Efficient algorithms to detect and restore *minimality*, an extension of the regular restriction of resolution

Bruce Spencer and J.D. Horton
University of New Brunswick
P.O. Box 4400, Fredericton, New Brunswick, Canada E3B 5A3
bspencer@unb.ca, jdh@unb.ca, http://www.cs.unb.ca

Abstract. A given binary resolution proof, represented as a binary tree, is said to be *minimal* if the resolutions cannot be reordered to generate an irregular proof. Minimality extends Tseitin's regularity restriction and still retains completeness. A linear time algorithm is introduced to decide whether a given proof is minimal. This algorithm can be used by a deduction system that avoids redundancy by retaining only minimal proofs, and thus lessens its reliance on subsumption, a more general but more expensive technique.

Any irregular binary resolution tree is made strictly smaller by an operation called *Surgery*, which runs in time linear in the size of the tree. After surgery the result proved by the new tree is non-strictly more general than the original result, and has fewer violations of the regular restriction. Furthermore any non-minimal tree can be made irregular in linear time by an operation called *Splay*. Thus a combination of splaying and surgery efficiently reduces a non-minimal tree to a minimal one.

Finally, a close correspondence between clause trees, recently introduced by the authors, and binary resolution trees is established. In that sense this work provides the first linear time algorithms that detect minimality and perform surgery on clause trees.

1. Introduction

The regular restriction [15] of binary resolution states that a resolution step resolving on a given literal should not be used to deduce a clause containing that literal. In other words, that resolution step should not be an ancestor of such a clause in the binary derivation tree. We extend this restriction so that it applies also when a reordering of the resolutions brings such a clause below that step. We use rotations of edges in the binary tree to reorder the resolution steps, and require that the rotations neither weaken what is proved nor increase the size of the tree. If a binary resolution proof cannot be made irregular by such rotations, we call it *minimal*.

This extension of regularity depends on sequences of rotations, and thus appears to be expensive to compute. However we characterize it with a condition that can be checked efficiently by examining the static tree. The condition is stated in terms of history paths in the binary resolution tree. Each *history path* tells the story of a given literal. The tail of the history path is a leaf of the tree and it tells where the literal was introduced in an input clause. The history path is said to *close* at the node where its literal is resolved away. When one history path closes at a node that occurs in another history path, and the two paths are disjoint, we say the first history path *directly precedes* the other. The *precedes* relation on history paths is the reflexive transitive closure of directly precedes. See Section 2 for examples of these definitions. History paths and the precedes relation are basic in our understanding of how binary resolution trees behave when rotations are performed. We provide simple conditions on history paths that characterize when one node can be rotated below another (we say it is *visible* from the other) and when one node cannot be rotated from below another (we then say it *supports* the other.) By examining the history paths in a static tree, we can decide much about what can and cannot be accomplished by sequences of rotations.

In particular, we can say whether rotations can convert a regular tree to an irregular one. A theorem prover can use this ability to screen the proofs it builds, and retain only the minimal ones. Since every clause with a non-minimal proof is subsumed by some clause with a minimal proof, this theorem prover uses minimality to lessen its reliance on subsumption. The running time of full subsumption grows with the number of retained clauses, which may become very large, while detecting minimality depends linearly on the size of each proof tree, which is typically much smaller.

Instead of using minimality simply as a filter, a theorem prover can convert a non-minimal proof to a minimal one, using operations defined in this paper. Consider the branch of an irregular derivation tree that makes it irregular. This branch contains the node where some given literal is resolved and, further down, contains a clause in which that literal occurs for a second time. Why resolve the literal away, only to have it reappear later? We can remove the resolution, and reconstruct the branch as closely as possible to the original, with this literal appearing additionally in some of the clauses. If the second occurrence of the literal is resolved away in the original tree, then we resolve both occurrences away in the constructed tree. The constructed tree is smaller and the result it proves is at least as general as the original one. We call this operation *Surgery*, and define a second operation, *Splay*, which rearranges a non-minimal tree so that it is irregular. Both operations run in time linear in the size of the tree. A combination of both operations will eventually reduce any non-minimal tree to some non-unique minimal one.

The first section below presents the regular restriction on binary resolution trees. In that section we introduce the surgery operation for irregular binary resolution trees. Then minimal binary resolution trees are introduced. This section also discusses rotations and the set of rotation equivalent trees, which is all of the trees that can be generated from a given tree by a sequence of these rotations. In the following section, we discuss the "precedes" relation of history paths, and use that to define the "holds" relation on nodes. We then relate holds and visibility by showing that if the nearest common descendant of two given nodes holds one of them, then that one is not seen by the other, *i.e.* it cannot be rotated below the other. Based on this condition, we give an efficient algorithm for deciding visibility. We show how a theorem prover can be restricted to retain only minimal proofs and disregard non-minimal ones, decreasing the number of proofs that must be considered. Then we show the splay operation for efficiently converting a non-minimal proof into an irregular one, so that surgery can further convert it to a minimal one. Thus another theorem prover can efficiently convert its non-minimal results to minimal ones if desired. In the next section we use history paths to characterize support: the condition where one node in a binary resolution tree must be a descendant of another after any sequence of rotations.

Both minimality and surgery were first developed for clause trees [5]. The clause tree is a tool for developing ideas in automated reasoning, and the binary resolution tree is an efficient, compact data structure to implement clause trees. In the second last section we show the close relationship between binary resolution trees and clause trees. Thus, this paper provides the first efficient algorithm for surgery in clause trees. We close with some remarks on the relation between clause trees and binary resolution trees, and related work.

This paper is an extension of [14].

2. Binary Resolution Trees

We use standard definitions [2] for atom, literal, substitution, unifier and most general unifier. In the following a *clause* is an unordered disjunction of literals. We do not use set notation because we do not want multiple occurrences of a literal to collapse to a single literal automatically. Thus our clauses can be viewed as multisets. An atom a *occurs in* a clause C if either a or $\neg a$ is one of the disjuncts of the clause. The clause C *subsumes* the clause D if there exists a substitution θ such that $C\theta \subseteq D$ (as sets, not as multisets). A *variable renaming substitution* is one in which every replacement of a variable maps to another variable, and no two variables map to the same variable. Two clauses C and D are *equal up to variable renaming* if there exists a variable renaming substitution θ such that $C\theta = D$ (as multisets). Two clauses are *standardized apart* if no variable occurs in both. Given two *parent* clauses $C_1 \vee a_1 \vee \ldots \vee a_m$ and $C_2 \vee \neg b_1 \vee \ldots \vee \neg b_n$ which are standardized apart (a variable renaming substitution may be required) their *resolvent* is the clause $(C_1 \vee C_2)\theta$ where θ is a most general unifier of $\{a_1, \ldots, a_m, b_1, \ldots, b_m\}$. The *atom resolved upon* is $a_1\theta$, and the set of *resolved literals* is $\{a_1, \ldots, a_m, \neg b_1, \ldots, \neg b_m\}$.

It is convenient to define a mapping ρ of literals for the resolution operation. This is used later to define history paths.

DEFINITION 1. (Resolution mapping). *For each resolution operation we define the resolution mapping ρ from each occurrence of a literal c in each parent clause either to the atom resolved upon if c is a resolved literal, or otherwise to the occurrence of $c\theta$ in the resolvent.*

Our presentation follows Robinson's definition of resolution [12], which does not use factoring as a separate operation on a clause. Factoring consists of applying a substitution that unifies two of its literals with the same sign and then removing one of these literals. This operation is not needed in binary resolution trees, since if a clause contains two identical or unifiable literals, both can be resolved upon whenever the clause is used in a resolution. By allowing several literals to be resolved on, instead of merging them before the resolution, we have just one type of internal node in our binary resolution tree, instead of two. (De Nivelle uses resolution nodes and factorization nodes [3].) Moreover, an implementation is free to merge or factor literals if desired. Factoring may be seen as an optimization if the factored clause can be used in several resolution steps, since the factoring is done only once.

A binary resolution derivation is commonly represented by a binary tree, drawn with its root at the bottom. Each edge joins a *parent* node, drawn above the edge, to a *child* node, drawn below it. The *ancestors* (*descendants*) of a node are defined by the reflexive, transitive closure of the parent (child) relation. The *proper ancestors* (*proper descendants*) of a node are those ancestors (descendants) not equal to the node itself. If T and H are nodes in a tree then $path(T, H)$ is the unique path from T to H. Here, T is called the *tail* and H the *head.*

DEFINITION 2. *A* binary resolution tree *on a set S of input clauses is a labeled binary tree. Each node N in the tree is labeled by a* clause label, *denoted $cl(N)$. Each node either has two parents and then its clause label is the result of a resolution operation on the clause labels of the parents, or has no parents and is labeled by an instance of an input*

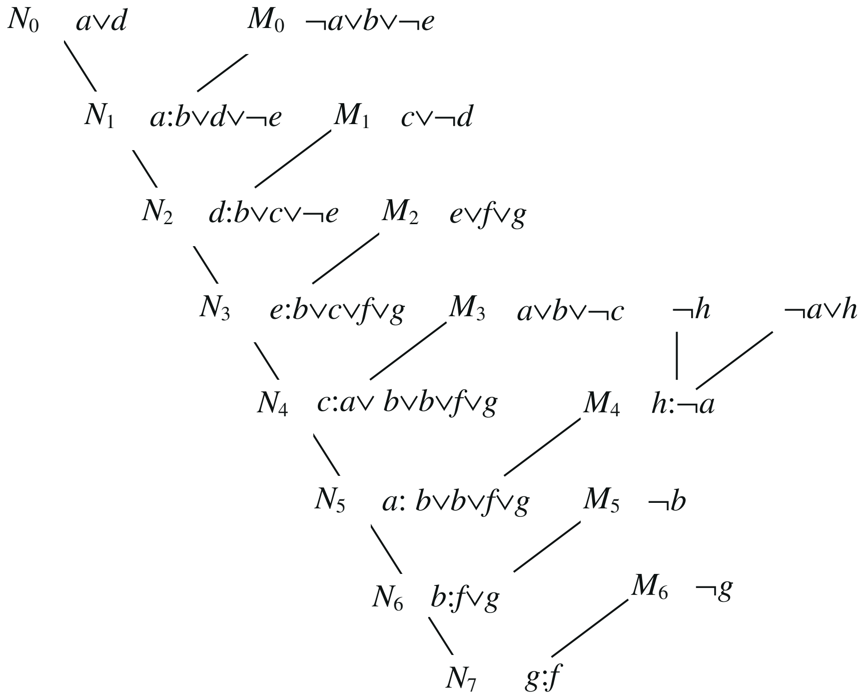

Figure 1. An irregular binary resolution tree.

clause from S. In the case of a resolution, the atom resolved upon is used as another label of the node: the atom label, denoted $al(N)$. Any substitution generated by resolution is applied to all labels of the tree. The clause label of the root of the binary resolution tree is called the result *of the tree, $result(T)$. A binary resolution tree is* closed *if its result is the empty clause, $\Box$.*

For the binary resolution tree in Figure 1, $S = \{a \vee d, \neg a \vee b \vee \neg e, c \vee \neg d, e \vee f \vee g, a \vee b \vee \neg c, \neg h, \neg a \vee h, \neg b, \neg g\}$. The labels of a node N are displayed beside the name of the node and separated by a colon, *e.g.* the node N_4 has atom label c, and clause label $a \vee b \vee b \vee f \vee g$. The order between the parents of a node is not defined.

If instead of labelling the internal nodes by atoms, one labels each edge with the complement of the literal resolved upon, a binary resolution tree would become an upside down semantic tree, and the leaves of the binary resolution tree would become failure nodes of the semantic tree.

