Forward and Backward Chaining in Linear Logic

James Harland[1] David Pym[2] Michael Winikoff[1]

[1] Department of Computer Science, Royal Melbourne Institute of Technology
[2] Queen Mary and Westfield College, University of London

Abstract. Logic programming languages based on linear logic are of both theoretical and practical interest, particulaly because such languages can be seen as providing a logical basis for programs which execute within a dynamic environment. Most linear logic programming languages are implemented using standard resolution or *backward chaining* techniques. However, there are many applications in which the combination of such techniques with *forward chaining* ones are desirable. We develop a proof-theoretic foundation for a system which combines both forms of reasoning in linear logic.

1 Introduction

Backward chaining is a standard technique in automated deduction, particularly in *logic programming* systems, often taking the form of a version of Robinson's *resolution rule* [18]. The fundamental question is to determine whether or not a given formula follows from a given set of formulæ, and there are various techniques which can be used to guide the search for a proof.

An instance of this approach is the analysis of logic programming in intuitionisitic logic [13]. The standard such analysis is that of Miller et al. [13], based around the notion of *uniform proofs*. These are defined in terms of the sequent calculus, which is well-known to be suited for analyses of backward-chaining.

With more recent interest in logic programming languages based on *linear logic* [6], the natural extension of these backward chaining techniques to linear logic has been much studied [1, 2, 4, 9, 17]; generally, it follows a similar pattern to intuitionistic logic. The details of the analysis are more intricate than in intuitionistic logic, and there are a number of points of diversion amongst the various approaches, but the same general procedure is followed.

Whilst the sequent calculus is a good basis for backward chaining, other systems for inference in intuitionistic logic provide forward chaining capabilities. Hilbert-type systems are the oldest and perhaps best-known of such systems [10]. Such systems allow different logics to be specified by different sets of axioms whilst maintaining modus ponens as the sole means of inference.

Another technical expression of forward chaining in intuitionistic logic may be found in the T_P operator used in the semantics of logic programs. Here, a mapping is made from interpretations to interpretations, in which the image is the result of applying the rules of the program to the initial interpretation via a combination of modus ponens and unification. The semantics of the program is then given by the least fixed point of this operator. It is interesting to note that this forward chaining system is traditionally used to provide a fixpoint semantics for SLD-resolution [3], a backward-chaining system.

It should be noted that a key property of the modus ponens rule in intuitionistic logic is that it preserves equivalence: $\phi \wedge (\phi \supset \psi) \equiv \phi \wedge \psi$. This strong property greatly simplifies the analysis of this rule of inference.

A combination of both backward and forward chaining may be found in deductive database systems such as Aditi [20]. In such systems, which are based on variants of Prolog, forward chaining is generally used in order to compute all answers to a query using efficient join algorithms and other techniques from relational databases, whilst backward chaining is used for less data-intensive computational tasks (such as format conversions).

The properties of forward and backward chaining systems for intuitionistic logic are generally well-understood. However, the question of how to best integrate the two models still remains. Give that Hilbert-type systems generally do not make any provision for backward chaining techniques, it seems reasonable to address the question of integration by investigating the incorporation of forward chaining features into the sequent calculus.

This is achieved by inserting *directed cuts* into an otherwise cut-free sequent calculus proof. Cut-free proofs are generally used in proof search to avoid a significant amount of non-determinism (*i.e.*, having to choose a cut formula arbitrarily); in the case of a directed cut, the cut formula will be calculated from the antecedent, and hence will not have the same problem.

The presence of both backward and forward chaining mechanisms in deductive databases suggests that a similar integration for linear logic will prove fruitful, especially as linear logic has been used to model database updates, state and action problems and concurrency.

However, the use of modus ponens in linear logic is not as simple as in intuitionistic logic since modus ponens does not preserve linear equivalence. For example, in linear logic $p \otimes (p \multimap q) \vdash q$ but in proving q we have to "consume" p.

In § 2, we discuss the issues for integrating forward chaining into intuitionistic logic; in § 3, we give a similar discussion for linear logic. In § 4, we present our formal results; in § 5, we show that our inference rules respect an encoding of intuitionistic logic programs into linear ones.

2 Forward Chaining in Intuitionistic Logic

The classic account of backward chaining (or *goal-directed*) inference in intuitionistic logic is [13]. There it is shown how the following class of formulæ, known as *hereditary Harrop formulæ*, can be used in a backward-chaining manner:

Definite formulæ: $D ::= A \mid D \wedge D \mid \forall x D \mid G \supset A$
Goal formulæ: $G ::= A \mid G \wedge G \mid G \vee G \mid \forall x G \mid \exists x G \mid D \supset G$

The key question is then how to introduce an appropriate notion of forward chaining into this framework. Our notion of forward chaining is based on *directed cuts*. It is standard in backward chaining systems to omit the cut rule in proof search; in general, the cut formula is entirely arbitrary, making backward chaining proof search involving this rule infeasible. This omission is safe, in that the cut-elimination theorem ensures that it is complete to search for only cut-free proofs. In our case, we wish to introduce a particular kind of cut known as a *directed* or *analytic* cut. Here, whilst the cut formula is not known in advance, we are able to determine it using certain inference rules, which thus form nthe forward chaining aspect of computation.

Hence our general procedure is to replace an inference with conclusion $\mathcal{P} \vdash G$, arrived at using the usual rules of the intuitionistic sequent calculus, with one of the form

$$\frac{\mathcal{P} \leadsto \mathcal{P}' \quad \mathcal{P}' \vdash G}{\mathcal{P} \vdash G}$$

where $\mathcal{P} \leadsto \mathcal{P}'$ represents a provability relation between programs. In particular, $\mathcal{P}'$ need not be known in advance, but will be calculated in the course of a proof. In more traditional terms, this represents a cut; when searching for a proof of $\mathcal{P} \vdash G$, we first determine a program $\mathcal{P}'$ such that $\mathcal{P} \vdash \bigwedge \mathcal{P}'$ and $\mathcal{P}' \vdash G$. The key technical question is then to find the appropriate rules for $\leadsto$.

This procedure is reminiscent of *interpolation* results, in particular those of [7], in which it is shown that for any uniform proof of $\mathcal{P} \vdash G$, there is a program $\mathcal{P}'$ such that $\mathcal{P} \vdash \bigwedge \mathcal{P}'$ and $\mathcal{P}' \vdash G$, where the latter proof is considerably simpler than the original one. In particular, $\mathcal{P}'$ can be constructed from G by replacing "indefinite" formulæ (such as $p \vee q$) with definite ones (such as p).

