Avoiding Duplicate Proofs with the Foothold Refinement

Bruce Spencer
Faculty of Computer Science
University of New Brunswick
P. O. Box 4400
Fredericton, New Brunswick, Canada
E3B 5A3
bspencer@unb.ca

Abstract

Wos has identified the problem of recomputing redundant information in the general setting of automated reasoning. We consider this problem in the setting of logic programming where we are given a formula and a goal and asked to find the instances of the goal that follow from the formula. We can use a proof procedure to find the result. The procedure exhibits redundancy when it finds two results such that one either duplicates or is more specific than the other.

We introduce the foothold format, a refinement of linear resolution that admits fewer duplicate proofs than Loveland's popular MESON format. The duplicates arise when reasoning by cases. It leads to proof procedures that compute fewer duplicate substitutions. In some of our examples all duplication is eliminated. The foothold refinement depends on a simple condition that can be checked quickly, and can detect redundancy before the proof is completely generated. This is important from a practical point of view, since the earlier redundancy is detected, the more unnecessary work can be avoided. For some examples the speedup is exponential. We show that the elimination of redundancy also applies to SLI resolution, a procedure for processing disjunctive logic programs.

1 Introduction

Wos [11] asks

> What strategy can be employed to deter a reasoning program from deducing a clause already retained, or from deducing a clause that is a proper instance of a clause already retained?

We consider the setting of first order logic without equality where we are given a goal literal and a set of clauses and we wish to know which instances of the literal follow from the clauses. To do this we use a proof procedure to compute substitutions for the variables in the literal. Prolog is a special case of this setting where all the clauses are Horn. A redundant substitution duplicates or is an instance of another substitution.

This paper presents the foothold format for linear resolution proofs that admits fewer duplicate proofs than MESON[3], a well-known proof format. The duplication we avoid arises in MESON when reasoning is done by cases. Reasoning by cases is required to handle clauses with more than one positive disjunct, clauses that do not fall into the Horn subset. Thus this type of duplication does not arise in Prolog, but it does arise in non-Horn extensions to Prolog, such as PTTP [10].

For readers familiar with Model Elimination (ME)[3], we avoid the duplication that can arise when the reduction operation is applied. Given an ME proof that includes a reduction between a goal and its complementary ancestor through a sequence of intervening ancestors, there must exist another ME proof which is identical except that the intervening ancestors are the complements of those from the first proof and these ancestors appear in reverse order (Lemma 2). To avoid the duplicate, we apply numerical labels to literals in each clause according to an arbitrary fixed initial ordering. Then a simple numerical condition based on these labels is imposed on the ME reduction operation. The condition allows only one of these two reductions to succeed. Thus one proof is avoided. By selecting the right arbitrary initial ordering we will accept the first proof and reject the second.

This paper builds upon the original presentation of the foothold format[8] with a more precise definition and more rigorous proofs. Since the original foothold paper we have proposed a different restriction on ME reduction[9]. Plaisted's positive refinement seems related [6], and other restrictions of resolution that avoid searching have been proposed [7].

2 Background: The MESON Proof Format

In this section we introduce negated ancestor proof graphs, which is a new definition for Loveland's MESON proof format. The new definition makes it convenient to add the foothold condition which eliminates certain redundant proofs.

2.1 Negated Ancestor Proof Graphs

Linear resolution [1, 3] can be used in the setting of first order logic to generate the instances for a goal literal by building a proof of the goal from the clauses, and applying the resulting substitutions to the goal. We consider one form of linear resolution, based on Loveland's MESON format, and we introduce a new definition of this format. Given a goal and a set of clauses we show that the goal follows from the clauses if and only if there is a proof graph of the goal that is built from (some of) the clauses.

We begin with two preliminary definitions.

Definition 1 If g is a literal then let $\overline{g}$ be the complement of g. That is, the overbar function adds the $\neg$ connective if it is not present in g, and removes it if it is.

Definition 2 For a clause $g_1 \vee \ldots \vee g_n$ there are n **contrapositive rules** as follows:

$$\begin{array}{l} \forall i = 1, \ldots, n \\ \qquad\qquad g_i \;\leftarrow\; \overline{g}_1 \;\; \cdots \;\; \overline{g}_{i-1} \;\; \overline{g}_{i+1} \;\; \cdots \;\; \overline{g}_n \end{array}$$

Informally, to construct a negated ancestor proof graph of a goal g, begin by constructing a node that contains g. This node will be the root of the graph. Given a partial negated ancestor proof graph, and a goal node containing the literal h (initially h is g) there are two ways to complete the graph. Case a is to find a node already in the graph containing $\overline{h}$ which is connected to this node by tree edges, and introduce a back edge from the goal node to this ancestor. Case b is to find a contrapositive rule with h on the left hand side, make a new node for each member of the right hand side, and introduce a tree edge from the goal node to each new node. Recursively prove each new node.

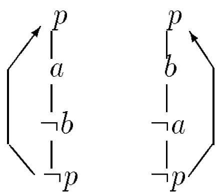


Figure 1: Negated Ancestor Proof Graphs

For example, consider the clauses

$$\begin{array}{ccc} p & \vee & \neg a \\ p & \vee & \neg b \\ a & \vee & b \end{array}$$

and the goal p. The contrapositive rules are

$$\begin{array}{cc} p \leftarrow a & \neg a \leftarrow \neg p \\ p \leftarrow b & \neg b \leftarrow \neg p \\ a \leftarrow \neg b & b \leftarrow \neg a \end{array}$$

There are two negated ancestor proof graphs of p, as shown in Figure 1. As in the conventional presentation of trees, the root is at the top, child nodes are lower than parent nodes, and tree edges have no arrows. Our back edges point upward.

Now we can formally present negated ancestor proof graphs. The definition depends on an auxilliary definition, NA1 proof graphs, which differ from negated ancestor proof graphs in that they have an additional set of ancestors that may be used to satisfy some goals.

Definition 3 Let P be a set of propositional clauses, and g a literal.

3.1 An NA1 proof graph of a node G containing the ancestor path A is a directed graph (V, E). $E = T \cup B$, where T is the set of tree edges, B is the set of back edges, T and B are disjoint and (V, T) is a tree. Each node in V contains a single literal. A is a sequence of nodes that represents a path of tree edges, which are the ancestors of G.[1] Let g be the literal in G. One of the two following conditions must be met:

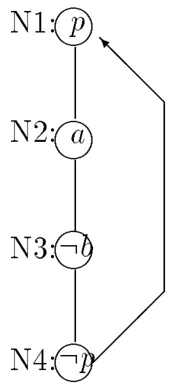
[1] G is *not* the root of the NA1 graph; rather, the first member of A is.

(case a) There exists a node N in A such that N contains the literal $\overline{g}$ and there exists a back edge (G, N) from G to N in the NA1 proof graph.

(case b) There exists a contrapositive rule $g \leftarrow g_1 \ldots g_n$ from some clause in P and for each $i = 1, \ldots, n$ there exists a node G_i containing g_i, a tree edge (G, G_i) and an NA1 proof graph of G_i containing the ancestor path (A, G).

