Toward Efficient Proof Search for Linear Logic

Greg Price

May 5, 2006

Abstract

Linear logic is a refinement of a mathematical logic that disallows weakening and contraction. This enables linear logic to treat propositions as resources that are part of a state and are generated and consumed, thus allowing modeling of state. Linear logic is useful in any realm that deals with reasoning about state; this includes planning problems and verification of hardware, software, and security protocols. We demonstrate two methods for improving the speed of an automated deduction system for linear logic: exploitation of a recently developed notion of left or right atomic bias and filtering of sequents. Each of these methods reduces the number of facts and iterations required by the prover to find a proof of a given theorem. Finally, we demonstrate that the resulting reduction in facts and iterations translates into a significant clock-time speedup, improving suitability for the various applications outlined above.

Contents

1 Introduction **3**
1.1 Definitions 4

2 Linear Logic **4**
2.1 State Transition Systems 6
2.2 Blocks World 7

3 Focused Inverse Method **9**
3.1 Focusing 10
3.2 Focusing the Inverse Method 11

4 Atomic Bias **12**

5 Filtering **14**

6 Experimental Results **15**
6.1 Atomic Bias 15
6.2 Filtering 18

7 Conclusion **19**

A Focusing Calculus **21**

B Derived Rules **23**

1 Introduction

Traditional non-linear logics are generally used to reason about truth. Most often, we see judgments of the form A true, A knows B, and the like. An important property of truth is that it is universal; if A is true, then it will always be true. Thus, if we know A and $A \Rightarrow B$ then we can conclude that A and B are both true. This becomes a problem, however, when we want to reason about state. If we know $have(cake)$ and $have(cake) \Rightarrow eat(cake)$, then we can have our cake and eat it, too. Linear logic [Gir87] extends a logic to enable reasoning about state. Linear implication, represented by $\multimap$, "consumes" its assumption; thus, knowing $have(cake)$ and $have(cake) \multimap eat(cake)$ only allows us to conclude $eat(cake)$.

This ability to reason about state enables reasoning about many different fields. These include concurrency, verification of hardware and software systems, and security protocols. Linearity has also been suggested as a basis for a security logic [GP06]. One of the most comprehensible examples is the domain of planning. Planning problems tend to be easily and clearly stated and translated into linear intuitionistic logic. In addition, a proof of a theorem representing a planning problem can easily be translated back into a plan of action. Thus, a proof-term-generating theorem prover can be used as a planner, but with the additional advantage of a richer language that allows much more useful reasoning in addition to the simple plan. One canonical planning problem is that of the *blocks world*. A blocks world problem involves determining the steps to move a set of blocks from one configuration into another. This example, described more thoroughly in Section 2.2, will be used as a running example throughout this paper to demonstrate the principles described.

Automated theorem provers play an important role in using logic to reason about substantial problems; they can find proofs many times faster than a person can. Even so, provers can be slower than is practical for real-world use. The development of a focusing optimization in an inverse method theorem prover for linear logic [CP05a] has provided a dramatic increase in the speed of linear logic theorem proving and increased the viability of applying automated reasoning in linear logic to the domains listed above.

The system still suffers, however, from various inefficiencies. This work seeks to address some of these issues through several methods. The first, described in Section 4, is to exploit the recently discovered notion of atomic bias [CPP06], which has major effects on the structure of the proof search. The second, described in Section 5, is recognizing facts that represent impossible situations, such as the blocks world's hand holding two blocks, and removing them from consideration.

These methods have a theoretical basis for decreasing the necessary work performed by the prover, and the expected result is a significant decrease in the wall clock time required.

The remainder of this paper is organized in three main parts: Sections 2 and 3 provide theoretical background in linear logic and proof search, respectively; Sections 4 and 5 describe the above methods for increasing efficiency; and Section 6 contains experimental results. The final section summarizes the findings and describes future work in this area.

1.1 Definitions

The examples throughout are shown using a *sequent calculus*. A sequent is written $P_1, \ldots, P_n \Longrightarrow Q$, meaning that Q is provable under the assumptions $P_1, \ldots, P_n$. The assumptions $P_1, \ldots, P_n$ are referred to as the *context*, and are often abbreviated using a capital Greek letter such as Γ. Certain kinds of sequents that have multiple contexts with different meanings will be introduced later. A sequent calculus represents a proof derivation using inference rules of the form

$$\frac{\Gamma \Longrightarrow P \quad \Gamma \Longrightarrow Q}{\Gamma \Longrightarrow P \wedge Q}$$

These rules mean that a proof of the conclusion (on the bottom of the rule) can be constructed from proofs of each the premises (on top). These rules can have zero or more premises; a rule with zero premises means that there is an immediate proof of the conclusion.

This paper considers a fragment of linear logic using the following connectives: multiplicative conjunction and its unit, represented by $\otimes$ and $\mathbf{1}$, additive conjunction and its unit, represented by $\&$ and $\top$, additive disjunction and its unit, represented by $\oplus$ and $\mathbf{0}$, the exponential, represented by !, and universal and existential quantification.

2 Linear Logic

The development of linear logic has greatly increased the ease of using logic to reason about stateful systems. The essential difference between a linear logic and its non-linear counterpart is that linear logic disallows weakening and contraction rules. The result is that an assumption cannot be unused or used more than once.

The difference can be seen clearly in a simple example:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \Longrightarrow A}\,\text{init} \quad \dfrac{\dfrac{}{A \Longrightarrow A}\,\text{init} \quad \dfrac{}{B \Longrightarrow B}\,\text{init}}{A, A \supset B \Longrightarrow B}\,\supset L}{A, A, A \supset B \Longrightarrow A \wedge B}\,\wedge L}{A, A \supset B \Longrightarrow A \wedge B}\,\text{contr}}{A \wedge (A \supset B) \Longrightarrow A \wedge B}\,\wedge L}{\cdot \Longrightarrow A \wedge (A \supset B) \supset A \wedge B}\,\supset R$$

It is clear in the above proof that the hypothetical assumption A is used twice, and the contraction rule is necessary to remove the "extra" A from the set of assumptions. Without contraction, as is the case in linear logic, the best we can do is the following proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \Longrightarrow A}\,\text{init} \quad \dfrac{\dfrac{}{A \Longrightarrow A}\,\text{init} \quad \dfrac{}{B \Longrightarrow B}\,\text{init}}{A, A \multimap B \Longrightarrow B}\,\multimap\text{L}}{A, A, A \multimap B \Longrightarrow A \otimes B}\,\otimes\text{R}}{A \otimes A, A \multimap B \Longrightarrow A \otimes B}\,\otimes\text{L}}{A \otimes A \otimes (A \multimap B) \Longrightarrow A \otimes B}\,\otimes\text{L}}{\cdot \Longrightarrow A \otimes A \otimes (A \multimap B) \multimap A \otimes B}\,\multimap\text{R}$$

This example uses the two linear connectives $\otimes$ and $\multimap$. The first, $\otimes$ ("tensor"), represents having resources simultaneously. The second, $\multimap$ ("lolli"), is like non-linear implication except in that it "consumes" the hypothetical assumption; it is often used to represent moving from one state to another. Thus, the above derivation merely proves that if you have two pieces of cake, you can eat one and have one left.

