Rank/Activity: A Canonical Form for Binary Resolution

J. D. Horton and Bruce Spencer

Faculty of Computer Science, University of New Brunswick
P.O. Box 4400, Fredericton, New Brunswick, Canada E3B 5A3
jdh@unb.ca, bspencer@unb.ca, http://www.cs.unb.ca

Abstract. The rank/activity restriction on binary resolution is introduced. It accepts only a single derivation tree from a large equivalence class of such trees. The equivalence classes capture all trees that are the same size and differ only by reordering the resolution steps. A proof procedure that combines this restriction with the authors' minimal restriction of binary resolution computes each minimal binary resolution tree exactly once.

1 Introduction

A new restriction of binary resolution is proposed in this paper. The restriction is complete in a strong sense, in that every binary resolution proof, up to reordering the resolution steps, is allowed. On the other hand, the restriction prevents multiple versions of the same proof from being constructed. If a given proof is allowed, then no other proof that can be obtained from it by reordering the steps is allowed.

Consider an automated reasoning procedure that takes a set of clauses as input, and resolves pairs of clauses containing complementary literals to generate new clauses. The possible resolutions must be restricted somehow. To do this, let each literal in a clause be either active or inactive, and only let pairs of active complementary literals resolve. Since at the beginning we have no idea which resolution it is best to perform first, all literals in all input clauses are active.

As resolutions occur, some of the literals in newly generated clauses must become inactive. Since we want to have only one acceptable order in which the literals are resolved, the literals in each clause are ordered by a rank function which assigns an integer value to each literal. This rank function must be consistent between a parent clause and a child clause, in that if $rank(a) < rank(b)$ in a parent clause, then $rank(a) < rank(b)$ in the child clause as well. When a clause is resolved on a literal of a given rank, in the newly created child clause all literals of a lesser rank (rank as defined in the parent clause) become inactive, and hence are not allowed to be resolved again. This activity condition also must be inherited from parent literal to child literal, with the following exception. When two literals that come from different parents merge (or factor) in a child clause, the literal becomes active, regardless of whether the literals in the parent clauses are active or inactive. This exception is very important as completeness is lost if this is not done.

Instead of just keeping the clauses produced, it is important to remember also how a clause is derived. This can be done using clause trees [5], which is where this material was first derived. In this paper we develop the ideas in binary resolution trees [7], which are much more closely related to the proof trees seen in many papers on resolution.

One simple rank function is to assign ranks from 1 to n arbitrarily in an n-literal input clause. When two clauses are resolved, the ranks of the literals in the clause containing the negative resolved literal are the same in the child clause as in the parent clause. The rank of a literal from the parent containing the positive resolved literal receives in the new clause a rank equal to its old rank plus the number of leaf literals in the binary resolution tree containing the negative resolved literal. This method guarantees that the rank function on every clause is one-to-one, and remains between one and the number of leaf literals in every binary resolution tree. In fact if the rank function is applied to the "history path" of every leaf literal, then the rank function remains one-to-one on the set of history paths. We must also specify the rank of merged literals. When two literals are merged, the new literal gets the smaller of the ranks of the literals being merged.

In this example, superscripts denote rank values and a superscript asterisk denotes an inactive literal. When the clause $a^1 \vee b^2 \vee d^3 \vee e^4$ resolves against $a^1 \vee c^2 \vee \neg d^3 \vee f^4$ the result is $a^1 \vee b^{6*} \vee c^{2*} \vee e^8 \vee f^4$. Resolving this against $\neg e^1$ results in a clause that is completely inactive and can never resolve. Note that the rank and activity of the literals in the original clauses are unaffected.

The only requirement on the procedure is that the choice of which resolution to do next has to be fair. If a resolution can be done, then it must be done eventually, at some time in the future, unless one of the resolving clauses is rejected. No resolution can be put off forever. Likewise, no resolution needs to be done more than once. Then the above restriction guarantees that every proof will be found by the procedure, and only once.