In order to produce the appropriate inference rules for $\leadsto$, recall that as $\leadsto$ is intended as a relation between programs (and hence definite formulæ), we may consider $\leadsto$ as a means of making *definite* inferences (*i.e.*, between sets of definite formulæ) and $\vdash$ as a means of making *potentially indefinite* inferences (*i.e.*, with definite formulæ in the antecedent and a goal formula in the succedent). As a result, we rewrite the inference rule $\wedge$L as below.

$$\frac{\mathcal{P}, D_1, D_2 \vdash G}{\mathcal{P}, D_1 \wedge D_2 \vdash G}\ \wedge L \qquad \frac{\dfrac{\mathcal{P}, D_1, D_2 \leadsto \mathcal{P}'}{\mathcal{P}, D_1 \wedge D_2 \leadsto \mathcal{P}'}\ \wedge L \quad \mathcal{P}' \vdash G}{\mathcal{P}, D_1 \wedge D_2 \vdash G}\ cut$$

Similar remarks apply to the rules $\forall$L, WL and CL.

The case for $\supset$L is somewhat more intricate (unsurprisingly). The instance of this rule specialized to definite and goal formulæ is as follows.

$$\frac{\mathcal{P} \vdash G \quad \mathcal{P}, D \vdash \Delta}{\mathcal{P}, G \supset D \vdash \Delta}$$

The left hand premise has a directed cut introduced into it. There are then two different forms of this rule, depending on the status of Δ.

If Δ is a goal formula G', then we have the rule

$$\frac{\mathcal{P} \leadsto \mathcal{P}' \quad \mathcal{P}' \vdash G \quad \mathcal{P}, D \vdash G'}{\mathcal{P}, G \supset D \vdash G'}\ \supset\vdash$$

If $\mathit{\Delta}$ is a program $\mathcal{P}''$, then we have the rule

$$\frac{\mathcal{P} \rightsquigarrow \mathcal{P}' \quad \mathcal{P}' \vdash G \quad \mathcal{P}, D \rightsquigarrow \mathcal{P}''}{\mathcal{P}, G \supset D \rightsquigarrow \mathcal{P}''} \supset\!\rightsquigarrow$$

This means that $\mathcal{P}$ can be "updated" to $\mathcal{P}'$ before being used for inference.

Turning to other rules for $\rightsquigarrow$, it seems reasonable to require that the relation $\rightsquigarrow$ be reflexive, transitive and monotonic, resulting in the following rules:

$$\frac{}{\mathcal{P} \rightsquigarrow \mathcal{P}}\ \mathit{Axiom} \rightsquigarrow \qquad \frac{\mathcal{P}_1 \rightsquigarrow \mathcal{P}_2 \quad \mathcal{P}_2 \rightsquigarrow \mathcal{P}_3}{\mathcal{P}_1 \rightsquigarrow \mathcal{P}_3}\ \mathit{Cut} \rightsquigarrow \qquad \frac{\mathcal{P}_1 \rightsquigarrow \mathcal{P}_3}{\mathcal{P}_1, \mathcal{P}_2 \rightsquigarrow \mathcal{P}_3, \mathcal{P}_2}\ \mathit{Weak} \rightsquigarrow$$

In addition, it would seem reasonable to include a version of the cut rule in which the conclusion is $\mathcal{P} \vdash G$, *i.e.*,

$$\frac{\mathcal{P} \rightsquigarrow \mathcal{P}' \quad \mathcal{P}' \vdash G}{\mathcal{P} \vdash G}\ \mathit{Cut} \vdash$$

Given these rules, it is possibly to simplify the implication rules.

In particular, we can omit the rightmost premise of the $\supset\!\rightsquigarrow$. We can recover the original rule by use of the Cut $\rightsquigarrow$ rule as follows:

$$\frac{\dfrac{\mathcal{P} \rightsquigarrow \mathcal{P}' \quad \mathcal{P}' \vdash G}{\mathcal{P}, G \supset D \rightsquigarrow \mathcal{P}, D} \supset\!\rightsquigarrow \qquad \mathcal{P}, D \rightsquigarrow \mathcal{P}''}{\mathcal{P}, G \supset D \rightsquigarrow \mathcal{P}''}\ \mathit{Cut} \rightsquigarrow$$

Also, the $\supset\vdash$ rule can be replaced by a combination of the (simplified) $\supset\!\rightsquigarrow$ rule and Cut $\vdash$, as follows:

$$\frac{\dfrac{\mathcal{P} \rightsquigarrow \mathcal{P}' \quad \mathcal{P}' \vdash G}{\mathcal{P}, G \supset D \rightsquigarrow \mathcal{P}, D} \supset\!\rightsquigarrow \qquad \mathcal{P}, D \vdash G'}{\mathcal{P}, G \supset D \vdash G'}\ \mathit{Cut} \vdash$$

Hence we arrive at the following set of inference rules:

$$\frac{}{\mathcal{P} \rightsquigarrow \mathcal{P}}\ \mathit{Axiom} \rightsquigarrow \qquad \frac{}{\mathcal{P}, A \vdash A}\ \mathit{Axiom} \vdash \qquad \frac{\mathcal{P} \rightsquigarrow \mathcal{P}'}{\mathcal{P}, D \rightsquigarrow \mathcal{P}'}\ W \rightsquigarrow \qquad \frac{\mathcal{P}, D, D \rightsquigarrow \mathcal{P}'}{\mathcal{P}, D \rightsquigarrow \mathcal{P}'}\ C \rightsquigarrow$$

$$\frac{\mathcal{P} \rightsquigarrow \mathcal{P}'' \quad \mathcal{P}'' \rightsquigarrow \mathcal{P}'}{\mathcal{P} \rightsquigarrow \mathcal{P}'}\ \mathit{Cut} \rightsquigarrow \qquad \frac{\mathcal{P} \rightsquigarrow \mathcal{P}' \quad \mathcal{P}' \vdash G}{\mathcal{P} \vdash G}\ \mathit{Cut} \vdash \qquad \frac{\mathcal{P} \rightsquigarrow \mathcal{P}'}{\mathcal{P}, \mathcal{P}'' \rightsquigarrow \mathcal{P}', \mathcal{P}''}\ \mathit{Weak}$$

$$\frac{\mathcal{P} \rightsquigarrow \mathcal{P}' \quad \mathcal{P}' \vdash G}{\mathcal{P}, G \supset D \rightsquigarrow \mathcal{P}, D} \supset\!\rightsquigarrow \qquad \frac{\mathcal{P}, D_1, D_2 \rightsquigarrow \mathcal{P}'}{\mathcal{P}, D_1 \wedge D_2 \rightsquigarrow \mathcal{P}'}\ \wedge \rightsquigarrow \qquad \frac{\mathcal{P} \vdash G_1 \quad \mathcal{P} \vdash G_2}{\mathcal{P} \vdash G_1 \wedge G_2}\ \wedge \vdash$$

$$\frac{\mathcal{P}, D[t/x] \rightsquigarrow \mathcal{P}'}{\mathcal{P}, \forall x D \rightsquigarrow \mathcal{P}'} \forall \rightsquigarrow \qquad \frac{\mathcal{P} \vdash G[y/x]}{\mathcal{P} \vdash \forall x G} \forall \vdash \qquad \frac{\mathcal{P} \vdash G_i}{\mathcal{P} \vdash G_1 \vee G_2} \vee \vdash \qquad \frac{\mathcal{P} \vdash G[t/x]}{\mathcal{P} \vdash \exists x G} \exists \vdash$$

The rule $\forall \vdash$ has the usual restriction that y is not free in $\mathcal{P}$ or G.