A similar effect occurs without the use of weakening. Consider the following simple proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \Longrightarrow A}\,\text{init}}{A, B \Longrightarrow A}\,\text{weak}}{A \wedge B \Longrightarrow A}\,\wedge\text{L}}{\cdot \Longrightarrow A \wedge B \supset A}\,\supset\text{R}$$

Without weakening, it is possible to prove $A \multimap A$ or $A \otimes B \multimap A \otimes B$, but not $A \otimes B \multimap A$. In a sense, all of the propositions must be accounted for. This requirement can be circumvented by using the $\top$ connective on the right, which can "consume" an arbitrary set of assumptions on the left (see the $\top$R rule in Appendix A.

Of course, not everything is a resource. When reasoning about state, it is still necessary to represent knowledge. This knowledge may include rules for manipulating world state, among other things. This can be done in linear logic by using the exponential connective, !. A proposition under a ! can be used an arbitrary number of times (including zero). This can also be useful for representing an unlimited source of some resource.

Because of this distinction between propositions that can be used arbitrarily and propositions that must be used exactly once, a linear sequent takes the following form, with two contexts:

$$\Gamma\,;\Delta \Longrightarrow Q$$

The first context, Γ, is the *unrestricted context*; it contains those propositions under a !, which can be used arbitrarily often. The second context, Δ, is the *restricted context* or *linear context*; it contains the linear propositions, which must be used exactly once.

Linear logic also contains various other connectives that represent internal choice, external choice, and other concepts. The connectives are all defined by inference rules in the sequent calculus is given in Appendix A.

2.1 State Transition Systems

Most applications of linear logic use the concept of a state transition system, with a "current" state and a set of transition rules for moving from state to state. Linear logic represents such systems in a particularly elegant way. Propositions represent aspects of a state, and a full state description consists of a conjunction of zero or more propositions; for example, $\mathit{have}(\mathit{cake})$ represents having a piece of cake, $\mathit{eat}(\mathit{cake})$ represents having eaten a piece of cake, and $\mathit{have}(\mathit{cake}) \otimes \mathit{have}(\mathit{cake}) \otimes \mathit{eat}(\mathit{cake})$ represents having eaten one piece of cake and having two left. Transition rules are represented by a simple linear implication of the form $\mathit{state}_1 \multimap \mathit{state}_2$. Often, these rules will be under a !, so they can be invoked arbitrarily.

Most interesting questions about these state transition systems involve the possibility of reaching a certain state from some start state. A theorem representing this question takes the following form:

$$[!]\mathit{transition}_1 \multimap \cdots \multimap [!]\mathit{transition}_n \multimap \mathit{state}_0 \multimap \mathit{state}_f$$

The proof of such a theorem actually uses the transition rules to model moving from one state to another. A detailed account of how the process works in this

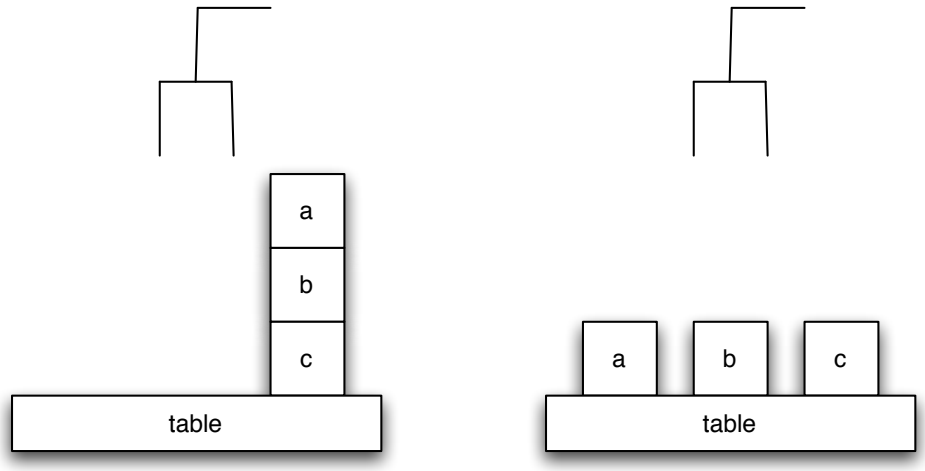

Figure 1: Two possible blocks world states

particular proof system is given in section 4. The resulting proof actually represents the steps taken to get from $state_0$ to $state_f$.

This property is incredibly useful in an application like verification of a security protocol. A theorem prover can be questioned about the possibility of reaching some undesirable state from the initial state. Should it find a proof, it not only tells you that your protocol is bad, it also provides a detailed account of how to get to the undesired state.

2.2 Blocks World

The motivating example for much of this work is a particular instance of a state transition system known as the *blocks world* planning problem. This problem consists of a table, a set of blocks, and a robotic hand that can pick up and put down a single block at a time. Each block can be directly on top of the table, directly on top of one other block, or held by the hand. One block directly on top of two blocks is not permitted, and two blocks directly on top of one block is likewise not permitted. Thus, blocks are arranged in a number of one-block-wide stacks on the table. Figure 1 diagrams two possible blocks world states. In addition, the hand can only pick up the top block in a stack and put a block down on top of a stack or on the table.

As mentioned above, blocks world states can be represented using propositions in linear logic. Each block not being held by the hand must be on either another block or on the table; these relationships are described by the propositions $on(x, y)$, meaning x is on y, and $ontab(x)$, meaning x is on the table. The state of the hand is described as $holds(x)$, if the hand is holding x, or as $empty$, if the hand is empty. In addition, $clear(x)$ is used to denote that x has no blocks on top of it, to avoid a complicated method of determining whether a block is at the top of the stack (and, thus, whether the hand can pick it up). The representation of a state is the conjunction of the applicable propositions, so the respective representations of the states in Figures 1(a) and 1(b) are $clear(a) \otimes on(a, b) \otimes on(b, c) \otimes ontab(c) \otimes empty$ and $clear(a) \otimes ontab(a) \otimes clear(b) \otimes ontab(b) \otimes clear(c) \otimes ontab(c) \otimes empty$.

Again, as mentioned above, transition rules are represented by linear implications with the starting state as the antecedent and the ending state as the consequent. The following four implications represent the possible transitions in the blocks world, where x and y represent any block:

$$! \forall x.\ clear(x) \otimes ontab(x) \otimes empty \multimap holds(x) \tag{1}$$

$$! \forall x.\ holds(x) \multimap clear(x) \otimes ontab(x) \otimes empty \tag{2}$$

$$! \forall x.\ \forall y.\ clear(x) \otimes on(x, y) \otimes empty \multimap holds(x) \otimes clear(y) \tag{3}$$

$$! \forall x.\ \forall y.\ holds(x) \otimes clear(y) \multimap clear(x) \otimes on(x, y) \otimes empty \tag{4}$$

Rule 1 represents picking a block up from the table, Rule 2 represents placing a block on the table, Rule 3 represents picking a block up from another block, and Rule 4 represents placing a block on another block.