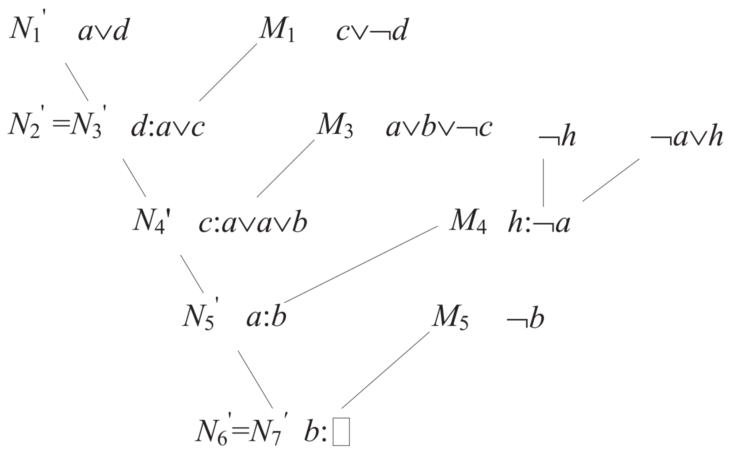

Figure 2. Surgery on Figure 1.

Using the resolution mapping ρ for each resolution operation in the tree, we can trace what happens to a literal from its occurrence in the clause label of some leaf, down through the tree until it is resolved away. Clearly if all literals are eventually mapped to the atom label of some internal node, the clause label of the root is empty. In this case by soundness of resolution, the clause labels of the leaves form an unsatisfiable set. Thus we are primarily concerned about tracing the "history" of a literal starting from its appearance in a leaf.

DEFINITION 3. (History Path). *Let the nodes $(N_0, \ldots, N_n)$ occur in a binary resolution tree T such that N_0 is a leaf whose clause label contains a literal a, and for each $i = 1, \ldots, n$, N_{i-1} is a parent of N_i. Let ρ_i be the resolution mapping from the parents of N_i to N_i. Also let $\rho_i \ldots \rho_2 \rho_1 a$ occur in $cl(N_i)$, so that a is not resolved away at any N_i. Suppose N_n either is the root of T, or has a child N such that $\rho_n \ldots \rho_1 a$ is mapped by the resolution at N to the atom resolved upon at N. Then $P = (N_0, \ldots, N_n)$ is a* history path *for a. The history path is said to* close *at N if N exists. The node N_n is the* head*, the leaf N_0 is the* tail *and a is the* literal *of P, written $head(P), tail(P)$ and $literal(P)$, respectively.*

For example in Figure 1, (M_1, N_2, N_3) is a history path for c which closes at N_4. The two history paths for b in Figure 1, corresponding to the two occurrences of b, are (M_3, N_4, N_5) and $(M_0, N_1, N_2, N_3, N_4, N_5)$.

Both of these close at N_6. The only history path that does not close is the one for f, which is $(M_2, N_3, N_4, N_5, N_6, N_7)$.

DEFINITION 4. (Regular). *A binary resolution tree T is* regular *if there does not exist a node N of T and a descendant M of N, such that $al(N)$ occurs in $cl(M)$.*

The tree in Figure 1 is irregular because $al(N_1)$ is a and a occurs in $cl(N_4)$. Irregular trees are never necessary. Why resolve away the a twice? One could choose to leave out the resolution at N_1, leaving the a in the clause, do the other resolutions as necessary (not all will be necessary) and later resolve a away, as was done at N_5. Operation *Surgery* makes this idea more formal.

A new binary resolution tree T' is constructed from T in which M_0 and all of its ancestors, and possibly other M_i are removed. However all leaves of T' are also leaves of T. Thus every history path in T' has a corresponding history path in T. The converse cannot be true, since the history paths of T through M_0 do not exist in T'. A new sequence $(N'_1, N'_2, \ldots, N'_n)$ is defined in which either $N'_i = N'_{i-1}$, (*i.e.*, M_i is removed) or N'_i corresponds to N_i in T.

OPERATION 1. (Surgery on irregular trees). *Let T be an irregular binary resolution tree with $a = al(N_1)$, $(N_1, N_2, \ldots, N_n)$ being the path of nodes from N_1 to the root N_n, and with N_j, $1 \leq j \leq n$, being the first node in this path whose clause label contains a. Let N_{i-1} and M_{i-1} be the parents of N_i for $i = 1, \ldots, n$. Let N_0 and M_0 be the parents of N_1, so that a occurs in the clause label of N_0 with the same sign as in $cl(N_j)$.*

procedure Surgery(T, N_1)
Let $N'_1 = N_0$ and let all ancestors of N_0 be in T'. (M_0 and all of its ancestors are removed.) Let N_k be the node at which some history path P_a for a in N_j closes, if k exists.
for $i = 2, \ldots, n$ **do**
  if there is a history path in T' containing N'_{i-1} for which the corresponding history path in T closes at N_i **then**
    Put M_{i-1} and all its ancestors into T'.
    Define N'_i as a new node in T' which is the child of N'_{i-1} and M_{i-1}.
    Let the resolution at N'_i be defined so that the history paths of T'_i that close at N'_i correspond to history paths of T that close at N_i.
    if $i = k$, **then**

Let the history path corresponding to P_a also close at N'_k. (That is, the same literals as far as possible are resolved at N'_i as at N_i with one possible addition.)

endif

else

There is no history path containing N'_{i-1} which corresponds to a path that closes at N_i.

Let $N'_i = N'_{i-1}$. (M_{i-1} and all of its ancestors are removed.)

endif

endfor

Note that if the occurrence of a in N_j is never resolved away, *i.e.* its history path continues to the root, then the two literals corresponding to a may occur in the root of T'. However, they are both at least as general as a.

Figure 2 shows the effect of surgery on Figure 1. Surgery is performed at N_1, using N_4 as N_j and $k = 5$. M_0, M_2 and M_6 are not needed in T'. By insisting that both occurrences of a are closed at the same node, we ensure that T' does not have a in its result when T does not. Thus N'_5 closes both history paths for a from N'_1 and M_3.

THEOREM 1. *Let T be an irregular binary resolution tree on a set S of clauses and let T' be constructed by Operation* Surgery *on T. Then T' is also a binary resolution tree on S, T' is smaller than T and the result of T' subsumes the result of T.*

Proof. We use the following lemma: If clauses C_1 and C resolve to give R_1, and if clause C_2 subsumes C_1 then either the resolution of C_2 and C is not possible and C_2 subsumes R_1, or it is possible and its result, R_2, subsumes R_1. This assumes that all literals from C resolved in the first resolution are also resolved in the second, and furthermore that literals from C_2 are resolved in the second resolution if they correspond (in the subsumption) to literals from C_1 resolved in the first resolution.

Each leaf in T' has the same label as a leaf in T and therefore T' is defined on S. Also, each internal node is defined by a resolution of its parents, so T' is a binary resolution tree. Note that $cl(N'_1) = cl(N_0)$ subsumes $cl(N_1) \vee a$ because $cl(N_1)$ contains all the literals in $cl(N_0)$ except possibly a. Using repeated applications of the lemma, it follows that $cl(N'_i)$ subsumes $cl(N_i) \vee a$ for $i = 2, \ldots, n$. Also $cl(N'_j)$ subsumes $cl(N_j)$ since a occurs in $cl(N_j)$. Then $cl(N'_i)$ subsumes $cl(N_i)$ for $i = j+1, \ldots, n$, so the result of T' subsumes that of T. Since M_0 is not in T' and since all other nodes in T' are taken at most once from T, it follows that T' has fewer nodes than T. $\square$

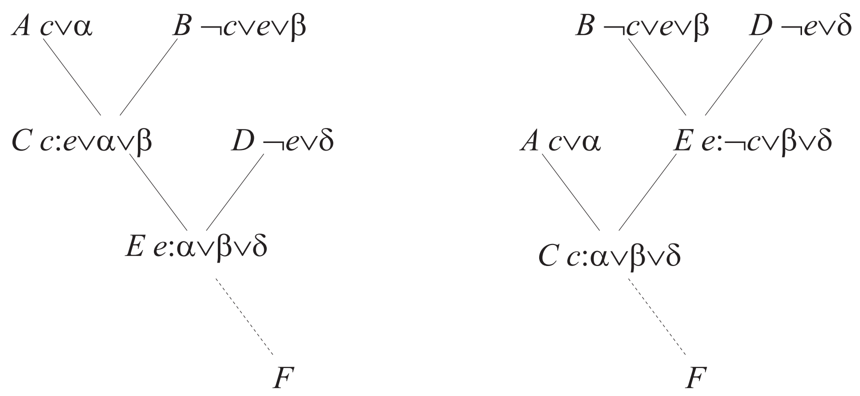

Figure 3. A binary tree rotation

THEOREM 2. (Completeness [15]). *If S is unsatisfiable there exists a closed regular binary resolution tree on S. Furthermore the smallest closed binary resolution tree is regular.*

Proof. If S is unsatisfiable, there exists a closed binary resolution tree [12]. If it is irregular, apply Operation *Surgery* repeatedly until it is regular. This process must terminate since the tree is smaller at each step.

If the smallest closed binary resolution tree is not regular, surgery can be applied to it, making a smaller closed tree. □

3. Minimal Binary Resolution Trees

A *rotation* of an edge in a binary tree is a common operation in computer science, for example with AVL trees [1]. Before we apply it to binary resolution trees, we review the operation on binary trees. Given the binary tree fragment on the left of Figure 3, a rotation is the reassignment of edges so that the tree on the right of Figure 3 is produced. The parent C of E becomes the child of E and the parent B of C becomes the parent of E. If E has a child in T, then C takes that child in T'. In other words, the edges (B, C), (C, E) and (E, F) if F exists, are replaced by the edges (B, E), (E, C) and (C, F) if necessary.

OPERATION 2. (Edge Rotation). *Let T be a binary resolution tree with an edge (C, E) between internal nodes such that C is the parent* *of E and C has two parents A and B. Further, suppose that no history path through A closes at E. Then the result of a* rotation *on this edge is the binary resolution tree T' defined by resolving $cl(B)$ and $cl(D)$ on $al(E)$ giving $cl(E)$ in T' and then resolving $cl(E)$ with $cl(A)$ on $al(C)$ giving $cl(C)$ in T'. Any history path closed at C in T is closed at C in T'; similarly any history path closed at E in T is closed at E in T'. Also, the child of E in T, if it exists, is the child of C in T'.*

A rotation may introduce tautologies to clause labels of internal nodes. For instance, if $al(C)$ occurs in $cl(D)$ in T, then $cl(E)$ in T' may be tautological. However the clause label of the root is not changed up to variable renaming (Corollary 4). We prove a slightly more general result first, which is also used later.

DEFINITION 5. *Let T_1 and T_2 be two binary resolution trees defined on the same set of input clauses. Then T_1 and T_2* resolve input literals similarly *if there is a one-to-one and onto mapping ν from nodes in T_1 to those in T_2, such that:*

1. *If N is a leaf then $\nu(N)$ is a leaf and both are labeled with the same instance (up to variable renaming) of an input clause. Thus there is a natural one to one correspondence, from literals in $cl(N)$ to those in $cl(\nu(N))$. Moreover this mapping of literals provides a mapping from history paths in T_1 to those in T_2, defined so that they start from the same literal in the input clause, up to variable renaming. We represent these other two mappings also with ν. We require for all history paths P in T_1 that $tail(\nu(P)) = \nu(tail(P))$ and $literal(\nu(P)) = literal(P)$ up to variable renaming.*

2. *For every history path P of T_1, P closes at a node N if and only if $\nu(P)$ closes at $\nu(N)$.*

Thus two binary resolution trees resolve input literals similarly if they resolve the same input literals against each other, albeit in a possibly different order.

