Counting the Number of Equivalent Binary Resolution Proofs

Joseph D. Horton*

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

Abstract. A binary resolution proof is represented by a binary resolution tree (brt) with clauses at the nodes and resolutions being performed at the internal nodes. A rotation in a brt can be performed on two adjacent internal nodes if the result of reversing the order of the resolutions does not affect the clause recorded at the node closer to the root. Two brt's are said to be rotationally equivalent if one can be obtained from the other by a sequence of rotations. Let $c(T)$ be the number of brt's rotationally equivalent to T. It is shown that if T has n resolutions, all on distinct atoms, and m merges or factors between literals, then

$$c(T) \geq 2^{2n-\Theta(m \log(n))}$$

Moreover $c(T)$ can be as large as $n!/(m+1)$.
A dynamic programming polynomial-time algorithm is also given to calculate $c(T)$ if T has no merges or factors.

1 Introduction

Binary resolution [10] is a commonly used technique in automated reasoning. There are many different ways to represent a binary resolution proof. The first method used was just to make a list of the clauses needed in whatever one is trying to prove, and record with each clause whether it was an input clause, or was produced by resolving two previous clauses in the list. However, the order in which many of the resolutions steps are performed is not important, so many effectively identical proofs would be considered to be different if a proof were only considered to be a list.

A better method is to consider a proof to be a binary tree, where the child of two nodes is the result of resolving the clauses at the parent nodes. Here the binary resolution tree (brt) of [12, 8] is used to represent such proofs. Many resolution-based automated reasoners construct proofs that can be effectively represented by brts.

Binary resolution sometimes is associative, in that for some clauses A, B and D, $(A *_c B) *_e D = A *_c (B *_e D)$. Resolution on an atom p is denoted by the operation $*_p$. Thus it is possible to change the order of the resolutions in the

* Research supported by a grant from NSERC of Canada

brt and produce essentially equivalent proofs. The clauses produced along the way are different, so that as lists of clauses, the proofs could look quite different. This *associative redundancy* of binary resolution is studied in [12, 8], altho it is not called that in those papers.

Binary resolution has *commutative redundancy* also, which is much easier to handle. Because $A *_c B = B *_c A$ for all clauses A and B which can be resolved, resolution procedures only perform one of these resolutions. One easy way to avoid the duplication is to order the clauses into a "waiting" list, select each one at a time as a "given" clause, resolve the given clause with any clause which previously had been a given clause, and put the resulting clauses into the "waiting" list. In this way, each pair of clauses is chosen only once. Assuming that clause A is chosen before clause B, the pair $\{A, B\}$ is resolved only when A is the chosen clause, and not when B is the chosen clause. If a resolution procedure does not avoid this commutative redundancy, then each brt with n resolutions could be constructed in 2^n different ways. By deleting identical clauses using forward subsumption, such a procedure would only produce each brt in two ways.

Associative redundancy is more difficult to deal with than commutative redundancy mainly because resolution always commutes but does not always associate. If two of the clauses have a common literal, and it is one of the resolved literals, then in one order of the resolutions it can be merged and then resolved away, but with the other order it is left in the result. For example:

$$(ace *_c b\overline{c}e) *_e d\overline{e} = abd$$

but

$$ace *_c (b\overline{c}e *_e d\overline{e}) = abde$$

The application of associativity requires two adjacent nodes to be rotated in a brt, reversing the order of the resolutions, and still produce the same resulting clause at the lower node. This rotation is similar to the rotation performed in AVL-trees when rebalancing [1]. A general example is shown in Figure 1. Two brts are said to be *rotationally equivalent* if one can be obtained from the other by a sequence of these rotations. Rotation equivalent captures precisely the associative redundancy of binary resolution.

An automated binary resolution procedure generally produces multiple brts that are rotationally equivalent to each other. Once produced, the extra brts are typically removed by subsumption. The rank/activity restriction of [8] combined with a fair resolution procedure (any allowed resolution is eventually performed) constructs exactly one brt from each equivalence class. Variants of resolution that order the literals with a clause, such as ordered resolution [9, 4] or lock resolution [2], also avoid constructing equivalent brts, but these procedures do not produce brts in all equivalence clauses. Thus the rank/activity restriction avoids precisely the associative redundancy of resolution.

How big are the equivalence classes of rotationally equivalent brts? In this paper it is shown that f a brt contains n internal nodes and m mergings of literals, then the number of brts rotationally equivalent to a given brt is $c(T) \geq$

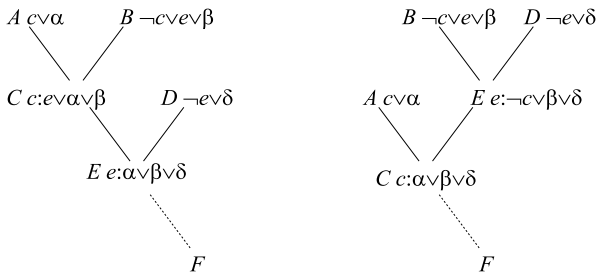

Fig. 1. A binary tree rotation

$2^{2n-\Theta(m\log(n))}$, and can be as great as $n!/(m+1)$. Deleting identical clauses allows one to construct a number of equivalent brts equal to the number of nodes which can be rotated to the root of the brt. In clause tree terms, this is the number of internal atom nodes which are not internal to a merge path. Supposing that m is not too big, then the r/a restriction saves a factor of between about 4 and n on the number of resolutions performed.

