Support Ordered Resolution

Bruce Spencer and J. D. Horton

Faculty of Computer Science, 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. In a binary tree representation of a binary resolution proof, rotating some tree edge reorders two adjacent resolution steps. When rotation is not permitted to disturb factoring, and thus does not change the size of the tree, it is invertible and defines an equivalence relation on proof trees. When one resolution step is performed later than another after every sequence of such rotations, we say that resolution supports the other.

For a given ordering on atoms, or on atom occurrences, a *support ordered* proof orders its resolution steps so that the atoms are resolved consistently with the given order without violating the support relation between nodes. Any proof, including the smallest proof tree, can be converted to a support ordered proof by rotations. For a total order, the support ordered proof is unique. The support ordered proof is also a rank/activity proof where atom occurrences are ranked in the given order.

Procedures intermediate between literal ordered resolution and support ordered resolution are considered. One of these, 1-weak support ordered resolution, allows to resolve on a non-maximal literal only if it is immediately followed by both a factoring and a resolution on some greater literal. In a constrained experiment where literal ordered resolution solves only six of 408 TPTP problems with difficultly between 0.11 and 0.56, 1-weak support ordered resolution solves 75.

1 Introduction

Automated theorem provers, in their search for a proof, must balance the deductive power of a calculus, telling what can be derived from a given point in the search, with restriction strategies, telling which deductions are to be avoided. Clearly the restriction strategy must not remove all of the choices that eventually lead to a proof, at least not without the user's being aware of its incompleteness. But even so, the restriction strategy may remove all shortest proofs, leading to another undesirable effect: the theorem prover takes longer to find a longer proof. An ideal restriction strategy would reduce the space to one richly populated with only short proofs, be simple to implement and quick to check. This is an unrealistic ideal. In this paper we give a reduction strategy is that is quick to check, simple to implement, admits smallest proofs trees, and is almost as restrictive as literal ordered resolution, a widely used and successful restriction.

Literal ordered resolution was used in the CADE-16 System Competition [9] by at least Bliksem, Gandalf, HOTTER, SPASS and Vampire.

In our setting we represent proofs as binary trees, labeled by clauses according to Robinson's resolution method[6]. A node is labeled both by a clause, referring to the conclusion drawn at this point by the resolution, and if it is not a leaf, by the atom that was resolved upon to give this conclusion. Our measure of size is the number of nodes in this tree. Often theorem provers build sequences of formulae where each deductively follows from previous ones. This sequence represents a traversal of a directed acyclic graph (dag) that underlies the tree. The size of an underlying dag is a more natural measure of proof size than the size of the tree. But often if one tree is smaller than another tree then the dag underlying the first tree is smaller than the dag underlying the other.

Support ordered resolution depends on the notion of support between two resolution steps, first defined in [4] and defined for proof trees in [7]. When compared with the literal ordered restriction, used by many theorem provers and explored in [3], support ordered resolution is less restrictive, in that it admits a very specific additional resolution step. On the other hand the support ordered restriction does not increase the size of the smallest tree, unlike literal ordered resolution which may restrict all smallest trees, and in some cases admit only exponentially larger trees and dags, as shown in Example 5 below.

Rotating some tree edge reorders two adjacent resolution steps. When rotation is not permitted to disturb factoring, and thus does not change the size of the tree, it is invertible and defines an equivalence relation on proof trees. For a given total order on atoms, there is a single support ordered tree in each rotation equivalence class. Since the equivalence classes typically contain an exponential number of trees, support ordered resolution substantially reduces the search space.

It is interesting to find a restriction of a resolution calculus that admits a smallest deduction (tree) while substantially reducing the search space, as support ordered resolution does. It is also interesting in that it brings together two apparently different restrictions, the rank/activity restriction[5] and literal ordered resolution. A given ordering on atoms can be used to set the ranks of literals in each clause, and then the rank/activity proof is the support ordered proof.

Viewed as a generalization of other restrictions, we can identify a number of other special cases of support ordered resolution. These suggest themselves as candidates for experiments. One set of these experiments has been done for the special case called 1-weak support ordered resolution, or 1-wso. This proof format depends on a restriction that can be quickly checked on partially closed binary resolution trees. It can be expressed in an intuitively clear way: Recall that the literal ordered restriction allows a resolution only on a maximal atom in each clause; 1-wso allows these and also allows a resolution on some non-maximal atom but then requires an immediate merge on a greater atom from different parents followed by a resolution step on that merged atom. Our experiments indicate that 1-wso provides deductive / reductive tradeoff that is worth taking.

