Ancestor Reduction in Binary Resolution Trees

Bruce Spencer and Joseph 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. It is shown how the operation of ancestor reduction, found in tableaux calculi, can be applied to the resolution calculus. An efficient algorithm tells how to reorder some of the previous resolution steps in a binary resolution tree, and thus enables an additional factoring step that removes a literal from a clause. This ancestor reduction operation has a small search space, constrained by the size of the proof tree. If the additional search is successful, the new operation can eliminate literals and reduce the overall time to find a proof. A calculus is defined combining atom (literal) ordered resolution and ancestor reduction. Atom ordered resolution restricts search, but may generate large proofs for some atom orderings. This proposed theorem prover is slightly less restrictive in that an additional search for some ancestor reductions is required, but it generates smaller proof trees under many atom orderings.

1 Introduction

Two major categories of deduction machines are based on resolution calculi and tableaux calculi. Often research into one of these can be applied to the other, but too often, in our opinion, the two communities are unable to take advantage each other's results.

They have many similarities. Given a logic formula, model elimination and tableaux calculi build rooted trees that represent deductions, in which nodes are labeled by subformula. Any leaf in the tree is closed by an operation known as ancestor reduction, in which an ancestor node labeled with a complementary formula is found, perhaps requiring an instantiation of variables. A branch from the root to an open leaf generates a partial model of the formula; a tree with no open leaves is a refutation. Commonly the given formula is in conjunctive normal form (CNF), each node is labeled by a literal, and the children of a node form (an instance of) a clause.

Given a logic formula, resolution systems choose subformulas with complementary parts, perhaps requiring instantiation, and, using the resolution rule of inference, generate new formulae. Subsequent steps may use (parts of) these new formula as well as the given formula. A tree or DAG induced by these steps serves as a deduction. We refer to this as the binary resolution tree (brt) or the binary resolution DAG. When the given formula is conjunctive normal, the subformulas are clauses and the complementary parts are literals.

Thus for refuting a given formula in CNF, the mechanisms are similar in several ways: deductions are tree-based, complementary literals are found, and variable instantiation is used, commonly unification.

Perhaps the most important similarity, when refuting a CNF, is the notion of eliminating a literal that is true in some partial model. In tableaux, a literal labeling a leaf is eliminated by either finding a complementary label on an ancestor, or by building a subtree below that leaf such that the leaves of the subtree are all closed. In resolution, a literal of a certain clause is resolved away (eliminated) either by resolving against a single-literal clause, or by resolving with a non-singleton clause such that all the other literals in that clause can be resolved away. The resolution step can be viewed as a form of extension[1].

Given that the two mechanisms have a similar task to perform, it is interesting to note that they choose different methods to do it: tableaux calculi use extension and ancestor reduction, while resolution calculi uses the resolution rule, and factoring. There are proposals for tableaux systems to use more factoring steps, where these steps are represented in tableax as introduced cut literals [4], or as paths from an open node to a node on a different path[1].

This paper addresses what we see as a gap in the literature: we introduce the use of ancestor reduction in resolution systems.

Consider the following example:

$$\{\neg a \vee d, a \vee \neg b, b \vee \neg c, b \vee c, \neg d\}$$

Given the brt shown in the top left of Figure 1, we see that after four resolution steps there is no refutation. The tableau in the top right is a connection tableau containing the four connections that correspond to these resolutions. It gives rise to a refutation. We might expect both mechanisms to achieve the same result. The tableau gains an advantage by eliminating b with an ancestor reduction two levels up. The brt on the bottom right has the same advantage as the tableau. It comes about by reordering the resolutions on b and c, so that both b's are resolved at once. By inspecting the tableau, we can find a reordering of resolution steps that gives the same effect as the ancestor reduction.

Here we introduce the notion of ancestor reduction in a brt. This reduction comes about by reordering the resolutions steps so that a given literal is eliminated as part of an existing resolution step. No new resolutions are necessary; one of the exisiting resolutions is made to eliminate the given literal in addition to the ones it already resolves.