As only the structure of the proofs are important in this paper, it can be assumed for the most part that all atoms are ground atoms. We can identify the nodes of the brt by the literal from the instances of the literals from the input clauses resolved at the node instead of the clause produced. Also one can consider in any given proof that each resolution is done on a different atom from all the other resolution in the proofs. These assumptions prevent consideration of improvements to the proof by inserting different factoring operations, or different merges. Indeed factors are often called merges thruout this paper. However the results of this paper apply to first order logic wihtout these restrictions just as well as to propositional logic.

The second section gives background concerning brts and clause trees [7]. Clause trees are another method of considering binary resolution proofs, and are essential for understanding the results in this paper. Section 3 considers the extreme cases for the number of equivalent brts when there are no merges of literals in the proofs. Section 4 gives a dynamic programming algorithm to calculate the exact number of brts equivalent to a given brt without any merges. Section 5 considers the extreme cases when the proof is allowed to have merges. Section 6 lists some open questions.

2 Background

The reader is assumed to be familiar with the standard notions of binary resolution [3]. Resolution proofs can be represented by the following type of proof tree.

Definition 1. *A* binary resolution tree, *or* brt *on a set $\mathcal{S}$ 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 $\mathcal{S}$, 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* sub-brt *of a brt T is a brt which consists of a node of T together with all its ancestor nodes, induced edges and their labels.*

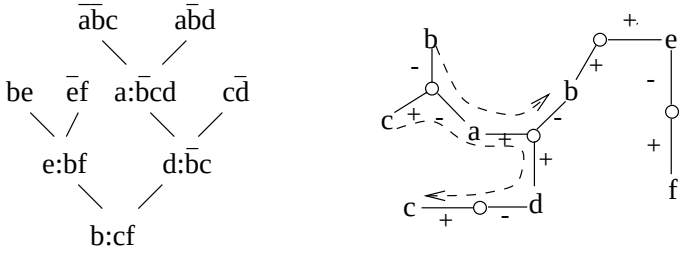

Fig. 2. A binary resolution tree and a corresponding clause tree

For the brt in Figure 2, $\mathcal{S} = \{\{b, e\}, \{\overline{e}, f\}, \{\overline{a}, \overline{b}, c\}, \{a, \overline{b}, d\}, \{c, \overline{d}\}\}$. The result of the brt is the clause $\{c, f\}$. The order of the parents of a node is not defined, thereby avoiding commutative redundancy.

Merging of literals during the construction of a proof is only done immediately after a resolution, at the first possible opportunity. This differs from [12] where factoring was delayed as long as possible instead. Either way there is no need to have nodes for factoring, as it always appears as part of the resolution step. Forcing factoring to be done early agrees better with what some automated reasoning procedures do. To avoid some trivial paths, in both brts and in the clause trees that are defined below, we assume that all general factors of the input clauses are available.

The *resolution mapping* ρ at an internal node in a brt maps each resolved literal to the atom resolved upon, and maps each unresolved literal c to the occurrence of $c\theta$ in the resolvent, where θ is the product of the unifications used in the resolution and any required factoring operations. Let the nodes $(N_0, \ldots, N_n)$ occur in a brt 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. However N is not considered to be on the 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.

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'. (See Figure 1).*

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. However the clause label of the root is not changed.

Lemma 1. *Given a binary resolution tree T with an internal node C and its child E, the rotation of edge (C, E) generates a new binary resolution tree and $cl(C) = cl(E)$ up to variable renaming.*

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. Rotation equivalence is an equivalence relation.

In a brt, the atom being resolved upon labels the nodes instead of the edges as is usually done in proof trees [3,6]. In the equivalent proofs studied in this paper, what is constant between them is what instances of input literals merge and resolve together. Which history paths close together is the important thing. Rotations do not affect what literal instances merge and resolve.

It is possible that a brt has two sub-brts which are isomorphic, in that the subtrees are isomorphic, and the atom and clause labels of corresponding nodes are the same. A theorem prover could thus have two different nodes with a common parent, and the proof found would become an acyclic directed graph (dag) instead of a tree.

Binary resolution proofs can also be represented by an entirely different tree structure, the clause tree, introduced in [7]. The definition of clause tree in this paper differs from that in [7]. There the definition is procedural, in that operations that construct clause trees are given. Here the definition is structural. Conceptually, a clause tree represents a clause together with a proof from a set of input clauses. An input clause is represented by a complete bipartite graph $K_{1,n}$ or claw, in which the leaves correspond to the atoms of the literals of the clause, modified by the sign on the edge connecting the leaf to the central vertex. Such a clause tree is said to be *elementary*. A new clause tree can be built by resolving two complementary literals from different elementary clause trees. Identify the two leaves, so the resolved literal becomes an internal node of the tree, thereby

building a clause tree with leaves still corresponding to the other literals of the clauses. Thus leaves of the clause tree correspond to the literals of the clause. If there are two leaves with unifiable or identical literals, this corresponds to two unifiable or identical literals occurring in the clause. When two such literals are factored or merged, one of the literals is removed from the clause. To represent this action in the clause tree, a *merge path* from the leaf corresponding to the removed literal to the other leaf corresponding to the now identical literal.