In the following sections we provide necessary background including the recent notion of support between nodes in a binary resolution tree[7]. This is followed by the introduction of support ordered resolution, the restriction of resolution closely related to literal ordered resolution but weakened in those situations where it conflicts with the support of one node for another. We also describe support ordered resolution as a generalization of both rank/activity and literal ordered resolution. This suggests a space of possible strategies. We describe one proof procedure in this space, 1-weak support ordered resolution which is a slight addition to a typical literal ordered resolution theorem prover. We also give the results of our experiments.

2 Background

A binary tree is a set of nodes and edges, where each edge joins a *parent* node to a *child* node, and where each node has one child or has zero and is then called the *root*, and each node has two parents or has zero and is then called a *leaf.* The descendant (ancestor) relation is the reflexive, transitive closure of child (parent).

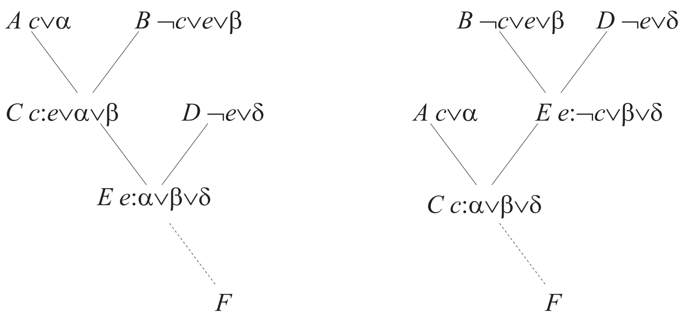

Fig. 1. A binary tree rotation

Given the binary tree fragment T on the left of Figure 1, a rotation is the reassignment of edges so that the tree T' on the right of Figure 1 is produced. The parent C of E becomes the child of E and the parent B of C becomes the parent of E. In other words, the edges (B, C) and (C, E) are replaced by (B, E) and (E, C). If E has a child F in T, then C takes that child in T', or equivalently the edge (E, F) is replaced by (C, F).

We use standard definitions [2] for atom, literal, substitution, unifier and most general unifier. A *clause* is a multiset of literals. 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) a *resolvent* is the clause $(C_1 \vee C_2)\theta$ where θ is a most general unifier of $\{a_1, \ldots, a_m, b_1, \ldots, b_n\}$. The *atom resolved upon* is $a_1\theta$, and the set of *resolved literals* is $\{a_1, \ldots, a_m, \neg b_1, \ldots, \neg b_n\}$.

Definition 1. *A* binary resolution tree*, or* brt *on a set S of input clauses is a binary tree where each node N in the tree is labeled by a* clause label*, denoted $cl(N)$. The clause label of a leaf node is an instance of a clause in S, and the clause label of a non-leaf is the resolvent of the clause label of its parents. A non-leaf node is also labeled by an* atom label*, $al(N)$, equal to the atom resolved upon. The clause label of the root is called the* result *of the tree, $result(T)$. A binary resolution tree is* closed *if its result is the empty clause, $\Box$.*

Our resolution is based on Robinson's original resolution, which we use to define resolution mapping and history path. The resolution mapping tells what happens to each literal in a given resolution step, and the history path tells what happens to it from the leaf where it is introduced to the node where it is resolved away.

The *resolution mapping* ρ at an internal node in a brt maps each resolved literal, $a_1, \ldots, a_m, \neg b_1, \ldots, \neg b_n$, to the atom resolved upon and maps each unresolved member c of C_1 or C_2 to the occurrence of $c\theta$ in the resolvent.

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 resolved upon. 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.

Let T be a binary resolution tree as in Figure 1 with an edge (C, E) between internal nodes such that E has a parent C 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'.

Two trees T_1 and T_2 are *rotation equivalent* if T_1 is the result of a rotation of an edge in T_2, or if T_1 and T_2 are both rotation equivalent to another tree.

The condition on a rotation that no history path through A closes at E is important, because otherwise the rotation would disturb factoring and thus more resolutions might be needed. Consider the left tree in Figure 2. There are history paths for e through both A and B that close at E. The tree after the edge rotation, shown on the right side of Figure 2 has an unresolved occurrence e in the result.