LEMMA 3. *If two binary resolution trees T_1 and T_2 resolve input literals similarly, the result of T_1 and the result of T_2 are the same, up to variable renaming.*

Proof. Note that $result(T_1)$ and $result(T_2)$ are composed entirely of literals from history paths that do not close, and since the same history paths are closed in each, the same literals are not resolved away. Also the composition of mgus in T_1 and that in T_2 are unique up to variable renaming, since given a node N, the same literals are unified at N and $\nu(N)$, up to variable renaming. □

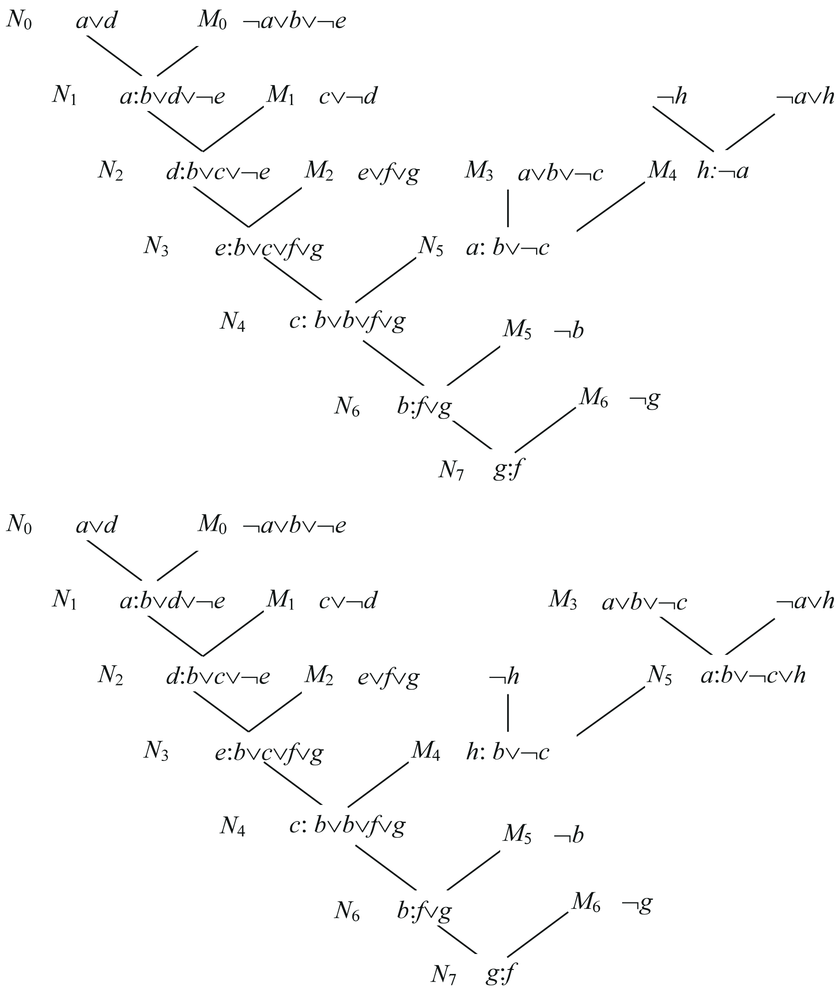

Figure 4. From Figure 1 rotate (N_4, N_5), then (M_4, N_5)

COROLLARY 4. *Given a binary resolution tree T with an internal node C and its child E, Operation* Edge Rotation *generates a new binary resolution tree and $cl(C) = cl(E)$ up to variable renaming.*

Proof. Observe that Operation *Edge Rotation* produces a tree which resolves input literals similarly. □

A rotation changes the order of two resolutions in the tree. Rotations are invertible; after a rotation, no history path through D closes at C, so another rotation at (E, C) can be done, which generates the original tree again. We say that two binary resolution trees are *rotation equivalent* if one can be generated from the other by a sequence of rotations. For instance, the first binary resolution tree in Figure 4 is produced by rotating the edge (N_4, N_5) in Figure 1. The second tree in Figure 4 is then produced by rotating the edge (M_4, N_5). Thus both trees are rotation equivalent to Figure 1. Rotation equivalent is an equivalence relation. It is not surprising that rotation equivalent binary resolution trees must resolve input literals similarly, but the converse is true as well.

THEOREM 5. *Two binary resolution trees T_1 and T_2 are rotation equivalent if and only if they resolve input literals similarly.*

Proof. ($\Leftarrow$) Since one rotation of T_1 creates a binary resolution tree that resolves input literals similarly to it, so too does the sequence of rotations creating T_2.

($\Rightarrow$) The converse is proved by induction on the number of internal nodes. Suppose T_1 and T_2 resolve input literals similarly. Then they must have the same number n of internal nodes since they have the same number of leaves. If $n = 0$ or $n = 1$ then no rotation is possible and the theorem holds. Let N be a node in T_1 with parents L_1 and L_2 that are leaves. Then in T_2, $\nu(N)$ has proper ancestors $\nu(L_1)$ and $\nu(L_2)$ which also are leaves, and $\nu(N)$ closes only history paths with tails $\nu(L_1)$ and $\nu(L_2)$. We create T_2' by rotating edges of T_2 so that $\nu(L_1)$ and $\nu(L_2)$ become parents of $\nu(N)$, if this is not already the case. Let C be either parent of $\nu(N)$ and let A and B be the parents of C. If $\nu(L_1)$ and $\nu(L_2)$ are both ancestors of C then neither is an ancestor of the other parent of $\nu(N)$. But $\nu(N)$ must close a history path from that other parent, contradiction. Thus the edge $(C, \nu(N))$ can be rotated, since not both A and B contain a history path closing at $\nu(N)$. This rotation reduces the total number of non-leaf ancestors of $\nu(N)$. After a finite number of such rotations, both parents of $\nu(N)$ are leaves. Call this tree T_2'.

Let T_1^* be T_1 with leaves L_1 and L_2 deleted, and let T_2^* be T_2' with leaves $\nu(L_1)$ and $\nu(L_2)$ deleted. Then T_1^* and T_2^* resolve input literals similarly since T_1 and T_2' resolve input literals similarly. By induction T_1^* and T_2^* are rotation equivalent. The sequence of rotations to convert T_1^* to T_2^* will also convert T_1 to T_2' which is rotation equivalent to T_2. $\square$

We focus on the set of rotation equivalent trees that do not contain an irregular binary resolution tree. Any tree in this set is said to be minimal, since surgery cannot be applied to make it smaller.

DEFINITION 6. *A binary resolution tree T is* minimal *if no sequence of rotations of edges generates a tree T' that is irregular.*

THEOREM 6. *If a binary resolution tree T on S is non-minimal, there exists a minimal binary resolution tree T' on S which is smaller than T and the result of T' subsumes the result of T.*

Proof. If T is not minimal, apply edge rotations and surgery operations so that a regular tree is produced. If this tree is minimal then let T' be this tree. Otherwise repeat from the beginning until T' is defined. This process must terminate because the tree is getting smaller at each application of Operation *Surgery*. Also the old result is subsumed by the new result at each step. $\square$

4. Checking Minimality

Determining whether a given binary resolution tree is minimal seems to be labourious, since the straightforward application of the definition, as is done in the proof of Theorem 6, checks every possible sequence of rotations, and there can be exponentially many. In this section we give an efficient algorithm for determining visibility – which nodes can be rotated below which – so deciding minimality is efficient.

DEFINITION 7. (Visible). *In a given binary resolution tree with internal nodes N and M, we say that M is* visible *from N, and that N can* see *M, if there exists a sequence of rotations such that M is a descendant of N. Otherwise M is invisible from N.*

Thus a node can see the nodes that can be rotated below it. Although this is a property defined in terms of rotations, it is possible to inspect a static tree, without doing any rotations, to determine visibility. Because of this, visibility can be computed in linear time. That static property, holds, is defined after one more concept, precedes.

DEFINITION 8. (Precedes). *A history path P* directly precedes *a history path Q if P and Q have no nodes in common, and P closes at some node in Q. We write $P \prec Q$. Moreover we say P* precedes *Q, and write $P \prec^* Q$ if there is a sequence of history paths $(P_1, \ldots, P_k)$ with $P = P_1$ and $Q = P_k$ and P_i directly precedes P_{i+1} for $i = 1, \ldots, k-1$. A history path P* precedes *a node N if N closes some history path Q and $P \prec^* Q$.*

The precedes relation on history paths is the reflexive and transitive closure of directly precedes. In particular a history path precedes itself, even though it does not directly precede itself. Also note that precedes defines a partial order on the set of history paths.

In most cases a rotation does not change the precedes relation on history paths.

LEMMA 7. *Let the history path P precede the history path Q in the binary resolution tree T and let P', Q' and T' be the images of P, Q and T respectively after a rotation of the edge (C, E) in T as in the definition of edge rotation. Further suppose that in T the head of Q is not C. Then P' precedes Q' in T'.*

Proof. If the tail of P is not an ancestor of E then the rotation cannot affect whether $P' \prec^* Q'$. Assume that $tail(P)$ is an ancestor of E. Let T_A, T_B and T_D be the subtrees of T rooted at A, B and D respectively. If the node at which P closes and $head(Q)$ are in the same one of these subtrees, again the rotation has no effect on whether $P' \prec^* Q'$. Assume that they are not in the same subtree. We know that $head(Q)$ is a descendant of the node at which P closes, so $head(Q)$ cannot be in any of the subtrees T_A, T_B or T_D. Also $head(Q)$ is not C so $head(Q)$ must be a (non-strict) descendant of E.

Now if $tail(P)$ and $tail(Q)$ are in the same subtree T_A, T_B or T_D, then the rotation does not affect $P' \prec^* Q'$ because the node of Q that P precedes is also in that subtree.

Assume that $tail(P)$ and $tail(Q)$ are not in the same subtree T_A, T_B or T_D. Let P_1 be the path closing at the node of Q that P precedes. Thus $P \prec^* P_1 \prec Q$. We have these cases, which are illustrated in Figure 5.

1. $tail(Q)$ is in T_A. If $tail(P)$ is in T_B then P_1 closes at C. After the rotation, P_1', which is one node longer, closes at C in T'. $P' \prec^* P_1' \prec Q'$.

   Otherwise $tail(P)$ is in T_D. Let P_0 denote a path through B closing at C. After the rotation $P' \prec^* P_1' \prec P_0' \prec Q'$.

2. $tail(Q)$ is in T_B. If $tail(P)$ is in T_A, then after the rotation $P' \prec^* P_1' \prec Q'$.

   Otherwise if $tail(P)$ is in T_D then after the rotation, $P' \prec^* P_1' \prec Q'$.

3. $tail(Q)$ is in T_D. Since no path through A closes at E, we know $tail(P)$ is not in T_A. Thus $tail(P)$ is in T_B. After the rotation, $P' \prec^* P_1' \prec Q'$.

4. $tail(Q)$ is not an ancestor of E. In T, there is a path R such that $P \prec R \prec Q$ and E is on R. By the cases above $P' \prec R'$ in T'. Also $R' \prec Q'$ in T' since the precedes relation of paths that close below E are not affected by the rotation. $\square$