To put it all together, the problem of going from the state in Figure 1(a) to the state in Figure 1(b) is represented by the following theorem:

$$\begin{aligned}
&(! \forall x.\ clear(x) \otimes ontab(x) \otimes empty \multimap holds(x)) \multimap \\
&(! \forall x.\ holds(x) \multimap clear(x) \otimes ontab(x) \otimes empty) \multimap \\
&(! \forall x.\ \forall y.\ clear(x) \otimes on(x, y) \otimes empty \multimap holds(x) \otimes clear(y)) \multimap \\
&(! \forall x.\ \forall y.\ holds(x) \otimes clear(y) - pclear(x) \otimes on(x, y) \otimes empty) \multimap \\
&clear(a) \otimes on(a, b) \otimes on(b, c) \otimes ontab(c) \otimes empty \multimap \\
&clear(a) \otimes ontab(a) \otimes clear(b) \otimes ontab(b) \otimes clear(c) \otimes ontab(c) \otimes empty
\end{aligned}$$

The proof for this theorem that is returned by the prover will be translatable into a plan for actually using the hand to move the blocks between the two states.

3 Focused Inverse Method

In general, proof search proceeds in one of two ways: backward, breaking a goal into subgoals and recursively attempting to prove those, or forward, starting from simple axioms and building more facts in the hope of generating a fact that matches the goal. Each of the two strategies has drawbacks. Backward search proceeds like a depth-first search of a graph, losing information if it pursues a dead end and has to backtrack. Forward search, on the other hand, never loses information, but it also generates many facts that are not along the path to the goal.

The inverse method [DV01] is one algorithm for forward-reasoning proof search. A basic implementation of the inverse method starts with an axiom of the form $p \Longrightarrow p$ for each atom p that appears in the goal theorem and appears both positively and negatively. Each of the axioms starts in the *set of support*, which is the database of known facts that are awaiting actions. Next, a set of rules is generated, with each rule being derived from the sequent calculus of the logic. The algorithm then proceeds with the OTTER loop, named for the OTTER theorem prover, which first implemented it [McC94]. The OTTER loop consists of the following steps:

1. Select a sequent from the set of support and move it to the *active set*
2. Apply derived rules to sequents in the active set to generate new sequents
3. Check whether each new sequent subsumes or is subsumed by other known sequents
4. Check if any new sequent subsumes the goal
5. Add new sequents to the set of support

There are, of course, many more considerations and details within each step, but the basic idea is in these five steps.

A simple example should clarify the algorithm. Suppose we are attempting to prove $A \multimap B \multimap A \otimes B$. The axioms are $\cdot\,; A \Longrightarrow A$ and $\cdot\,; B \Longrightarrow B$, and there are various rules such as the following:

$$\frac{\Gamma\,; \Delta_1, A \Longrightarrow A \quad \Gamma\,; \Delta_2, B \Longrightarrow B}{\Gamma\,; \Delta_1, \Delta_2, A, B \Longrightarrow A \otimes B} \qquad\qquad \frac{\Gamma\,; \Delta, B \Longrightarrow A \otimes B}{\Gamma\,; \Delta \Longrightarrow B \multimap A \otimes B}$$

Because there are no exponentials in this example, the unrestricted context will always be empty and will be omitted for the remainder of the example. At the beginning, the active set is empty, and the set of support contains the three axioms;

the algorithm then proceeds in the method stated above.

Iteration	Selected	Active	Generated	Set of support
0			$A \Longrightarrow A, B \Longrightarrow B$	
1	$A \Longrightarrow A$	$A \Longrightarrow A$	$B \Longrightarrow B$	
3	$B \Longrightarrow B$	$B \Longrightarrow B; A \Longrightarrow A$	$A, B \Longrightarrow A \otimes B$	$A, B \Longrightarrow A \otimes B$
4	$A, B \Longrightarrow A \otimes B$	$A, B \Longrightarrow A \otimes B; \ldots$	$A \Longrightarrow B \multimap A \otimes B$	$A \Longrightarrow B \multimap A \otimes B$
5	$A \Longrightarrow B \multimap A \otimes B$	$A \Longrightarrow B \multimap A \otimes B; \ldots$	$\cdot \Longrightarrow A \multimap B \multimap A \otimes B$	$\cdot \Longrightarrow A \multimap B \multimap A \otimes B$

The sequent generated in iteration 5 matches the goal we are trying to achieve, so the proof search terminates successfully. Of course, more complex examples will generate many more sequents, with multiple new sequents from each iteration. In addition, subsumption occurs in most reasonably complicated examples. Subsumption is when a stronger sequent makes a weaker sequent redundant; for example, $\forall x.\ f(x)$ is a much stronger statement than $f(a)$, so $\forall x.\ f(x)$ subsumes $f(a)$. If $\forall x.\ f(x)$ is generated and $f(a)$ is in the set of support, then $f(a)$ will be removed because it is no longer necessary.

3.1 Focusing

An important step to improve the efficiency of backward search was the idea of *focusing*, orginally developed for logic programming by Andreoli [And92]. A simplistic backward search has a huge number of choice points. For example, when considering the sequent $A \otimes B, C \otimes D, E \otimes F \Longrightarrow P \otimes Q$, the prover can proceed by decomposing any of the four conjunctions; if that particular route does not lead to success, the prover will then try decomposing another of the four and follow a path that is largely redundant with that of the first decomposition. The basic idea behind focusing is to perform such decompositions in a deterministic fashion, reducing the total number of choices the prover must make, thus reducing the size of the search space.

Consider an attempt to prove $A \otimes B \multimap C \multimap D$. An unfocused proof search would first decompose the top-level implication, getting the sequent $A \otimes B \Longrightarrow C \multimap D$. At this point, the prover could proceed by decomposing the $\otimes$ on the left or the $\multimap$ on the right. Suppose the prover decomposes the $\otimes$ first. When that branch of the search fails, the prover will then try composing the $\multimap$ first. A focusing prover treats the two decompositions as part of a single phase in which order is immaterial. Thus, only a single ordering will need to be considered. On a simple example such as this one, focusing cuts the amount of work required by a prover almost in half. It is easy to see how focusing can drastically cut down the work required for a more complex theorem with many more choice points.

The focusing sequent calculus uses four different sequent forms to ensure determinism. In these sequents, Ω is called the the active context and consists of formulas that are awaiting decomposition. A and Q are arbitrary formulas:

1. $\Gamma\,;\Delta \Longrightarrow Q$, a *neutral sequent*;
2. $\Gamma\,;\Delta\,;\Omega \Longrightarrow Q$, an *active sequent*;
3. $\Gamma\,;\Delta \gg Q$, a *right-focused sequent*; and
4. $\Gamma\,;\Delta\,;A \ll Q$, a *left-focused sequent*.