3.2 A **negated ancestor proof graph** of g is an NA1 proof graph of G that contains the empty ancestor path.

To illustrate the definition we start with one of the proof graphs in Figure 1. We show it again, with labels on the nodes.



We will illustrate the definition by showing how this graph satisfies the conditions. It is a negative ancestor proof graph of p if it is an NA1 proof graph of $N1$ with the empty ancestor path. So we check $N1$ against Definition 3.1 with the ancestor path A set to (). Case a cannot hold since there is no member of A, so we try case b. Choosing the rule $p \leftarrow a$, we note that there is a node $N2$ containing a, and a tree edge to from $N1$ to $N2$. If we can satisfy ourselves that there is an NA1 proof of $N2$ containing the ancestor path $(N1)$ we will be done. Apply Definition 3.1 again. Case b applies, with the rule $a \leftarrow \neg b$, and there is a node $N3$ containing $\neg b$ and a tree edge from $N2$ to $N3$. We now must check that there is an NA1 proof of $N3$ with the ancestor path $(N1, N2)$. Again case b of the definition applies and, using the clause $\neg b \leftarrow \neg p$ we see that there is a node $N4$, which contains $\neg p$ and a tree edge from $N3$ to $N4$. Finally we ask if there is an NA1 proof graph of $N4$ containing the ancestor path $(N1, N2, N3)$. This time case a applies since there the node $N1$ in A contains p which is the complement of $\neg p$. Therefore we are satisfied that there is an NA1 proof graph of $N4$, which allows us to conclude there is an NA1 proof graph of $N3$, and so forth for $N2$ and $N1$. We conclude that this is a negated ancestor proof graph of p.

The part of the MESON format we have extracted preserves the important properties of soundness and completeness.

Theorem 1 Let P be a consistent set of propositional clauses, and g a literal. $P \models g$ if and only if there exists a negative ancestor proof graph of g using clauses in P.

Proofs for the theorems in this paper are found in Appendix A.

2.2 First Order Proofs

We will assume that the reader is familiar with first order logic, as presented in, for example [1].

Suppose we are given a set P of clauses and a literal g and we want to find the instances of g that follow from P. Procedures that build first order MESON proofs can also build substitutions for the variables in g. For example if $P = \{g(a)\}$ then the proof of $g(x)$ returns the substitution $x := a$. There may also exist indefinite instances of the goal, as defined by a disjunctive set of substitutions. For example if $P = \{g(a) \vee g(b)\}$ then the proof of $g(x)$ should return $x := a \vee x := b$. Green [2] provides a method for computing disjunctive answers. Stickel [10] describes how disjunctive answers can be computed within the MESON format:

> Indefinite answers can be obtained by solving the query with its negation included among the axioms, and examining the proof to find the query's instantiations.

Completeness of the MESON format extends from the propositional case to the first order case [3]. To see why, recall Herbrand's theorem which states: a set of clauses is unsatisfiable if and only if there is a finite unsatisfiable set of ground instances of these clauses. If $P \models g$ then $P \cup \{\overline{g}\}$ is unsatisfiable, so there is an unsatisfiable set $P' \cup \{\overline{g}_1, \ldots, \overline{g}_n\}$ where P' is made up of ground instances of clauses of P, and each g_i is a ground instance of g. Therefore $P \cup \{\overline{g}_2, \ldots, \overline{g}_n\} \models g_1$. Since ground literals may be treated as propositions, and since the propositional MESON format is complete there is a propositional MESON proof of g_1 from $P' \cup \{\overline{g}_2, \ldots, \overline{g}_n\}$. A lifting argument can now be made to show that from the propositional proof, the appropriate first order proof can be constructed which produces the disjunction of substitutions.

3 Foothold Proof Graphs

In this section we present foothold proof graphs, a refinement of negated ancestor proof graphs that eliminates some redundant proofs, but retains completeness. The basic idea is to break up the symmetry in duplicate negated ancestor proofs. In Figure 1, the duplication is due to the symmetry of the two contrapositive rules of $a \vee b$, namely $a \leftarrow \neg b$ and $b \leftarrow \neg a$. In a foothold proof we assign labels from the set {-1,0,+1} to the literals on the right hand side of the contrapositive rules. For this example the symmetry is broken by assigning the label $(+1)$ to $\neg b$ in $a \leftarrow \neg b$, and the label (-1) to $\neg a$ in $b \leftarrow \neg a$.

To build the labeled contrapositive rules for a general clause, first separate the positive literals from the negative literals. Here positive and negative refer to the sign of the literal in the original clause. Put all of the literals of each type into an ordered sequence. Any ordering will do. For each p_i from the positive literals, build a contrapositive rule as follows. Put p_i on the left hand side. On the right hand side put the complements of the other literals in the clause. To each positive literal that appears before p_i in the ordered sequence assign the label $(+1)$, and to each positive literal that appears after p_i assign (-1). To each negative literal assign (0). Also, build a rule for each negative literal n_i as follows. Assign $(+1)$ to each negative literal that appears before n_i, assign (-1) to each negative literal that appears after n_i and assign (0) to each positive literal.

In the example the labeled contrapositive rules are

$$\begin{array}{ll} p \leftarrow a^{(0)} & \neg a \leftarrow \neg p^{(0)} \\ p \leftarrow b^{(0)} & \neg b \leftarrow \neg p^{(0)} \\ a \leftarrow \neg b^{(+1)} & b \leftarrow \neg a^{(-1)} \end{array}$$

This labelling is based on the ordering b, a for the literals in the clause $a \vee b$.

Now, build the negated ancestor proof graph using these labeled contrapositives. During the search for a negated ancestor, sum up the labels of all of the literals encountered. If the sum is positive when we reach the negated ancestor, then accept the proof. Otherwise reject it.

In the example the foothold proofs graphs are

$$\begin{array}{ccc} p & \quad & p \\ | & & | \\ a^{(0)} & & b^{(0)} \\ | & & | \\ \neg b^{(+1)} & & \neg a^{(-1)} \\ | & & | \\ \neg p^{(0)} & & \neg p^{(0)} \end{array}$$

The first of them is accepted since the sum is $0 + 1 + 0$ and the second is rejected because the sum is $0 - 1 + 0$.

The name "foothold" refers to an analogy between reaching the negated ancestor and climbing a tree. If there are more positive labels than negative labels, there is a foothold on which we can climb. If there are more negative labels than positive ones, there is no foothold, so it is impossible to climb.

This simple idea can now be presented more formally.

3.1 Propositional Foothold Format

Definition 4 Let P be a set of propositional clauses.