2 Background

We use standard definitions [2] for atom, literal, substitution, unifier and most general unifier. Most of the rest of this section originated from [7, 8]. 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. A *variable renaming substitution* is one in which every variable maps to a new variable (not in the expression in question), 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 $C\theta = D$. 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 $C_2 \vee \neg b_1 \vee \ldots \vee \neg b_n$ that are standardized apart (a variable renaming substitution may be required), their *resolvent* is the clause $(C_1 \vee C_2)\theta$ where θ is the most general unifier of $\{a_1, \ldots, a_m, b_1, \ldots, b_n\}$. The *atom resolved upon* is $a_1\theta$, and the set of *resolved literals* is $\{a_1, \ldots, a_m, \neg b_1, \ldots, \neg b_n\}$.

For each resolution operation we define the resolution mapping ρ from each occurrence of a literal c in each parent clause to either the atom resolved upon if c is a resolved literal, or otherwise to the occurrence of $c\theta$ in the resolvent. We use ρ later to define history paths.

The reader may be missing the usual factoring operation on a clause, which consists of applying a substitution that unifies two of its literals with the same sign and then removing one of these literals. The original definition of resolution [6] does not have this operation. 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. Thus the root is a descendant of every node in the tree.

Definition 1. *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 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 $S = \{a \vee d, \neg a \vee b \vee \neg e, c \vee \neg d, e \vee f \vee g, a \vee b \vee \neg c, \neg a \vee h, \neg h, \neg b, \neg g\}$. The labels of a node N are displayed beside the name of the node and separated by a colon if both labels exist. For example 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.

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. If all literals are eventually

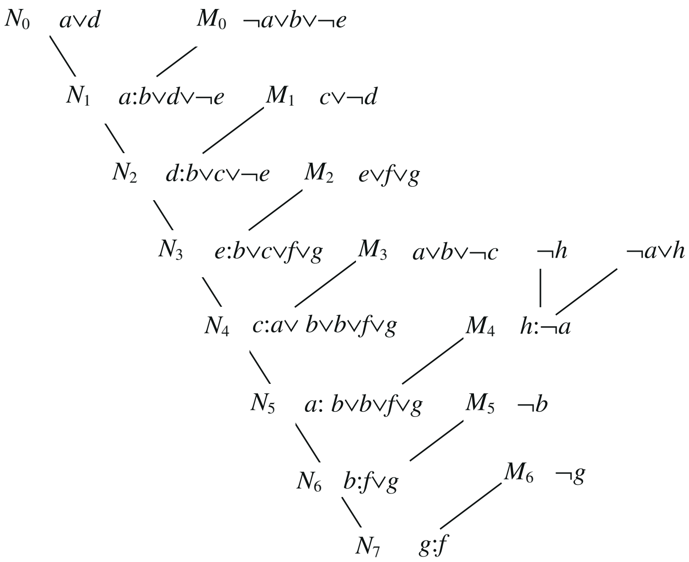

Fig. 1. A binary resolution tree.

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 is an unsatisfiable set of clauses. Thus we are primarily concerned about tracing the "history" of a literal starting from its appearance in a leaf.

Definition 2 (History Path). *Let a be an occurrence of a literal in the clause label of a leaf node N_0 of a binary resolution tree T. Let $P = (N_0, N_1, \ldots, N_n)$ be a path in T where for each $i = 1, \ldots, n$, N_i is the child of N_{i-1}, ρ_i is the resolution mapping into N_i, and $\rho_i \ldots \rho_2 \rho_1 a$ occurs in $cl(N_i)$, so that a is not resolved upon in P. Further suppose N_n either is the root of T, or has a child N such that $\rho_n \ldots \rho_1 a$ is the atom resolved upon at N. Then P is a* history path *for a in T. 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 $(M_2, N_3, N_4, N_5, N_6, N_7)$.