Note that the Cut $\vdash$ and $\supset\rightsquigarrow$ rules have the same premises.

Note also that the rules $\wedge \rightsquigarrow$ and $\forall \rightsquigarrow$ are reminiscent of the $[-]$ mapping of [13]. In fact, if we were to replace the $\wedge$L and $\forall$L rules of the intuitionistic sequent calculus with $\wedge \rightsquigarrow$ and $\forall \rightsquigarrow$ respectively, then these two rules would effectively construct the $[-]$ mapping.

One final issue is that as $\rightsquigarrow$ is a proof relation between definite formulæ, we need to take some care when "synchronising" an inference $\mathcal{P} \rightsquigarrow \mathcal{P}'$ with $\mathcal{P}' \vdash G$. For example, we have that $\forall x\ p(x) \vdash \exists z\ p(f(z))$ and $\forall x\ p(x) \rightsquigarrow p(f(y))$, but $p(f(y)) \not\vdash \exists z\ p(f(z))$. Hence when interpreting a succedent $\mathcal{P}'$ in $\mathcal{P} \rightsquigarrow \mathcal{P}'$, we need to "repackage" the formulæ in $\mathcal{P}'$. We shall revisit this issue in more detail in the next section.

3 Forward Chaining in MAILL

Here we look to extend the above analysis to the fragment of linear logic known as MAILL (Multiplicative Additive Intuitionistic Linear Logic), in which the connective $\parr$ is absent. There is no great technical problem with extending the analysis to the full logic, but the development is conceptually simpler without it.

One of the key technical issues when extending the above considerations to logics which contain multiplicative rules is that modus ponens no longer preserves equivalence. As the rules of contraction and weakening do not apply, we now have to make a choice as to whether to use a given instance of a formula as an "input" to a rule, or to preserve it. For example, in the multiplicative fragment of linear logic, we have that $p, p \multimap q \vdash q$, but that $p, p \multimap q \not\vdash p \otimes q$, and $p, p \multimap q \not\vdash p \,\&\, q$. Hence the modus ponens rule acts more like a committed choice, as, unlike the pure intuitionistic case, there is no way to preserve equivalence.

The mixture of multiplicative and additive fragments introduces further complexities. For example, consider the program $p, !(p \multimap q)$. Here we can either use p as an "input" to the rule $!(p \multimap q)$, but as we cannot make a copy of p, in doing so we "lose" the use of p. Hence, we have both $p, !(p \multimap q) \vdash p$ and $p, !(p \multimap q) \vdash q$, according to whether we preserve or use p. One way to express this succinctly is to note that in this case we have that $p, !(p \multimap q) \vdash p \,\&\, q$.

Hence the interaction between the multiplicative and additive fragments in this way is an important technical issue.

As for the intuitionistic case above, we wish to be able to replace (roughly speaking) any occurrence of $\mathcal{P} \vdash \mathcal{G}$ in a backward chaining proof with $\mathcal{P} \rightsquigarrow \mathcal{P}'$ and $\mathcal{P}' \vdash \mathcal{G}$ (*i.e.,* incorporating into this inference a linear version of the Cut $\vdash$ rule), thus converting it into a mixed forward chaining and backward chaining proof.

First, let us consider the $\multimap$L rule of intuitionistic linear logic (specialized appropriately to definite formula and goal formulæ):

$$\frac{\mathcal{P}_1 \vdash G \qquad \mathcal{P}_2, D \vdash \Delta}{\mathcal{P}_1, \mathcal{P}_2, G \multimap D \vdash \Delta}$$

As above, we readily obtain the rule

$$\frac{\mathcal{P}_1 \leadsto \mathcal{P}' \quad \mathcal{P}' \vdash G \quad \mathcal{P}_2, D \vdash G'}{\mathcal{P}_1, \mathcal{P}_2, G \multimap D \vdash G'} \multimap\vdash$$

and similarly the rule

$$\frac{\mathcal{P}_1 \leadsto \mathcal{P}' \quad \mathcal{P}' \vdash G \quad \mathcal{P}_2, D \leadsto \mathcal{P}_3}{\mathcal{P}_1, \mathcal{P}_2, G \multimap D \leadsto \mathcal{P}_3} \multimap\leadsto$$

As above, the $\multimap\leadsto$ rule can be simplified (by omitting the rightmost premise) and the $\multimap\vdash$ rule eliminated by an appropriate use of the rules Cut $\leadsto$ and Cut $\vdash$.

Also as above, we re-write the left rules ($\forall$L, $\otimes$L, &L, !L, **1**L, W!L, C!L) in the obvious manner.

Two further points of difference with the intuitionistic case remain. One is that, as discussed above, the linear version of modus ponens does not preserve equivalence. Hence it is possible for there to be two programs $\mathcal{P}_1$ and $\mathcal{P}_2$ such that $\mathcal{P} \leadsto \mathcal{P}_1$ and $\mathcal{P} \leadsto \mathcal{P}_2$ where $\mathcal{P}_1$ and $\mathcal{P}_2$ are not equivalent, which makes it seem reasonable to adopt the following rule (modulo the comments on reconstruction below):

$$\frac{\mathcal{P} \leadsto \mathcal{P}_1 \quad \mathcal{P} \leadsto \mathcal{P}_2}{\mathcal{P} \leadsto (\bigotimes \mathcal{P}_1) \,\&\, (\bigotimes \mathcal{P}_2)} \; collect$$

It is this rule that enables us to conclude that $p, !(p \multimap q) \leadsto p \,\&\, q, !(p \multimap q)$, which enables us to state completeness results in a straightforward manner.

The other is that in order to be able to make appropriate conclusions involving exponentials (and ! in particular), it seems necessary to include a version of the !R of the linear sequent calculus.

As noted above, as $\leadsto$ is a relation between programs, in proof-theoretic terms it is a relation between antecedents. Hence given $\mathcal{P} \leadsto \mathcal{P}'$, it will generally be the case that $\mathcal{P}'$ is in clausal form, which means that reconstruction of the program corresponding to $\mathcal{P}'$ will be necessary.

