If we were to augment the proof system for $\mathcal{L}$ with a form of weakening for just the special formula $\mathbb{1}$, as in:

$$\frac{\Gamma; \Delta \longrightarrow C}{\Gamma; \Delta, \mathbb{1} \longrightarrow C}\ \mathbb{1}_L$$

then affine reasoning for a formula A could be simulated by replacing instances of A with instances of $(A \,\&\, \mathbb{1})$. The interpreter could weaken such a formula by first using the $\&_{L_2}$ rule to select the $\mathbb{1}$ portion of the formula, which could then be discarded by using the new rule. The switch program would then be replaced by:

```
init Name State G :- (state Name State & true) -o G.
set Name NewState G :- state Name State,
                       (state Name NewState & true) -o G.

toggle Name G :- state Name off, (state Name on & true) -o G.
toggle Name G :- state Name on, (state Name off & true) -o G.

setting Name State :- state Name State.
```

which would behave as desired.

Unfortunately, this rule cannot be added directly to $\mathcal{L}$ without compromising the completeness of uniform proofs and correspondingly complicating the proof procedure. Even then, the resulting programs would be somewhat less readable than one would hope.

A similar problem occurs when the programmer wants to use relevant reasoning, as in artificial intelligence applications. In such a setting the answer to

"Does A imply B?" should be "yes" only if A was actually used to demonstrate B, not if B is true regardless. On the other hand, it is not generally of interest whether the assertion A was referenced more than once in the proof of B. Relevant behavior can be simulated in $\mathcal{L}$ goals by adding the assumption to both contexts simultaneously. Adding it to the linear context guarantees that it must be used at least once; adding it to the intuitionistic context allows it to be used as many additional times as needed. Thus the relevant goal $A \stackrel{R}{\rightarrow} B$ can be replaced by the $\mathcal{L}$ goal $A \Rightarrow (A \multimap B)$. Unfortunately the execution profile of programs encoded in this way is somewhat less than ideal, since the interpreter spends a good deal of time enforcing the linear constraint needlessly.

4 An Omnibus Logic

A logic programming language with direct support for relevant reasoning was first proposed by Bollen in his work on Conditional Logic Programming (CLOG-PROG) [1]. While that system shares much of its philosophical and formal foundations with $\mathcal{L}$ it is somewhat weaker in that arbitrary nesting of quantifiers and implications is not allowed.[5] In addition it says nothing about the affine and linear constraints which have been shown to have many useful applications.

In this section we introduce a new system, $\mathcal{O}$, whose rules are given in Figs. 3 and 4. The system is similar to $\mathcal{L}$, but the left hand sides of its sequents are composed of four separate multiset contexts. Left to right these are the intuitionistic, relevant, affine, and linear contexts.

As with $\mathcal{L}$, the structure of the axioms and the *absorb* rules determines much of the behavior of the system. In the identity axioms, both the intuitionistic and affine contexts may have arbitrary contents, while the relevant context must be empty and the linear context must contain only the formula being matched. This enables implicit weakening in the intuitionistic and affine contexts but not the other two.

The abs_I rule, which corresponds to the *absorb* rule in $\mathcal{L}$, makes a copy of a formula in the intuitionistic context and makes it available for use in the linear context. In contrast, the abs_A rule removes the formula being absorbed from the affine context when it is added to the linear context, so that the formula cannot be reused. Finally, the abs_R rule removes its formula from the relevant context but places copies in both the intuitionistic and linear contexts. Thus, once the formula has been used once, it can then be used zero or more additional times.

As with the system $\mathcal{L}$, there is an implication operator corresponding to each of the contexts used to load clauses into that context.