A *rotation* of an edge in a binary tree is a common operation, 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 2, a rotation is the reassignment of edges so that the tree on the right of Figure 2 is produced. The parent C of E becomes the child of E and the parent B of C

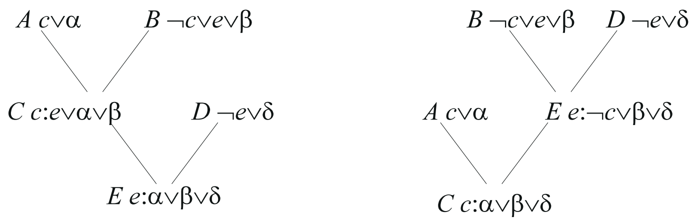

Fig. 2. A binary tree rotation

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 it exists, are replaced by the edges (B, E), (E, C) and (C, F) if necessary.

Operation 1 (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 $cl(D)$ then $cl(E)$ in T' may be tautological.

Note that before the rotation, no history path through A closes at E. We do not allow the rotation if history paths through both parents of C close at E. If we did then after the rotation, one of them would not have the opportunity to close, and thus the clause label of the root would change. Before showing that the clause label of the root is not changed (Corollary 1), we prove a slightly more general result, which is also used later.

Definition 3. *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* close history paths 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 instances of the same 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 close history paths similarly if they resolve the same literals against each other, albeit in a possibly different order.

Lemma 1. *If two binary resolution trees T_1 and T_2 close history paths 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 mgu's in T_1 and the composition of mgu's 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. □

Corollary 1. *Given a binary resolution tree T with an internal node C and its child E, Operation 1 generates a new binary resolution tree and $cl(C)$ in $T' = cl(E)$ in T, up to variable renaming.*

Proof. Observe that Operation 1 produces a tree which closes history paths 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 3 is produced by rotating the edge (N_4, N_5) in Figure 1. The second tree in Figure 3 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 close history paths similarly, but the converse is true as well.

Theorem 2. *Two binary resolution trees T_1 and T_2 are rotation equivalent if and only if they close history paths similarly.*

Proof. Since one rotation of T_1 creates a binary resolution tree that closes history paths similarly to it, so too does the sequence of rotations creating T_2.

The converse is proved by induction on the number of internal nodes. Suppose T_1 and T_2 close history paths 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 so that $\nu(L_1)$ and $\nu(L_2)$ are 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 it is not possible that both A and B contain a history path