From a neutral sequent, some formula is selected to be under focus. The system then enters a *focused phase*, in which it decomposes the formula under focus as long as it is synchronous. The left-synchronous connectives are $\&$, $\top$, $\multimap$, and $\forall$. The left-synchronous connectives are $\otimes$, $\mathbf{1}$, $\oplus$, $\mathbf{0}$, !, and $\exists$. When an asynchronous connective is under focus, the systems enters an *active phase*, wherein all asynchronous connectives in the active context and on the right side are decomposed (left-synchronous connectives are right-asynchronous and vice versa); the order of decomposition in the active phase is unimportant. When the asynchronous decompositions are complete, the system reaches another neutral sequent, where another choice is made for a proposition to receive focus.

The full set of rules for the focusing sequent calculus for intuitionistic linear logic is reproduced in Appendix A.

3.2 Focusing the Inverse Method

While it was developed for backward search and inherently involves decomposing a goal formula to subgoals, focusing has also been applied to the inverse method [CP05b]. To generate the rules that are used to generate new facts, a focusing inverse method prover selects a proposition to receive focus and simulates a backward focusing phase on that proposition. The leaves of the simulated backward phase become the premises for a large-step rule. For example, focusing on the proposition $\forall x.\, \mathit{ontab}(x) \otimes \mathit{clear}(x) \otimes \mathit{empty} \multimap \mathit{holds}(x)$ results in the following rule, where a can be any block:

$$\frac{\Gamma\,;\Delta, \mathit{holds}(a) \Longrightarrow Q}{\Gamma\,;\Delta, \mathit{ontab}(a), \mathit{clear}(a), \mathit{empty} \Longrightarrow Q}$$

This is a fairly straightforward interpretation of the transition rule. It is possible to go from a state with a clear on the table and an empty hand to a state with

a in the hand; thus, if there is a path from the state of holding a to some state Q, then there is a path from the state with a clear on the table and an empty hand to that state Q (namely, invoking the transition and then following the given path). The interested reader can refer to Appendix B for a more thorough description of how focused phases work to generate large-step rules.

The use of focusing in the inverse method for linear logic is a relatively recent development. The present work examines and refines the proof system and implementation presented in [CP05a]. The full sequent calculus for the focusing system on which the implemented prover is based can be found in Appendix A.

4 Atomic Bias

An interesting property of the focusing sequent calculus is the way it treats atomic propositions under focus. When running a backward search, if an atomic proposition is under left focus, then the right side must be the same proposition; otherwise, that branch of the proof search fails. If the atomic proposition is under right focus, then focus is immediately lost. This is represented by the following two rules from the focusing sequent calculus:

$$\frac{}{\Gamma\,;\cdot\,;\,p \ll p}\ \text{rinit} \qquad\qquad \frac{\Gamma\,;\,\Delta\,;\,\cdot \Longrightarrow p}{\Gamma\,;\,\Delta \gg p}\ \text{rb}^*$$

There is a symmetrical and equally valid system that does the opposite. If an atomic proposition is under right focus, then the linear context must be a singleton containing the same proposition, or the branch fails. If the atomic proposition is under left focus, then focus is immediately lost. This is represented by the two alternative rules below:

$$\frac{}{\Gamma\,;\,p \gg p}\ \text{linit} \qquad\qquad \frac{\Gamma\,;\,\Delta\,;\,p \Longrightarrow R}{\Gamma\,;\,\Delta\,;\,p \ll R}\ \text{lb}^*$$

These two different systems are called *right-biased* and *left-biased*, respectively. Because of the domain for which focusing was originally developed (logic programming), the former behavior has always been historically chosen. However, the choice turns out to be nontrivial, but deserves much more careful consideration.

Despite the seemingly minor change in the system, the choice of bias has an extraordinary impact on the operation of proof search. This is especially apparent in linear logic encodings of state transition systems, because many branches in the

large-step rule derivations terminate with initial sequents. The previously shown rule that is derived from focusing on $\forall x.\ ontab(x) \otimes clear(x) \otimes empty \multimap holds(x)$ assumes that all atoms are left-biased. It is reproduced below, along with the rule that is derived under the assumption that all atoms are right-biased.

$$\frac{\Gamma\ ;\ \Delta, holds(a) \Longrightarrow Q}{\Gamma\ ;\ \Delta, ontab(a), clear(a), empty \Longrightarrow Q}$$

$$\frac{\Gamma\ ;\ \Delta_1 \Longrightarrow ontab(a) \quad \Gamma\ ;\ \Delta_2 \Longrightarrow clear(a) \quad \Gamma\ ;\ \Delta_3 \Longrightarrow empty}{\Gamma\ ;\ \Delta_1, \Delta_2, \Delta_3 \Longrightarrow holds(a)}$$

There is a very clear and elegant symmetry between these two derived rules. With left-biased atoms, the relevant changes in state occur on the let side of the sequent, and the rule works backward relative to the direction of the transition. With right-biased atoms, the rule works forward and operates on the right side of the sequents.

In addition to affecting the shapes of derived rules, atomic bias also changes the initial set of axioms used by the inverse method. In a non-focused inverse method, there is an axiom $p \Longrightarrow p$ for every p in the formula that appears both positively and negatively. In the focused inverse method, a left-biased atom appears only if it appears negatively directly under a right-synchronous connective ($\otimes, \mathbf{1}, !$), and a right-biased atom appears only if it appears positively directly under a left-synchronous connective ($\&, \top, \multimap$). These differences are once again illustrated clearly by the blocks world example. With right-biased atoms, there is an axiom for each of the five atoms (*on*, *ontab*, *clear*, *empty*, *holds*). With left-biased atoms, there are no such axioms; the only sequent initially in the set of support actually comes from a derived rule with zero premises.

It should be readily apparent from the large operational differences caused by atomic bias selection that there are likewise large differences in running times. Looking at the blocks world example in particular, the derived rules with a single premise are preferable to the derived rules with multiple premises. Those with many premises arbitrarily combine the contexts of multiple sequents, resulting in many useless sequents that do not correspond to actual blocks world states. The single-premise rules show a much clearer correlation with the actual state transitions in the blocks world problem. Thus, the left-biased system starts with the end state and works backward to other reachable states until it finds the start state. This functionality is not only conceptually cleaner but also reduces the number of sequents under consideration. Because there are fewer sequents to deal with, the clock time for the actual proof search should be greatly reduced.

Actual experimental results for several different classes of problems are shown in Section 6.1.

5 Filtering

Another interesting tactic for improving proof search efficiency comes not from purely practical, rather than theoretical, observations. In examining the list of sequents generated by the prover under consideration, one can find interesting contexts that include the both $holds(a)$ and $empty$ or three instances of on with a two-block world. Such sequents represent situations that are impossible in a particular blocks world.