Let us consider an example. Consider the provable sequent

$$\forall x\ p(f(x)), \forall x\ q(g(x)), !\forall x\ p(x) \multimap r(x), !\forall y\ q(y) \multimap s(y) \vdash \forall x\ r(f(x)) \otimes \forall y\ s(g(y))$$

which has the following proof in the linear sequent calculus (in which, for simplicity, we refer to $\forall x\ p(f(x))$ as A, $\forall x\ q(g(x))$ as B, $!\forall x\ p(x) \multimap r(x)$ as $!C$ and $!\forall y\ q(y) \multimap s(y)$ as $!D$):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p(f(z)) \vdash p(f(z))} \quad \dfrac{}{r(f(z)) \vdash r(f(z))}}{p(f(z)), p(f(z)) \multimap r(f(z)) \vdash r(f(z))}\multimap L}{A, p(f(z)) \multimap r(f(z)) \vdash r(f(z))}\forall L}{A, C \vdash r(f(z))}\forall L}{A, !C \vdash r(f(z))}\,!L}{A, !C \vdash \forall x\; r(f(x))}\forall R \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{q(g(w)) \vdash q(g(w))} \quad \dfrac{}{s(g(w)) \vdash s(g(w))}}{q(g(w)), q(g(w)) \multimap s(g(w)) \vdash s(g(w))}\multimap L}{B, q(g(w)) \multimap s(g(w)) \vdash s(g(w))}\forall L}{B, D \vdash s(g(w))}\forall L}{B, !D \vdash s(g(w))}\,!L}{B, !D \vdash \forall y\; s(g(y))}}{A, B, !C, !D \vdash \forall x\; r(f(x)) \otimes \forall y\; s(g(y))}\otimes R$$

For the version containing $\rightsquigarrow$, we get the following proof, made up of two sub-proofs joined by a cut.

Sub-proof 1:

$$\dfrac{\dfrac{\dfrac{}{p(f(z)) \rightsquigarrow p(f(z))} \quad \dfrac{}{p(f(z)) \vdash p(f(z))}}{p(f(z)), B, p(f(z)) \multimap r(f(z)), !C, !D \rightsquigarrow r(f(z)), B, !C, !D}\multimap}{A, B, !C, !D \rightsquigarrow r(f(z)), B, !C, !D}\forall L, C!L, !L$$

Sub-proof 2:

$$\dfrac{\dfrac{\dfrac{}{q(g(z)) \rightsquigarrow q(g(w))} \quad \dfrac{}{q(g(z)) \vdash q(g(w))}}{r(f(z)), q(g(w)), q(g(w)) \multimap s(g(w)), !C, !D \rightsquigarrow r(f(z)), s(g(w)), !C, !D}\multimap}{r(f(z)), B, !C, !D \rightsquigarrow r(f(z)), s(g(w)), !C, !D}\forall L, C!L, !L$$

Overall:

$$\dfrac{\begin{array}{c}\vdots\\ A, B, !C, !D \rightsquigarrow r(f(z)), B, !C, !D\end{array} \quad \begin{array}{c}\vdots\\ r(f(z)), B, !C, !D \rightsquigarrow r(f(z)), s(g(w)), !C, !D\end{array}}{A, B, !C, !D \rightsquigarrow r(f(z)), s(g(w)), !C, !D}\,Cut \rightsquigarrow$$

Note that in order to derive the final result of $\forall x\ r(f(x)) \otimes \forall y\ s(g(y))$, we need to "package up" the final state reached, essentially by using the equivalent of the left inference rules ∀L and ⊗L. This is just the inverse of the clausal decomposition mapping $[-]$ of [17]; in order to recover the appropriate program from the derived multiset of clausal formulæ, we need to use the inverse of this mapping.

For example, if we find that $\mathcal{P} \rightsquigarrow \{p(x), q(x), r(y)\}$, then we interpret this as showing that $\mathcal{P} \vdash (\forall x\ p(x) \otimes q(x)) \otimes \forall y\ r(y)$.

4 A Calculus and Its Properties

First we define the class of definite and goal formulæ under consideration. This is essentially the class of formulæ from [17] restricted to MAILL (*i.e.*, no occurrences of $\invamp$ or ?). We also omit goals containing implications, again for reasons of technical simplicity. However, we permit formulæ of the form $G \multimap D$ rather than just $G \multimap A$, which are more natural for forward chaining proofs.

Definition 1. *Definite and goal formulæ are defined as follows:*
Definite *formulæ:* $D ::= A \mid D \otimes D \mid D \,\&\, D \mid \mathbf{1} \mid \forall x D \mid !D \mid G \multimap D$
Goal *formulæ:* $G ::= A \mid G \otimes G \mid G \,\&\, G \mid G \oplus G \mid \mathbf{1} \mid \forall x G \mid \exists x G \mid !G$

A program is a multiset of closed definite formulæ.

As the example of the previous section shows, we need to be able to reconstruct a program from its "decomposed" form. As noted in [17], it is important to know whether a given variable has been quantified within the scope of a ! or not. For example, in the formula $\forall x!\forall y p(x,y)$ x is *global*, *i.e.*, it *does not* require renaming when the formula is copied, and y is *local*, *i.e.*, it *does* require renaming when the formula is copied. As in [17], we will assume that there is a means of identifying global and local variables from their names alone, which will resolve the ambiguity above.

Note also that $!\forall x\forall y\ p(x,y)$ and $!\forall x!\forall y\ p(x,y)$ are equivalent, and hence we do not need any finer distinctions than global and local in the above sense.

It should also be noted that unlike in classical logic (and to a lesser extent, intuitionistic logic) we cannot simply move quantifiers past formulæ which do not contain the quantified variable. For example, $\forall x\ p \otimes q(x)$ is not equivalent to $p \otimes \forall x\ q(x)$. However, it is possible to argue that the latter version is more natural than the former. Hence we introduce below the notion of *tightly quantified* formulæ.

The key point is to determine the appropriate place for quantifiers. This is not simply a matter of finding all formulæ which contain the quantified variable. For example, given the formula $p(x,y) \otimes q(y,z) \otimes r(z,x)$, it should be clear that the appropriate quantified formula is $\forall x\ \forall y\ \forall z\ p(x,y) \otimes q(y,z) \otimes r(z,x)$. However, for the formula $p(x,y) \otimes q(y,z) \otimes r(v,w)$, it would be $(\forall x\ \forall y\ \forall z\ p(x,y) \otimes q(y,z)) \otimes (\forall v\ \forall w\ r(v,w))$.

Thus we arrive at the definitions below.

We denote the free variables of a formula F as $\mathrm{free}(F)$, and $\bigcup_{F \in P} \mathrm{free}(F)$ as $\mathrm{free}(P)$.

Definition 2. *Let P be a multiset of definite formulæ, and let $x, y \in \mathrm{free}(P)$. The variables x and y are* connected *in P if $\exists F \in P$ such that $x, y \in \mathrm{free}(F)$, or $\exists z, F$ such that $x, z \in \mathrm{free}(F)$ and z and y are connected in P.*

A free variable x in free(P) is connected to a formula $F \in P$ if x is connected to a free variable in F.