4.1 The **foothold rule set** of P is the union over clauses C in P of the set of foothold rules of C.

4.2 Let C be a propositional clause. Let $(\neg l_1, \ldots \neg l_n)$ be an ordered sequence of the negative literals in C and $(r_1, \ldots r_m)$ be an ordered sequence of the positive literals. The **foothold rules** of C with respect to these orderings are

$$\begin{array}{l} \forall i = 1, \ldots, n \\ \neg l_i \;\leftarrow\; l_1^{(+1)} \ldots l_{i-1}^{(+1)} \quad l_{i+1}^{(-1)} \ldots l_n^{(-1)} \quad \neg r_1^{(0)} \ldots \neg r_m^{(0)} \end{array}$$

$$\begin{array}{l} \forall i = 1, \ldots, m \\ r_i \;\leftarrow\; \neg r_1^{(+1)} \ldots \neg r_{i-1}^{(+1)} \quad \neg r_{i+1}^{(-1)} \ldots \neg r_m^{(-1)} \quad l_1^{(0)} \ldots l_n^{(0)} \end{array}$$

4.3 The superscript (s) on each literal on the right hand side of a foothold rule is called the **foothold label**. A foothold label written $(*)$ is one whose value we disregard (a "don't care").

Definition 5 Let P be a set of clauses, g a literal, and F_P be a foothold rule set of P.

5.1 A **labeled negated ancestor proof graph** containing F_P is a negated ancestor proof graph with an integer label assigned to each node and to each back edge as follows:
  (1) to the root, assign an unspecified label, $(*)$.
  (2) to each child node G_i, containing g_i assign (l_i) where the parent node contains g, where the contrapositive rule chosen for g was $g \leftarrow g_1, \ldots, g_i, \ldots, g_n$, and where the corresponding foothold rule is $g \leftarrow g_1^{(l_1)}, \ldots, g_i^{(l_i)}, \ldots, g_n^{(l_n)}$.
  (3) to each back edge (N_n, N_1) assign $\Sigma_{i=2}^{n} l_i$ where $N_1, \ldots, N_n$ is the path of tree edges such that N_n and N_1 contain complementary literals, and N_i has the label (l_i).

5.2 A **foothold proof graph** is a labeled negated ancestor graph such that every back edge has a positive label.

Theorem 3 Let P be a consistent set of propositional clauses, and g a literal. Let F_P be a foothold rule set of P. $P \models g$ if and only if there exists a foothold proof graph of g from F_P.

3.2 First Order Foothold Proofs

Foothold labels can be assigned to first order clauses in exactly the same way that they are assigned to propositional clauses. First order foothold proofs are defined as first order negated ancestor proofs, with the additional restriction that each backedge from a goal to its negated ancestor has a positive foothold sum.

Completeness of first order foothold proofs is also preserved. The argument in Section 2.2 for the completeness of first order negated ancestor proofs also applies to first order foothold proofs. The only property of propositional negated ancestor proofs that was used in that proof was propositional completeness, a property shared by propositional foothold proofs. Since foothold proofs are a special case of negated ancestor proofs, the same lifting lemma may be applied to generate first order foothold proofs.

3.3 Restricting other Types of Proofs

The basic idea behind the foothold restriction is that it is not necessary to apply ancestor resolution both to a chain of extensions built in one direction and to that chain built in the other direction. We have found that this idea can be applied to other proof methods that use extension and ancestor resolution, such as SLI (or LUST)[4]. SLI refutations provide the procedural interpretation for disjunctive logic programs.

We define SLI-FH derivations which are SLI derivations that adhere to the foothold restriction. We refer the reader to the literature [4] for the definitions of SLI derivation, t-clause, admissibility and minimality conditions.

Literals in the SLI-FH trees are assigned a foothold label when they are introduced by t-extension. As with the foothold rules of Definition 4 the foothold labels are either $+1$ or -1 according to some ordering of the literals. We assume that the lexical order of the literals in the input clauses is used. When literals become marked by * (i.e., they go from being B literals to A literals) they retain their foothold labels. During t-ancestry foothold labels on the path from L^* to M are summed to compute the foothold condition.

Definition 6 An *SLI-FH-derivation* of a t-clause E from a set of t-clauses, S, with top t-clause C is a sequence of t-clauses $D = (C_1, ..., C_n)$ such that:

C_1 is C, and C_n is E;

C_{i+1} is obtained from C_i by either t-extension, t-factoring, t-ancestry, or t-truncation;

if C_{i+1} is obtained from C_i by t-extension or t-truncation, then C_i satisfies the admissibility condition;

if C_{i+1} is obtained from C_i by t-extension, t-ancestry, or t-factoring, then C_i satisfies the minimality condition.

C_{i+1} is obtained from C_i by *t-extension* with input t-clause B_j iff

(1) C_i is $(\varepsilon^* \alpha_1 L \beta_1)$;

(2) B_j is $(\varepsilon^* \alpha_2 M \beta_2)$;

(3) L and M are complementary and unify with mgu θ;

(4) C_{i+1} is $(\varepsilon^* \; \alpha_1\theta \; (L\theta^* \; \alpha_2\theta \; \beta_2\theta) \; \beta_1\theta)$;

(5) if M is positive then all positive literals of $\alpha_2\theta$ are assigned a foothold label of $+1$ and all of the positive literals of $\beta_2\theta$ are assigned a foothold label of -1. If M is negative then all negative literals of $\alpha_2\theta$ are assigned a foothold label of $+1$ and all of the negative literals of $\beta_2\theta$ are assigned a foothold label of -1.

C_{i+1} is obtained from C_i by *t-factoring* iff

(1) C_i is $(\alpha_1 \; L \; \alpha_2 \; M \; \alpha_3)$ or C_i is $(\alpha_1 \; M \; \alpha_2 \; L \; \alpha_3)$;

(2) L and M have the same sign and unify with mgu θ;

(3) L is in γ_M (i.e., L is in a higher level of the tree);

(4) C_{i+1} is $(\alpha_1 \; \theta \; L\theta^* \; \alpha_2\theta \; \alpha_3\theta)$ when C_i is $(\alpha_1 \; L \; \alpha_2 \; M \; \alpha_3)$, or C_{i+1} is $(\alpha_1\theta \; \alpha_2\theta \; L\theta^* \; \alpha_3\theta)$ when C_i is $(\alpha_1 \; M \; \alpha_2 \; L \; \alpha_3)$.

C_{i+1} is obtained from C_i by *t-ancestry* iff

(1) C_i is $(\alpha_1(L^*\alpha_2(\alpha_3 M \alpha_4)\alpha_5)\alpha_6)$;

(2) L and M are complementary and unify with mgu θ;

(3) σ is the sum of the foothold labels on the A-literals on the path from L^* to M including that on M but not including the foothold label on L^*, and $\sigma > 0$;

(4) C_{i+1} is $(\alpha_1\theta(L\theta^* \; \alpha_2\theta(\alpha_3\theta \; \alpha_4\theta)\alpha_5\theta)\alpha_6\theta)$.

C_{i+1} is obtained from C_i by *t-truncation* iff either

C_i is $(\alpha(L^*)\beta)$ and C_{i+1} is $(\alpha\beta)$

or

C_i is (ε^*) and C_{i+1} is $\Box$.