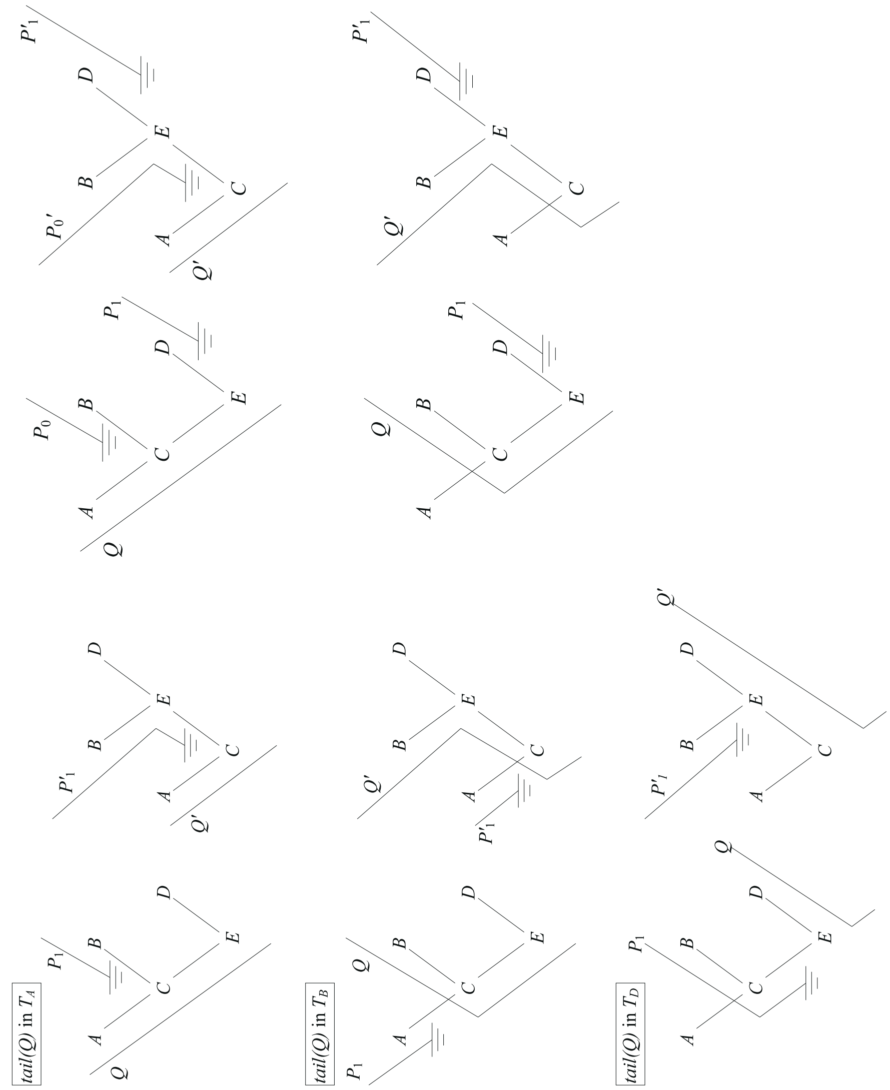

Figure 5. Cases of Lemma 7

DEFINITION 9. (Hold). *Two history paths* P_1 *and* Q_1 directly hold *an internal node* M *of a binary resolution tree if* M *is the first node that occurs on both* P_1 *and* Q_1*, (i.e. neither parent of* M *occurs on both.) We say that history paths* P *and* Q hold M *if there exist history paths* P_1 *and* Q_1 *such that* $P_1 \prec^* P, Q_1 \prec^* Q$ *while* P_1 *and* Q_1 *directly hold* M*. A node* N holds M *if* P *and* Q *hold* M *and both* P *and* Q *close at* N*. Also, if* P *and* Q *hold* M *and are in a set of history paths, we say that the set holds* M*.*

The following theorem relates "invisible", a property that depends on all rotation equivalent trees, to "holds", a property that can be checked by examining just the one tree of interest. Consider the second tree of Figure 4. In this tree N_5 is visible from N_4, and hence also from N_1. This can be seen by undoing the rotations performed, resulting in Figure 1, where N_5 is a descendant of N_4. This can also be observed directly, without undoing the rotations, by noting that N_4 does not hold N_5 in the second tree of Figure 4. The history paths above M_4 are shown in the first part of Figure 6. No path from L_2 precedes N_4.

When N_5 is rotated below N_4 it becomes a descendant of N_1 also, and this brings about an opportunity for surgery to be applied.

Now if $\neg c$ had occurred at L_2, and was resolved away at N_4, then a new history path would exist from L_2 to M_4. Then N_5 would be directly held by N_4 via the two paths for $\neg c$, as shown in the second part of Figure 6. If $\neg c$ were instead added to L_1, then a new history path from L_1 to M_4 would exist and N_5 would be directly held by the path for h and a path for $\neg c$. N_5 would be held by both paths for $\neg c$ closing at N_4, since the path for h directly precedes the new path for $\neg c$. This is shown in the third part of Figure 6. In both the second and third trees N_5 is not visible from N_4.

THEOREM 8. *In a binary resolution tree, the nearest common descendant of two internal nodes* M *and* N *holds* M *if and only if* M *is invisible from* N*.*

Proof. ($\Rightarrow$) We show that if the nearest common descendant of M and N holds M, then after a rotation, the (possibly new) nearest common descendant of M and N holds M. Thus M can never be a descendant of N, for if it were then the nearest common descendant would be M and a node cannot hold itself. Cases in the proof of the forward direction are shown in Figure 7.

Let F be the nearest common descendant of N and M, let the rotated edge be (C, E), and let nodes A, B and D adjacent to it be as defined in Operation *Edge Rotation*. Let P and Q hold M and close at F, while P_1 and Q_1 directly hold M, and $P_1 \prec^* P$ and $Q_1 \prec^* Q$.

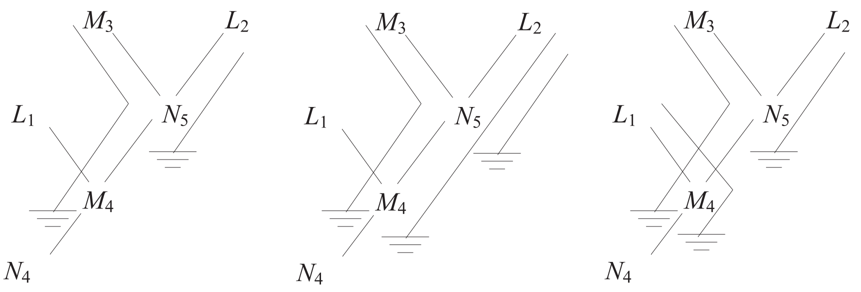

Figure 6. N_4 does not hold N_5 in the first tree, but does in the others.

In Cases 0 and 1, suppose $F \neq E$, so that after the rotation F is still the nearest common descendant of M and N. Then C is not the head of P or Q. By Lemma 7, $P_1' \prec^* P'$ and $Q_1' \prec^* Q'$. Suppose (Case 0) $M \neq E$. Then after the rotation, M is still the first common node on P_1' and Q_1', so F still holds M.

Now suppose (Case 1) that $M = E$. Without loss of generality assume that P_1 contains C and Q_1 contains D. If (Case 1a) P_1 contains B then after the rotation, P_1' and Q_1' still hold E, so F holds M. If (Case 1b) P_1 contains A then consider the path R containing B and closing at C. After the rotation, $R' \prec P_1' \prec^* P'$, so that R' and Q_1' hold E, so again F holds M.

For Cases 2 and 3, suppose that $F = E$. If (Case 2) M is an ancestor of C then N is either E or an ancestor of D. Since no history path can contain A and C and close at E, $M \neq C$. For the same reason, P and Q contain B and close at E. If (Case 2a) M is an ancestor of A then the paths that directly precede P and Q close at C and hold M. Thus C holds M. After the rotation the nearest common descendant of M and N is C, and C still holds M. Otherwise (Case 2b) if M is an ancestor of B then after the rotation the nearest common descendant of M and N is E and E still holds M.

Finally consider the case (Case 3) where M is an ancestor of D. If (Case 3a) N is either E or an ancestor of B then after the rotation, E still holds M and is still the nearest common descendant of N and M. If (Case 3b) N is C or is an ancestor of A, then consider path R with head at B which closes at C. After the rotation, the nearest common descendant of M and N is C, while R' directly precedes both P' and Q'. Thus C holds M after the rotation.

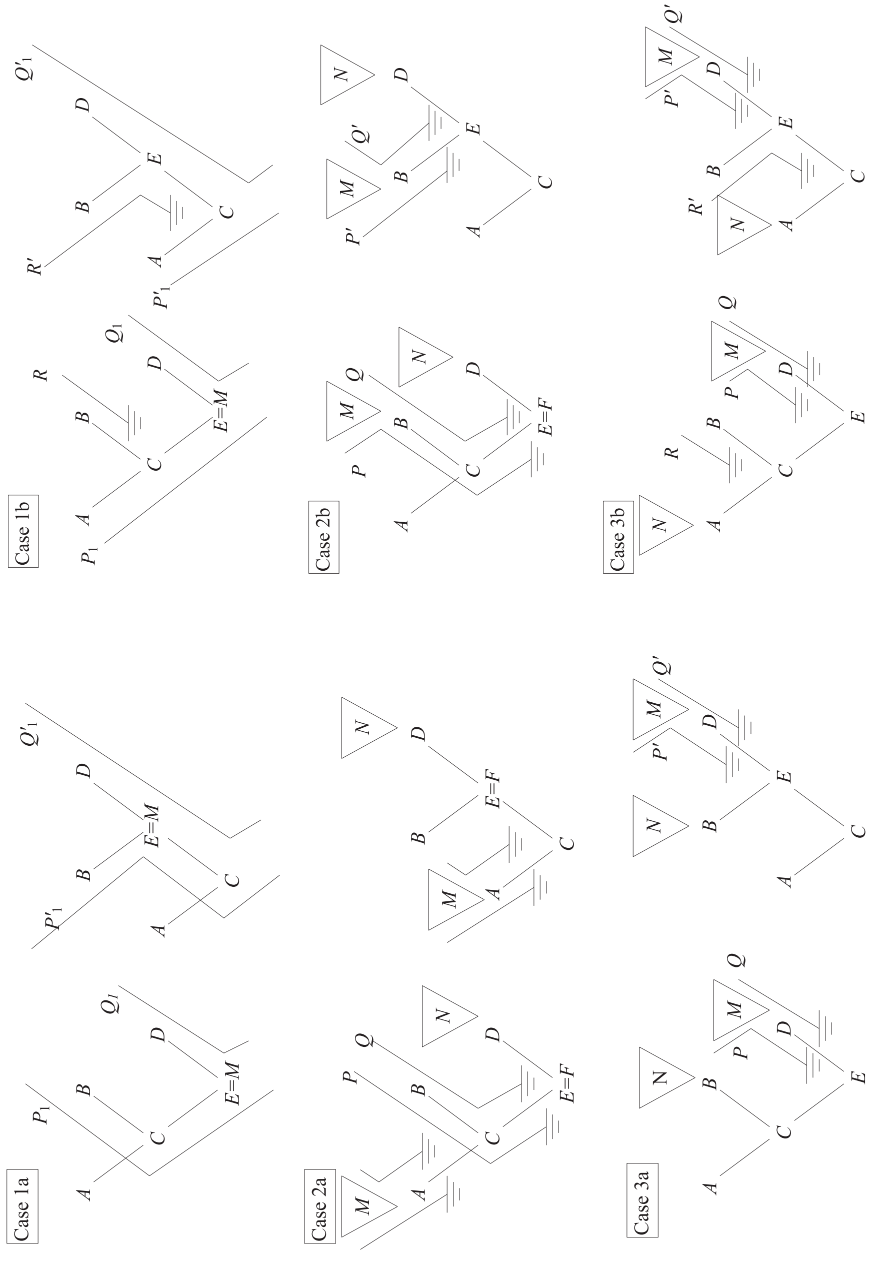

Figure 7. Cases of Theorem 8

($\Leftarrow$) Conversely, suppose that M is not held by the nearest common descendant F of M and N. We generate a sequence of rotations so that M is a descendant of N. If $F = M$ then M is a descendant of N and therefore visible from N. If F is the child of M then the edge (M, F) can be rotated because M is not held by F. This rotation makes M a descendant of F and therefore of N. Now suppose F is a proper descedant of the child of M. There exists a path $(H_0, H_1, \ldots, H_n)$ for $n > 1$ where $F = H_n$, $M = H_1$, so that H_{i-1} and K_{i-1} are the parents of H_i, for $i = 1, \ldots, n$. We use induction on n. Without loss of generality K_0 is chosen so that $al(H_2)$ occurs in $cl(K_0)$; thus a history path through K_0 closes at H_2. If there exists i in $\{2, \ldots, n\}$ such that no history path through H_{i-2} closes at H_i then the edge (H_{i-1}, H_i) can be rotated, shown as Rotation 1 in Figure 8. We say that H_i has been rotated to the side of the $path(M, F)$, and so M is now closer to F and the theorem holds by induction.