A formula $\forall x_1 \ldots \forall x_n F_1 \otimes F_k$ is tightly quantified *if x_i is connected to each F_j, $1 \le j \le k$, and each F_j is tightly quantified.*

Note that we need only consider formulæ of the form $\forall x_1 \ldots \forall x_n F_1 \otimes F_k$ in this definition, as $\forall x p \,\&\, q(x)$ is equivalent to $p \,\&\, \forall x q(x)$, as are $\forall x p \multimap q(x)$ and $p \multimap \forall x q(x)$. Hence we may consider that all universal quantifiers are pushed innermost in this manner.

We will assume that all programs under consideration in this paper from hereon are tightly quantified.

We are now in a position to define the "recomposition"operator.

Definition 3. *Let $\{P_1, \ldots P_n\}$ be a maximal partition of a multiset of definite formulæ P such that $\mathrm{free}(P_i) \cap \mathrm{free}(P_j) = \emptyset$ for $i \neq j$.*

We define $\lozenge P$ *as* $\bigcup_{i=1}^{n} \forall(\otimes P_i)$. *We define* $\spadesuit P$ *as* $\bigotimes_{i=1}^{n} \forall(\otimes P_i)$.

Note that as all formulæ are tightly quantified, all we need to do to determine where the quantifier should be is to determine the minimal scope which includes all occurrences of the (free) variable in question.

Below we present the inference rules for the system.

Definition 4. *A* derivation tree *is a proof tree governed by the following rules:*

$$\frac{}{\mathcal{P} \rightsquigarrow \mathcal{P}}\ \text{Axiom} \rightsquigarrow \qquad \frac{}{A \vdash A}\ \text{Axiom} \vdash \qquad \frac{\mathcal{P} \rightsquigarrow \mathcal{P}'}{\mathcal{P}, !D \rightsquigarrow \mathcal{P}'}\ W \rightsquigarrow \qquad \frac{\mathcal{P}, !D, !D \rightsquigarrow \mathcal{P}'}{\mathcal{P}, !D \rightsquigarrow \mathcal{P}'}\ C \rightsquigarrow$$

$$\frac{\mathcal{P} \rightsquigarrow \mathcal{P}'' \quad \mathcal{P}'' \rightsquigarrow \mathcal{P}'}{\mathcal{P} \rightsquigarrow \mathcal{P}'}\ Cut \rightsquigarrow \qquad \frac{\mathcal{P} \rightsquigarrow \mathcal{P}' \quad \lozenge\mathcal{P}' \vdash G}{\mathcal{P} \vdash G}\ Cut \vdash \qquad \frac{\mathcal{P} \rightsquigarrow \mathcal{P}'}{\mathcal{P}, \mathcal{P}'' \rightsquigarrow \mathcal{P}', \mathcal{P}''}\ Weak$$

$$\frac{\mathcal{P}_1 \rightsquigarrow \mathcal{P}' \quad \lozenge\mathcal{P}' \vdash G}{\mathcal{P}_1, \mathcal{P}_2, G \multimap D \rightsquigarrow \mathcal{P}_2, D}\ \multimap\rightsquigarrow \qquad \frac{\mathcal{P}, D_i \rightsquigarrow \mathcal{P}'}{\mathcal{P}, D_1 \,\&\, D_2 \rightsquigarrow \mathcal{P}'}\ \& \rightsquigarrow \qquad \frac{\mathcal{P}, D_1, D_2 \rightsquigarrow \mathcal{P}'}{\mathcal{P}, D_1 \otimes D_2 \rightsquigarrow \mathcal{P}'}\ \otimes \rightsquigarrow$$

$$\frac{\mathcal{P}, D \rightsquigarrow \mathcal{P}'}{\mathcal{P}, !D \rightsquigarrow \mathcal{P}'}\ ! \rightsquigarrow \qquad \frac{\mathcal{P} \rightsquigarrow \mathcal{P}'}{\mathcal{P}, \mathbf{1} \rightsquigarrow \mathcal{P}'}\ \mathbf{1} \rightsquigarrow \qquad \frac{\mathcal{P}, D[t/x] \rightsquigarrow \mathcal{P}'}{\mathcal{P}, \forall x D \rightsquigarrow \mathcal{P}'}\ \forall \rightsquigarrow$$

$$\frac{\mathcal{P} \rightsquigarrow \mathcal{P}_1 \quad \ldots \quad \mathcal{P} \rightsquigarrow \mathcal{P}_n}{\mathcal{P} \rightsquigarrow (\spadesuit\mathcal{P}_1) \,\&\, \ldots \,\&\, (\spadesuit\mathcal{P}_n)}\ collect \qquad \frac{!\mathcal{P} \rightsquigarrow \mathcal{P}'}{!\mathcal{P} \rightsquigarrow !\lozenge\mathcal{P}'}\ !M$$

$$\frac{}{\vdash \mathbf{1}}\ 1 \vdash \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}\ \bot \vdash \qquad \frac{}{\Gamma \vdash \top, \Delta}\ \top \vdash$$

$$\frac{P \vdash G_1 \quad P' \vdash G_2}{P, P' \vdash G_1 \otimes G_2}\ \otimes \vdash \qquad \frac{P \vdash G_1 \quad P \vdash G_2}{P \vdash G_1 \,\&\, G_2}\ \& \vdash \qquad \frac{P \vdash G_i}{P \vdash G_1 \oplus G_2}\ \oplus \vdash$$

$$\frac{!P \vdash G}{!P \vdash !G}\ ! \vdash \qquad \frac{P \vdash G[y/x]}{P \vdash \forall x G}\ \forall \vdash \qquad \frac{P \vdash G[t/x]}{P \vdash \exists x G}\ \exists \vdash$$

We will often use $\mathcal{P} \rightsquigarrow \mathcal{P}'$ as a shorthand for the statement that $\mathcal{P} \rightsquigarrow \mathcal{P}'$ has a derivation tree.

We now show that the rules above are sound, *i.e.*, that any proof in the above system has a corresponding proof in the linear sequent calculus.

Proposition 1 (Soundness). *Let* $\mathcal{P}$ *be a multiset of definite formulæ. Then if* $\mathcal{P} \rightsquigarrow \mathcal{P}'$ *then* $\mathcal{P} \vdash \spadesuit\mathcal{P}'$.

Proof. A simple induction on the length of the derivation.

The following corollary is immediate.

Corollary 1. *Let $\mathcal{P}$ be a multiset of definite formulæ and let G be a goal formula. Then if $\mathcal{P} \rightsquigarrow \mathcal{P}'$ and $\diamondsuit\mathcal{P}' \vdash G$ then $\mathcal{P} \vdash G$.*

In order to prove a corresponding completeness result, we need the concept of the depth of a formula, which will form a bound on the indefinite part of the proof (*i.e.*, the part that uses rules for $\vdash$ rather than $\rightsquigarrow$).