The rest of this section takes the form of a series of propositions about the formal properties of $\mathcal{O}$, in particular its relationship to $\mathcal{L}$ and other systems. These propositions are proved in full in the author's dissertation [7]. As with the papers which introduced $\mathcal{L}$, the bulk of the propositions here are stated in

[5] Miller and Hodas have demonstrated in several papers the advantages of allowing such arbitrary nesting. These issues are summarized in Hodas' dissertation [7].

$$\frac{}{\Gamma;\emptyset;\Psi;B \longrightarrow B}\ identity \qquad \frac{}{\Gamma;\Upsilon;\Psi;\Delta \longrightarrow \top}\ \top R$$

$$\frac{\Gamma,B;\Upsilon;\Psi;\Delta,B \longrightarrow C}{\Gamma,B;\Upsilon;\Psi;\Delta \longrightarrow C}\ abs_I \qquad \frac{\Gamma,B;\Upsilon;\Psi;\Delta,B \longrightarrow C}{\Gamma;\Upsilon,B;\Psi;\Delta \longrightarrow C}\ abs_R \qquad \frac{\Gamma;\Upsilon;\Psi;\Delta,B \longrightarrow C}{\Gamma;\Upsilon;\Psi,B;\Delta \longrightarrow C}\ abs_A$$

$$\frac{\Gamma;\Upsilon;\Psi;\Delta,B_i \longrightarrow C}{\Gamma;\Upsilon;\Psi;\Delta,B_1 \,\&\, B_2 \longrightarrow C}\ \&L_i \qquad \frac{\Gamma;\Upsilon;\Psi;\Delta \longrightarrow B \quad \Gamma;\Upsilon;\Psi;\Delta \longrightarrow C}{\Gamma;\Upsilon;\Psi;\Delta \longrightarrow B \,\&\, C}\ \&R$$

$$\frac{\Gamma;\Upsilon;\Psi_1;\Delta_1 \longrightarrow B \quad \Gamma;\Upsilon;\Psi_2;\Delta_2,C \longrightarrow E}{\Gamma;\Upsilon;\Psi_1,\Psi_2;\Delta_1,\Delta_2,B \multimap C \longrightarrow E}\ \multimap L \qquad \frac{\Gamma;\Upsilon;\Psi;\Delta,B \longrightarrow C}{\Gamma;\Upsilon;\Psi;\Delta \longrightarrow B \multimap C}\ \multimap R$$

$$\frac{\Gamma;\emptyset;\Psi_1;\emptyset \longrightarrow B \quad \Gamma;\Upsilon;\Psi_2;\Delta,C \longrightarrow E}{\Gamma;\Upsilon;\Psi_1,\Psi_2;\Delta,B \overset{A}{\multimap} C \longrightarrow E}\ \overset{A}{\multimap} L \qquad \frac{\Gamma;\Upsilon;\Psi,B;\Delta \longrightarrow C}{\Gamma;\Upsilon;\Psi;\Delta \longrightarrow B \overset{A}{\multimap} C}\ \overset{A}{\multimap} R$$

$$\frac{\Gamma;\Upsilon;\emptyset;\emptyset \longrightarrow B \quad \Gamma;\Upsilon;\Psi;\Delta,C \longrightarrow E}{\Gamma;\Upsilon;\Psi;\Delta,B \overset{R}{\multimap} C \longrightarrow E}\ \overset{R}{\multimap} L \qquad \frac{\Gamma;\Upsilon;\Psi;\Delta,B \longrightarrow C}{\Gamma;\Upsilon;\Psi;\Delta \longrightarrow B \overset{R}{\multimap} C}\ \overset{R}{\multimap} R$$

$$\frac{\Gamma;\emptyset;\emptyset;\emptyset \longrightarrow B \quad \Gamma;\Upsilon;\Psi;\Delta,C \longrightarrow E}{\Gamma;\Upsilon;\Psi;\Delta,B \Rightarrow C \longrightarrow E}\ \Rightarrow L \qquad \frac{\Gamma,B;\Upsilon;\Psi;\Delta \longrightarrow C}{\Gamma;\Upsilon;\Psi;\Delta \longrightarrow B \Rightarrow C}\ \Rightarrow R$$

$$\frac{\Gamma;\Upsilon;\Psi;\Delta,B[x \mapsto t] \longrightarrow C}{\Gamma;\Upsilon;\Psi;\Delta,\forall x.B \longrightarrow C}\ \forall L \qquad \frac{\Gamma;\Upsilon;\Psi;\Delta \longrightarrow B[x \mapsto c]}{\Gamma;\Upsilon;\Psi;\Delta \longrightarrow \forall x.B}\ \forall R$$

provided that c is not free in the lower sequent.