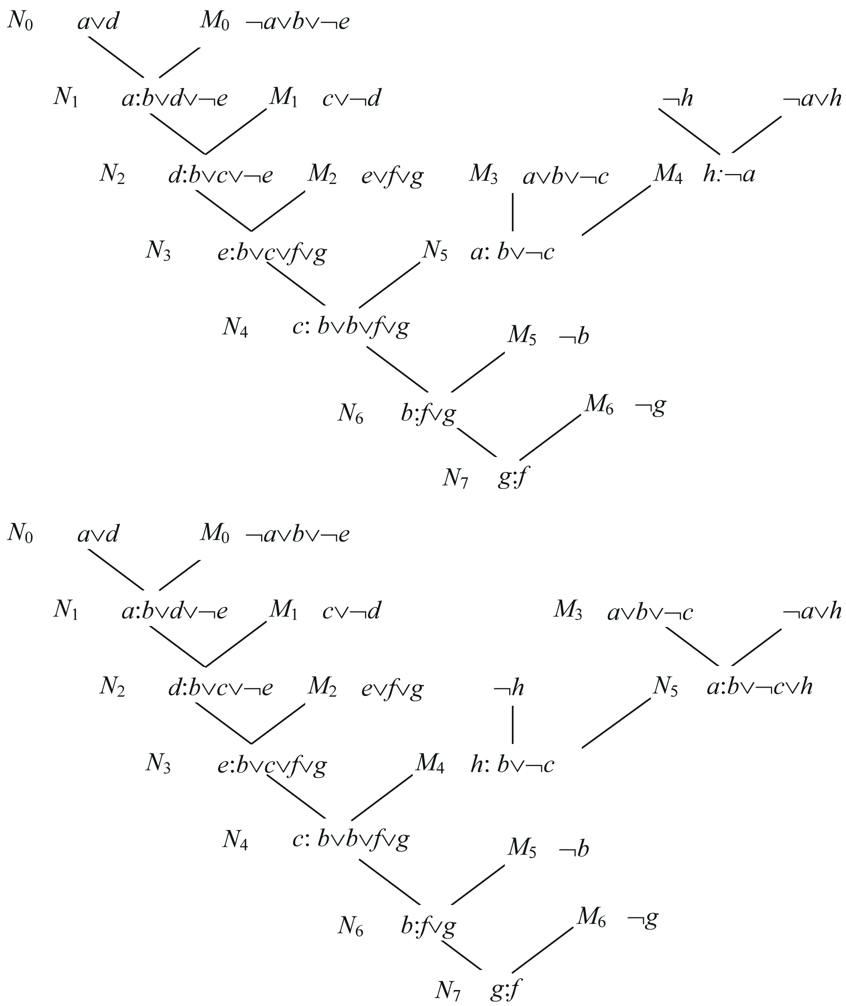

Fig. 3. From Figure 1 rotate (N_4, N_5), then (M_4, N_5)

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^* close history paths similarly since T_1 and T_2' close history paths 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. □

3 The rank/activity calculus

A rank function must assign a value to every literal in the clause at each node in a given binary resolution tree, in such a way that it orders history paths consistently. Moreover it must assign values to sets of literals if they are unified by a resolution closer to the root of the binary resolution tree. In the following definition a rank function is required to assign values to every set of unifiable literals, even if they are not unified later in the tree. In the informal discussion in the introduction, the rank of a set of literals was given as the minimum of the ranks of the literals unified, but this is not a requirement. The maximum or any other number could be used instead. In the following, if H is a set of history paths in a binary resolution tree T such that (1) some node occurs on all these paths and (2) the literals of these paths are unifiable, then let $literal(H)$ be the multiset of these literals.

Definition 4 (Rank function). *Let $\mathcal{F}$ be a set of binary resolution trees, closed under taking subtrees. Let r assign an integer value to every set of unifiable literals at every node of every tree. Then r is a* rank function *for $\mathcal{F}$ if r satisfies the following condition in every binary resolution tree T:*

For every pair of disjoint sets H_1 and H_2 of history paths that have two nodes N_1 and N_2 in common:

$$r(literal(H_1), N_1) < r(literal(H_2), N_1) \iff$$

$$r(literal(H_1), N_2) < r(literal(H_2), N_2).$$

Thus r is a rank function if it orders the sets of history paths consistently. In fact the reflexive transitive closure of this relation between sets of history paths, is a partial order.

For example, let T_0 be a binary resolution tree containing a node N_1 and its child N_2. $cl(N) = s \vee p(x) \vee p(y) \vee q(u) \vee q(v)$ where x, y, u and v are variables. At N_1, let r map s to 2, $\{p(x), p(y)\}$ to 4 and $\{q(u), q(v)\}$ to 6. The resolution at N_2 resolves away the s so that $cl(N_2) = p(x) \vee p(y) \vee q(u) \vee q(v)$. If r at N_2 maps $\{p(x), p(y)\}$ to 12, then it must map $\{q(u), q(v)\}$ to a value greater than 12 if r is to be a rank function. In this example H_1 is the history paths for $p(x)$ and $p(y)$ so $literal(H_1)$ is $\{p(x), p(y)\}$, while $literal(H_2)$ is $\{q(u), q(v)\}$.

Next we want to define those binary resolution trees that can be built using the rank/activity restriction. Let N be a node other than the root of a binary resolution tree T. Let $H(N)$ be the set of history paths with N as their head. Then these paths close at the child of N.