Definition 5. *We define the* depth *of a formula F as follows:*
depth(A) = 0 where A is atomic
depth($\clubsuit F$) = 1 + depth(F) where $\clubsuit$ is a unary connective
depth($F_1 \heartsuit F_2$) = 1 + max(depth(F_1), depth(F_2)) where $\heartsuit$ is a binary connective
We define depth$(\mathcal{F})$ = $\Sigma_{F \in \mathcal{F}} \mathrm{depth}(F)$.

Hence we are now in a position to consider completeness.

Proposition 2 (Completeness). *Let $\mathcal{P}$ be a multiset of definite formulæ, D be a definite formula and G be a goal formula. Then*

1. *If $\mathcal{P} \vdash \spadesuit\mathcal{P}'$, then $\mathcal{P} \rightsquigarrow \mathcal{P}'$.*
2. *If $\mathcal{P} \vdash D$, then $\mathcal{P} \rightsquigarrow D, !\mathcal{P}'$ for some $\mathcal{P}'$.*
3. *If $\mathcal{P} \vdash G$, then $\exists\mathcal{P}'$ such that $\mathcal{P} \rightsquigarrow \mathcal{P}'$ and $\spadesuit\mathcal{P}' \vdash G$.*
4. *If $\mathcal{P} \vdash G$, then $\exists\mathcal{P}'$ such that $\mathcal{P} \rightsquigarrow \mathcal{P}'$ and $\spadesuit\mathcal{P}' \vdash G$ where the size of the proof of $\mathcal{P}' \vdash G$ is not more than depth(G).*

Proof.
1. (Sketch) An inductive argument based on the size of the proof. The only interesting cases are for the right rules, such as $\otimes$R. In this case, if $\spadesuit\mathcal{P}' = D_1 \otimes D_2$, then given the premises $\mathcal{P}_1 \vdash D_1$ and $\mathcal{P}_2 \vdash D_2$, by the hypothesis $\mathcal{P}_1 \rightsquigarrow \mathcal{P}_1'$ and $\mathcal{P}_2 \rightsquigarrow \mathcal{P}_2'$ where $D_i = \spadesuit\mathcal{P}_i'$, and so by a combination of Cut $\rightsquigarrow$ and Weak $\rightsquigarrow$ we get $\mathcal{P}_1, \mathcal{P}_2 \rightsquigarrow \mathcal{P}_1', \mathcal{P}_2'$ as required.
2. Follows from (1) and the ability to permute W $\rightsquigarrow$ upwards (see result below).
3. Trivial, as $\mathcal{P} \rightsquigarrow \mathcal{P}$.
4. (Sketch) An inductive argument based on the size of the proof. This based on (3) and transformations which "transfer" occurrences of left rules in the proof of $\spadesuit\mathcal{P}' \vdash G$ to inferences in the proof of $\mathcal{P} \rightsquigarrow \mathcal{P}'$.

Note that (1) above directly reflects that $\rightsquigarrow$ is an alternative characterization of provability between definite formulæ. (2) above is a slight restatement of this, in that a given definite formula could be the result of a proof (using $\vdash$) involving weakening, which corresponds to a proof (using $\rightsquigarrow$) in which $W \rightsquigarrow$ is used. (3) is a moderately general completeness result, and (4) is an extreme instance of it. As noted above, (3) is trivially true in that $\mathcal{P} \rightsquigarrow \mathcal{P}$. However, note that there may be many $\mathcal{P}'$ with this property, including one, as shown in (4), in which the forward chaining aspect is maximized, as the size of the backward chaining one is limited to the minimum number of inferences required to decompose G into atoms. Hence (3) and (4) between them show the range of possibilities for mixing the two approaches to inference.

It should be clear that the amount of mandatory forward chaining is inversely proportional to the degree of freedom allowed in the proof of $\mathcal{P}' \vdash G$; the greater the freedom, the more scope there is for backward chaining. Hence strategies which are neither maximal nor minimal in this respect will require an appropriate statement of the requirements for the proof.

One aspect of the inference rules above is that the W $\leadsto$ rule (like its counterpart in the linear sequent calculus) can always be permuted upwards. This means that consequences are not "lost" in the computation, but in the manner of 2 above, can be "picked out" of the context. In practice, we would not expect to use the W $\leadsto$ rule, in order to retain as much information as possible.

Another observation is that it is always possible to use a version of the $\multimap\leadsto$ rule in which the left hand premise is an axiom. Basically, this normalized form of the rule corresponds to a greater degree of forward chaining, in that we do not have an intermediate result $\mathcal{P}'$, but directly determine the consequences of the program.

This leads us to the definitions below.

Definition 6. *An instance of the $\multimap\leadsto$ rule is called* basic *if the left-hand premise is an axiom, i.e., $\mathcal{P} = \mathcal{P}'$. A proof is* basic *if every instance of $\multimap\leadsto$ in the proof is basic.*

A proof is strong *if it contains no occurrences of W $\leadsto$.*

Basic proofs are such that repeated occurrences of the $\multimap\leadsto$ rule are not nested, but occur sequentially (combined by occurrences of Cut $\leadsto$, where necessary). Hence basic proofs are "flatter" with respect to occurrences of $\multimap\leadsto$ than non-basic ones. This may be considered the technical effect of ensuring that only "definite" information is used in forward chainAnne64ing inferences, in that the information used to prove the body of the clause is directly present in the program.

Strong proofs give a technical characterization of the observation that information should be preserved, whenever possible, and hence the W $\leadsto$ rule should be avoided.

It is not hard to establish the following results.

Proposition 3. *If $\mathcal{P} \leadsto \mathcal{P}'$, then*

1. *there is a basic proof of $\mathcal{P} \leadsto \mathcal{P}'$*
2. *there is a strong proof of $\mathcal{P} \leadsto \mathcal{P}', \mathcal{P}''$ for some $\mathcal{P}''$*

Proof.

1. (Sketch)
   A simple induction based on the transformation

$$\dfrac{\begin{array}{c}\vdots\\ \mathcal{P} \leadsto \mathcal{P}'\end{array} \quad \begin{array}{c}\vdots\\ \mathcal{P}' \vdash G\end{array}}{\mathcal{P}, G \multimap D \leadsto D}\,\multimap\leadsto \quad \Longrightarrow \quad \dfrac{\begin{array}{c}\vdots\\ \mathcal{P} \leadsto \mathcal{P}'\end{array} \quad \dfrac{\dfrac{}{\mathcal{P}' \leadsto \mathcal{P}'}\,Ax \quad \begin{array}{c}\vdots\\ \mathcal{P}' \vdash G\end{array}}{\mathcal{P}', G \multimap D \leadsto D}\,\multimap}{\mathcal{P}, G \multimap D \leadsto D}\,Cut \leadsto$$

2. (Sketch) A simple induction based on the upward permutability of W $\leadsto$ and its absorption into the Axiom $\leadsto$ rule.