[1] In fact the story is a bit more complicated in resolution. Given multiple literals in a clause, the mechanism may perform some of the steps that eliminate one literal, leaving some work unfinished, and perform some of the steps to eliminate the others, and then go back to work left over from the first, and so on. Atom ordered resolution, which eliminates literals according to some atom order, does this jumping about. But the deduction tree can always be restructured so that the proof is seen as eliminating each literal in turn. This may require some duplication of subtrees during the restructuring. We do not advocate the restructuring – the point here is only that the mechanism can be *viewed* as eliminating literals one at a time.

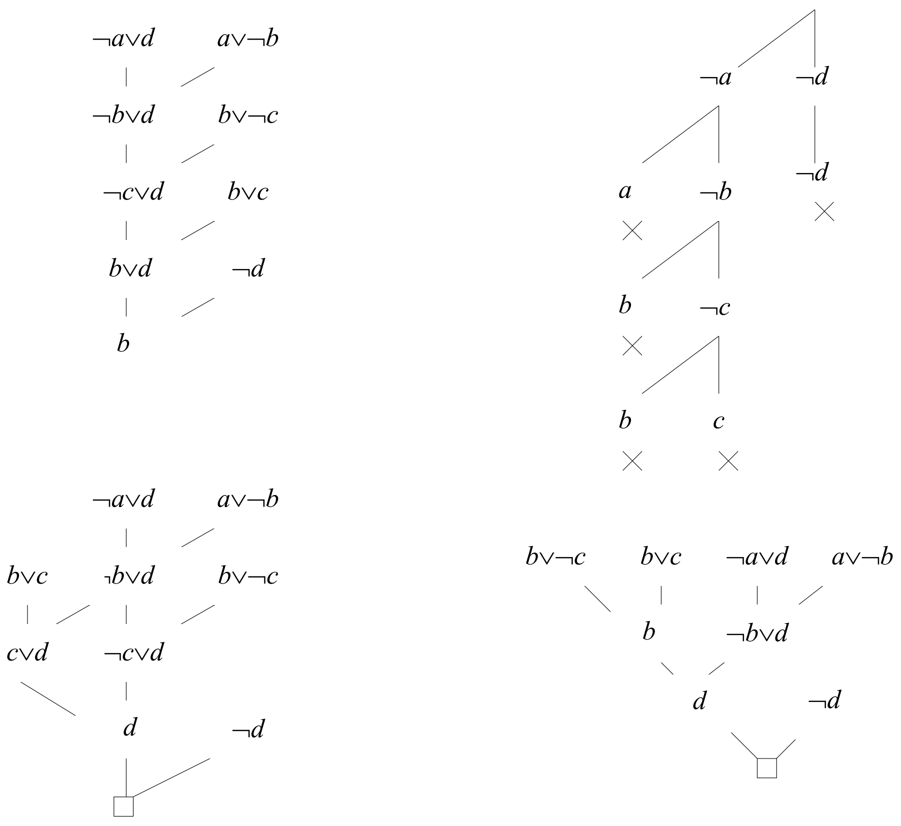

Fig. 1. Clockwise from top left: a brt, a corresponding but stronger tableau, a reordered brt with the strength of the tableau, and a larger atom-ordered brt

One can always make a brt's result at least as strong as a tableau with the same connections by considering different orderings. Conventional wisdom says a theorem prover does not have the time to search for the most effective ordering, and that it is better to fix an ordering and recover from its disadvantages by quickly making more inferences. The tree in bottom left of Figure 1 shows inferences ordered lexically on the atoms. Five resolutions are needed to build a refutation. So only one more resolution is required.

Now consider the following clause set:

$$\{a \vee b, a \vee \neg b, \neg a \vee c, \neg a \vee \neg c\}$$

If resolutions must be on lexically first atoms, then seven resolutions must be done to refute the clauses, whereas the opposite order requires three. Exponentially many more resolutions may be required for an unfortunately chosen order [7]. Our contention is that a small amount of searching for a better order of resolutions may be worth the effort, especially if there are no serious ill effects other than the extra search.