Definition 5 (r-compliant). *Let r be a rank function for a binary resolution tree and all its subtrees. Then T is* r-compliant *if the following condition is true. Let N and M be any two nodes such that $H(N)$ also have M in common. Thus M is an ancestor of N. Then $r(literal(H(M)), M) \leq r(literal(H(N)), M)$.*

Returning to the example T_0, let $N_2 = M$ and suppose that a new node N is made the child of M. Both $p(x)$ and $p(y)$ are resolved away at N so $cl(N) = q(u) \vee q(v)$. Thus $H(M) = H_1$, the history paths for $p(x)$ and $p(y)$, so $r(literal(H(M)), M) = 12$. Then suppose the child of N resolves away the remaining literals so that $literal(H(N)) = \{q(u), q(v)\}$. Call the resulting tree T_1. We have already assumed that $r(literal(H(M)), M) < r(literal(H(N)), M)$ to make r a rank function. This condition also makes T_1 r-compliant. Note that if one rotates the edge between N and the root, one gets a tree which is rotation equivalent to T_1 but resolves away the q's before the p's. Since r still ranks the p's in M lower than the q's, this tree is not r-compliant. The resolution on the q's has deactivated the p's.

In the general case, the resolution at M's child does not deactivate the set of history paths with head at N. Moreover, this set of history paths is not affected by what happens before they are drawn together at some node by a resolution. Therefore it is created as an active set of literals, which justifies the re-activation of literals when they are merged together. Hence the set is active in N and can be resolved by a rank/activity procedure at N's child. Thus the r-compliant binary resolution trees are precisely those trees that can be constructed using the rank/activity restriction of binary resolution, using the function r as the rank function.

Theorem 3 (Completeness and uniqueness). *Let T be a binary resolution tree. Let r be a rank function for the set of all binary resolution trees that are rotation equivalent to T. Then there is a binary resolution tree T' that is rotation equivalent to T and is r-compliant. Moreover, if the rank function r maps disjoint sets of literals at any given node to different values, then T' is unique.*

Proof. First we prove the existence of T', which implies that the rank/activity restriction is complete. The proof is an induction on the number of nodes in T. If Tconsists of a single node, then T itself is r-compliant for any rank function defined on T.

We now consider the case in which T has more than one node. Consider any leaf L. The literals of L correspond to history paths that either close at some internal node of T, or pass through to the root of T. Let $N_1, N_2, ..., N_k$ be the nodes that are the heads of history paths with L as the tail, excluding those paths whose heads are the root. Let P_i be the set of history paths with L as the tail and N_i as the head, for $i = 1, 2, ..., k$. We define a pointer $P(L)$, which points from L at some internal node of T. $P(L)$ is the child of the N_j where:

1. $P_j = H(N_j)$;

2. $r(literal(P_j), L) \leq r(literal(P_i), L)$ for all i such that $P_i = H(N_i)$.

The first condition asserts that the literals of P_j are not merged or factored with literals from any other leaf, before being resolved. The second condition asserts that this set of literals has the minimum rank of all sets of literals that satisfy condition (1). If there is more than one choice for $P(L)$ because r is not one-to-one, then any choice of the nodes that satisfy the conditions can be made. Note that if the rank function satisfies the condition of the second part of the theorem, then $P(L)$ must necessarily be unique, even if r itself is not one-to-one. There is some P_i that satisfies condition $P_i = H(N_i)$, since L itself is one of the N_i, as at least one literal of $cl(L)$ must close at L's child, and thus cannot be merged with literals from other clauses first.

Now $P(L)$ is a function that points from the leaves of T into the interior nodes of T. Since the number of leaves is one more than the number of interior nodes, by the pigeon hole principle there must be some node N pointed at by two different leaves, L_1 and L_2. Then $P(L_1) = P(L_2) = N$.