Definition 7 An SLI-FH refutation from the set of t-clauses S with top t-clause C is an SLI-derivation of the null clause $\Box$ from the top t-clause C.

For example, given the clauses (from [4])

(1) $(\varepsilon^* p(f(X)) \; \neg q(X) \; \neg r(X))$,

(2) $(\varepsilon^* p(f(X)) \; q(X))$,

(3) $(\varepsilon^* r(X))$

and the goal clause

$(\varepsilon^* \neg p(f(a))$

the following is an SLI-FH refutation:

$(\varepsilon^* \neg p(f(a)))$	goal clause
$(\varepsilon^* (\neg p(f(a))^* \; \neg q(a) \; \neg r(a)))$	t-extension with (1)
$(\varepsilon^* (\neg p(f(a))^* \; (\neg q(a)^* \; p(f(a))^{(+1)}) \; \neg r(a)))$	t-extension with (2)
$(\varepsilon^* (\neg p(f(a))^* \; (\neg q(a)^*) \; \neg r(a)))$	t-ancestry where $\sigma = +1$
$(\varepsilon^* (\neg p(f(a))^* \; \neg r(a)))$	t-truncation
$(\varepsilon^* (\neg p(f(a))^* \; (\neg r(a)^*)))$	t-extension with (3)
$(\varepsilon^* (\neg p(f(a))^*))$	t-truncation
(ε^*)	t-truncation
$\square$	t-truncation

However, the following sequence of t-clauses cannot be extended to complete an SLI-FH refutation since $p(f(a))$ cannot be eliminated:

$(\varepsilon^* \neg p(f(a)))$	goal clause
$(\varepsilon^* (\neg p(f(a))^* \; q(a)^{(-1)}))$	t-extension with (2)
$(\varepsilon^* (\neg p(f(a))^* \; (q(a)^{*(-1)} \; p(f(a)) \; \neg r(a)^{(+1)})))$	t-extension with (1)

Elimination by t-ancestry is blocked because of the foothold condition with $\sigma = -1$, and neither t-extension nor t-factoring applies. If we ignore the foothold condition, we can apply t-ancestry, and then complete the redundant SLI proof:

$(\varepsilon^* \neg p(f(a)))$	goal clause
$(\varepsilon^*(\neg p(f(a))^*\ q(a)^{(-1)}))$	t-extension with (2)
$(\varepsilon^*(\neg p(f(a))^*\ (q(a)^{*(-1)}\ p(f(a))\ \neg r(a)^{(+1)})))$	t-extension with (1)
$(\varepsilon^*(\neg p(f(a))^*\ (q(a)^{*(-1)})\ \neg r(a)))$	t-ancestry where σ is ignored
$(\varepsilon^*(\neg p(f(a))^*\ \neg r(a)))$	t-truncation
$(\varepsilon^*(\neg p(f(a))^*\ (\neg r(a)^*)))$	t-extension with (3)
$(\varepsilon^*(\neg p(f(a))^*))$	t-truncation
(ε^*)	t-truncation
$\square$	t-truncation

4 Evaluation

4.1 How easily can foothold proofs be computed?

Like the MESON format, the foothold format leads to proof procedures that are simple to implement. Appendix B contains a Prolog procedure for computing foothold proofs. Any theorem prover that is based on the MESON format, such as PTTP [10] can easily be converted to the foothold format.

Procedures that compute MESON proofs can take advantage of an important pruning rule. If a goal is encountered that is the exact duplicate of a goal in the ancestor sequence, then this branch of the tree can be ignored and no success is reported. There is no reason to pursue a proof of this goal, because this condition arises when the procedure has entered a loop. Procedures that compute foothold proofs can take advantage of the same pruning rule, for the same reason. The Prolog procedure in the appendix implements this pruning rule.

4.2 When should the foothold format be used?

The foothold format should be used whenever one is reasoning with clauses that extend beyond the Horn subset and the objective of the search is to find more than one result. If the objective of the search is to find a single solution, or to show that a set of clauses is unsatisfiable, then foothold proof procedures are not necessarily better or worse than MESON proof procedures.

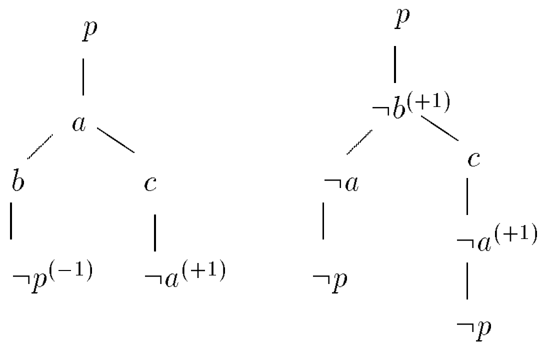


Figure 2: The Foothold restriction does not preserve minimum proofs

If the objective is to find a proof with minimum height, where the height of a proof is the maximum size of the ancestor set, then the foothold refinement should not be used. The shortest negated ancestor proof may be rejected by the foothold condition, while a longer proof, composed of the same clauses in a different arrangement, meets the condition. In Figure 2 the proof on the left is a negated ancestor proof graph, but not a foothold proof graph because of the label on $\neg p$. The proof on the right is the smallest negated ancestor proof graph for the orderings chosen in the foothold rules.

Proof height is an important consideration for "iterative deepening" proof methods, such as the PTTP [10]. By restricting the search to a preset limit, and iteratively increasing this limit, depth first search strategies can be prevented from running down infinite paths. They will find each proof up to that limit. Thus depth first search achieves the completeness of breadth-first search, without also requiring exponential space.

The iterative deepening strategy can be used with the foothold refinement. Because the minimum height proof may be eliminated, the searcher may be forced to look deeper to generate the same number of distinct proofs. In Section 4.5 we discuss how the reduction in search space helps offset the cost of the additional depth.

4.3 How many proofs are avoided?

The amount of redundancy reduced depends on the clauses, but we are guaranteed that the foothold procedure will generate no more redundancy than the negated ancestor procedure, since the foothold format is strictly more specific than the negated ancestor format.

There are some examples where the foothold format still admits an exponential amount of redundancy, in the sense that more than one proof can be built from the same set of clauses. In other words, given the clauses used in the construction of some proof, other proofs can be built from the same clauses. Redundancy in these examples is due to interaction between features of the clauses, including the non-Horn feature. In the following example different parts of the proof must use different subproofs to prove the same literal. Redundancy arises because in yet another part of the proof the same literal can be proved by any of the subproofs. There are $\Pi_{i=1}^{n} i!$ foothold proofs of p from the following:

$$
\begin{array}{ll}
p \vee \neg p_1 \vee \ldots \vee \neg p_n & \\
p_i \vee \neg p_{i+1} \vee \neg a & \forall i = 1 \ldots n-1 \\
p_n \vee \neg a & \\
p_i \vee a & \forall i = 1 \ldots n
\end{array}
$$

There are some examples where the foothold procedure eliminates all redundancy. Consider the following:

$$
\begin{array}{ll}
p \vee \neg p_1 \vee \ldots \vee \neg p_n & \\
p_i \vee \neg a_i & \forall i = 1 \ldots n \\
p_i \vee \neg b_i & \forall i = 1 \ldots n \\
a_i \vee b_i & \forall i = 1 \ldots n
\end{array}
$$

For each n there is exactly one foothold proof of p from the above, but there are 2^n negated ancestor proofs. In the table in Figure 3 we compare the time to calculate all proofs of p from this set of clauses for different values of n. (All programs in this paper were written in Quintus Prolog, and run on a VAX 8600. Times are reported in milliseconds.)