5 Encoding IL in MAILL

In [17], the following mapping was used to encode hereditary Harrop formulæ into linear logic, and it was shown that this encoding respects uniform provability.

$$\begin{array}{ll}(A)^- & = A \\ (G_1 \wedge G_2)^- & = (G_1)^- \mathbin{\&} (G_2)^- \\ (G_1 \vee G_2)^- & = (G_1)^- \oplus (G_2)^- \\ (\exists \boldsymbol{x}.G)^- & = \bigvee \boldsymbol{x} \,.\, (G)^- \\ (\forall \boldsymbol{x}.G)^- & = \bigwedge \boldsymbol{x} \,.\, (G)^- \\ (D \supset G)^- & = (D)^+ \multimap (G)^- \end{array} \qquad \begin{array}{ll}\mathcal{D}^+ & = \bigcup_{D\in\mathcal{D}} (D)^+ \\ (A)^+ & = \,!\, A \\ (D_1 \wedge D_2)^+ & = \,!\,(D_1)^+ \otimes \,!\,(D_2)^+ \\ (\forall \boldsymbol{x}.D)^+ & = \,!\,(\bigwedge \boldsymbol{x} \,.\, (D)^+) \\ (G \supset A)^+ & = !((G)^- \multimap A). \end{array}$$

If we take this encoding of hereditary Harrop formulæ into a fragment of the linear counterpart, then there is a proof of the original sequent in intuitionistic logic iff there is a proof of the encoded sequent in linear logic.

Applying this encoding to the rules governing a derivation tree in linear logic yields the rules governing a derivation tree in intuitionistic logic. In the following we use a subscript L or I to indicate which system of rules we are using. More formally we have that

1. $\mathcal{P} \vdash_I G$ iff $(\mathcal{P})^+ \vdash_L (G)^-$ [17], and
2. $\mathcal{P} \leadsto_I \mathcal{P}'$ iff $(\mathcal{P})^+ \leadsto_L (\mathcal{P}')^+$

In order to show the second of these results, firstly we observe that a number of rules ($Cut \leadsto$, $Cut \vdash$, $Axiom \leadsto$, and $Weak$) are common to the two systems.

Next we observe that applying the encoding $()^+$ to a program always yields a collection of clauses whose outermost connective is !, that is if $(\mathcal{P})^+ = \mathcal{P}'$ then $\mathcal{P}' = !\mathcal{P}''$ for some $\mathcal{P}''$. In order to derive a suitable $\mathcal{P}''$ we may need to use the equivalence $(!\mathcal{P}) \otimes (!\mathcal{Q}) \equiv !(\mathcal{P} \mathbin{\&} \mathcal{Q})$.

Given that the programs produced by the encoding are of this form the structural rules ($W \leadsto$ and $C \leadsto$) in the two systems are obviously equivalent. Also, the intuitionistic axiom rule

$$\frac{}{\mathcal{P}, A \vdash_I A}\ Axiom \vdash$$

reduces to the inference:

$$\cfrac{\cfrac{\cfrac{}{A \leadsto_L A}\ Axiom \leadsto}{!A \leadsto_L A}\ ! \leadsto}{!\mathcal{P}, !A \leadsto_L A}\ W \leadsto$$

We now briefly sketch the proof of equivalence under the encoding of the $\wedge \vdash$ rule for both systems. The proof for the other $\vdash$ rules is similar. We want to prove that $\mathcal{P} \vdash_I F_1 \wedge F_2$ iff $(\mathcal{P})^+ \vdash_L (F_1 \wedge F_2)^-$. The latter sequent is (by definition of the encoding) $(\mathcal{P})^+ \vdash_L (F_1)^- \mathbin{\&} (F_2)^-$ giving the desired result:

$$\cfrac{\begin{array}{cc}\vdots & \vdots \\ \mathcal{P} \vdash_I F_1 & \mathcal{P} \vdash_I F_2\end{array}}{\mathcal{P} \vdash_I F_1 \wedge F_2}\ \wedge \vdash \qquad \cfrac{\cfrac{\begin{array}{cc}\vdots & \vdots \\ (\mathcal{P})^+ \vdash_L (F_1)^- & (\mathcal{P})^+ \vdash_L (F_2)^-\end{array}}{(\mathcal{P})^+ \vdash_L (F_1)^- \mathbin{\&} (F_2)^-}\ \&}{(\mathcal{P})^+ \vdash_L (F_1 \wedge F_2)^-}\ \equiv$$

Finally, we sketch the equivalence proof for $\wedge \leadsto$. We want to show that $\mathcal{P}, D_1 \wedge D_2 \leadsto_I \mathcal{P}_2$ iff $(\mathcal{P}, D_1 \wedge D_2)^+ \leadsto_L (\mathcal{P}_2)^+$. This is shown by the following inference. Note that the final step in the right inference relies on the encoding producing nonlinear programs.

$$\dfrac{\begin{matrix}\vdots\\ \mathcal{P}, D_1, D_2 \leadsto_I \mathcal{P}_2\end{matrix}}{\mathcal{P}, D_1 \wedge D_2 \leadsto_I \mathcal{P}_2} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots\\ (\mathcal{P}, D_1, D_2)^+ \leadsto_L (\mathcal{P}_2)^+\end{matrix}}{(\mathcal{P})^+, !(D_1)^+, !(D_2)^+ \leadsto_L (\mathcal{P}_2)^+}\,\equiv}{(\mathcal{P})^+, !(D_1)^+ \otimes !(D_2)^+ \leadsto_L (\mathcal{P}_2)^+}\,\otimes}{(\mathcal{P})^+, (D_1 \wedge D_2)^+ \leadsto_L (\mathcal{P}_2)^+}\,\equiv}{(\mathcal{P}, D_1 \wedge D_2)^+ \leadsto_L (\mathcal{P}_2)^+}\,\equiv$$

6 Acknowledgements

Harland is grateful for the hospitality of the Department of Computer Science of Queen Mary and Westfield College, University of London during a period of sabbatical leave. Pym is partially supported by a Fellowship from the EPSRC.