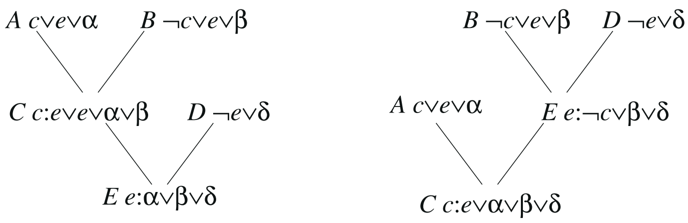

Fig. 2. Not a brt rotation

The calculus of binary resolution trees consists of the following:

1 A node labeled by (an instance of) an input clause is a brt.
2 If T is a brt and θ is a variable substitution then $T\theta$ is a brt formed by replacing each label l in T by $l\theta$.
3 Suppose T_1 and T_2 are brts and no variable appears in both, R_1 and R_2 are the clause labels of the roots of T_1 and T_2 respectively, and R is the clause formed by resolving R_1 and R_2 on atom A with substitution θ. Then T is a brt formed by creating a new node N with atom label A, clause label R and the roots of $T_1\theta$ and $T_2\theta$ are R's parents.
4 If T is a brt and (C, E) is an edge then T' formed by rotating the edge (C, E) is a brt (shown in Figure 1).

Because not all rotations are allowed, sometimes a node N in a brt T remains below another node M, under all sequences of rotations. When this occurs, we say that N *supports* M. Support is a transitive relation, and support cannot be circular. Although it is not exploited in this paper, support can be determined from the history paths of T [7].

3 Support Ordered Resolution

To simplify the discussion we assume that the literal ordering is independent of sign, and thus is an atom ordering. The same results can be developed for literal ordering. Because we use atom ordering, the internal nodes of a brt are ordered in atom ordering on the atom labels of the nodes, *i.e.* the atom resolved upon at that node.

Definition 2. *Given an ordering $\preceq$ of atoms and a binary resolution tree T, we say that a node N is* support ordered *if no descendant of N has higher order than N unless it supports N. T is support ordered if all its nodes are.*

In effect, support ordering is the lexical composition of the support relation and atom ordering.

Theorem 1. *For a given partial ordering $\preceq$ on atoms and a given brt T, there is a support ordered proof tree T^* that is rotation equivalent to T. If $\preceq$ is total, T^* is unique.*

Proof. We proceed by induction on the size of T. If T has one or three nodes, T is trivially support ordered. Suppose T has k nodes. Consider N a node ordered highest in $\preceq$ not supporting any other node in T. Rotate edges above N so that both parents of N are leaves. These rotations are possible because N supports no nodes. Let C_N be the clause label of N in the resulting tree T_0. From this tree, remove the parents L_1 and L_2 of N, so that N is a leaf, and call the resulting smaller tree T_1. By induction there exists T_1^*, a support ordered binary resolution tree of $k-2$ nodes that is rotation equivalent to T_1. N is a leaf of T_1^*. Construct T^* by replacing the parents L_1 and L_2 of N in T^* so that the resolution done at N is the same as in T_0. Because T and T^* are rotation equivalent, the support relations in T^* and T are the same. All nodes in T^* that are also in T_1^* are support ordered, with the possible exception of N. But since any descendants of N that are ordered higher than N are supports of N, N is support ordered.

To argue uniqueness, consider for each leaf L the highest ordered descendant D_L that supports no other descendants of L. Such a node must exist because circular support relations are not possible. Since $\preceq$ is total, D_L is unique for L. But D_L must be the child of L in T^*; otherwise T^* is not support ordered. Thus each leaf of T^* has a unique child. For any node N in T^*, other than the root, this argument can be combined with an induction on the height of the tree above N to show that N has a unique child. Thus T^* is unique. □

Although the support ordering restriction admits a unique proof among the trees rotation equivalent to T, it is not easy for a theorem prover to compute. Support cannot be determined until after a proof is complete. To be useful, a restriction must be applicable to a partial proof. Also the check should require only a simple computation, preferably one with low complexity (constant or linear time) and require information that is local to the proof step. Therefore we define weak support ordered resolution, which can be checked quickly, locally, and on a partial proof. Unfortunately the check cannot be made with just the partial proof we have so far – more steps must be done. We will revisit this problem.