If either parent of N is not L_1 or L_2, then N can be rotated with that parent, since the literals closing at N through that parent cannot come from distinct grandparents of N. Since the set of literals closing at N are not changed by rotation, N can be rotated upward in the tree until its parents are L_1 and L_2 themselves. Now remove L_1 and L_2 from the tree, making N a leaf of a new smaller tree T_1. The function r, restricted to the nodes of T_1 and all its rotation equivalent binary resolution trees, is still a rank function for T_1. By induction T_1 is rotation equivalent to a binary resolution tree T_1' that is r-compliant. Replace the leaf N in T_1' by the subtree with leaves L_1 and L_2, and root N, to get another binary resolution tree T'. T' is rotation equivalent to T, since the rotations of T_1 to T_1' can be mirrored by rotations in binary resolution trees with N replaced by the $L_1 - N - L_2$ subtree.

It remains to show that T' is r-compliant. The only nodes that must be checked are the new nodes L_1 and L_2, as a possible M in the definition of r-compliant. Consider any non-root node N' such that the history paths $H(N')$ have L_1 in common. Then by the choice of $P(L_1), r(literal(H(L_1)), L_1) \leq r(literal(H(N')), L_1)$ by the definition of $P(L_1)$, so that the r-compliant condition is always satisfied at L_1. The same situation applies at L_2. Thus T' is r-compliant. Hence the rank/activity restriction is complete.

If the condition is added that the rank function r does not map two disjoint sets of literals at any node to the same value, then the pointer function chooses a unique node $P(L)$ for any leaf L. Let L, N_i, P_i, be as defined above. Let T^* be any binary resolution tree that is r-compliant and rotation equivalent to T. The history paths of $H(L)$ close at the child of L. Then $r(literal(H(L)), L) \leq r(literal(P_i), L)$ for $i = 1, 2, ..., k$, because T^* is r-compliant. But by the uniqueness condition on r, these ranks must all be distinct. Thus there is a unique j such that $H(L) = P_j$, and $P(L) =$ the child of N_j, by the definition of $P(L)$. Hence $N_j = L$, and $P(L)$ must be the child of L in T^*.

This argument shows that each leaf of T^* has a unique child. The argument can be extended to all the nodes of T^*, by inducting on the height of the subtree above the node. Thus every node, other than the root node, has a uniquely defined child node. It follows that the binary resolution tree T* is unique. □

Consider any bottom up binary resolution proof procedure that keeps all clauses that it generates, and uses the rank/activity restriction. It only constructs proofs that correspond to r-compliant binary resolution trees, where r is the rank function used. One consequence of this theorem is that as long as the proof procedure is fair, in that if a resolution is allowed then it will eventually be performed once, the procedure must construct every possible proof, up to reordering the resolutions. That is, the procedure is refutationally complete. Moreover it will produce a binary resolution tree of minimal size (number of nodes), if it is not halted after the first is found. However only one binary resolution tree is produced for each possible proof, if the rank function is distinct on disjoint sets of literals. As the number of reorderings is typically exponential in the size of the binary resolution tree, this amounts to a considerable saving of work compared to a proof procedure that does not use this restriction.

4 Combining with minimality

The rank/activity restriction combines well with minimality, another restriction of binary resolution. Minimality [5, 7, 8] is an extension of the better known regularity restriction [9]. A binary resolution tree is regular if, for every internal node N, the atom label of N is not in the clause label of any descendant of N.

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. We call this operation *surgery* [7, 8].

A binary resolution tree is minimal if it is not rotation equivalent to a irregular binary resolution tree. There is a linear time (in the size of the tree) algorithm to detect whether the resolution of two binary resolution trees creates binary resolution tree that is minimal or is non-minimal [7].

Theorem 4. *Let $\mathcal{C}$ be an unsatisfiable set of clauses. Let r be a rank function defined on the binary resolution trees that can be constructed with $\mathcal{C}$ as the clauses of the leaves. Then there is a minimal r-compliant binary resolution tree with an empty clause at its root. Moreover, one of the smallest binary resolution trees on $\mathcal{C}$ with the empty clause at the root is minimal and r-compliant.*