Returning to our right-biased derived rule for $\forall x.\ ontab(x) \otimes clear(x) \otimes empty \multimap holds(x)$ (shown again below), it is clear how these strange contexts come about.

$$\frac{\Gamma\ ;\ \Delta_1 \Longrightarrow ontab(a) \quad \Gamma\ ;\ \Delta_2 \Longrightarrow clear(a) \quad \Gamma\ ;\ \Delta_3 \Longrightarrow empty}{\Gamma\ ;\ \Delta_1, \Delta_2, \Delta_3 \Longrightarrow holds(a)}$$

Rules such as this one combine three contexts arbitrarily, without regard to the result. Thus, impossible situations, such as having a hand that is both empty and holding a block, are considered by the proof system. A large number of generated spurious sequents contributes to the difference in real time of finding a proof for a blocks world problem in the left- and right-biased systems.

Unfortunately, there is no easy theoretical solution to the problem. Rather, it is possible to introduce hand-made filters that remove the spurious sequents from consideration when they are generated. This avoids the cost in time of considering sequents that do not correspond to "real" situations. Such filtering, however, is on theoretically shaky ground. An important consideration is that linear logic proofs of the encoding of a state transition problem, such as those of the blocks world, correspond to actual plans in the original domain. It is reasonable, then, to assume that a proof containing "impossible" sequents, which would translate to a plan containing impossible states, cannot exist, or that if such a proof exists, there is also a proof that does not use impossible sequents. For this reason, despite the lack of a formal proof, we contend that an implementation that includes filtering remains complete (that is, if a proof exists, it can be found even with the filter in place).

Filtering also has an interesting interaction with atomic bias. In the blocks world problems, for example, it is shown above how a system with right-biased atoms generates spurious sequents. Filtering is expected to be highly effective

when used with such a system. On the flip side, a system with left-biased atoms does not generate any impossible sequents. In this case, filtering requires computation time without providing any useful benefit. However, filtering is expected to represent a relatively insignificant portion of the total computation time, and should thus be a practical strategy overall. Of course, the computation time required by the filter depends on its complexity, so it may be that some filters do substantially more harm than good. Experimental results for filtering can be found in Section 6.2.

6 Experimental Results

6.1 Atomic Bias

The following experiments were originally performed for a related work by Chaudhuri, Pfenning, and the author [CPP06], and the problem descriptions and tables of results are adapted from that work. These experiments were run on a 3.4GHz Pentium 4 machine with 1MB L1 cache and 1GB main memory; the provers were compiled using MLTon version 20060213 using the default optimization flags. In the tables, iters is the number of iterations of the OTTER loop required to find a proof, gen is the number of sequents generated during the proof search, subs is the number of generated sequents that are subsumed, and time is the wall-clock time in seconds, including garbage collection time.

Stateful system encodings In these examples, we encoded the state transition rules for stateful systems such as a change machine, a blocks world problem with a fixed number of blocks, a few sample Petri nets. For the blocks world example, we also compared a version that uses the CLF monad [CPWW02] and one without.

	right-biased				**left-biased**			
name	iters	gen	subs	time	iters	gen	subs	time
`blocks`	20	43	18	0.001	12	84	61	0.001
`blocks-clf`	27	65	26	0.002	5	24	7	**<0.001**
`change`	16	22	7	0.001	11	20	6	0.001
`petri-1`	23	38	23	**0.001**	284	1099	921	0.062
`petri-2`	57	133	105	**0.003**	393	1654	1433	0.068

Graph exploration algorithms In these examples we encode the algorithm for exploring graph for calculating Euler or Hamiltonian tours. The problems have an

equal balance of proofs (i.e., a tour exists) and refutations (i.e., no tour exists).

	right-biased				left-biased			
name	iters	gen	subs	time	iters	gen	subs	time
`euler-1`	6291	11853	5565	9.010	6291	11853	5565	**8.570**
`euler-2`	15640	34329	18689	152.12	15640	34329	18689	**145.9**
`euler-3`	64360	159194	94834	3043.35	64360	159194	94834	**2938.55**
`hamilton`	708	911	185	0.11	165	178	0	**<0.001**

The Euler tour computation uses a symmetric algorithm, so both backward and forward chaining generate the same facts, though, interestingly, a left-biased search performs slightly better than the right-biased system. For the Hamiltonian tour examples, the left-biased search is vastly superior.

Affine logic encoding Linearity is often too stringent a requirement for situations where we simply need *affine* logic, i.e., where every hypothesis is consumed *at most* once. Affine logic can be embedded into linear logic by translating every affine arrow $A \to B$ as either $A \multimap B \otimes \top$ or $A \,\&\, \mathbf{1} \multimap \top$. Of course, one might select complex encodings; for example choosing $A \,\&\, !(\mathbf{0} \multimap X) \multimap B$ (for some arbitrary fresh proposition X) instead of $A \,\&\, \mathbf{1} \multimap B$. Even though the two translations are equivalent, the prover performs dismally on the former.

	right-biased				left-biased			
encoding	iters	gen	subs	time	iters	gen	subs	time
$A \multimap B \otimes \top$	38	108	73	0.003	34	107	73	**0.002**
$A \,\&\, \mathbf{1} \multimap B$	252	1103	828	0.098	62	229	126	**0.019**
$A \,\&\, !(\mathbf{0} \multimap X) \multimap B$	264	7099	6793	2.028	235	841	578	**0.042**

Quantified Boolean formulas In these examples we used two variants of the algorithm from [**?**] for encoding QBFs in linear logic. The first variant uses exponentials to encode reusable "copy" rules for the input branching in the computed circuit; this variant performs very well in practice, so the table below collates the results of all the example QBFs in one entry. The second variant is completely non-exponential and explicitly copies the signals in the circuit as needed. This variant produces problems that are considerably harder, so we have divided the problems in three sets in increasing order of complexity.

	right-biased				left-biased			
encodings	iters	gen	subs	time	iters	gen	subs	time
`qbf-exp`	1508	1722	140	**0.13**	7948	17610	9590	2.69
`qbf-nonexp-1`	1457	5590	4067	**0.54**	1581	4352	2612	0.58
`qbf-nonexp-2`	15267	517551	502174	368.92	9469	49777	37716	**29.55**
`qbf-nonexp-3`	28556	990196	961494	2807.64	21233	89542	115917	**308.24**

For these examples, when the number of iterations is low (i.e., the problems are simple), the right-biased search appears to perform better than the left-biased system. However, as the problems get harder, the left-biased system becomes dominant.