Thus for each i in $\{2, \ldots, n\}$, some history path through H_{i-2} closes at H_i. If there is no j in $\{1, \ldots, n\}$ such that H_j is held by F then in particular H_{n-1} is not held so the edge (H_{n-1}, H_n) can be rotated, rotating N_{n-1} off the bottom of the path. This is shown in Figure 8 as Rotation 2. But then the distance from M to the nearest common descendant of M and N is $n - 1$ edges, and the theorem holds by induction. Choose the smallest j in $\{1, \ldots, n\}$ so that H_j is held by F. Note that $j \neq 1$ by assumption. If there is a history path through K_{j-2} closing at H_j then this path and the path through H_{j-2} closing at H_j directly hold H_{j-1}. Thus H_{j-1} is held by F, contradicting the definition of j. Therefore no such path through K_{j-2} exists and the edge (H_{j-1}, H_j) can be rotated as illustrated in Rotation 3 of Figure 8. If $j = 2$ then H_1 is closer to F after the rotation; we say that H_j has been rotated off the top of the $path(M, F)$, and the theorem holds by induction. Otherwise consider the history paths through H_{j-2} closing at H_j. If one of these includes H_{j-3} and another includes K_{j-3} then H_{j-2} is held by F after the rotation. This means that H_{j-2} was held by F before the rotation, contradicting the definition of j. Thus the edge (H_{j-2}, H_{j-1}) can be rotated. The result of this rotation is that H_j will be rotated either to the side of the $path(M, F)$, or closer to M. By a second induction on the distance from M to H_j, H_i will eventually be rotated off the top or to the side. Thus the distance from M to F will decrease by the constructed sequence of rotations. $\square$

The sequence of rotations, constructed in the proof of the converse direction of Theorem 8 rotates the visible node M to below the node N that can see it. Thus if the tree were regular but non-minimal, the rotations would make it irregular, exposing the non-minimal parts.

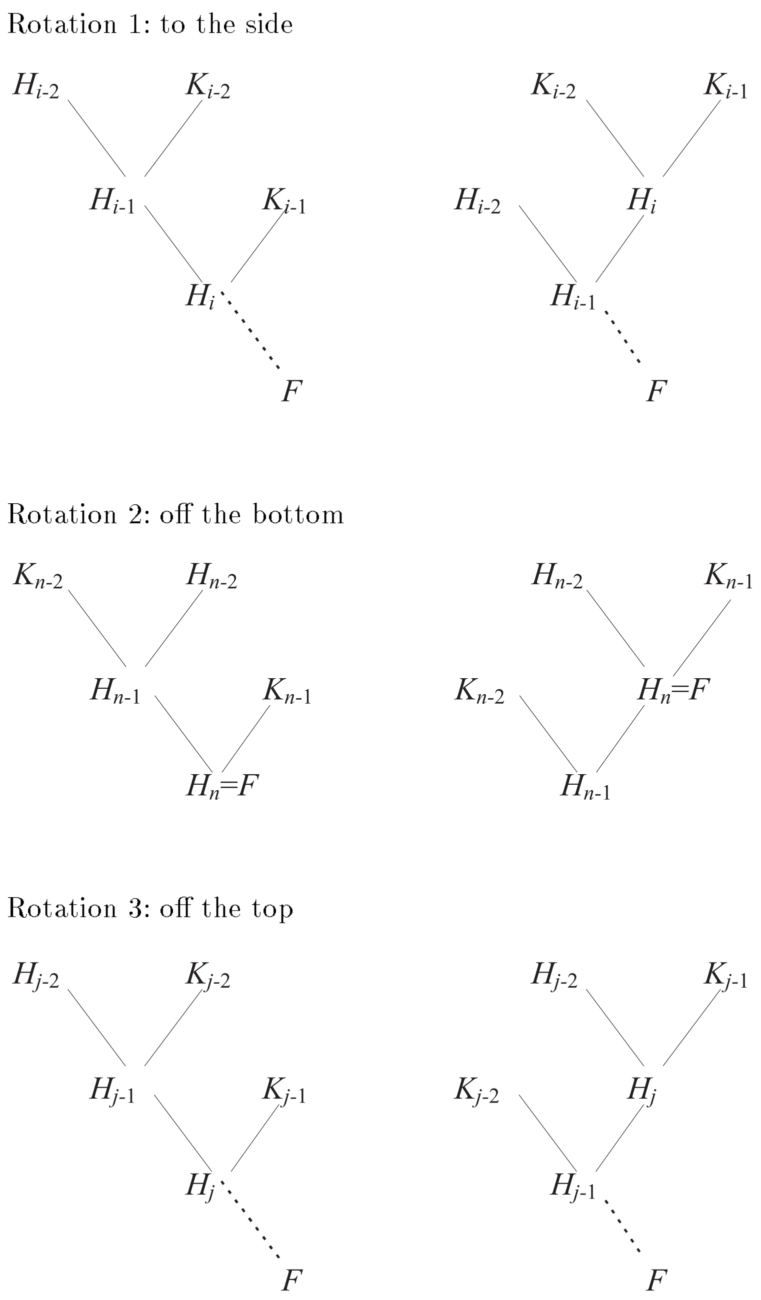

Figure 8. Rotations in the proof of Theorem 8

Using surgery, we can remove all non-minimal sections of a tree, and reconstruct a minimal tree from the pieces left behind.

As an example, consider the second tree of Figure 4, where N_5 is visible from N_1. We generate the sequence of rotations to bring N_5 below N_1. The common descendant is N_4. The path of H's is (M_3, N_5, M_4, N_4). The first step of the construction in the converse proof above, to show the visibility, is to rotate edge (N_5, M_4), as no path through M_3 closes at M_4. This rotates M_4 to the side, to give the top tree of Figure 4. Now the path of H's is (M_4, N_5, N_4). Since no path through M_4 closes at N_4, N_4 can be rotated to the side by rotating edge (N_5, N_4). N_5 is now a descendant of N_1 as in Figure 1.

Unfortunately the number of rotations required to expose a non-minimality may be quadratic in the length of the path in the tree. Since the trees are not balanced, this is quadratic in the size of the tree in the worst case. Later we give a linear time operation, called *Splay*, to bring a visible node to a descendant. In the next section, however, we detect and avoid non-minimal trees.

5. Minimality as a restriction

Now we turn our attention to a theorem prover that keeps only minimal binary resolution trees. Since every non-minimal tree is subsumed by some minimal tree, this strategy uses the minimal criteria to reduce redundancy, lessening reliance on subsumption.

DEFINITION 10. *Let T be a binary resolution tree. Then $atoms(T) = \{al(N) | N$ is a node of $T\}$ is called the* set of atoms *of T. A* subbrt *of T is a binary resolution tree whose node set consists of a node of T called the root of the subbrt, and all of its ancestors. For a subbrt T' of T, $Vis(T') = \{al(N) | N$ is a node of T' and visible from the root of $T\}$ is called the set of* visible atoms of T'.

THEOREM 9. *Let a binary resolution tree T consist of a root node R and two subbrt's T_1 and T_2. T is minimal if and only if*

1. *T_1 and T_2 are minimal,*
2. *no atom in $cl(R)$ is in $atoms(T)$,*
3. *$atoms(T_1) \cap Vis(T_2) = \phi$, and*
4. *$atoms(T_2) \cap Vis(T_1) = \phi$.*

Proof. Assume that T is minimal. If T_1 or T_2 were not minimal, then there would be a sequence of edge rotations which would make the subbrt irregular. The same sequence performed on T would make T irregular as well. Hence the first condition is true. If the second condition were false, then T would be irregular immediately. Assume that the third condition is false. Then there are two nodes, $N \in T_1$ and $M \in T_2$ whose atom labels are the same, and M is visible from R. Also M can be rotated below R, without rotating any edges in T_2. Now M is a descendant of N, making T irregular, because the parent of M between N and M must have $al(N)$ in its clause label. The fourth condition is symmetric to the third.

Conversely, assume that T is not minimal. Then there is a sequence of rotations that create an irregular tree T'. Some node N has a descendant M_1 in T' such that $al(N)$ occurs in $cl(M_1)$. Since the rotations do not change $cl(R)$, if $al(N)$ occurs in the result of T', it occurs in $cl(R)$ in T and then T violates the second condition. Thus $al(N)$ does not occur in $cl(R)$ so there is a descendant M of N in T' such that $al(N) = al(M)$. Back in T, M is visible from N. Hence in T by Theorem 8, the nearest common descendant of M and N, call it F, does not hold M. If M and N are in the same T_i then F is also in T_i and does not hold M. Thus M is visible from N in T_i. Then T_i is not minimal and violates the first condition.

Assume M and N are in different T_i. Since M has been rotated below N, M is visible from N in T, and by Theorem 8 M is not held by the nearest common descendant R of M and N. Thus M is visible from R. Therefore $al(M)$ is in $Vis(T_i)$ while $al(N)$ is in $atoms(T_{3-i})$. This violates condition 3 or 4. □

Any theorem prover based on binary resolution that keeps only minimal trees has already satisfied the first condition in Theorem 9 for any newly constructed tree, since only minimal trees are used in the construction. It is easy to check that the new result does not contain an atom in $atoms(T)$. What is left is to find is an easy way to calculate those atoms in each subbrt T_i which are visible from the root of T.

$Vis(N, \mathcal{P}_N)$ returns the atoms labels of nodes at and above node N in T visible from the root on T, where $\mathcal{P}_N$ is the paths through N that precede the root. The idea in procedure *Vis* is that a node is visible in from the root if and only if it is not held by the root, by Theorem 8. So for each node N we need to calculate the history paths going through it that precede the root. If some of these paths go through one parent of N, and some go through the other, then N is held by the root; otherwise N is visible from the root.

In each procedure call $Vis(N, \mathcal{P}_N)$, the first argument N is a node in the tree and we want to know whether or not it is held by the root.

Initially it is one of the parents of the root, but as we traverse upward it becomes instantiated with each ancestor of this parent. The invariant we maintain is that the second argument, $\mathcal{P}_N$, is the set of paths that include N and precede the root. The paths through the parent of the root that precede the root are simply those whose heads are that parent, so the invariant is easy to establish in the first place. Suppose N has parents A and B. To calculate the paths $\mathcal{P}_B$ through B that precede the root, we start with $\mathcal{P}_N$, and remove those paths that go through A if there are any. If there are none then $\mathcal{P}_B$ is $\mathcal{P}_N$. If there is some path in $\mathcal{P}_N$ through A, we need to add to $\mathcal{P}_B$ any path with head at B, since these paths precede that path through A, and thus precede the root.

Since we know that the paths in $\mathcal{P}_N$ must go through at least one of the parents of N, we assume that B is that parent.

procedure Vis$(N, \mathcal{P}_N)$
if N is a leaf **then return** ϕ
else
Let A and B be the parents of N and partition $\mathcal{P}_N$ into $\mathcal{P}_A$ and $\mathcal{P}_B$, which are the sets of paths which go through A and through B, respectively. Assume without loss of generality that B is chosen so that $\mathcal{P}_B$ is nonempty.
Let $\mathcal{C}_A$ and $\mathcal{C}_B$ be the sets of history paths with heads at A and B respectively, and hence close at N.
if $\mathcal{P}_A$ is nonempty **then**
//N is held
return Vis$(A, \mathcal{P}_A \cup \mathcal{C}_A) \cup Vis(B, \mathcal{P}_B \cup \mathcal{C}_B)$;
else
//$\mathcal{P}_A = \phi$ and N is not held, so it is visible
return $\{al(N)\} \cup Vis(A, \mathcal{C}_A) \cup Vis(B, \mathcal{P}_B)$
endif
endif

The third and fourth conditions in Theorem 9 require us calculate $Vis(T_i)$ which is done by calling $Vis(R_i, \mathcal{Q}_i)$ where R is the root of the binary resolution tree, R_1 and R_2 are the parents of R, and $\mathcal{Q}_i$ is the set of paths with heads at R_i.

Procedure *Vis* runs in a number of set union calculations which is proportional to the number of nodes in the tree. With hashing, these operations can in principle be performed in time proportional to the size of the clauses. Hence Vis is a linear time algorithm, which is as fast as one could expect.

6. Restoring Minimality

There are two basic ways to restore minimality. If a binary resolution tree is irregular then the surgery operation removes the non-minimality. For a binary resolution tree that is regular but non-minimal, some rotations need to be done to make it irregular – we call this *exposing* the non-minimality. This irregularity involves two internal nodes M and N that resolve upon the same atom a and neither is a descendant of the other but (at least) one is visible from the other. Suppose M is visible from N. When M is rotated below N, one parent of M also becomes a descendant of N. Since a occurs in that parent's clause label, surgery from N can be done.