Definition 3. *Given an ordering $\preceq$ and a brt T, an edge between a parent N_1 and its child N_2 in a brt is weak support ordered if the N_1 is ordered higher than N_2 or if the edge is not rotatable. T is weak support ordered if every edge of T is.*

Support ordered resolution is strictly more restrictive than weak support ordered, in that it admits strictly fewer proofs. Consider the following brt: The nodes on a branch are (N_d, N_a, N_c, N_b) where each is a parent of the next. Let a be the atom resolved at N_a, and similarly for b, c, d, and let the atom ordering be reverse alphabetical so it would dictate that a should be resolved before b, *etc.* Suppose N_a supports N_d and N_b supports N_c, so a must be resolved after d and b after c. Thus according to the support relation, we have only two choices to resolve first, d or c, and according to the atom order, we must resolve c first. The resulting support ordered tree is illustrated in the left of Figure 3. All edges in this tree are also weak support ordered; the only rotatable edge is (N_a, N_c)

but it conforms to the atom ordering without being rotated. But weak support order applys the atom ordering only locally. A rotation equivalent tree, shown on the right of Figure 3 has the branch (N_c, N_b, N_d, N_a) which is also weak support ordered, since the only rotatable edge (N_b, N_d) and it also conforms. Thus wso is a weaker restriction, but easier to check, than support order.

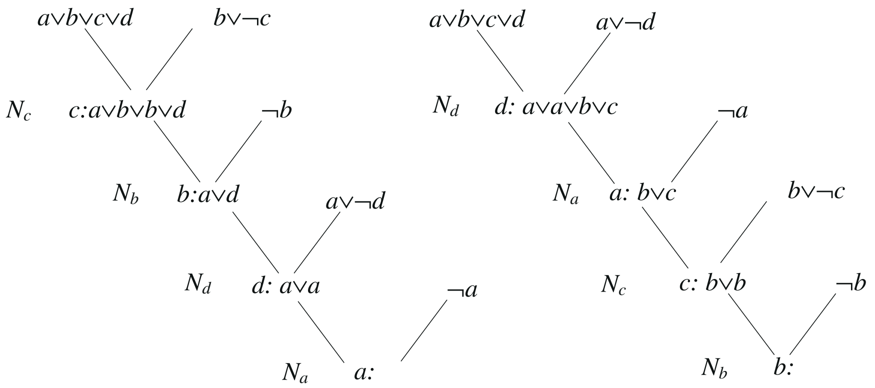

Fig. 3. A support ordered tree and a weak support ordered tree

4 Relation to the Rank/Activity Restrictions

The rank/activity restriction [5], or r/a, can be stated in terms of history paths and where they close.

Given a rank function r that orders atoms in each clause, a brt T is defined to be r-*compliant* if for each leaf L of T the literals of L are resolved away either in r-order, or in the opposite order only if there is another history path that closes with the higher ordered literal's history path, but does not intersect lower ordered one. That is, for each pair l_1, l_2 of literals in L that close at descendants d_1 and d_2 respectively, if $r(l_2) < r(l_1)$ then either (maximal case) d_2 is a descendant of d_1 or (non-maximal case) d_1 is a descendant of d_2 and some history path that does not intersect the path for l_2 also closes at d_1. This is illustrated in Figure 4. The ovals represent nodes in the brt's and the lines with ground symbols represent history paths and where they close. The tree on the left illustrates a maximal resolution, where $l_2 \preceq l_1$ so that l_1 is maximal and should be resolved first. The tree on the right is a non-maximal resolution, where l_1 is resolved later, only after it shares nodes with another history path for l_1. Note that the shared node may be d_2, and that d_1 may be the child of d_2, as it will be in the case of 1-wso resolution, below.

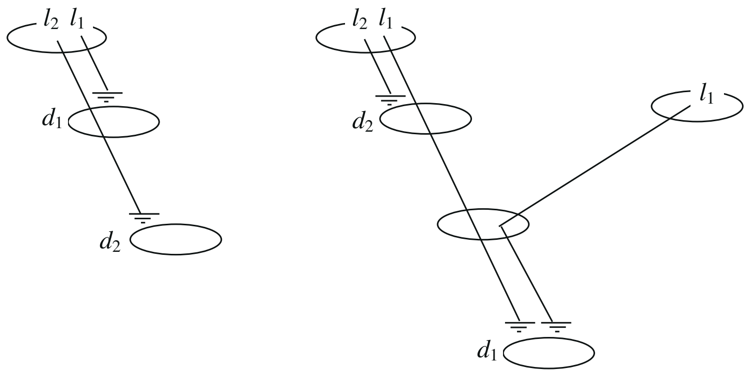

Fig. 4. A maximal and a non-maximal resolution

Alternately one can state the r/a restriction in terms of an activity level associated with each literal. Initially all literals are active, *i.e.* are available for resolution. When a literal of a given rank is resolved, all literals of higher rank are turned off, and are turned back on only if merged at this or some later resolution step. In the definition above, if l_1 is resolved before l_2, l_2 is turned off, and then the other history path closing at d_2 is the one that re-activates l_2.