Proof. An irregular binary resolution tree can be manipulated by surgery so that the second identical literal on a branch is resolved away at the same time as the first identical literal. The resulting binary resolution tree is smaller, and the resulting clause at the root subsumes the clause of the original binary resolution tree. (See [8] for a proof.) Thus the smallest binary resolution tree that results in the empty clause must be regular. Since every binary resolution tree is rotation equivalent to one that is r-compliant, so is the smallest one. Rotating nodes cannot turn a minimal binary resolution tree into one that is non-minimal, for if the resulting tree can be rotated into an irregular proof, so too can the original tree. Moreover rotating nodes does not change the size of the binary resolution tree. The theorem follows. □

Corollary 2. *A fair binary resolution procedure that uses both the rank/activity and the minimality restrictions and that keeps all other clauses produced, is refutationally complete.*

Proof. Any subtree of a minimal binary resolution tree is also minimal, since if a subtree can be rotated to be irregular, so can the supertree. Thus to produce a given minimal tree, only minimal trees have to be resolved. By the same proof as Theorem 3, every minimal tree must be rotation equivalent to some tree produced by this procedure. □

5 A simple example

The rank/activity restriction, like resolution itself, does not require any specific procedure or approach, so any example is rather arbitrary. However an example may clarify some points. The procedure used below is not recommended as a efficient theorem prover, but has been chosen because it is simple and straightforward.

The procedure grows a list of clauses, starting from a list of the input clauses. The clauses are processed from top to bottom. Each clause in the list, when its turn comes, is resolved in all possible ways with the clauses above it. The generated clauses, if they contain an active literal, are added to the bottom of the list. Those clauses with no active literals are discarded. This procedure is fair so it is complete with the rank/activity restriction. Subsumption is not used.

The rank of a literal is denoted by a superscript following it. If a literal is inactive, an asterisk is placed after the rank. Ranks are kept as discussed in the introduction, so a "size" must be kept for each clause, representing the number of literals in the corresponding binary resolution tree. When a clause that consists only of inactive literals would be generated by a given resolution, the word "inactive" is placed in the diagram where the clause would otherwise appear, but the clause itself is not inserted into the list, and in fact does not even need to be generated. The clause will be inactive if the two resolving literals are the highest ranked active literals in the parent clauses, there are other literals and no merging of literals is possible.

The clauses and literals resolved upon for each new clause are indicated by the notation clause1#:literal-position - clause2#:literal-position.

# Clause	Size	Source
1. p^1	1	input
2. $\neg s^1$	1	input

3. $s^1 \vee \neg r^2$	2	input
4. $\neg q^1 \vee \neg p^2$	2	input
5. $q^1 \vee \neg p^2 \vee r^3$	3	input

Processing clauses 1 and 2 generates no clauses.
Processing clause 3 generates one clause.

6. $\neg r^3$	3	3:1-2:1

Processing clause 4 generates a clause with no active literals, $\neg q^{1*}$.
Such inactive clauses are discarded.

inactive		4:1-1:1

Processing clause 5 generates three clauses, one of which is inactive, and another has a merge.

7. $q^{1*} \vee r^3$	4	5:2-1:1
inactive		5:3-3:2
8. $\neg p^2 \vee \neg p^4 \vee r^5$	5	5:1-4:1
merged to $\neg p^2 \vee r^5$		

Processing clause 6 generates one inactive clause.

inactive		6:1-5:3

In processing clause 7, only the active literal needs to be resolved.

inactive		7:2-3:2
inactive		7:2-6:1

Processing clause 8:

9. r^6	6	8:1-1:1
inactive		8:2-3:2
inactive		8:2-6:1

Processing clause 9 ends the procedure.

inactive		9:1-3:2
10. $\square$	8	9:1-6:1

The resulting binary resolution tree is in Figure 4.