In the proof of the converse of Theorem 8 we gave a quadratic length sequence of operations for moving visible nodes down to become descendants. Operation *Splay* uses a linear number of operations. It is related to the splay operation [13] for binary trees in that it brings a node closer to the root. In our case, to splay a binary resolution tree at M, we divide the descendants of M into two sets: the observers O for those that can see M, and the non-observers $\overline{O}$ for those that cannot. Then we rearrange the tree so that all the nodes in O are ancestors of M, leaving just those nodes in $\overline{O}$ as descendants of M. Thus M is brought as close to the root as possible. This is done in a way that guarantees N is an ancestor of M, so that surgery can be accomplished.

The first step in the *Splay* is to determine the *observers* of M, which are the descendants of M that can see M. Procedure *Obs* does this. Given a node M in a binary resolution tree T such that $\mathcal{P}_1$ is the complete set of history paths containing both M and one parent of M and $\mathcal{P}_2$ is the paths containing M and the other parent, $Obs(M, \mathcal{P}_1, \mathcal{P}_2)$ returns the set of proper descendants of M that can see M.

procedure Obs($M, \mathcal{P}_1, \mathcal{P}_2$)
 if M is the root of T **then**
  return ϕ
 else
  Let D be the child of M and let K be the other parent of D.
  Let $\mathcal{C}_1$ be the paths from $\mathcal{P}_1$ that close at D and $\mathcal{C}_2$ be the paths from $\mathcal{P}_2$ that close at D.
  Let $\mathcal{K}$ be the paths that contain both K and D.
  if $\mathcal{C}_1 \neq \phi$ and $\mathcal{C}_2 \neq \phi$ **then**
   //D holds M
   return $Obs(D, \mathcal{K} \cup \mathcal{P}_1 - \mathcal{C}_1, \mathcal{K} \cup \mathcal{P}_2 - \mathcal{C}_2)$
  else if $\mathcal{C}_2 = \phi$ **then**
   return $\{D\} \cup Obs(D, \mathcal{K} \cup \mathcal{P}_1 - \mathcal{C}_1, \mathcal{P}_2)$

else
$//\mathcal{C}_1 = \phi$
return $\{D\} \cup Obs(D, \mathcal{P}_1, \mathcal{K} \cup \mathcal{P}_2 - \mathcal{C}_2)$
endif
endif

Every time *Obs* is called, every pair of paths one from $\mathcal{P}_1$ and the other from $\mathcal{P}_2$ holds the initial M. If we ever find a descendant D that closes a member of each, then D holds M and so cannot see M and therefore D is not returned; otherwise it is returned as part of the final set of observers. To maintain the invariant as we go from M to D, if D closes paths in $\mathcal{P}_i$ we need to add to $\mathcal{P}_i$ all of the paths $\mathcal{K}$ that come from K through D, because each one of these paths precedes a path that contains M. If D does not close any path through $\mathcal{P}_i$, then we have nothing to add to $\mathcal{P}_i$. In either case we remove from $\mathcal{P}_i$ any path that closes at D since these do not contain D.

Operation *Splay* uses code similar to procedure *Obs*, but not the procedure itself, to decide how the resolutions below M should be reordered in the new tree. The parents S_1 and S_2 of M are considered stubs to be built upon. The observers of M are put into one of two queues Q_1 and Q_2, and the non-observers are put into Q_3. Along with each node put in the queue, we also put the subtree rooted at the other parent of the node, so that the resolution can be reconstructed later. Those put into Q_1 become descendants of S_1, while those in Q_2 become descendants of S_2. After this is done, the two subtrees are joined by a resolution corresponding to M. Finally all of the nodes in Q_3 are made descendants of M. Each of the resolutions is done so that all of the history paths closed at a node in the given tree are closed at the same node in the constructed tree. Thus the resulting tree resolve input literals similarly with the given tree.

OPERATION 3. (Splay). *A splay at an internal node M in a binary resolution tree T produces a new binary resolution tree T' such that all descendants of M cannot see M in T'.*

All resolutions in this operation must be done so that the resulting binary resolution tree resolves input literals similary with T.

procedure Splay(T, M)
Initially Q_1, Q_2 and Q_3 are three empty queues.
Initially $\mathcal{P}_1$ is the set of history paths through M and one parent S_1 of M, while $\mathcal{P}_2$ is the set of history paths through M and the other parent S_2 of M.
call Splay1$(M, \mathcal{P}_1, \mathcal{P}_2)$
return ProcessQ(
resolve(ProcessQ(S_1, Q_1), ProcessQ(S_2, Q_2)),

Q_3)

procedure Splay1($M, \mathcal{P}_1, \mathcal{P}_2$)
 if M is the root of T **then**
   return
 else
   Let D be the child of M and let K be the other parent of D.
   Let $\mathcal{C}_1$ be the paths from $\mathcal{P}_1$ that close at D and $\mathcal{C}_2$ be the paths from $\mathcal{P}_2$ that close at D.
   Let $\mathcal{K}$ be the paths that contain both K and D.
   if $\mathcal{C}_1 \neq \phi$ and $\mathcal{C}_2 \neq \phi$ **then**
     //D holds M
     enqueue D and the subtree rooted at K into Q_3
     Splay1($D, \mathcal{K} \cup \mathcal{P}_1 - \mathcal{C}_1, \mathcal{K} \cup \mathcal{P}_2 - \mathcal{C}_2$)
   else if $\mathcal{C}_2 = \phi$ **then**
     enqueue D and the subtree rooted at K into Q_1
     Splay1($D, \mathcal{K} \cup \mathcal{P}_1 - \mathcal{C}_1, \mathcal{P}_2$)
   else
     //$\mathcal{C}_1 = \phi$
     enqueue D and the subtree rooted at K into Q_2
     Splay1($D, \mathcal{P}_1, \mathcal{K} \cup \mathcal{P}_2 - \mathcal{C}_2$)
   endif
 endif

procedure ProcessQ(T, Q)
 if Q is empty **then**
   return T
 else
   dequeue T_1 and N from Q leaving Q_1
   construct T_2 by resolving T and T_1 making them parents of N
   return ProcessQ(T_2, Q_1)
 endif

For example, the binary resolution tree in Figure 9 shows the result of performing a splay from the node $M = N_5$ on the second tree in Figure 4. The subtrees S_1 and S_2 are the leaves labelled $a \vee b \vee \neg c$, and $\neg a \vee h$, respectively. The descendants of N_5 are M_4, N_4, N_6 and N_7, of which N_4, N_6 and N_7 are put into Q_1, while M_4 is put into Q_2. These end up above N_5 in the resulting tree, and N_5 becomes the root since every node can see N_5.

LEMMA 10. *If T is a binary resolution tree and M is an internal node of T, then the result T' of $Splay(T, M)$ is a binary resolution tree. T'*

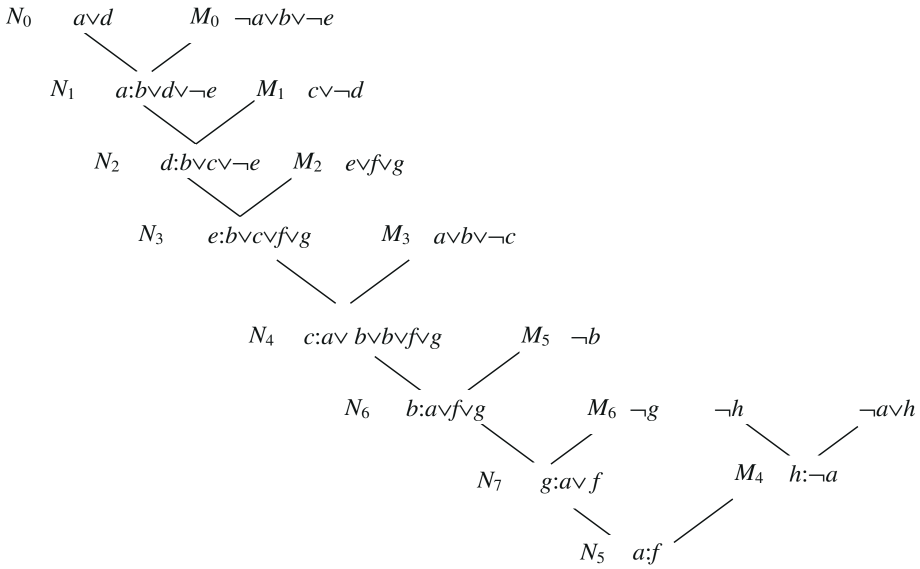

Figure 9. Result of Operation *Splay* at N_5 on the second tree in Figure 4

is defined on the same set of clauses as T, is the same size as T and has the same result as T.

Proof. T' resolves input literals similarly to T. $\square$

OPERATION 4. (SplaySurgery). *Let T be a binary resolution tree T containing a pair M, N of nodes such that $al(M) = al(N)$ and N can see M*

procedure SplaySurgery(T, M, N).
 if M is a descendant of N in T **then**
 return Surgery(T, N)
 else
 return Surgery(Splay(T, M), N)
 endif

To see that a splay actually brings M below N, so that surgery is possible, suppose that the nearest common descendant of M and N is not M. Thus either N is that nearest common descendant or N and M are on different branches. N is either D or an ancestor of K at some point in *Splay1*. We also know N can see M so it must be put into one of the queues Q_1 or Q_2, and cannot be part of Q_3. Thus when the tree is rebuilt, N is put in while we are processing either Q_1 or Q_2. M is put in after Q_1 and Q_2 are empty, so it is below N.

Returning to our example, after the splay, the node N_5 in Figure 9 is below N_1 and surgery from N_1 can be done. The result of this surgery is the same as that shown in Figure 2. Notice that in this example we also have the option to splay at N_1 and do surgery from N_5, since N_1 is also visible from N_5. The resulting tree would be different: it would not require the leaf $\neg a \vee h$ nor $\neg h$, and its result would have been f. Thus splay surgery does not produce a unique result. However its result is at least as general, and the tree is always smaller.

SplaySurgery does a linear number of set operations since it does at most n resolution steps where n is the length of a branch, and then performs surgery, which itself does a number of resolutions limited by the length of a branch. Thus it is effectively a linear time algorithm.

7. Support

Often a node that is visible from another can be rotated so that it is not a descendant, but sometimes no sequence of rotations can bring a node from below another. In this case we say the node *supports* the other, since it is always beneath it. Support, like visibility, is a property that depends on the entire set of rotation equivalent trees. In this section we characterize support in terms of history paths, so it can be checked by examining a static tree.

DEFINITION 11. (Support). *A node N in a binary resolution tree T supports a node M if after every sequence of rotations, M is an ancestor of N.*

DEFINITION 12. (Tightly holds). *Two history paths $P = P_m$ and $Q = Q_n$* tightly hold *a node N if there exist two sequences $P_1, \ldots, P_m$ and $Q_1, \ldots, Q_n$ of history paths such that*

1. *for $i = 1, \ldots, m-1, P_i \prec P_{i+1}$,*
2. *for $j = 1, \ldots, n-1, Q_j \prec Q_{j+1}$,*
3. *the pair P_1 and Q_1 directly holds M, and*
4. *for no pair $i, j, 1 \leq i < m$ and $1 \leq j < n$ does the head of P_i equal the head of Q_j.*

A node N tightly holds *M if both P and Q close at N.*

Tightly holds is the special case of holds, where no two paths in the sequences have the same head except the bottom two. This turns out to be the exact condition for one node being below another after every sequence of rotations.