As reported in Figure 3 the time to build all 2^n negated ancestor proofs increases exponentially with n, but the time to build the only foothold proof and to decide it is the only one, increases more slowly with n.

Value of n	Foothold	Negated Ancestor
1	16	17
2	33	33
3	33	100
4	67	216
5	83	417
6	150	1033
7	184	2583
8	200	5483
9	300	12100
10	367	26400
$\vdots$	$\vdots$	$\vdots$
30	3850	$\approx 10^{10}$

Figure 3: Times (msec) to compute all proofs of each type

4.4 Why is so much time saved?

Time is saved because so much redundant searching is avoided. To illustrate this, we compare a naive MESON proof procedure with and without the foothold restriction.

For example, we are given the following clauses

$$q \vee \neg p \vee \neg s \vee \neg t$$
$$p \vee \neg a$$
$$p \vee \neg b$$
$$a \vee b$$
$$s \vee \ldots$$

and the query q. Suppose it is possible, but very expensive to prove s, and it is impossible to prove t. If a naive MESON theorem prover pursues the clauses in the order they are given and pursues the literals left to right, then it will attempt to prove q using the first clause. It will need to prove p, which succeeds via reasoning by cases as shown in Section 2.1. Then it will prove s incurring its associated high cost. But it will be unable to deal with the t

literal. Failure will cause backtracking and it will build the second proof of p, reprove s, incurring the high cost a second time, and then fail at t once more.

The same theorem prover with the foothold restriction will proceed exactly as above and build one proof of s, until the failure at t occurs. It will fail to find the second proof of p and so avoid building the expensive proof of s again.

More savings arise when we complicate the example slightly. Let us insert a new literal, say $\neg p_1$ between between $\neg p$ and $\neg s$ in the first clause, and add the clauses

$$\begin{array}{ccc} p_1 & \vee & \neg a_1 \\ p_1 & \vee & \neg b_1 \\ a_1 & \vee & b_1 \end{array}$$

Then the MESON proof procedure will build four proofs of s whereas the prover with the foothold restriction, still, will only build one proof of s.

4.5 How much time is saved?

In Figure 4 we compare two procedures for producing proofs. We report on the time to build all proofs and the number of proofs built. The procedure NA builds all negated ancestor proofs; FH builds all negated ancestor proofs that conform to the foothold condition.[2] If FH finds that a part of the proof does not conform to the foothold condition then it does not continue expanding that proof. The procedures are otherwise identical. They prune branches with identical ancestors, and they prune the search after a height of 10. We have run these programs on the tests suggested by Pelletier[5] that do not include equality axioms.

Pelletier's tests are unsatisfiable sets of clauses ; we converted each set of clauses into a problem of showing that a literal is entailed by a set of clauses by introducing a new literal into the one clause in the set, and trying to show that literal followed.

[2]Checking this condition adds an extra addition operation each time we search up the ancestor path and an extra less than check if and when a complementary ancestor is found. The extra costs for these operations are negligible.

Figure 4: Pelletier's Problems: Times (in msec) and number of proofs

	#	Time	Count
FH	1	16	1
NA	1	17	1
FH	2	33	1
NA	2	33	1
FH	3	17	1
NA	3	0	1
FH	4	33	1
NA	4	17	1
FH	5	33	1
NA	5	33	1
FH	6	34	1
NA	6	17	1
FH	7	17	1
NA	7	17	1
FH	8	17	1
NA	8	0	1
FH	9	50	1
NA	9	50	4
FH	11	33	1
NA	11	33	1
FH	12	766	1
NA	12	222450	4096
FH	13	17	1
NA	13	33	1
FH	14	50	1
NA	14	50	4
FH	15	17	1
NA	15	33	1
FH	16	0	1
NA	16	17	1
FH	17	67	1
NA	17	67	2
FH	18	33	1
NA	18	16	1

	#	Time	Count
FH	19	17	1
NA	19	17	1
FH	20	50	1
NA	20	33	1
FH	21	67	2
NA	21	167	12
FH	22	4800	207
NA	22	50167	3782
FH	23	167517	4620
NA	23	1437117	97200
FH	24	100	1
NA	24	200	8
FH	25	2583	5
NA	25	4533	36
FH	27	67	1
NA	27	83	2
FH	28	34	1
NA	28	50	1
FH	30	50	2
NA	30	67	4
FH	31	50	1
NA	31	17	1
FH	32	50	1
NA	32	50	1
FH	35	0	1
NA	35	16	1
FH	36	50	4
NA	36	67	4
FH	37	483	29
NA	37	550	36
FH	39	33	1
NA	39	67	8
FH	44	50	1
NA	44	66	2
FH	45	16683	16
NA	45	1524550	9589

	#	Count at each height							
		3	4	5	6	7	8	9	10
FH	22	1	14	17	23	29	35	41	47
NA	22	4	73	165	298	467	672	913	1190
FH	23	2	10	108	276	516	828	1212	1668
NA	23	8	82	1350	4716	9990	17208	26370	37476
FH	25				2	2	1		
NA	25			5	9	10	6	4	2
FH	37	1	4	4	4	4	4	4	4
NA	37	1	5	5	5	5	5	5	5
FH	45						9		7
NA	45				5	7	147	708	8722

Figure 5: Proof Heights for Selected Problems

We report on problems for which a proof can be found in a reasonable amount of time. Some problems required sound unification, with the occur-check. Because we used Prolog's unification, these could not be run.

In these tests the foothold refinement is never significantly more expensive than the negated ancestor procedure, and is more expensive only in simple problems, such as 3 and 8, where the times are so small they have little significance. On large meaningful problems, the foothold procedure is strongly favoured. This suggests that for still larger problems the savings will continue to be significant.

Figure 4 indicates that the foothold format is never significantly slower than the negated ancestor format, and can be much faster.

Figure 5 reports the number of proofs of each height for some of the more complex problems. In Problems 25 and 45 the minimum height proof is rejected. Proofs are found after 1 and 2 more levels, respectively. In these two examples the search spaces are especially reduced. This result suggests the the search space reduction will compensate for the loss of the minimum height proof.

The effect of the foothold restriction is very pronounced on problems that are inherently redundant, such as the famous pigeon-hole problems. To show that it is impossible to put 4 pigeons in 3 holes where no more that one pigeon can occupy any hole, the NA procedure found all 20736 proofs in 538,618 milliseconds. The FH procedure needed only 416 milliseconds to discover that only one proof remained.