In all the procedure has done five resolutions, only one of which, #7, is not used in the proof. A total of eight resolutions were not done because the resulting

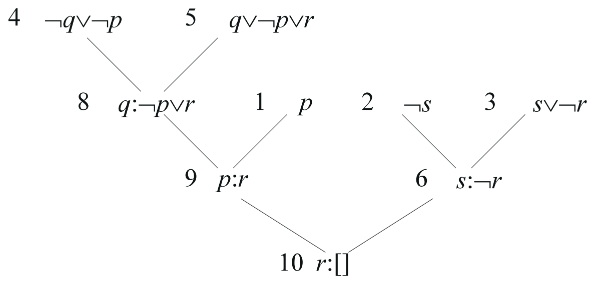

Fig. 4. The example's unique r-compliant binary resolution tree

clause would be inactive. There are nine rotation equivalent binary resolution trees; there is only one r-compliant one, unless the merge is clause 8 is optional.

6 Discussion

This paper presents a canonical form for a large equivalence class of binary resolution trees. For a rank function r that gives different values to disjoint sets of literals, there is a unique r-compliant binary resolution tree that is rotation equivalent to a given binary resolution tree. Rotation equivalent trees do the same resolutions in a different order so this is a natural class to study. The restriction leads to an exponential reduction in the number of proofs in the search space. Implementations need only store an integer rank and a one-bit activity status with each literal, and the check is inference local so it adds little to the overall execution time. Meanwhile it does not eliminate the proof tree of minimum size. Most restrictions of resolution, such as linear, selection function, A-ordered, and lock, do not have all of these properties. Moreover, the minimal restriction [5], an extension of the regular restriction, can be combined with rank/activity.

A-ordered resolution appears to be a somewhat closely related restriction; it makes the smallest atom in the A-ordering the only active atom. However, A-ordering cannot consistently order the literals in first-order logic and all their instances. The rank/activity restriction applies directly to first order logic, since the rank of an occurrence of a literal is not affected by taking instances. In effect, rank/activity resembles an A-ordering restriction that sometimes allows literals to be skipped over, and brought back into the order after they are factored.

In one sense the rank/activity restriction is just a subrestriction of subsumption, because every unclosed binary resolution tree that is prevented from being formed by rank/activity, would eventually rejected by subsumption too. But it is much easier to check activity than it is to check subsumption since subsumption depends on the, often large, size of the set of retained clauses. This work addresses a problem posed by Wos [10]:

If a strategy could be found whose use prevented a reasoning program from deducing redundant clauses, we would have a solution far preferable to our current one of using subsumption.

Full scale subsumption cannot be combined with the rank/activity restriction, since totally separate proofs of the same clause need not have the same rank orderings of the literals or activity conditions on the literals. However it is possible to combine a considerable portion of subsumption with the rank/activity restriction. To use rank activity completely, one can delete a subsumed clause D if the subsuming clause C corresponds to a "smaller" binary resolution tree than D does. It is also possible to combine a good deal of the rank/activity restriction with full subsumption while maintaining completeness. To use subsumption fully, one can activate all of the literals of a clause if that clause subsumes another clause, thus treating the subsuming clause like a new input clause. Of course subsumption guarantees uniqueness of clauses, but in a sense uniqueness is lost in this latter combination, in that proof of a clause can be constructed and then subsumed by a rotation equivalent proof of the same clause. This has been investigated more fully using clause trees [4].

References

1. G. M. Adelson-Velskii and E. M. Landis. An algorithm for the organizaton 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. 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 Brunwsick, PO Box 4400, Fredericton, New Brunswick, Canada, 1997.
5. J. D. Horton and Bruce Spencer. Clause trees: a tool for understanding and implementing resolution in automated reasoning. *Artificial Intelligence*, 92:25–89, 1997.
6. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12:23–41, 1965.
7. 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.
8. Bruce Spencer and J.D. Horton. Efficient procedures to detect and restore minimality, an extension of the regular restriction of resolution. *Journal of Automated Reasoning*, 1998. accepted for publication.
9. G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics*, Seminars in Mathematics: Mathematicheskii Institute, pages 115–125. Consultants Bureau, 1969.
10. L. Wos. *Automated Reasoning : 33 Basic Research Problems*. Prentice-Hall, Englewood Cliffs, New Jersey, 1988.