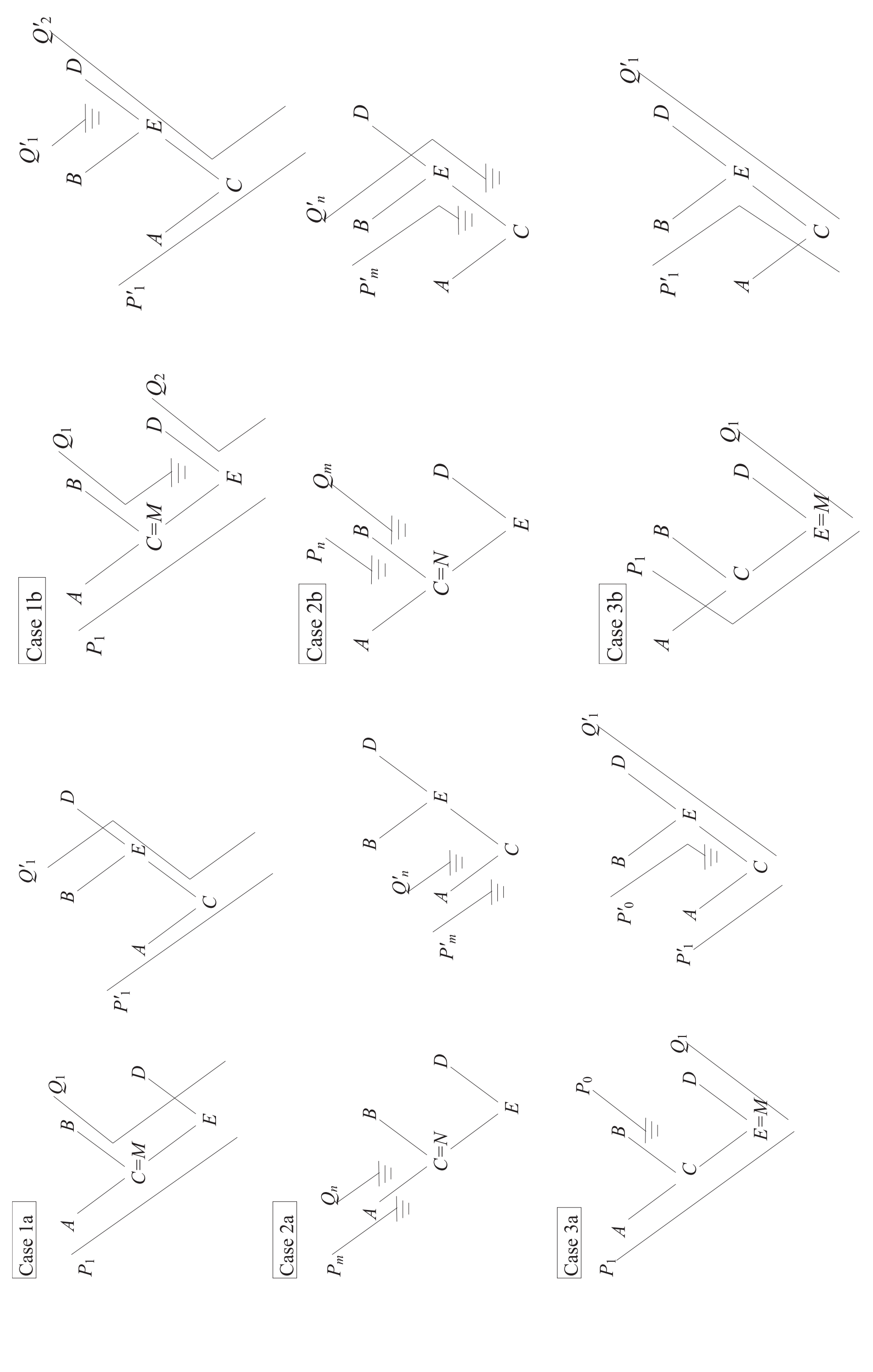

Figure 10. Cases showing tightly hold is invariant under rotations

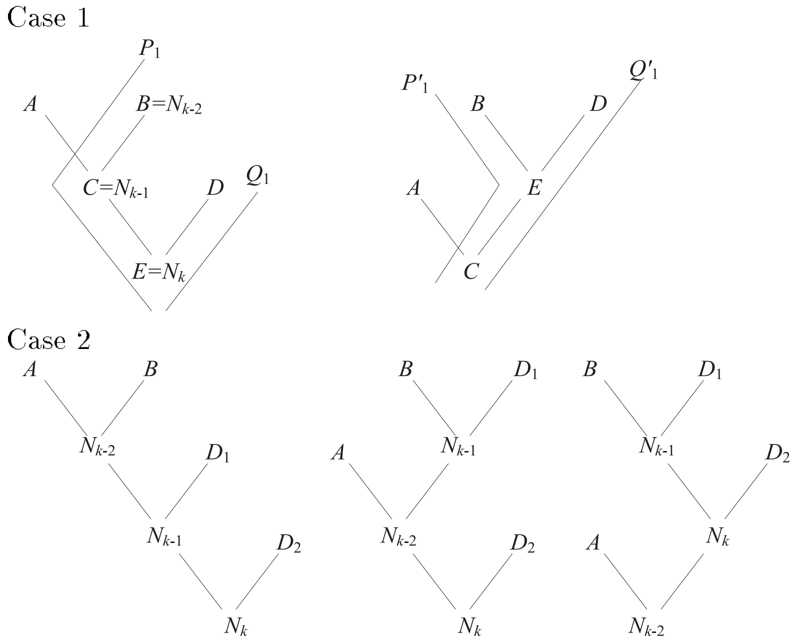

Figure 11. Cases showing support implies tightly holds

THEOREM 11. *In a binary resolution tree, an interior node N tightly holds an interior node M if and only if N supports M.*

Proof. ($\Rightarrow$) We show that if N tightly holds M before a rotation, it also does afterwards, and thus must be a descendant of M. Consider a rotation of edge (C, E) where nodes A, B, C, D and E are defined as in Operation *Rotation*. Also let $P_1, \ldots, P_m$ and $Q_1, \ldots, Q_n$ be the paths by which N tightly holds M. We indicate the image of any history path P after the rotation by P'.

At first assume that $\{C, E\}$ is disjoint from $\{M, N\}$. By Lemma 7, $P_1' \prec^* P_m'$ and $Q_1' \prec^* Q_n'$. Before the rotation all heads of paths in the sequences were distinct so they must be distinct after the rotation, unless there is a new path in either of the sequences. There is one case where such a new path occurs. In Lemma 7, the new path P_0 is introduced when $tail(Q)$ is in T_A and $tail(P)$ is in T_D. In that case the head of new history path was not on $path(M, N)$. Thus the heads of paths of the new sequences $(P_1', \ldots, P_m')$ and $(Q_1', \ldots, Q_n')$ are distinct except $head(P_n') = head(Q_n')$. Also M is still directly held by P_1' and Q_1' so N tightly holds M.

Now suppose $\{C, E\}$ is not disjoint from $\{M, N\}$. (Case 1) If $C = M$ then without loss of generality let P_1 contain A and Q_1 contain B. We know that P_1 does not close at E since the rotation is possible. Suppose first that Q_1 does not close at E, as in Figure 10.1a. Then after the rotation C is directly held by P_1' and Q_1' and so is tightly held by N. Now suppose Q_1 closes at E, as in Figure 10.1b. Note that Q_2 must exist, since otherwise E holds M. Thus Q_1 and P_1 both close at $E = N$ and then the rotation is not possible. After the rotation, C is directly held by P_1' and Q_2', and so is still tightly held by N. (Case 2) Now let $C = N$. Both P_m and Q_n close at N, and either they both include A, as in Figure 10.2a, or they both include B as in Figure 10.2b. In either case, after the rotation, N tightly holds M by the sequences $(P_1', \ldots, P_m')$ and $(Q_1', \ldots, Q_n')$. (Case 3) Now if $E = M$ then assume without loss of generality that C is in P_1 and D is in Q_1. If A is also in P_1 as in Figure 10.3a, then there is a path P_0 with head at B and $P_0 \prec P_1$. After the rotation, the sequences $(P_0', P_1', \ldots, P_m')$ and $(Q_1', \ldots, Q_n')$ ensure that N tightly holds M. If B is in P_1, as in Figure 10.3b, then after the rotation the paths $(P_1', \ldots, P_m')$ and $(Q_1', \ldots, Q_n')$ ensure N tightly holds M. (Case 4) Finally if $E = N$, then either P_m and Q_n both contain C or both contain D. If they both contain C then note that they both contain the same parent B of C, else the rotation is not possible. Then after the rotation, B is a parent of E and although P_m' and Q_n' are one node shorter than before, N tightly holds M by the paths $(P_1', \ldots, P_m')$ and $(Q_1', \ldots, Q_n')$. If P_m and Q_n both contain D, then after the rotation there is no change to the history paths; thus N tightly holds M.

($\Leftarrow$) Assume M supports N. We construct the pair of sequences of history paths by which M tightly holds N. We induct on the length of $path(M, N) = (N_1, N_2, \ldots, N_k)$ where $M = N_1, N = N_k$, and N_i is a parent of N_{i+1} for $i = 1, \ldots, k-1$. If $k = 2$, M is a parent of N and then a rotation of the edge (M, N) is possible unless there exist history paths that tightly hold M and close at N. Suppose $k > 2$. First assume (case 1) that N_k does not support N_{k-1}. Then by the first half of this theorem N_k does not tightly hold N_{k-1} and so the edge (N_{k-1}, N_k) can be rotated. Since M must remain an ancestor of N, M must be an ancestor of B in the rotation, as shown in Figure 11.1. Thus the path from M to N is shorter, and by induction N tightly holds M in the new binary resolution tree. By rotating the edge back again, one sees that N tightly holds M in T, because tightly holds is invariant under rotation. As this case is covered, from now on we assume that N_k supports N_{k-1}.

Next assume (case 2) that N_k does not support N_{k-2}. Then N_{k-1} does not support N_{k-2}, for otherwise N_k supports N_{k-1} supports N_{k-2} and, by transitivity of support, there is a contradiction. By the first half of this theorem, we may rotate the edge (N_{k-2}, N_{k-1}). Then we may rotate the edge (N_{k-2}, N_k), since N_k does not tightly hold N_{k-2}. In this second rotation, N_k must remain a descendant of N_{k-1} so it must be as shown in the third binary resolution tree in Figure 11.2. Thus $path(M, N)$ is shorter, so by induction N tightly holds M. Because tightly holds is invariant under rotation, N tightly holds M in T also. From now on we assume that N_k supports N_{k-2}. By similar arguments we assume that N_k supports each of $N_{k-3}, \ldots, N_2$.

(Case 3) If the edge (M, N_2) can be rotated in T then $path(M, N)$ is shorter in the resulting tree T' and by induction N tightly holds M in T'. Since tightly holds is invariant under rotation, N tightly holds M in T.

Finally if the edge (M, N_2) cannot be rotated then there are paths from each parent of M closing at N_2 in T. We know by induction that N tightly holds N_2 and assume that $(P_1, \ldots, P_m)$ and $(Q_1, \ldots, Q_n)$ are the paths that make this so. Either P_1 or Q_1 includes $N_1 = M$. Assume without loss of generality that P_1 includes N_1. Then Q_1 must include the other parent of N_2 since N_2 is the first node that P_1 and Q_1 have in common. No matter which parent of M is in P_1, there is a path closing at N_2 containing M and M's other parent. Call this path Q_0. Since the head of Q_0 is M, it is distinct from the heads of the other P_i and Q_j in the sequences. Thus N tightly holds M by the paths $(Q_0, Q_1, \ldots, Q_n)$ and $(P_1, \ldots, P_m)$. $\square$

In this final theorem, we relate the notions of visibility and support.

THEOREM 12. *In a binary resolution tree, a node M is invisible from a node N iff there is a support S of M on $path(M, N)$.*

Proof. ($\Rightarrow$) Assume M is invisible from N. Let D be the nearest common descendant of M and N. Consider the path sequences $P_1, \ldots, P_m$ and $Q_1, \ldots, Q_n$ by which D holds M. Consider the least i and j such that $head(P_i) = head(Q_j)$ and let S be the node where P_i and Q_j both close. These i and j must exist since the $head(P_m) = head(Q_n)$. Since the heads of paths before P_i and Q_j are distinct, S tightly holds M and thus S supports M.