$$\frac{\Gamma_1;\Upsilon_1;\Psi_1;\Delta_1 \longrightarrow B \quad \Gamma_2;\Upsilon_2;\Psi_2;\Delta_1,B \longrightarrow C}{\Gamma_1,\Gamma_2;\Upsilon_1,\Upsilon_2;\Psi_1,\Psi_2;\Delta_1,\Delta_2 \longrightarrow C}\ cut_L$$

$$\frac{\Gamma_1;\emptyset;\emptyset;\emptyset \longrightarrow B \quad \Gamma_2,B;\Upsilon;\Psi;\Delta \longrightarrow C}{\Gamma_1,\Gamma_2;\Upsilon;\Psi;\Delta \longrightarrow C}\ cut_I$$

$$\frac{\Gamma_1;\emptyset;\Psi_1;\emptyset \longrightarrow B \quad \Gamma_2;\Upsilon;\Psi_2,B;\Delta \longrightarrow C}{\Gamma_1,\Gamma_2;\Upsilon;\Psi_1,\Psi_2;\Delta \longrightarrow C}\ cut_A$$

$$\frac{\Gamma_1;\Upsilon_1;\emptyset;\emptyset \longrightarrow B \quad \Gamma_2;\Upsilon_2,B;\Psi;\Delta \longrightarrow C}{\Gamma_1,\Gamma_2;\Upsilon_1,\Upsilon_2;\Psi;\Delta \longrightarrow C}\ cut_R$$

Fig. 3. System $\mathcal{O}$ for Intuitionistic, Relevant, Affine, and Linear Implication

$$\frac{}{\Gamma;\emptyset;\Psi;\emptyset \longrightarrow \mathbf{1}}\ \mathbf{1}_R \qquad \frac{\Gamma;\Upsilon;\Psi_1;\Delta_1 \longrightarrow B_1 \quad \Gamma;\Upsilon;\Psi_2;\Delta_2 \longrightarrow B_2}{\Gamma;\Upsilon;\Psi_1,\Psi_2;\Delta_1,\Delta_2 \longrightarrow B_1 \otimes B_2}\ \otimes_R$$

$$\frac{\Gamma;\emptyset;\emptyset;\emptyset \longrightarrow C}{\Gamma;\emptyset;\emptyset;\emptyset \longrightarrow !C}\ !_R$$

$$\frac{\Gamma;\Upsilon;\Psi;\Delta \longrightarrow B[x \mapsto t]}{\Gamma;\Upsilon;\Psi;\Delta \longrightarrow \exists x.B}\ \exists_R \qquad \frac{\Gamma;\Upsilon;\Psi;\Delta \longrightarrow B_i}{\Gamma;\Upsilon;\Psi;\Delta \longrightarrow B_1 \oplus B_2}\ \oplus_{R_i}$$

Fig. 4. Additional operator rules for $\mathcal{O}$

terms of the core system. The additional operator rules can however be added once the view is restricted to uniform, cut-free proofs. The first two propositions describe the relationship between $\mathcal{L}$ and $\mathcal{O}$:

Proposition 1. *The system $\mathcal{O}$ is complete for $\mathcal{L}$. That is, if the $\mathcal{L}$ sequent $\Gamma;\Delta \longrightarrow C$ is provable, then $\Gamma;\emptyset;\emptyset;\Delta \longrightarrow C$ is provable in $\mathcal{O}$.*