References

1. J.-M. Andreoli. Logic Programming with Focusing Proofs in Linear Logic. *J. Logic Computat.* 2(3), 1992.
2. J.-M. Andreoli and R. Pareschi. Linear Objects: Logical Processes with Built-in Inheritance. Proceedings of the International Conference on Logic Programming, 496-510, Jerusalem, June, 1990.
3. M.H. van Emden and R.A. Kowalski, The Semantics of Predicate Logic as a Programming Language, *Journal of the Association for Computing Machinery* 23:4:733-742, October, 1976.
4. D. Galmiche and G. Perrier. On proof normalization in Linear Logic. Theoretical Computer Science 135:76-100, 1994.
5. G. Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift* 39, 176-210, 405-431, 1934.
6. J.-Y. Girard. Linear Logic. *Theoretical Computer Science* 50, 1-102, 1987.
7. J. Harland. A Proof-Theoretic Analysis of Goal-Directed Provability. *Journal of Logic and Computation* 4:1:69-88, January, 1994.
8. J. Harland, D. Pym, and M. Winikoff. Programming in Lygon: An Overview. Proceedings of the Fifth International Conference on Algebraic Methodology and Software Technology 391-405, Munich, July, 1996.
9. J. Hodas, D. Miller. Logic Programming in a Fragment of Intuitionistic Linear Logic: Extended Abstract. Proceedings of the Symposium on Logic in Computer Science, 32-42, Amsterdam, July, 1991.
10. S.C. Kleene. *Introduction to Metamathematics*. North Holland, 1952.
11. S.C. Kleene. *Mathematical Logic*. Wiley and Sons, 1968.
12. D. Miller. A Logical Analysis of Modules in Logic Programming. *Journal of Logic Programming:6:1&2:79-108*, 1989.
13. D. Miller, G. Nadathur, F. Pfenning and A. Ščedrov. Uniform Proofs as a Foundation for Logic Programming. *Annals of Pure and Applied Logic:51:125-157*, 1991.

14. G. Nadthur and D. Miller. Higher-Order Horn Clauses. *JACM 37:4: 777-814*, 1990.
15. G. Plotkin. Structural Operational Semantics (lecture notes). Technical Report DAIMI FN-19, Aarhus University, 1981 (reprinted 1991).
16. D. Pym. On Bunched Predicate Logic *Proceedings of the 14th IEEE Symposium on Logic in Computer Science*, 183-192, Trento, Italy, July, 1999. IEEE Computer Society, 1999.
17. D.J. Pym, J.A. Harland. A Uniform Proof-theoretic Investigation of Linear Logic Programming. Journal of Logic and Computation 4:2:175-207, April, 1994.
18. J.A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the Association for Computing Machinery* 12:1:23-41, 1965.
19. L. Sterling and E. Shapiro. *The Art of Prolog (2nd ed.).* MIT Press, 1994.
20. J. Vaghani, K. Ramamohanarao, D. Kemp, Z. Somogyi, P. Stuckey, T. Leask and J. Harland. The Aditi Deductive Database System. *VLDB Journal* 3:2:245-288, April, 1994.

A Inference rules for MAILL

$$\frac{}{\phi \vdash \phi}\ \text{axiom} \qquad \frac{\Gamma \vdash \phi, \Delta \quad \Gamma', \phi \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ \text{cut}$$

$$\frac{\Gamma, \phi, \psi, \Gamma' \vdash \Delta}{\Gamma, \psi, \phi, \Gamma' \vdash \Delta}\ \text{X-L} \qquad \frac{\Gamma \vdash \Delta, \phi, \psi, \Delta'}{\Gamma \vdash \Delta, \psi, \phi, \Delta'}\ \text{X-R}$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, \mathbf{1} \vdash \Delta}\ \mathbf{1}\text{-L} \qquad \frac{}{\vdash \mathbf{1}}\ \mathbf{1}\text{-R}$$

$$\frac{}{\bot \vdash}\ \text{L}^{\bot} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}\ \text{R}^{\bot}$$

$$\frac{}{\Gamma, \mathbf{0} \vdash \Delta}\ \mathbf{0}\text{-L} \qquad \frac{}{\Gamma \vdash \top, \Delta}\ \top\text{-R}$$

$$\frac{\Gamma \vdash \phi, \Delta}{\Gamma, \phi^{\bot} \vdash \Delta}\ \bot\text{-L} \qquad \frac{\Gamma, \phi \vdash \Delta}{\Gamma \vdash \phi^{\bot}, \Delta}\ \bot\text{-R}$$

$$\frac{\Gamma, \phi, \psi \vdash \Delta}{\Gamma, \phi \otimes \psi \vdash \Delta}\ \otimes\text{-L} \qquad \frac{\Gamma \vdash \phi, \Delta \quad \Gamma' \vdash \psi, \Delta'}{\Gamma, \Gamma' \vdash \phi \otimes \psi, \Delta, \Delta'}\ \otimes\text{-R}$$

$$\frac{\Gamma, \phi \vdash \Delta}{\Gamma, \phi \mathbin{\&} \psi \vdash \Delta} \qquad \frac{\Gamma, \psi \vdash \Delta}{\Gamma, \phi \mathbin{\&} \psi \vdash \Delta}\ \&\text{-L} \qquad \frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \mathbin{\&} \psi, \Delta}\ \&\text{-R}$$

$$\frac{\Gamma, \phi \vdash \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \phi \oplus \psi \vdash \Delta}\ \oplus\text{-L} \qquad \frac{\Gamma \vdash \phi, \Delta}{\Gamma \vdash \phi \oplus \psi, \Delta} \qquad \frac{\Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \oplus \psi, \Delta}\ \oplus\text{-R}$$

$$\frac{\Gamma \vdash \phi, \Delta \quad \Gamma', \psi \vdash \Delta'}{\Gamma, \Gamma', \phi \multimap \psi \vdash \Delta, \Delta'}\ \multimap\text{-L} \qquad \frac{\Gamma, \phi \vdash \psi, \Delta}{\Gamma \vdash \phi \multimap \psi, \Delta}\ \multimap\text{-R}$$

$$\frac{\Gamma, \phi \vdash \Delta}{\Gamma, !\phi \vdash \Delta}\ !\text{-L} \qquad \frac{!\Gamma \vdash \phi, ?\Delta}{!\Gamma \vdash !\phi, ?\Delta}\ !\text{-R}$$

$$\frac{!\Gamma, \phi \vdash ?\Delta}{!\Gamma, ?\phi \vdash ?\Delta}\ ?\text{-L} \qquad \frac{\Gamma \vdash \phi, \Delta}{\Gamma \vdash ?\phi, \Delta}\ ?\text{-R}$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, !\phi \vdash \Delta}\ W!\text{-L} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash ?\phi, \Delta}\ W?\text{-R}$$

$$\frac{\Gamma, !\phi, !\phi \vdash \Delta}{\Gamma, !\phi \vdash \Delta}\ C!\text{-L} \qquad \frac{\Gamma \vdash ?\phi, ?\phi, \Delta}{\Gamma \vdash ?\phi, \Delta}\ C?\text{-R}$$

$$\frac{\Gamma, \phi[t/x] \vdash \Delta}{\Gamma, \forall x \,.\, \phi \vdash \Delta}\ \forall\text{-L} \qquad \frac{\Gamma \vdash \phi[y/x], \Delta}{\Gamma \vdash \forall x \,.\, \phi, \Delta}\ \forall\text{-R}$$

$$\frac{\Gamma, \phi[y/x] \vdash \Delta}{\Gamma, \exists x \,.\, \phi \vdash \Delta}\ \exists\text{-L} \qquad \frac{\Gamma \vdash \phi[t/x], \Delta}{\Gamma \vdash \exists x \,.\, \phi, \Delta}\ \exists\text{-R}$$

where y is not free in Γ, Δ.