5 Conclusion

In settings where results are computed from proofs, such as first order logic, duplicate proofs produce the same result. Time is wasted computing the duplicate proof and more time is wasted checking the result for redundancy. We have presented the foothold format, a refinement of the MESON format that admits fewer duplicate proofs. Procedures that compute these proofs can detect redundancy before the entire proof is constructed. Empirical evidence shows a simple foothold procedure is never significantly slower and sometimes much faster then the same procedure without the foothold refinement. The savings are greater for more complex examples.

6 Acknowledgements

The author is grateful for assistance from Bell-Northern Research and NSERC (OGP 0106290) and for helpful comments from the reviewers.

A Proofs

We provide this proof to demonstrate that the part of the MESON proof format we have extracted has the important properties of soundness and completeness.

Theorem 1 Let P be a consistent set of propositional clauses, and g a literal. $P \models g$ if and only if there exists a negative ancestor proof graph of g using clauses in P.

Proof Completeness ($\Rightarrow$ by induction on the number of clauses in P)
We claim for all P, g and A if P is a consistent set of propositional clauses, g is a literal, A is a sequence of nodes representing a path of tree edges[3],

[3]Note that the tree edges for A do not necessarily come from this P.

B is the set of all literals whose complements appear in the nodes of A, $P \cup B \models g$, $P \cup B$ is consistent, and G is a node containing g then there is an NA1 proof graph of G containing the ancestor path A. If this claim is true then by setting A to the empty sequence, B becomes $\{\}$ and we have a negative ancestor proof graph of g.

If $P = \{\}$ then $P \cup B \models g$ means $g \in B$, so there is a node in A containing $\overline{g}$, which means there is an NA1 proof, using case a of Definition 3.1. Let P have $k+1$ clauses. If $g \in B$ then there is an NA1 proof, also by case a. Otherwise we construct P', a subset of P such that $P' \cup B \cup \{\overline{g}\}$ is inconsistent, and for all proper subsets P'' of P', $P'' \cup B \cup \{\overline{g}\}$ is consistent. (P' can be obtained by exhaustively considering all possible subsets of P.) Assume P' does not contain a clause C which mentions g. Then since $P' \cup B$ is consistent and $g \notin B$, $P' \cup B \cup \{\overline{g}\}$ is consistent, which is false. Thus there is a clause $C \in P$ and without loss of generality assume $C = g \vee g_1 \vee \ldots \vee g_n$. If $n = 0$ then the NA1 proof graph of G consists of G and the ancestor path A. Assume $n > 0$. Since $P' \setminus \{C\}$ is a proper subset of P', $P' \setminus \{C\} \cup B \cup \{\overline{g}\}$ is consistent. If there is a g_j in C such that $P' \setminus \{C\} \cup B \cup \{\overline{g}\} \cup \{g_j\}$ is consistent, then $P' \cup B \cup \{\overline{g}\}$ is consistent, which is false. Thus $P' \setminus \{C\} \cup B \cup \{\overline{g}\} \models \overline{g}_j$ for all j. Then by induction, setting P in the claim to $P' \setminus \{C\}$, setting B to $B \cup \{\overline{g}\}$, and setting g to $\overline{g}_j$ if we construct nodes G_j containing $\overline{g}_j$ then there is an NA1 proof graph of each G_j with ancestor path (A, G). These graphs can be combined by constructing tree edges from G to each G_j to form an NA1 proof of G containing the ancestor path A.

Soundness ($\Leftarrow$ by induction on the height of the tree (V, T))
We claim that if there is an NA1 proof graph of some node G with respect to the ancestor path A, then $P \cup B \models g$, where B is the set of literals whose complements appear in A and g is the literal in G. If the claim is true then by setting A to the empty path, B becomes $\{\}$ and $P \models g$.

Let the tree (V, T) in the NA1 graph have height 0. Then either there is a back edge from G to a node in A so $g \in B$, or there is a contrapositive rule g with no right hand side so $g \in P$. In either case $P \cup B \models g$. Suppose the tree has height $k+1$. If there is a back edge from G to a node in A then again $P \cup B \models g$. Otherwise there is a contrapositive rule $g \leftarrow g_1 \ldots g_n$, and for all $i = 1, \ldots, n$ there is a node G_i containing g_i and an NA1 proof graph of G_i containing ancestor path (A, G). These smaller graphs contain trees with height at most k, so by induction $P \cup B \cup \{\overline{g}\} \models g_i$. Since $P \models g \leftarrow g_1 \ldots g_n$ it follows that $P \cup B \cup \{\overline{g}\} \models g$. Thus $P \cup B \cup \{\overline{g}\}$ is inconsistent so $P \cup B \models g$.∎

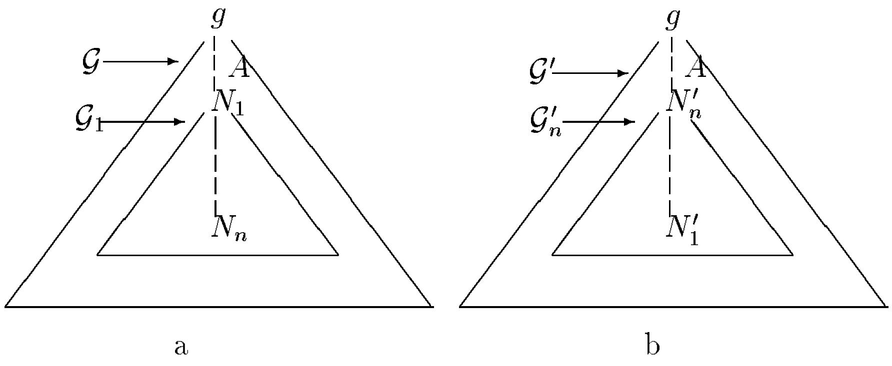


Figure 6: From $\mathcal{G}$ with a backedge (N_n, N_1) to $\mathcal{G}'$ with a back edge (N'_1, N'_n)

The following lemma is used in the proof of Theorem 3. Essentially it proves that if a proof graph has tree branches from $h_1, \ldots, h_n$ where $h_n = \overline{h}_1$ then the branches can be "flipped" to produce another proof with branches from $\overline{h}_n, \ldots, \overline{h}_1$. For example, Lemma 2 defines the mapping from the first to the second proof graph in Figure 1.

Lemma 2 Let $\mathcal{G}$ be a negative ancestor proof graph using clauses from P that contains a path of tree edges represented by the sequence of nodes $N_1, \ldots, N_n$, and a back edge (N_n, N_1). For $i = 1, \ldots, n$, let N_i contain the literal h_i. Then there exists a negative ancestor proof graph $\mathcal{G}'$ using clauses from P, that contains a path of tree edges $N'_n, \ldots, N'_1$, where N'_i contains $\overline{h}_i$, and a back edge (N'_1, N'_n).