Proof. The proof is immediate, since each step in the $\mathcal{L}$ proof can be mapped directly to an $\mathcal{O}$ step by inserting the two empty contexts into the antecedents of the sequents. That the middle two contexts never need to be involved is clear since the Γ, Δ and $\{C\}$ are multisets of formulas with none of the new operators from $\mathcal{O}$, and it is only the left-hand rules for the two new implications that lead formulas to be moved into the two middle contexts. □

Proposition 2. *The system $\mathcal{O}$ is sound for $\mathcal{L}$ augmented with the rule:*

$$\frac{\Gamma;\Delta \longrightarrow C}{\Gamma;\Delta,\mathbb{1} \longrightarrow C}\ \mathbb{1}_L$$

in the sense that if C° is the result of recursively replacing all instances of $(D \stackrel{R}{\rightarrow} E)$ and $(D \stackrel{A}{\rightarrow} E)$ in C with $(D \Rightarrow (D \multimap E))$ and $((D \,\&\, \mathbb{1}) \multimap E)$, respectively, and if $\Gamma^\circ = \{C^\circ | C \in \Gamma\}$ and $\Gamma^{\&\mathbf{1}} = \{(C \,\&\, \mathbb{1}) | C \in \Gamma\}$, then, if the $\mathcal{O}$-sequent $\Gamma;\Upsilon;\Psi;\Delta \longrightarrow C$ is $\mathcal{O}$-provable, then $\Gamma^\circ,\Upsilon^\circ;\Upsilon^\circ,(\Psi^\circ)^{\&\mathbf{1}},\Delta^\circ \longrightarrow C^\circ$ is provable in the augmented $\mathcal{L}$.

The proof is by induction on the structure of proofs.

Hodas and Miller showed that there is an encoding of hereditary Harrop formulas, and their corresponding sequents, into $\mathcal{L}$ such that a sequent is provable in intuitionistic logic if and only if the encoded sequent is provable in $\mathcal{L}$ [7, 9]. By Propositions 1 and 2, then, the same holds true for $\mathcal{O}$. (The "if and only if" is maintained, because the new $\mathbb{1}_L$ rule in Proposition 2 will not occur in the proof of an encoded Harrop sequent.) A similar encoding and proof can be used to show that the same property holds for relevant and affine logic (over implication, conjunction and universal quantification) relative to $\mathcal{O}$.

The last two propositions demonstrate that, in spite of its enrichment relative to $\mathcal{L}$, $\mathcal{O}$ still has two crucial properties that justify using it as the foundation of a logic programming language:

Proposition 3. *Cut elimination holds for system $\mathcal{O}$. That is, if there is an $\mathcal{O}$-proof of the sequent $\Gamma_I; \Gamma_R; \Gamma_A; \Gamma_L \longrightarrow C$, then there is a proof which does not include occurrences of the cut_I, cut_R, cut_A, or cut_L rules.*

The proof is an extension of the proof of cut-elimination for $\mathcal{L}$ also given in the author's thesis [7]. It consists, in the usual manner, of a terminating algorithm for removing instances of *cut*. In this case the algorithm proceeds in phases, each of which removes *cuts* of a particular type.

Proposition 4. *Uniform proofs are complete for $\mathcal{O}$. That is, if there is a cut-free $\mathcal{O}$ proof of the sequent $\Gamma; \Upsilon; \Psi; \Delta \longrightarrow C$, then there is a uniform proof, i.e. one in which any occurrence of a sequent with a non-atomic succeedent is the conclusion of the right hand rule for the principal operator of the succeedent.*

Again, the proof is an extension of the proof for $\mathcal{L}$ and appears partially in [9] and fully in [7].

As with $\mathcal{L}$, though, the completeness of uniform proofs is not enough to yield an efficient interpreter, for the resulting programming language, due to the need to partition the affine and linear contexts in applying many of the system's rules. Fortunately, the *IO* proof system developed for $\mathcal{L}$ [9] which shows how to delay this process (in much the same way that unification delays the choice of substitution in traditional logic programming) can be extended to the new system with only a few changes.