But how can this search be performed? Converting the resolution deduction into a tableau is expensive. Moreover it may not be beneficial. Consider any connection tableau from the clauses in Figure 1 with $b \vee \neg c$ as the top clause. Then no two-level ancestor reduction is possible, and such a tableau has seven connections. For other examples, the connection tableau will indicate a better resolution order only if c-literal reductions are also considered [5].

Say that a given literal in a tableau can *see* all of its ancestors; they are *visible* from the literal [3]. Thus visible literals are potential candidates for an ancestor reduction. Extend the term to visible literals in a brt. A given literal in the root clause of a brt can see those previously resolved literals that are potential candidates for ancestor reduction.

This paper provides the visible algorithm for literals in a brt. Given a literal in the root clause that we want to eliminate, visible returns a literal already resolved upon in the brt if some reordering of resolution steps puts these two literals in the same clause somewhere in new tree. If both the given literal and the visible literal are identical or unifiable, then the resolution that eliminated the visible literal can also eliminate the given literal. Visible returns all such literals in one pass through the tree, so its complexity is linear in the tree size.

We depend on the splay algorithm [6] for restructuring the brt. Given an internal node in the brt, splay restructures the tree so that that node is the new root, if possible, or failing that, it is brought as close as possible to the root of the tree. When applied for ancestor reduction, we have identified some resolved-away literal that is visible from a literal in the root, and we want both of these literals to occur in the same clause. The node where that literal is resolved upon is splayed. The old root becomes an internal node that is a parent of this splayed node. In so doing, both literals are in the clause of this old root, and both can be resolved at the splayed node.

The restructuring done by a splay could be accomplished by reordering pairs of resolution steps, but splay performs the restructuring in a pass through the branch from the internal node to the root. Thus splay is also linear in the tree size.

Our search for an ancestor reduction can also reveal an irregularity in the sense of tableau calculus, which is more specialized than irregularity in resolution calculus. Removing this irregularity corresponds to removing from a tableau branch a section with two identical literals, a loop in the reasoning. This can have the effect of removing literals other than the one for which the ancestor was sought. This result is already known in the literature as the surgery operation [6]. Surgery applies to the more general circumstance of irregularity in a brt where some atom resolved upon at a given node arises again in the clause of a descendant.

The background section gives our definition of similar binary resolution trees, which means that they differ only by the order of the steps given. Within the set of similar trees, we identify when a given pair of nodes can be put on the same branch (visible) and when they must stay on the same branch (support). Using this, we define the ancestor reduction operation, which uses a combination of the visible and splay algorithms. Following this, we point out that ancestor reduction and atom ordered resolution can be used together.

2 Background

We use standard definitions for atom, literal, substitution, unifier and most general unifier. A clause is a disjunction of literals. We use Robinson's notion of resolution; one or more literals selected from one clause resolve against one or more literals selected from another clause, if a unifying substitution exists for the atoms of all the selected literals. This gives rise to a resolvent clause comprised of the remaining literals with the substitution applied. This operation from a pair of clauses to a resolvent clause gives rise to a binary tree, drawn with the resolvent clause below the resolved clauses.

A *binary resolution tree* (brt) 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 root of a clause tree is the newest, lowest node, and its clause label is called the brt's *result*. A binary resolution tree is *closed* if the result is the empty clause, $\Box$. By soundness of resolution, a closed brt on S is a refutation of S.

We are interested in the "history" of a literal in a brt, from its introduction to the tree in a leaf to either its being resolved away, or to its occurence at the root. The path of nodes whose clause labels contain a given literal is called the *history path* of that literal. The tail of the path is a leaf where the literal occurs in (an instance of) an input clause. The head of the path is the lowest clause where that literal's occurrence appears. In the case where the literal is resolved away, the node where it resolves is said to *close* the path. This node is the child of the head of the path. For instance, given the top left brt in Figure 1, the history path for d is every node on the vertical spine of the tree, except the bottom node, where d closes. The history path for b does not close.