Note that in [5] it is the lower ranked literal that gets turned off. This arbitrary decision was reversed for this paper to be more consistent with the usual description of literal ordered resolution.

Thus the literal ordered resolution restriction is a special case of the rank/activity restriction, in which the ranks of literals in a clause are specified according to an ordering on the atoms, and non-maximal resolutions is not used. Since there is no reactivation of literals, there is no chance that a resolution step on a non-maximal literal will end in a refutation proof – it can never be resolved away. Thus a procedure to compute literal ordered resolution proofs needs to consider only maximal literals and has much less choice at each step; this leads to a much reduced search space, and accounts for the speed and success of such procedures. R/A procedures have considerable fan out since non-maximal literals are often active as well.

Yet in terms of proof size, the reduced choice of literal ordered resolution can lead to the elimination of all proofs rotation equivalent to smallest proof tree. In Example 5 the smallest proof, found when reverse alphabetical ordering of the atoms is used, has 15 resolutions, but the proof has 32,767 resolutions when the order is alphabetical. This example, for n=4, generalizes to 2^n clauses with n literals each, which can take from $2^n - 1$ to $2^{2^n-1} - 1$ resolutions depending on the ordering.

In terms of deletion strategies, both rank/activity and literal ordered resolution retain completeness when used with tautology deletion. In fact, rank/activity can also be used with the regular and the surgery-minimal restrictions [7] which

$$
\begin{array}{l}
\{\ a,\ \ b1,\ \ c11,\ \ d111\} \\
\{\ a,\ \ b1,\ \ c11, \neg d111\} \\
\{\ a,\ \ b1, \neg c11,\ \ d112\} \\
\{\ a,\ \ b1, \neg c11, \neg d112\} \\
\{\ a, \neg b1,\ \ c12,\ \ d121\} \\
\{\ a, \neg b1,\ \ c12, \neg d121\} \\
\{\ a, \neg b1, \neg c12,\ \ d122\} \\
\{\ a, \neg b1, \neg c12, \neg d122\} \\
\{\neg a,\ \ b2,\ \ c21,\ \ d211\} \\
\{\neg a,\ \ b2,\ \ c21, \neg d211\} \\
\{\neg a,\ \ b2, \neg c21,\ \ d212\} \\
\{\neg a,\ \ b2, \neg c21, \neg d212\} \\
\{\neg a, \neg b2,\ \ c22,\ \ d221\} \\
\{\neg a, \neg b2,\ \ c22, \neg d221\} \\
\{\neg a, \neg b2, \neg c22,\ \ d222\} \\
\{\neg a, \neg b2, \neg c22, \neg d222\}
\end{array}
$$

Fig. 5. 32767 alphabetical a-ordered resolutions are required

are strictly more restrictive. Literal ordered resolution with the surgery-minimal restriction is not complete.

Subsumption deletion works well with literal ordered resolution, but only a weakened form works with rank/activity – if a clause with some inactive literals is used as the subsuming clause, it must have all of its literals reactivated. This can be seen just by observing that otherwise the subsuming clause would not be able to draw a (non-strictly stronger) conclusion in cases where the subsumed clause could.

With respect to the ordering on literals, the r/a restriction does not require that the ordering (or rank) of literals in a clause is consistent with an overall literal ordering. In fact the ordering in r/a is applied on the literal occurrences and identical literals occuring in different clauses need not be ranked the same. Thus the ordering can be total without being liftable. An ordering is said to be liftable if $a \preceq b$ iff $a\theta \preceq b\theta$ for all substitutions θ. In literal ordered resolution, the orderings must be liftable to maintain completeness. Since liftable orderings are often are not total (but see [3]), in these cases the restriction cannot choose a unique maximal literal, leading to fan out in the search space.