5 Conclusion

We have shown that the system $\mathcal{O}$ forms an attractive refinement of $\mathcal{L}$ which provides new implication operators corresponding to all the possible variants of context management. This system will be implemented in the next public release of the Lolli linear-logic programming interpreter.

So, returning to the original motivation, programs can now be written which use the new forms of reasoning directly. For instance, if affine implication is given the concrete syntax `--@`,[6] the switch example can be written as:

```
init Name State G :- state Name State --@ G.
set Name NewState G :- state Name State,
                       state Name NewState --@ G.

toggle Name G :- state Name off, state Name on --@ G.
toggle Name G :- state Name on, state Name off --@ G.

setting Name State :- state Name State.
```

[6] Finding a reasonable concrete syntax for the new arrows of this system has been a challenge. The hope is that the at-sign in '`--@`' will at least be mnemonic for 'affine'. Suggestions for better choices are welcome.

which will behave properly, even when multiple switches have been defined.

Similarly, if relevant implication is given the concrete syntax '->>' then it is possible to implement relevant reasoning systems like the following one, which is taken from Bollen's article [1].

```
state zone1 (downwind-of zone2).
state zone4 toxic-dump.
state zone4 populated.

state Z polluted :- state Z1 factory, state Z (downwind-of Z1).
state Z danger-to-pop :- state Z toxic-dump, state Z populated.
```

In this setting we can have the following interaction:

```
?- state zone2 factory ->> state polluted zone2.
yes.
?- state zone2 factory ->> state danger-to-pop zone4.
no.
?- state danger-to-pop zone4.
yes.
```

While this particular interaction does not actually make use of the allowed contraction, it is not hard to conceive of queries that would.

It is important to note the fact that the simplicity of most of the proofs of properties of $\mathcal{L}$ and $\mathcal{O}$ is the result of the careful restriction of these systems to a few well behaved operators. Any attempt to integrate these different forms of reasoning over a broad set of operators is likely to prove quite difficult. Witness the complexity of Girard's system $\mathcal{LU}$, which unifies classical and intuitionistic reasoning [4]. The fact that a useful language results from this work demonstrates that sequential logic programming is really based mostly on the logic of implication.

6 Acknowledgments

The author is thankful to Dale Miller, Frank Pfenning, and many others for their helpful comments on this work.

References

1. A. W. Bollen. Relevant logic programming. *Journal of Automated Reasoning*, 7(4):563–586, December 1991.
2. D. M. Gabbay and U. Reyle. N-Prolog: An extension of Prolog with hypothetical implications. I. *Journal of Logic Programming*, 1:319 – 355, 1984.
3. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
4. Jean-Yves Girard. On the unity of logic. Technical Report 26, Université Paris VII, June 1991.

5. Joshua S. Hodas. Lolli: An extension of λProlog with linear logic context management. In Dale Miller, editor, *Proceedings of the 1992 λProlog Workshop*, 1992.
6. Joshua S. Hodas. Specifying filler-gap dependency parsers in a linear-logic programming language. In Krzysztof R. Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming, Washington D.C.*, pages 622 – 636, 1992.
7. Joshua S. Hodas. *Logic Programming in Intuitionistic Linear Logic: Theory, Design, and Implementation.* PhD thesis, University of Pennsylvania, Department of Computer and Information Science, August 1993.
8. Joshua S. Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic: Extended abstract. In G. Kahn, editor, *Sixth Annual Symposium on Logic in Computer Science*, pages 32 – 42, Amsterdam, July 1991.
9. Joshua S. Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Journal of Information and Computation*, 1994. To appear.
10. Dale Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, 6:79 – 108, 1989.
11. 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.
12. Gopalan Nadathur and Dale Miller. An Overview of λProlog. In *Fifth International Logic Programming Conference*, pages 810–827, Seattle, Washington, August 1988. MIT Press.