We use a strict notion of correspondence between two brts, which allows us to state that they differ only in the order in which the resolutions are done, but the individual resolutions themselves are identical. Two brts T_1 and T_2 on the same set of input clauses are said to *resolve input literals similarly* (or briefly, are *similar*) if, by following the history paths, the same input literals are found to resolve against each other in T_1 as in T_2. More formally, there must exist a mapping ν (1-to-1 and onto) from nodes of T_1 to nodes of T_2 such that leaves map to leaves labeled with the same input clauses, up to variable renaming. This induces a natural mapping from literals in the leaves of T_1 to those in T_2, and by extension, a mapping from history paths in T_1 to those in T_2. Suppose ν can be extended so that a history path closes at node N in T_1 if and only if the corresponding history path closes at $\nu(N)$ in T_2. Then we say T_1 and T_2 resolve input literals similarly.

Note that for similar brts, corresponding nodes are not necesarily labeled with the same clauses, except for the leaves and the root. Also corresponding history paths are not necessarily the same length.

Definition 1 (Visible [6]). *In a given brt with internal nodes N and M, we say that M is* visible *from N, and that N can* see *M, if some similar brt exists in which M is a descendant of N. Otherwise M is invisible from N.*

Definition 2 (Support [6]). *In a given brt with internal nodes N and M, we say that N is* supports *M, and that M* is supported by *N, if N is a descendant of M in every similar tree.*

To simplify the discussion we assume that the literal ordering is independent of sign, and thus is an atom ordering. In the following, the internal nodes of a brt are ordered by applying this atom ordering to the atom labels of the nodes, *i.e.* the atom resolved upon at that node.

Definition 3 (Support ordered resolution [7]). *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 (Completeness and uniqueness of support ordered resolution [7]). *For a given partial ordering $\preceq$ on atoms and a given brt T, there is a support ordered proof tree T^* that is similar to T. If $\preceq$ is total, T^* is unique.*

3 Identifying and performing ancestor reductions

Given a brt with a some literal k in the clause label of the root, we want to know if there is some way to reorder the resolutions so that k can also be removed. In other words we want to find a similar brt, but one that affords an opportunity to close the history path for k. Such an opportunity would arise in the new brt if the new history path for k now contains a node where the literal resolved away matches k. Thus we need to consider the set of similar brt's. On the surface, this search seems time-comsuming, and so does the task of rebuilding the proof, but we want both the search and the restructuring to have low complexity.

The Vis algorithm identifies the internal node where an occurrence of k is already resolved, and the splay adjusts the tree so that the both occurrences of k are in the same internal node. The Vis and Splay algorithms appeared in [6]. Vis is adjusted so it returns literals instead of atoms. The sign of the literal returned is decided by the occurrence of the literal in $cl(B)$.

```
procedure Vis($N, \mathcal{P}_K$)
 if $N$ is a leaf then return $\phi$
 else
   Let $A$ and $B$ be the parents of $N$ and partition $\mathcal{P}_K$ into $\mathcal{P}_A$ and $\mathcal{P}_B$, which
   are the sets of history paths that 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
     return Vis($A, \mathcal{P}_A \cup \mathcal{C}_A$) $\cup$ $Vis(B, \mathcal{P}_B \cup \mathcal{C}_B)$;
   else
     Let literal $a \in cl(B)$ such that $al(N)$ occurs in $a$.
     return $\{a\} \cup Vis(A, \mathcal{C}_A) \cup Vis(B, \mathcal{P}_B)$
   endif
 endif
```

AncestorReduce is given a brt T and a literal k in the clause label of the root of T. It returns T itself if no reduction is possible, or otherwise, it returns a tree similar to T except that k is no longer the root, and the history path for k is closed.

```
procedure AncestorReduce($T, k$)
 Let $R$ be the root $T$.
 Let $\mathcal{P}_K$ be the set of history paths in $T$ for $k$, that include $R$.
 Let $V$ = Vis($R, \mathcal{P}_K$).
 if $k \in V$ then
   Let $K$ be the node in $T$ where the visible $k$ was found.
   Let $T' = splay(T, K)$
```

```
    Consider $R$ in $T'$. It is no longer the root, but it is a parent of $K$. $K$ is on
    all the history paths $\mathcal{P}_K$, which do not close. Also $K$ is on the history paths
    for the visible $k$ and these do close. Extend the resolution at $K$ so that all
    history paths from both sets close.
    return $T'$
  endif
```

A call to splay(T, K) [6] is given an internal node K in a binary resolution tree T and returns a new binary resolution tree T' such that all descendants of K cannot see K in T'. Thus K is as close to the root as possible.