Lock resolution [1], incidentally, is closely related. In lock resolution, as in rank/activity, the ranks are assigned to literal occurrences, chosen in any order, not according to an overall literal ordering. Like literal ordered resolution, lock resolution uses only the maximal case of the resolutions in Figure 4. Unfortunately lock resolution does not retain completeness with either tautology deletion or subsumption.

From this discussion we imagine a space of strategies, shown in Figure 6, where the systems on the top all depend on orderings that can not in general be

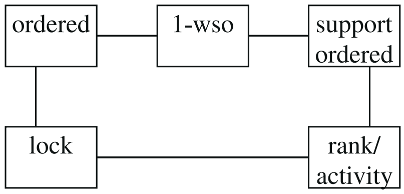

Fig. 6. A space of restrictions. The X axis ranges from the restriction to only maximal resolutions to the allowance of non-maximal resolutions. The Y axis indicates that ordering is defined on literal occurrences vs. defined on the literal set

liftable, thus are not total, and those on the bottom depend on arbitrary ranks within a clause, which can be total. Those on the left depend only on the maximal case of the resolutions in Figure 4, and those on the right, support ordered and rank/activity, use both cases. The system labelled 1-wso is one system in this space, described in the next section. With 1-wso, a very restrictive form of the non-maximal case is permitted: a non-maximal atom may be resolved on, but then a merge and a resolution on a greater atom must follow immediately.

5 Building Support Ordered Proofs

One should not construct support ordered proofs directly from the definition, since the support part of the restriction cannot be determined until the tree is completed. The rank/activity calculus with the ranks set according to the literal or atom ordering should be used instead. It then computes support ordered proofs. Even so, the number of allowable deductions at each stage may be too high, especially in the early stages when all literals are active.

On the other hand the advantages of r/a may be useful: preservation of the smallest proof and compatibility with the surgery-minimal restriction.

The 1-weak support ordered resolution keeps some of the advantages of both. It uses the maximal case and a restricted form of the non-maximal case, from Figure 4. Because the maximal case is sufficient to ensure completeness of the procedure, we are free to further restrict the non-maximal case. In the 1-wso non-maximal case, a non-maximal atom may be resolved but only if some greater atoms occur in each parent and those greater atoms can be merged or factored. Then the factoring is performed, even if a substitution is required. Moreover, the resulting clause is forced to resolve in a conventional way on this factored literal, whether or not it is maximal in the clause and whether or not a non-maximal resolution would otherwise be permitted.

We defined wso resolution to make a slightly modified support condition which was easier to compute. Although weak support ordered resolution looked promising, we noted the restriction could not be applied to a partial proof without knowing more information. Specifically when a resolution on a non-maximal literal is made, it is not immediately knowable that this new node will eventually be supported by the resolution on a greater literal that has been deferred. This guarantee is now made by 1-wso by forcing such a supporting resolution to be the next step. One can see that this new node supports the non-maximal one because the rotation between these two nodes would disturb the merge, as in Figure 2. We need to refer the reader to [7] for the proof that no other sequence of rotations could somehow invert these nodes.

Because the distance from the resolution producing the merge or factor is one node away from the resolution on the merged literal, we call the resulting system 1-weak support ordered resolution.

There is a surprising distinction betweem wso and 1wso. While they are closely related, neither is a special of the other. In Figure 7, the tree on the left is wso but not 1-wso. It is not 1-wso because the non-maximal resolution on b does not end up merging a greater literal. It is wso because of the two possible rotations, the upper one would unorder the nodes and the lower one would disturb a merge on a. The tree on the right of Figure 7 is 1-wso but not wso. It is 1-wso because after the non-maximal resolution on c, the resolution on the greater, merged atom b follows immediately. But it is not wso, because the lower resolutions should be inverted, according to the atom ordering. In Figure 7 wso and support order correspond, so the tree on the left is support ordered while the tree on the right is not.

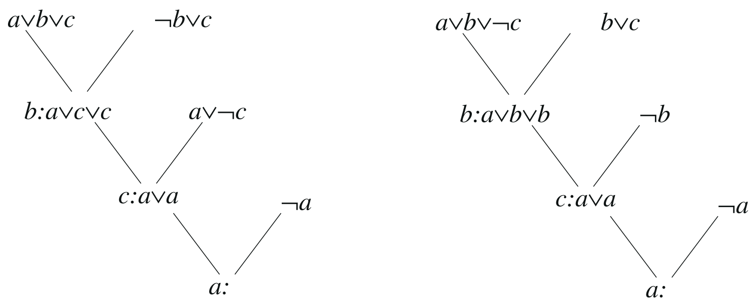

Fig. 7. A binary tree rotation

One of the design decisions of 1-wso was to prevent a great increase in the number of choices beyond literal ordered resolution. The rather complex condition for a non-maximal resolution, and the identification of the forced literal for the next resolution help to prevent such an increase. So 1-wso resembles ordered resolution in that only one literal in each clause is ever available for resolution.