Proof Let $\mathcal{G}_1$ be the NA1 proof graph of N_1, the node containing h_1. We shall construct $\mathcal{G}'$ by replacing $\mathcal{G}_1$ in $\mathcal{G}$ with $\mathcal{G}'_n$, an NA1 proof graph of N'_n, the node containing $\overline{h}_n$. (See Figure 6.) Note that $h_1 = \overline{h}_n$ since there is a back edge (N_n, N_1). Let A be the ancestor path from the root of $\mathcal{G}$ to N_1. We construct $\mathcal{G}'_n$ by inductively constructing an NA1 proof graph of N'_i with ancestor path $(A, N'_n, \ldots, N'_i)$, such that there is no back edge leading to any one of $N'_{n-1}, \ldots, N'_i$. (See Figure 7.)

The NA1 proof graph of N'_1 consists mostly of the long ancestor path $(A, N'_n, \ldots, N'_2)$. By case a, it contains a back edge (N'_1, N'_n), since N'_1 con-

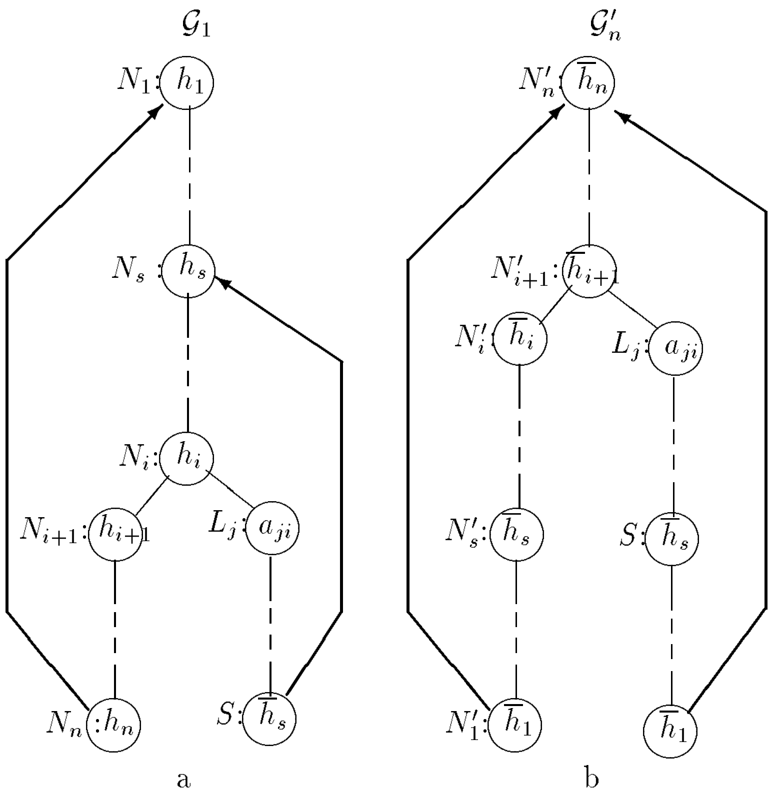


Figure 7: Constructing $\mathcal{G}'_n$ from $\mathcal{G}_1$

tains $\overline{h}_1$ and N'_n contains h_1. There is no back edge leading to N'_1 since N'_1 has no descendants. To construct the NA1 proof graph of N'_{i+1} with ancestor path $(A, N'_n, \ldots, N'_{i+2})$, note that there is a clause in P from which the contrapositive rule

$$h_i \leftarrow h_{i+1} a_{1i} \ldots a_{mi}$$

can be constructed, without loss of generality. Therefore the contrapositive rule

$$\overline{h}_{i+1} \leftarrow \overline{h}_i a_{1i} \ldots a_{mi}$$

can also be constructed. So we apply case b and construct for each $j = 1, \ldots, m$ a node L_j containing a_{ji}, a tree edge (N'_{i+1}, L_j) and an NA1 proof graph of L_j with ancestor path $(A, N'_n, \ldots, N'_{i+1})$. We have already constructed an NA1 proof graph of N'_i with ancestor path $(A, N'_n, \ldots, N'_{i+1})$ by induction.

To construct the NA1 proof graph of L_j, note that part of $\mathcal{G}_1$ is an NA1 proof graph of L_j with ancestor path $(A, N_1, \ldots, N_i)$. (See Figure 7a.) Make a copy of that graph, replacing the ancestor path $(A, N_1, \ldots, N_i)$ with the ancestor path $(A, N'_n, \ldots, N'_{i+1})$. Any back edge (S, N_s) for $1 \leq s < i$ in the proof graph for L_j must also be removed since its destination has been removed. To construct a new proof graph for S note that by induction we have already constructed a proof graph for N_s, a node which also contains $\overline{h}_s$. This graph has an ancestor path $(A, N'_n, \ldots, N'_{s+1})$, but no back edges to any of $N'_{n-1}, \ldots, N'_s$. Make a copy of this graph, and in the copy give the name S to N_s. Since there were no back edges to $N'_{n-1}, \ldots, N'_s$, we can replace the ancestor path in the new graph with $(A, N'_n, \ldots, N'_{i+1}, L_j, \ldots, S)$. (See Figure 7b.) This completes the construction of the proof graph for L_j. To complete the induction we point out that there are no back edges to N'_{i+1}.∎

Theorem 3 Let P be a consistent set of propositional clauses, and g a literal. Let F_P be a foothold rule set of P. $P \models g$ if and only if there exists a foothold proof graph of g from F_P.

Proof ($\Leftarrow$) Since the foothold proof graph is a negative ancestor proof graph, the result follows by Theorem 1.

($\Rightarrow$) Let $\mathcal{G}$ be a negative ancestor proof graph of g from P, provided by Theorem 1. According to Definition 5 label $\mathcal{G}$ with respect to F_P. If there are no negative back edges then this is a foothold proof graph, and we are done. Otherwise select a lowest node N that is the destination of some negative back edge. We shall construct a labelled NA1 proof graph of N that has no negative back edges leading to N or to any descendant of N. By selecting a new lowest node that is the destination of some negative back edge and repeating the construction of an NA1 proof graph of it with no negative back edges we can eliminate all negative back edges. This completes the construction.