This section extends the work reported in [6], where all visibility was defined in terms of visible nodes and visible atoms. Here we consider visible literals.

4 Combining with ordered resolution

We propose the following reasoning procedure, based upon an atom ordering. Literals are ordered in a clause by their atoms. A largest literal in a clause can be resolved, or a smaller literal be resolved as long as larger literals can be ancestor reduced. We show that such a system builds all support ordered proofs. Thus it will produce minimally sized brts, independent of the atom ordering.

Definition 4 (SOAR tree). *Let $\mathcal{C}$ be a clause set and let $\preceq$ be a total order on the atoms of $\mathcal{C}$. An brt constructed by* support ordered resolution using ancestor reduction*, or SOAR tree, is a binary resolution tree on $\mathcal{C}$, according to one of the following:*

- *A leaf node whose clause label is an instance of a clause from $\mathcal{C}$.*
- *A brt comprised of a root node and two SOAR subtrees T_1 and T_2, where the clause label of the root is the resolvent of T_1 and T_2 on the maximal atom from each.*
- *A brt T constructed as follows: Let T' be a new brt from resolving two SOAR trees T_1 and T_2 on a possibly non-maximal literal from each, such that for $i = 1, 2$ all literals from the T_i clause greater than the literal resolved upon, are visible in T_{2-i}. Let T be T' after ancestor reduction removes all of these higher ordered literals.*

Note that the third item above is strictly more general than the second. This third rule adds support ordered nodes to the tree. Thus the calculus generates many support ordered brts. One might think that the SOAR calculus generates all support ordered trees. Assuming the atoms are ordered alphabetically, consider the clauses

$$\{\neg a, a \vee b \vee d, a \vee c \vee \neg d, \neg b, \neg c\}$$

There is a four-resolution refutation, using each clause once. But the SOAR calculus must make two resolutions with the $\neg a$ clause, and so requires five.

5 Summary

In this paper we propose a strategy for identifying, in a binary resolution tree, any opportunity for ancestor reduction. Ancestor reduction is accomplished by restructuring a brt so that the same resolutions are done in a different order, and the new tree allows an extra literal to be resolved upon in an existing resolution.

The SOAR calculus has the property of being slightly more permissive than atom ordered resolution alone, in that it requires more searching and allows more resolutions. But it gives smaller sized trees, under many atom orders. Unlike the rank/activity resolution [2] it does not have a large number of initial choices. Unlike weak support ordered resolution [7], it works well with subsumption.

References

1. Peter Baumgartner, J.D. Horton, Bruce Spencer. Merge path improvements for minimal model hypertableau. In *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 51–65. Springer, 1999.
2. 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, July 1998.
3. J. D. Horton and Bruce Spencer. Clause trees: a tool for understanding and implementing resolution in automated reasoning. *Artificial Intelligence*, 92:25–89, 1997.
4. R. Letz, K. Mayr, C. Goller. Controlled integration of the cut rule into connection tableau calculi. *Journal of Automated Reasoning*, 13:297–337, 1994.
5. R. E. Shostak. Refutation graphs. *Artificial Intelligence*, 7:51–64, 1976.
6. Bruce Spencer and J. D. Horton. Efficient algorithms to detect and restore minimality, an extension of the regular restriction of resolution. *Journal of Automated Reasoning*, 25:1–34, 2000.
7. Bruce Spencer and J. D. Horton. Support ordered resolution. In David McAllester, editor, *Automated Deduction – CADE-17*, number 1831 in Lecture Notes in Artificial Intelligence, pages 385–400. Springer, June 2000.