Contrast this with rank/activity which exactly computes support ordered resolution but has all literals available for resolution initially, so it must rely on other heuristics to limit the search.

Rank/activity guarantees that some shortest proof tree is in its search space. 1-wso does not, but it does guarantee that some tree in its search space is non-strictly smaller than the smallest one in the search space for support ordered resolution. For the problem in Figure 5, a 1-wso tree of size 1511 nodes is found by the theorem prover described below. While this is greater than the minimal 15, it is much better that 32767. There may be a smaller 1-wso tree for this example.

The 1-wso system inherits full subsumption from literal ordered resolution between clauses without forced literals. We may include in this any clause with a forced literal that happens to be maximal, which is the most common case. If a clause has a forced literal, and its other literals are subsumed by some clause C with no forced literals, then the subsumption should also be allowed; any clause generated when the forced literal is resolved upon will also be subsumed by C. A clause with a forced literal can also be subsumed by a clause with a forced literal if the forced literals are not different, *i.e.* if they correspond in the subsumption's subset. The subsumption is not allowed if a clause with a forced non-maximal literal tries to subsume a clause with a different literal either forced or maximal, because the candidate subsuming clause is not adequate to be used in place of the candidate subsumed clause.

6 Experiments

The experiments were conducted on a 400MHz Pentium II with 64 Mb RAM running RedHat Linux 6.0 with a theorem prover called pBliksem written by the first author in about 5000 lines of Prolog. The experiments depended on SWI-Prolog 3.2.8. pBliksem borrows heavily from the design of Bliksem, but as it is written in Prolog it has no claims to Bliksem's speed. Nevertheless it is possible to overhaul the underlying data structures in a matter of minutes or hours, and to experiment with unconventional resolution steps. Written in Prolog it has a degree of dependability and clarity. It relies mainly on forward calls, instead of backtracking, so it makes heavy demands on Prolog's garbage collection and tail recursive optimization. Overall it can manage about 5000 inferences in ten minutes, but seldom manages significantly more inferences, because of memory usage. The inference rate varies from 150 inferences per second initially, to under ten per second after ten minutes. While this is clearly not a competitive system, there is nothing to suggest that the results we have obtained with this experimental system would not also be obtained by a better theorem prover. In particular the inferences it chooses are almost identical to those chosen by Bliksem. Occasionally the choices to be made are not determined by the settings of the flags and parameters, and in these tied cases the systems may chose differently.

The experiments were performed on about 80% of the problems in TPTP-v2.3.0 [8] with difficulties in the range strictly above zero and strictly below 0.6. We selected 408 of the 528 such problems. It was configured to use Bliksem's liftable literal order, in which lexicographical ordering is used, but literals that are lexically identical up to a variable are not comparable. The clause ordering depends first on the complexity of the clause, which is the sum of the number of function symbols, predicate symbols and variables. If the complexity of clauses is identical then the sizes of the underlying brts are compared. Two configurations of the theorem prover were tried: literal ordered resolution and 1-wso resolution. Ten minutes of computing time was given to each problem.

Literal ordered resolution solved only six problems, where as 1-wso solved 75, including the six. 1-wso used about 10% more time than literal ordered resolution to solve those six. The numbers of non-maximal resolutions used are given in Table 1. This table shows that non-maximal resolutions were used in most proofs, except the six ones solved by the literal ordered resolution system. Most did not require many non-maximal resolutions but a few did: BOO035-1 required 20 and SYN074-1 required 19.

Number of non-maximal resolutions in the tree	Number of solved	Percentage
0	6	8%
1	32	43%
2	14	19%
3	11	15%
>3	12	16%

Table 1. Counting non-maximal resolutions used by 1-wso

The 1-wso program did best on the problems with difficulty between 0.2 and 0.3 and worst on the problems between 0.3 and 0.4, as shown by Table 2. Both above and below these ranges almost 20% percent of the problems were solved, indicating that the correlation between the difficulty measure and the success of the algorithm is rather small. Perhaps this is because the difficulty of a theorem is a very hard property to measure.