First-order stateful systems The first experiments were with first-order encodings of various stateful systems. We selected a first-order blocks world encoding (both with and without the CLF monad), Dijkstra's Urn Game, and an AI planning problem for a certain board game. The left-biased system performs consistently better than the right-biased system for these problems.

	right-biased				left-biased			
problem	iters	gen	subs	time	iters	gen	subs	time
`blocks`	58	530	396	0.15	32	484	421	**0.04**
`blocks-clf`	81	872	515	0.33	19	102	87	**0.007**
`urn`	37	91	34	0.30	17	73	69	**0.14**
`board`	437	8777	3923	4.08	208	6621	2191	**1.10**

Purely intuitionistic problems The prover was tested on some problems drawn from the SICS benchmark [SFH92]. These intuitionistic problems were translated into linear logic in two different ways– the first uses Girard's original encoding of classical logic in classical linear logic where every subformula is affixed with the exponential, and the second is a focus-preserving encoding as described in [CP05b]. $\times$ denotes inability to find a proof within one hour.

	right-biased				left-biased			
problem	iters	gen	subs	time	iters	gen	subs	time
SICS1-gir	451	2435	1743	1.64	461	3622	2727	0.78
SICS1-foc	70	457	392	0.07	81	620	519	0.05
SICS2-gir	3794	20489	14666	13.8	4326	33990	25487	7.32
SICS2-foc	612	3917	3361	0.59	770	5841	4878	0.47
SICS3-gir	26198	1414779	1012607	952.89	16156	1269440	951897	273.39
SICS3-foc	4222	27074	23308	41.37	2875	21831	18712	29.12
SICS4-gir	×	×	×	×	×	×	×	×
SICS4-foc	11121	71320	61309	108.98	7680	58523	49992	77.80

Horn examples from TPTP The final test consisted of 20 non-trivial Horn problems selected from the TPTP version 3.1.1 [SS98]. The selection of problems was not systematic but was not constrained to any particular section of the TPTP. Translation was performed as described in [CPP06]. The list of selected problems can be found from Chaudhuri's web-page.[1]

right-biased				left-biased			
iters	gen	subs	time	iters	gen	subs	time
5170	331201	302110	**487.22**	6620	741560	553903	672.44

6.2 Filtering

The following experiments used simple hand-coded constraints to filter sequents. The constraints limit the number of times a resource can occur in the linear context of a sequent. If the constraint is not met, then a sequent is removed from consideration. An example constraint for a blocks world problem dealing with two blocks is $on \leq 2 \wedge empty + holds \leq 1$; with only two blocks, the atom *on* can only occur twice, and only one of *empty* or *holds* can occur.

These experiments were run on a 1GHz PowerPC G4 machine with 1GB main memory; the provers were compiled using MLTon version 20051202 using the default optimization flags. In the tables, iters is the number of iterations of the OTTER loop required to find a proof, gen is the number of sequents generated during the proof search, filt is the number of sequents filtered by the specified constraint, subs is the number of generated sequents that are subsumed, and time is the wall-clock time in seconds, including garbage collection time.

[1] `http://www.cs.cmu.edu/~kaustuv/papers/ijcar06`

	right-biased					left-biased				
problem	iters	gen	filt	subs	time	iters	gen	filt	subs	time
`blocks-filt`	276	9982	6093	3615	13.835	152	14892	462	14351	**11.194**
`blocks-nofilt`	×	×	×	×	×	214	24727	n/a	24582	**93.682**

7 Conclusion

As was expected, the left-biased system significantly outperforms the right-biased system for the blocks world problems. The same holds true for other state transition systems, such as encodings of a coin-changing machine. Interestingly, there are also classes of problems for which the right-biased system performs better than the left-biased system, so the left-biased system is not universally superior.

In addition, filtering successfully increased speed for both the left- and right-biased system. However, no gain was expected for the left-biased case; a cursory examination of the prover's output indicates that the implementation may not exactly match the theoretical operation of the system. Nonetheless, filtering showed particularly impressive results for the right-biased system; the prover went from being unable to prove the theorems in under an hour (vs. about 90 seconds for the left-biased system) to taking a mere 24 percent longer.

Unfortunately, it is still necessary to construct filters and select bias by hand. We have shown here that, for state transition systems like the blocks world, left bias is the clear winner. When considering other classes of problems, the derived rules, axioms, and/or actual sample results are necessary to determine the proper bias. The fact that atomic bias can be ascribed to individual atoms rather than just to the system as a whole also complicates the matter. Instead of just 2 options, there are 2^n for a theorem with n distinct atoms.

Thus, the most interesting future work arising from this is finding a better theoretical grounding for the practical concerns seen herein. First, it is desirable to have a proof of the completeness of a filtering system. Currently, it may be possible to prove that the particular filters used for the blocks world problem do not affect the completeness of the system. To generalize the completeness proof to all problems, it will likely be necessary to find some automatic way of generating a filter from a problem. Of course, this also has the desired effect of eliminating the work required in writing a filter by hand. In the same vein, it would be extremely useful to have an algorithm for automatically selecting bias on a per-atom basis. This would again greatly reduce the amount of human time necessary to prepare for the most efficient proof search.

One other interesting possible method for increasing proof search efficiency is the use of heuristics to determine which fact to select from the set of support. The order in which facts are selected can have a huge impact on how quickly a proof is found. Proof search is similar to traversing a tree with high degree; the difference in number of nodes visited is hugely different between finding a direct path and a breadth-first search. Using a smart heuristic will allow a prover to perform much more closely to finding a direct path than to breadth-first search. This advancement will further increase the prover's speed and make it an even more viable option for reasoning about stateful systems.

Acknowledgements

I would like to thank my advisor, Frank Pfenning, for driving my initial interest in and teaching me about logics, proof theory, and automated deduction, as well as for guiding this project as its purpose evolved and for insightful discussions of the issues herein. I would also like to thank Kaustuv Chaudhuri for providing the theoretical foundation and implementation that form a basis for this work, as well as for aiding my understanding of linear logic in general and this proof system in particular and for helping me with various problems related to the prover implementation. Finally, I extend my gratitude to everyone who listened to me as I attempted to explain linear logic and focusing and helped me to figure out the best way to do so.

References

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[CP05a] Kaustuv Chaudhuri and Frank Pfenning. A focusing inverse method theorem prover for first-order linear logic. In *Proceedings of CADE-20*, pages 69–83, Tallinn, Estonia, July 2005. Springer-Verlag LNAI-3632.

[CP05b] Kaustuv Chaudhuri and Frank Pfenning. Focusing the inverse method for linear logic. In Luke Ong, editor, *Proceedings of CSL 2005*, pages 200–215, Oxford, UK, August 2005. Springer-Verlag LNCS-3634.

[CPP06] Kaustuv Chaudhuri, Frank Pfenning, and Greg Price. A logical characterization of forward and backward chaining in the inverse method. In *Proceedings of IJCAR 2006*, Seattle, WA, USA, August 2006.

[CPWW02] Iliano Cervesato, Frank Pfenning, David Walker, and Kevin Watkins. A concurrent logical framework I & II. Technical Report CMU-CS-02-101 and 102, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.

[DV01] Anatoli Degtyarev and Andrei Voronkov. The inverse method. In James Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 179–272. MIT Press, September 2001.

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

[GP06] Deepak Garg and Frank Pfenning. Non-interference in constructive authorization logic. In *Proceedings of CSFW 19*, Venice, Italy, July 2006.

[McC94] William McCune. OTTER 3.0 Reference manual and guide. Technical Report ANL-94/6, Argonne National Laboratory, Argonne, IL, 1994.