To construct the NA1 proof graph of N as claimed, let $N_1 = N$ and select a back edge (N_n, N_1) with label $k < 0$. We know $k = \Sigma_{i=2}^{n} l_i$ where l_i is assigned by the foothold rule

$$h_i \leftarrow h_{i+1}^{(l_i)} a_{1i}^{(*)} \ldots a_{mi}^{(*)}$$

Apply Lemma 2 to (N_n, N_1) to replace that back edge with (N_1', N_n') with label $k' = \Sigma_{i=2}^{n} l_i'$ where l_i' is assigned by the foothold rule

$$\overline{h}_{i+1} \leftarrow \overline{h}_i^{(l_i')} a_{1i}^{(*)} \ldots a_{mi}^{(*)}$$

If h_i and h_{i+1} are both positive or both negative literals then in the ordering chosen in F_P either h_i comes before h_{i+1}, in which case $l_i = +1$ and $l_i' = -1$, or h_i comes after h_{i+1}, in which case $l_i = -1$ and $l_i' = +1$. If one of h_i and h_{i+1} is positive and the other is negative then $l_i = l_i' = 0$. In either case $l_i = -l_i'$ so $k = -k'$. Thus $k' > 0$. We have replaced a negative back edge with a positive one. We claim that we have not negated any positive backedge so that the number of positive backedges has strictly increased. By continuing to select a negative back edge and apply Lemma 2, we strictly increase the number of positive back edges. Since there is a finite maximum number of back edges we can eliminate all negative back edges in a finite number of steps.

It remains to show our claim that an application of Lemma 2 to a negative back edge does not negate any positive back edge to N_1. Let (M_m, N_1) be a positive back edge, and let the path of tree edges from N_1 to M_m be $N_1, \ldots, N_i, M_1, \ldots, M_m$, as in Figure 8a. Let $X = \Sigma_{j=2}^{i} l_j$, let $X_N = \Sigma_{j=i+2}^{n} l_j$, let X_M be the sum of the labels of M_2 to M_m, let k be the label of M_1. After the application of Lemma 2 to (N_1, N_n) we have the graph shown in

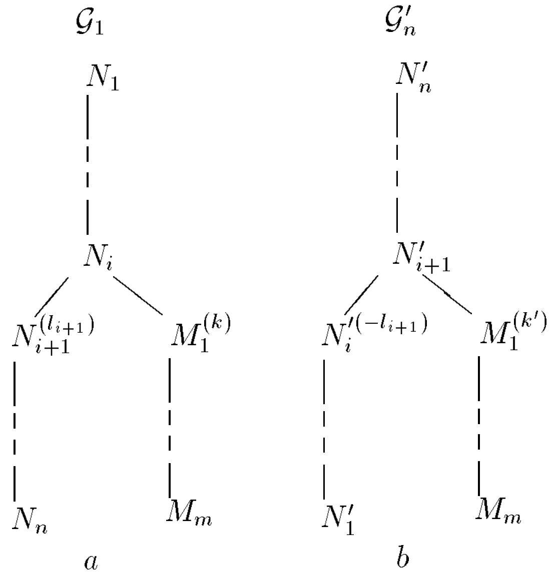


Figure 8: The effect Lemma 2 on other back edges

8b, where k' is the new label for M_1.

$$\begin{array}{lrcl}
\text{Given} & X + l_{i+1} + X_N & < & 0 \\
\text{it follows} & X + l_{i+1} + X_N & \leq & -1 \\
\text{so} & X & \leq & -l_{i+1} - X_N - 1 \\
\text{Given} & X + k + X_M & > & 0 \\
\text{it follows} & X & > & -k - X_M \\
\text{Transitively} & -l_{i+1} - X_N - 1 & > & -k - X_M \\
\text{so} & X_M - X_N & > & l_{i+1} + 1 - k
\end{array}$$

$$\begin{array}{lrcl}
\text{Suppose} & X_M - X_N + k' & < & 0 \\
\text{Then} & X_M - X_N & \leq & -1 - k' \\
\text{Transitively} & -1 - k' & > & l_{i+1} + 1 - k \\
\text{so} & k - k' - l_{i+1} & > & 2
\end{array}$$

Since k, k' and l_{i+1} are restricted to values in $-1, 0, +1$ it follows that $k = 1, k' = -1$ and $l_{i+1} = -1$. Therefore in the clause in question the ordering chosen to build the foothold rules put the literal in N_i before the literal in N_{i+1} since $l_{i+1} = -1$. It put the literal in N_{i+1} before the literal in M_1 since $k' = -1$. And it put the literal in M_1 before the literal in N_i since $k = 1$. But this is absurd so the supposition that $X_M - X_N + k' < 0$ is false.∎

B Computing Foothold Proofs in Prolog

```
% Propositional Foothold Proof Procedure

:- op(250, fx, (¬)).

prove(G) :- fh_prove(labelled_literal(G, _), [ ]).

fh_prove(labelled_literal(G, _), Anc) :-
        negate(G, Neg_G),
        member( labelled_literal(Neg_G, _), Anc),
        !, fail. % Cut is allowed when literals are propositions
fh_prove(labelled_literal(G, Label), Anc) :-
```

```
        ancestor_search(G, Label, Total, Anc),
        !, %Cut is allowed when literals are propositions
        Total > 0.
fh_prove(labelled_literal(G, Label), Anc) :-
        foothold_contrapositive_rule(G, Body),
        negate(G, Neg_G),
        fh_prove_all(Body, [labelled_literal(Neg_G, Label) | Anc]).

fh_prove_all([ ], _).
fh_prove_all([ G1 | G ], Anc) :-
        fh_prove(G1, Anc),
        fh_prove_all(G, Anc).

ancestor_search(Goal, S0, S0, [labelled_literal(Goal, Label) | Anc]).
ancestor_search(Goal, S0, S2, [labelled_literal(G, Label) | Anc]) :-
        S1 is S0 + Label,
        ancestor_search(Goal, S1, S2, Anc).

negate(X, ¬X) :- \+ X = ¬_.
negate(¬X, X).
```

References

[1] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York and London, 1973.

[2] C. Cordell Green. Theorem-proving by resolution as a basis for question-answering systems. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence 4*, pages 183–205. American Elsevier Publishing Company, Inc., 1969.

[3] Donald Loveland. *Automated Theorem Proving: A Logical Basis.* North Holland, Amsterdam, 1978.

[4] Jack Minker and Arcot Rajasekar. A fixpoint semantics for disjunctive logic programs. *The Journal of Logic Programming*, 9(1):45–74, July 1990.

[5] Francis Jeffry Pelletier. Seventy-five problems for testing automated theorem provers. *Journal of Automated Reasoning*, 2:191–216, 1986.

[6] D. Plaisted. A sequent-style model elimination strategy and a positive refinement. *Journal of Automated Reasoning*, 6:389–402, 1990.

[7] Rolf Socher-Ambrosius. How to avoid the derivation of redundant clauses in reasoning systems. *Journal of Automated Reasoning*, 9:77–97, 1992.

[8] Bruce Spencer. Avoiding duplicate proofs. In Saumya K. Debray and Manuel Hermenegildo, editors, *Proceedings of the North American Conference on Logic Programming*, pages 569–584. MIT Press, 1990.

[9] Bruce Spencer. Linear resolution with ordered clauses. In *Proceedings of the Workshop on Disjunctive Logic Programming at the International Symposium of Logic Programming*, 1991. SanDiego, California.

[10] Mark E. Stickel. A prolog technology theorem prover. *Journal of Automated Reasoning*, 4:353–380, 1989.

[11] Larry Wos. *Automated Reasoning : 33 Basic Research Problems.* Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

Contents