The times taken to solve the problems, reported in Table 3, were scattered somewhat logarithmically with more toward the upper end of the time allocated. Each factor of two allowed about 10 more problems to be solved.

Table 4 shows the number of inferences generated by the prover, which would not be affected by the inefficiency of our implementation. The effect of allowing more inferences is similar to allowing more time, but there is a slightly larger shift to the upper end of the range. Each factor of two allowed about 12 more problems to be solved.

Difficulty d	Number of Problems	Solved	%
$0.1 \leq d < 0.2$	66	12	18%
$0.2 \leq d < 0.3$	84	32	38%
$0.3 \leq d < 0.4$	170	15	9%
$0.4 \leq d < 0.5$	48	8	17%
$0.5 \leq d < 0.6$	40	8	20%
	Total 408		

Table 2. Difficulty of problems and number solved by 1-wso

Time range (seconds)	Number of problems
$0 \leq t \leq 1$	13
$1 \leq t < 2$	4
$2 \leq t < 4$	2
$4 \leq t < 8$	1
$8 \leq t < 16$	11
$16 \leq t < 32$	2
$32 \leq t < 64$	6
$64 \leq t < 128$	13
$128 \leq t < 256$	9
$256 \leq t < 512$	12
$512 \leq t < 600$	2

Table 3. Times taken to solve problems

Inferences Generated g	Number of problems
$0 < g < 10$	1
$10 \leq g < 20$	0
$20 \leq g < 40$	2
$40 \leq g < 80$	9
$80 \leq g < 160$	7
$160 \leq g < 320$	6
$320 \leq g < 640$	11
$640 \leq g < 1280$	11
$1280 \leq g < 2560$	16
$2560 \leq g < 5120$	11
$5120 \leq g \leq 5357$	1

Table 4. Number of inferences required to solve problems

Table 5 shows the number of problems of each category solved. The 1-wso procedure seems to be much better at some categories than others. Note that there is no special treatment of the equality literal in our implementation.

Category	Number of problems	Solved	Percent
BOO	5	1	20%
CAT	11	0	0%
COL	15	2	13%
FLD	26	0	0%
GEO	57	19	33%
GRP	20	7	35%
HEN	5	5	100%
LCL	33	23	70%
LDA	6	1	17%
NUM	3	1	33%
PLA	15	0	0%
PUZ	1	0	0%
RNG	6	0	0%
ROB	1	0	0%
SET	113	11	10%
SYN	91	15	5%
	408	75	18%

Table 5. Problems solved in each category

7 Conclusions

A good restriction strategy in an automated theorem prover depends balancing the economy of choice with the economy of cuts. Literal ordered resolution strongly limits choices but cuts away many proofs, sometimes leaving only very big proofs. Support ordered resolution, or rank/activity, allows more choices, and cuts away all but one of a set of rotation equivalent proofs, and leaves a smallest proof tree. 1-weak support ordered resolution is a step away from literal ordered resolution toward support ordered resolution, allowing very specific extra choices. It does not eliminate all the redundancy of rotation equivalence, nor does it preserve the smallest proof, but it is guaranteed to find non-strictly smaller proofs than literal ordered resolution, occasionally finding much smaller ones. Experiments indicate that the balance here is in favor of increasing choice beyond literal ordered resolution to increase the number of possible proofs. This leaves open the question whether and how to move toward support ordered resolution, allowing modestly more choices while leaving more and smaller proofs in the search space.

References

1. R. S. Boyer. *Locking: A Restriction of Resolution.* PhD thesis, University of Texas at Austin, 1971.
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 B. Spencer. Clause trees: a tool for understanding and implementing resolution in automated reasoning. *Artificial Intelligence*, 92:25–89, 1997.
5. 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.
6. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12:23–41, 1965.
7. B. Spencer and J. D. Horton. Efficient procedures for detecting and restoring minimality, an extension of the regular restriction of resolution. *Journal of Automated Reasoning*, to appear.
8. G. Sutcliffe, C. Suttner, and T. Yemenis. The TPTP problem library. In D. Kapur, editor, *Automated Deduction CADE-12*, number 814 in Lecture Notes in Artificial Intelligence, pages 252–266. Springer-Verlag, Berlin, 1994.
9. Geoff Sutcliffe. The cade-16 system competition. *Journal of Automated Reasoning*, 24:371–396, 2000.