[SFH92] Dan Sahlin, Torkel Franzén, and Seif Haridi. An intuitionistic predicate logic theorem prover. *Journal of Logic and Computation*, 2(5):619–656, 1992.

[SS98] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

A Focusing Calculus

$\boxed{\Gamma\,;\,\Delta \gg A \qquad \text{right-focal}}$

$$\frac{q \text{ left-biased}}{\Gamma\,;\,q \gg q}\ \text{linit} \qquad \frac{}{\Gamma\,;\,\cdot \gg \mathbf{1}}\ \mathbf{1}R \qquad \frac{\Gamma\,;\,\Delta_1 \gg A \quad \Gamma\,;\,\Delta_2 \gg B}{\Gamma\,;\,\Delta_1, \Delta_2 \gg A \otimes B}\ \otimes R$$

$$\frac{\Gamma\,;\,\Delta \gg A_i}{\Gamma\,;\,\Delta \gg A_1 \oplus A_2}\ \oplus R_i \qquad \frac{\Gamma\,;\,\Delta \gg [t/x]A}{\Gamma\,;\,\Delta \gg \exists x.\, A}\ \exists R \qquad \frac{\Gamma\,;\,\cdot\,;\,\cdot \Longrightarrow A}{\Gamma\,;\,\cdot \gg\, !\,A}\ !\,R$$

$\boxed{\Gamma\,;\Delta\,;A \ll Q \qquad \text{left-focal}}$

$$\frac{p \text{ right-biased}}{\Gamma\,;\cdot\,;p \ll p}\ \text{rinit} \qquad \frac{\Gamma\,;\Delta\,;A_i \ll Q}{\Gamma\,;\Delta\,;A_1 \,\&\, A_2 \ll Q}\ \&L_i$$

$$\frac{\Gamma\,;\Delta_1\,;B \ll Q \quad \Gamma\,;\Delta_2 \gg A}{\Gamma\,;\Delta_1,\Delta_2\,;A \multimap B \ll Q}\ \multimap L \qquad \frac{\Gamma\,;\Delta\,;[t/x]A \ll Q}{\Gamma\,;\Delta\,;\forall x.\,A \ll Q}\ \forall L$$

$\boxed{\text{focus}}$

$$\frac{\Gamma\,;\Delta\,;P \ll Q}{\Gamma\,;\Delta,P \Longrightarrow Q}\ \text{lf} \qquad \frac{\Gamma\,;\Delta\,;p \ll Q \quad p \text{ right-biased}}{\Gamma\,;\Delta,p \Longrightarrow Q}\ \text{lf}^*$$

$$\frac{\Gamma\,;\Delta \gg Q}{\Gamma\,;\Delta \Longrightarrow Q}\ \text{rf} \qquad \frac{\Gamma\,;\Delta \gg q \quad q \text{ left-biased}}{\Gamma\,;\Delta \Longrightarrow q}\ \text{rf}^* \qquad \frac{\Gamma,A\,;\Delta\,;A \ll Q}{\Gamma,A\,;\Delta \Longrightarrow Q}\ \text{copy}$$

$\boxed{\Gamma\,;\Delta\,;\Omega \Longrightarrow R\,;\cdot \qquad \text{right-active}}$

$$\frac{\Gamma\,;\Delta\,;\Omega \Longrightarrow A\,;\cdot \quad \Gamma\,;\Delta\,;\Omega \Longrightarrow B\,;\cdot}{\Gamma\,;\Delta\,;\Omega \Longrightarrow A \,\&\, B\,;\cdot}\ \&R \qquad \frac{}{\Gamma\,;\Delta\,;\Omega \Longrightarrow \top\,;\cdot}\ \top R$$

$$\frac{\Gamma\,;\Delta\,;\Omega \cdot A \Longrightarrow B\,;\cdot}{\Gamma\,;\Delta\,;\Omega \Longrightarrow A \multimap B\,;\cdot}\ \multimap R \qquad \frac{\Gamma\,;\Delta\,;\Omega \Longrightarrow [a/x]A\,;\cdot}{\Gamma\,;\Delta\,;\Omega \Longrightarrow \forall x.\,A\,;\cdot}\ \forall R^a$$

$$\frac{\Gamma\,;\Delta\,;\Omega \Longrightarrow \cdot\,;Q}{\Gamma\,;\Delta\,;\Omega \Longrightarrow Q\,;\cdot}\ \text{ract} \qquad \frac{\Gamma\,;\Delta\,;\Omega \Longrightarrow \cdot\,;p \quad p \text{ right biased}}{\Gamma\,;\Delta\,;\Omega \Longrightarrow p\,;\cdot}\ \text{ract}^*$$

$\boxed{\Gamma\,;\Delta\,;\Omega \cdot L \cdot \Omega' \Longrightarrow \gamma \qquad \text{left-active}}$

$$\frac{\Gamma\,;\Delta\,;\Omega \cdot A \cdot B \cdot \Omega' \Longrightarrow \gamma}{\Gamma\,;\Delta\,;\Omega \cdot A \otimes B \cdot \Omega' \Longrightarrow \gamma}\ \otimes L \qquad \frac{\Gamma\,;\Delta\,;\Omega \cdot \Omega' \Longrightarrow \gamma}{\Gamma\,;\Delta\,;\Omega \cdot \mathbf{1} \cdot \Omega' \Longrightarrow \gamma}\ \mathbf{1}L$$

$$\frac{\Gamma\,;\Delta\,;\Omega \cdot A \cdot \Omega' \Longrightarrow Q \quad \Gamma\,;\Delta\,;\Omega \cdot B \cdot \Omega' \Longrightarrow \gamma}{\Gamma\,;\Delta\,;\Omega \cdot A \oplus B \cdot \Omega' \Longrightarrow \gamma}\ \oplus L \qquad \frac{}{\Gamma\,;\Delta\,;\Omega \cdot \mathbf{0} \cdot \Omega' \Longrightarrow \gamma}\ \mathbf{0}L$$

$$\frac{\Gamma\,;\Delta\,;\Omega \cdot [a/x]A \cdot \Omega' \Longrightarrow \gamma}{\Gamma\,;\Delta\,;\Omega \cdot \exists x.\,A \cdot \Omega' \Longrightarrow \gamma}\ \exists L^a \qquad \frac{\Gamma,A\,;\Delta\,;\Omega \cdot \Omega' \Longrightarrow \gamma}{\Gamma\,;\Delta\,;\Omega \cdot\, !\,A \cdot \Omega' \Longrightarrow \gamma}\ !\,L$$

$$\frac{\Gamma\,;\Delta,P\,;\Omega \cdot \Omega' \Longrightarrow \gamma}{\Gamma\,;\Delta\,;\Omega \cdot P \cdot \Omega' \Longrightarrow \gamma}\ \text{lact} \qquad \frac{\Gamma\,;\Delta,q\,;\Omega \cdot \Omega' \Longrightarrow \gamma \quad q \text{ left-biased}}{\Gamma\,;\Delta\,;\Omega \cdot q \cdot \Omega' \Longrightarrow \gamma}\ \text{lact}^*$$