($\Leftarrow$) Assume that S on $path(M, N)$ is a support of M. Let $P_1, \ldots, P_m$ and $Q_1, \ldots, Q_n$ be the sequences of paths by which S tightly holds M. Since S is on $path(M, N)$, and S is a descendant of M, S must be an ancestor of the nearest common descendant D of M and N. Thus there are paths $R_1, \ldots, R_k$ where $R_i \prec R_{i+1}$ for $i = 1, \ldots, k-1$ such that S is on R_1 and R_k closes at D. Thus D holds M via the sequences $(P_1, \ldots, P_m, R_1, \ldots, R_k)$ and $(Q_1, \ldots, Q_n, R_1, \ldots, R_k)$. $\square$

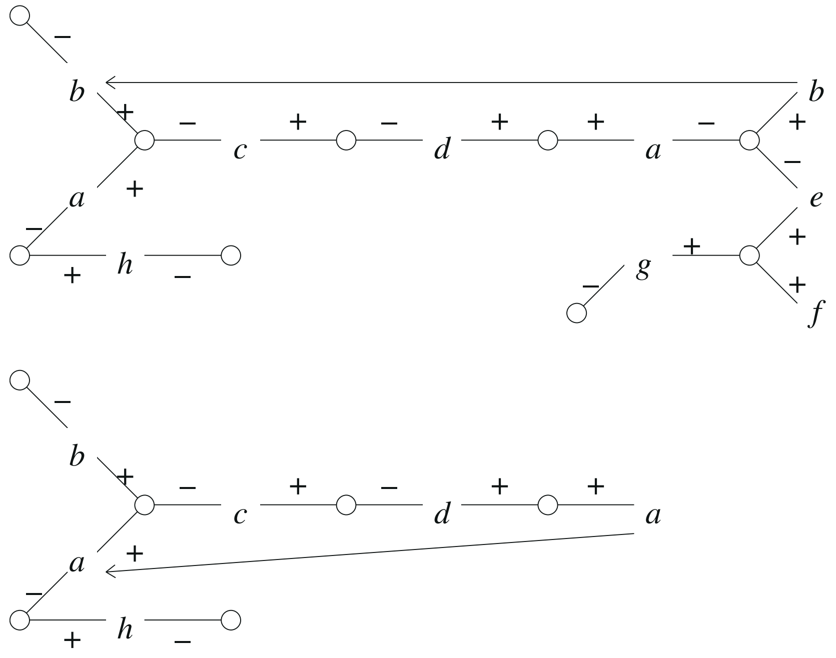

Figure 12. Clause trees corresponding to Figures 1, 4, and 9 and to Figure 2

8. Relation to Clause Trees

The results in this paper were developed after we understood clause trees [5], and then primarily as a means to implement clause trees. We eventually found that most of our ideas from clause trees could be expressed in binary resolution trees. Binary resolution trees are simpler in some ways since they are easier to implement, but often clause trees are easier to use when exploring new ideas. In particular, visibility and support can be read almost directly from a clause tree, whereas the properties held and tightly held are somewhat harder to see in binary resolution trees. In this section we relate the two.

Refering once again to the example, the rotation equivalent binary resolution trees in Figures 1, 4 and 9 all correspond to the first clause tree in Figure 12. The binary resolution tree in Figure 2 corresponds to the second clause tree in Figure 12. A reader familiar with clause trees will note that the second clause tree in Figure 12 can be constructed directly from the first by adding a merge path between the a atom nodes and then doing surgery to remove the subtree beyond the tail of the new merge path.

Some of the correspondences between binary resolution trees and clause trees are obvious. The leaves of a binary resolution tree are the clause nodes of the clause tree; both are labeled by an instance of an input clause. The internal nodes of a binary resolution tree are the atom nodes of a clause tree. A history path in a binary resolution tree, which we associate with a literal, corresponds to a labeled edge in a clause tree, also associated with a literal. If the history path is not closed, it corresponds to an edge incident with an atom node leaf; a closed history path corresponds to an edge incident with a closed atom node.

Minimal clause trees correspond to minimal binary resolution trees whose result does not contain two identical atoms. The minimal condition on clause trees does not allow any legal unchosen merge path or legal tautology path, including leaf to leaf paths. A minimal binary resolution tree may correspond to a clause tree with a legal unchosen leaf to leaf merge path, or legal leaf to leaf tautology path. This is because the regularity condition in binary resolution trees requires one of the nodes to be internal.

Finally, visibility (resp. support) between internal nodes in a binary resolution tree correspond to visibility (resp. support) between closed atom nodes in a clause tree. Table I shows a number of the corresponding notions.

9. Related, Past and Future Work

Regularity is one of the most important restrictions and some form of it is used in many theorem proving methods related to resolution, including tableau [11], and all variants of model elimination[9].

By Theorem 6 a smallest binary resolution tree is minimal. Goerdt has shown [4] that a smallest regular binary resolution directed acyclic graph (DAG) may be exponentially larger than an irregular binary resolution DAG. Thus by considering only regular or minimal binary resolution DAGs, a theorem prover may not find the smallest proof, and hence may require more inferences to prove a given theorem. However in most cases we believe that the space of minimal binary resolution DAGs is much smaller than the space of all binary resolution DAGs and that considerable time can be saved by restricting to minimal ones. (Similar arguments can be made for and against most restrictions of resolution, including set-of-support and hyperresolution.)

Table I. Corresponding notions

Clause Trees	**Binary Resolution Trees**
clause node	leaf node
internal atom node	internal node
edge	history path
open leaf atom node	literal in result
$cl(T)$	result
internal-to-internal surgery	splay surgery
internal-to-leaf surgery	surgery
merge path	two history paths with a common head
merge set	all history paths closing at a given node
equivalence classes of reversal equivalent	equivalence classes of rotation equivalent
minimal (up to leaf-to-leaf)	minimal
visible internal atom nodes	visible (not held by a common descendant)
support on internal atom nodes	support, tightly hold
path reversal	no structural change
no structural change (change in derivation)	rotation

Permuting inference steps has been investigated by Kleene [8], in the context of Gentzen's sequent calculus, both classical and intuitionistic. Kleene's permutations sometimes increase the size of the proof. It is interesting to note that he defines the ancestor relation between instances of formulas in each inference, allowing him to state which instances of formulas in the deduction belong to a given instance in the end sequent. This is analogous to our notion of history paths.

In the context of binary resolution derivations, de Nivelle [3], has two types of nodes: resolution nodes have two parents and factoring nodes have one. He defines four types of edge rotations, depending on the type of nodes incident with the edge. We disallow the rotation where a factorization node is parent to a resolution node, because in this case the size of the derivation must be increased. His application is to construct *resolution games* which are then used to show various completeness results for restrictions of resolution based on ordering literals. Both de Nivelle's paper and ours show that basic properties of resolution may be explored by considering the set of trees equivalent modulo permutations or resolutions.

The main contribution of this paper is to present the minimal restriction of resolution, originally developed in terms of clause trees, using the well known proof format of binary resolution derivations. The original motivation for doing so was to implement bottom up algorithms for constructing minimal clause trees. A direct implementation, based on the structural definition of clause trees was done, but each new tree needed its own storage space and the visibility algorithm was cumbersome. To remedy both of these problems we used the notion of binary resolution derivations and implemented each new clause tree as a single storage cell with two parent clause trees. (For propositional logic this solves the space problem, since each tree can be used as part of other trees.) The surprising result is that visibility can be expressed easily, and requires a linear algorithm using this data structure. We then determined to explain as much of our clause tree work as we could using binary resolution trees, to make them more accessible to readers familiar with resolution. This task turned out to be difficult until we discovered that edge rotations, history paths and the "precedes" relation between history paths are the fundamental concepts needed. Then we related visibility and support to history paths using the "holds" and "tightly holds" relations. In retrospect the authors still believe that it is easier to work conceptually with clause trees. We have anecdotal evidence, from a graduate course in automated reasoning given once with clause trees only and twice with binary resolution trees followed by clause trees, that the intuitions for support and visibility are quite understandable using either data structure.

We are presently developing the full theorem provers described here, one to use minimality as a restriction and another to use splay and surgery to improve proofs after they are constructed by resolution. Both retain only minimal binary resolution trees so that the recursive calls in the first condition of Theorem 9 are not needed. Our implementations actually build binary resolution DAGs, instead of trees, to save space. This is important because theorem provers are limited by space as well as by time. Note that procedure *Vis* traverses the entire implicit tree, so it may visit a single stored node more than once. Thus its runtime is not guaranteed to be linear in the size of the DAG.

Redundancy elimination by subsumption is always an important consideration for theorem provers. Unfortunately the minimal restriction is not complete with full subsumption. For instance one can not refute $\{p, \neg p \vee q, \neg p \vee \neg q\}$ with a minimal binary resolution tree if one resolves p against $\neg p \vee q$, generating q, and then uses back subsumption to remove $\neg p \vee q$ before the resolution between $\neg p \vee q$ and $\neg p \vee \neg q$ is done. The latter resolution is part of the only minimal binary resolution refutation of these clauses. The former resolution step leads to a binary resolution tree in which p is resolved at two different nodes on one branch. However, we have discovered [7] that one can retain completeness by giving up some of the power of minimality, without giving up any of the power of subsumption.

The implementations optionally include the rank/activity restriction, discussed in [6], which ensures that from each set of rotation equivalent binary resolution trees, exactly one is found.

The space of minimal binary resolution trees is interesting for the following reasons: (1) it is refutationally complete, (2) it extends the well known regularity restriction of resolution, (3) it contains the smallest binary resolution tree, (4) non-minimal (sub)trees can be identified in time linear in the size of the tree, and (5) non-minimal trees can be reduced to minimal ones efficiently.

Acknowledgments

The authors are grateful to NSERC of Canada for funding, and to the anonymous reviewers for their constructive comments. The first author did some initial work while visiting the Technical University of Darmstadt.

References

1. G. M. Adelson-Velskii and E. M. Landis. An algorithm for the organization of information. *Soviet Math. Doklady*, 3:1259–1263, 1962.
2. Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York and London, 1973.
3. Hans de Nivelle. Resolution games and non-liftable resolution orderings. *Collegium Logicum, Annals of the Kurt Gödel Society*, 2:1–20, 1996.
4. Andreas Goerdt. Regular resolution versus unrestricted resolution. *SIAM Journal on Computing*, 22:661–683, 1993.
5. J. D. Horton and B. Spencer. Clause trees: a tool for understanding and implementing resolution in automated reasoning. *Artificial Intelligence*, 92:25–89, 1997.
6. J. D. Horton and B. Spencer. Rank/activity: a canonical form for binary resolution. In C. Kirchner and H. Kirchner, editors, *Automated Deduction – Cade-15*, number 1421 in Lecture Notes in Artificial Intelligence, pages 412–426. Springer-Verlag, Berlin, July 1998.
7. J. D. Horton and Bruce Spencer. Bottom up procedures to construct each minimal clause tree once. Technical Report TR97-115, Faculty of Computer Science, University of New Brunswick, PO Box 4400, Fredericton, New Brunswick, Canada, 1997.

8. S. Kleene. On the permutability of inferences. *Memoirs of the AMS*, 10:1–26, 1952.
9. Donald Loveland. Mechanical theorem proving by model elimination. *J. ACM*, 15:236–251, 1968.
10. W. W. McCune. Otter 3.0 users guide. Technical Report ANL-94/6, Mathematics and Computer Science Division, Argonne National Laboratories, Argonne, IL, 1994.
11. R. Letz, K. Mayr, C. Goller. Controlled integration of the cut rule into connection tableau calculi. *Journal of Automated Reasoning*, 13:297–337, 1994.
12. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12:23–41, 1965.
13. D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32:652–686, 1985.
14. Bruce Spencer and J.D. Horton. Extending the regular restriction of resolution to non-linear subdeductions. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 478–483. AAAI Press/MIT Press, 1997.
15. G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics*, Seminars in Mathematics: Matematicheskii Institute, pages 115–125. Consultants Bureau, 1969.