$\boxed{\text{blur}}$

$$\frac{\Gamma\,;\Delta\,;L \Longrightarrow \cdot\,;Q}{\Gamma\,;\Delta\,;L \ll Q}\ \text{lb} \qquad \frac{\Gamma\,;\Delta,q\,;\cdot \Longrightarrow \cdot\,;Q \quad q \text{ left-biased}}{\Gamma\,;\Delta\,;q \ll Q}\ \text{lb}^*$$

$$\frac{\Gamma\,;\Delta\,;\cdot \Longrightarrow R\,;\cdot}{\Gamma\,;\Delta \gg R}\ \text{rb} \qquad \frac{\Gamma\,;\Delta\,;\cdot \Longrightarrow p\,;\cdot \quad p \text{ right-biased}}{\Gamma\,;\Delta \gg p}\ \text{rb}^*$$

B Derived Rules

As described in Section 3.2, large-step rules for the inverse method are generated by focusing on a subformula and decomposing it. The mini-proof search actually undergoes one focused phase and one active phase. When it reaches a neutral sequent, the search terminates. Thus, the leaves of this backward search are initial or neutral sequents. The neutral sequents are then treated as the premises of a large-step rule. The intermediate steps in the generation of a large-step rule are unimportant, thus reducing the search space for the inverse method. An example of a simulated phase appears below for a blocks world transition rule:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma\,;\,\Delta_1, holds(a) \Longrightarrow \gamma}{\Gamma\,;\,\Delta_1, holds(a)\,;\,\cdot \Longrightarrow \gamma}}{\Gamma\,;\,\Delta_1\,;\,holds(a) \Longrightarrow \gamma}}{\Gamma\,;\,\Delta_1\,;\,holds(a) \ll \gamma} \qquad \dfrac{}{\Gamma\,;\,\Delta_2 \gg ontab(a)} \qquad \dfrac{}{\Gamma\,;\,\Delta_3 \gg clear(a)} \quad \dfrac{}{\Gamma\,;\,\Delta_4 \gg empty}}{\Gamma\,;\,\Delta_3, \Delta_4 \gg clear(a) \otimes empty}}{\Gamma\,;\,\Delta_2, \Delta_3, \Delta_4 \gg ontab(a) \otimes clear(a) \otimes empty}}{\Gamma\,;\,\Delta_1, \Delta_2, \Delta_3, \Delta_4\,;\, ontab(a) \otimes clear(a) \otimes empty \multimap holds(a) \ll \gamma}}{\Gamma\,;\,\Delta_1, \Delta_2, \Delta_3, \Delta_4\,;\, ontab(a) \otimes clear(a) \otimes empty \multimap holds(a) \ll \gamma}}{\Gamma\,;\,\Delta_1, \Delta_2, \Delta_3, \Delta_4\,;\, \forall x.\, ontab(x) \otimes clear(x) \otimes empty \multimap holds(x) \ll \gamma}}{\Gamma = \Gamma', \forall x.\, ontab(x) \otimes clear(x) \otimes empty \multimap holds(x)\,;\,\Delta_1, \Delta_2, \Delta_3, \Delta_4 \Longrightarrow \gamma}$$

In order for this proof to succeed, Δ_2 must be the singleton $ontab(a)$. Otherwise, $\Gamma\,;\,\Delta_2 \gg ontab(a)$ is not a valid initial sequent, and this branch of the proof search immediately fails. Likewise, Δ_3 must be the singleton $clear(a)$, and Δ_4 must be $empty$. Thus, the generated rule is

$$\dfrac{\Gamma\,;\,\Delta_1, holds(a) \Longrightarrow Q}{\Gamma\,;\,\Delta_1, ontab(a), clear(a), empty \Longrightarrow Q}$$

The focusing phase shown above is under left bias; for the same formula in a right-biased system, the backward phase appears as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma\,;\,\Delta_1, holds(a) \ll \gamma}{\Gamma\,;\,\Delta_1\,;\,holds(a) \ll \gamma} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma\,;\,\Delta_2 \Longrightarrow ontab(a)}{\Gamma\,;\,\Delta_2\,;\,\cdot \Longrightarrow \cdot\,;\,ontab(a)}}{\Gamma\,;\,\Delta_2\,;\,\cdot \Longrightarrow ontab(a)\,;\,\cdot}}{\Gamma\,;\,\Delta_2 \gg ontab(a)} \qquad \dfrac{\dfrac{\dfrac{\Gamma\,;\,\Delta_3 \Longrightarrow clear(a)}{\Gamma\,;\,\Delta_3\,;\,\cdot \Longrightarrow \cdot\,;\,clear(a)}}{\Gamma\,;\,\Delta_3\,;\,\cdot \Longrightarrow clear(a)\,;\,\cdot}}{\Gamma\,;\,\Delta_3 \gg clear(a)} \quad \dfrac{\dfrac{\dfrac{\Gamma\,;\,\Delta_4 \Longrightarrow empty}{\Gamma\,;\,\Delta_4\,;\,\cdot \Longrightarrow \cdot\,;\,empty}}{\Gamma\,;\,\Delta_4\,;\,\cdot \Longrightarrow empty\,;\,\cdot}}{\Gamma\,;\,\Delta_4 \gg empty}}{\Gamma\,;\,\Delta_3, \Delta_4 \gg clear(a) \otimes empty}}{\Gamma\,;\,\Delta_2, \Delta_3, \Delta_4 \gg ontab(a) \otimes clear(a) \otimes empty}}{\Gamma\,;\,\Delta_1, \Delta_2, \Delta_3, \Delta_4\,;\, ontab(a) \otimes clear(a) \otimes empty \multimap holds(a) \ll \gamma}}{\Gamma\,;\,\Delta_1, \Delta_2, \Delta_3, \Delta_4\,;\, ontab(a) \otimes clear(a) \otimes empty \multimap holds(a) \ll \gamma}}{\Gamma\,;\,\Delta_1, \Delta_2, \Delta_3, \Delta_4\,;\, \forall x.\, ontab(x) \otimes clear(x) \otimes empty \multimap holds(x) \ll \gamma}}{\Gamma = \Gamma', \forall x.\, ontab(x) \otimes clear(x) \otimes empty \multimap holds(x)\,;\,\Delta_1, \Delta_2, \Delta_3, \Delta_4 \Longrightarrow \gamma}$$

In order for this proof to succeed, $\Gamma\,;\,\Delta_1\,;\,holds(a) \ll \gamma$ must be a valid initial sequent, so Δ_1 must be empty, and γ must be $holds(a)$. Thus, the derived rule is the following:

$$\frac{\Gamma\,;\,\Delta_2 \Longrightarrow ontab(a) \quad \Gamma\,;\,\Delta_3 \Longrightarrow clear(a) \quad \Gamma\,;\,\Delta_4 \Longrightarrow empty}{\Gamma\,;\,\Delta_2, \Delta_3, \Delta_4 \Longrightarrow holds(a)}$$