Definition 2 (Clause Tree). *$\mathcal{T} = (N, E, L, M)$ is a clause tree on a set S of input clauses if:*

1. *(N, E) as a graph is an unrooted tree.*
2. *L is a labeling of the nodes and edges of the tree. $L : N \cup E \to S \cup A \cup \{+, -\}$, where A is the set of instances of atoms in S. Each node is labeled either by a clause in S and called a clause node, or by an atom in A and called an atom node. Each edge is labeled $+$ or $-$.*
3. *No atom node is incident with two edges labeled the same.*
4. *Each edge $e = \{a, c\}$ joins an atom node a and a clause node c; it is associated with the literal $L(e)L(a)$.*
5. *For each clause node c, $\{L(a,c)L(a) | \{a, c\} \in E\}$ is an instance of $L(c)$. A path $(v_0, e_1, v_1, \ldots, e_n, v_n)$ where $0 \leq i \leq n$, $v_i \in N$ and $e_j \in E$ where $1 < j < n$ is a* merge path *if $L(e_1)L(v_0) = L(e_n)L(v_n)$. Path $(v_0, \ldots, v_n)$* precedes *$(\prec)$ path $(w_0, \ldots, w_m)$ if $v_n = w_i$ for some $i = 1, \ldots, m-1$.*
6. *M is the set of merge paths called chosen merge paths such that:*
   (a) the tail of each is a leaf (called a closed *leaf),*
   (b) the tails are all distinct and different from the heads, and
   (c) the relation $\prec$ on M can be extended to a partial order, that is, does not contain a cycle.

An *open leaf* is an atom node leaf that is not the tail of any chosen merge path. The disjunction of the literals at the open leaves of a clause tree $\mathcal{T}$ is called the *clause* of $\mathcal{T}$, $cl(\mathcal{T})$, and is identical to the clause at the root of a corresponding brt.

Some relationships between brts and clause trees are discussed in [12]. Among them are: internal nodes of brts correspond to atom nodes of the clause tree; leaves of a brt correspond to clause nodes of the clause tree; a history path in a brt corresponds to an edge of a clause tree. See Figure 2 to see a clause tree that corresponds to a brt.

In this paper we disallow merge paths of length two since they correspond to factoring an input clause. Any most general factor of an input clause is allowed to form an elementary clause tree instead.

When a merge path is chosen between two open leaves, there is no reason to choose one direction over the other, unless one specifies some arbitrary heuristic. The corresponding proofs remain exactly the same. One can define a *path reversal* operation which changes the clause tree except that one merge path runs in the opposite direction, which may cause some other merge paths to be modified somewhat. Then two clause trees are said to be *reversal equivalent* if there is

a sequence of path reversals which transform one tree to the other. Perhaps a better alternative, developed in [11] in a slightly different context and put into general clause trees in [7], is the foothold restriction, which can be used to make a arbitrary choice that is consistant regardless of the order of the resolutions.

Since both clause trees and brts are simply ways to write down resolution proofs, they are also equivalent to each other. The rotational equivalence classes of brts are in one-to-one correspondence with the reversal equivalence classes of clause trees [12].

3 Proofs without merges

A resolution proof that does not contain any step in which two literals merge, or factor, corresponds to a clause tree with no chosen merge paths. The resolutions can be done in any order. Such proofs have been considered in several ways. Given a set S of clauses, it is known that the following statements are equivalent: S has an input refutation; S has a unit refutation; the set of factors of S contains a relative Horn subset which is unsatisfiable. These are equivalent to S admitting a clause tree without merge paths [7]. If there are n resolutions, then there are $n!$ different proofs, written as a sequence of resolutions, for which every resolution is relevant to the proof. The clause tree corresponding to these proofs is unique, since there are no merge paths to be reversed. Many brts, all rotationally equivalent, can correspond to these proofs, as a sequence of resolutions, yet each brt may correspond to many of these proofs.

Given a brt T with n atom nodes, let $c(T)$ be the number of brt's rotationally equivalent to T. Similarly define the number of brt's corresponding to a given clause tree $\mathcal{T}$ to be $c(\mathcal{T})$.

Theorem 2. *If T is a mergeless brt with n internal nodes, so that it corresponds to a mergeless clause tree $\mathcal{T}$, then*

$$C_n \leq c(\mathcal{T}) \leq n!$$

where $C_n = ((2n)!/(n!(n+1)!)$ is the n^{th} Catalan number. These bounds are tight.

Proof. Once the order of the n resolutions is determined, so is the brt. Therefore $c(T) \leq n!$, the number of possible orderings of the resolutions. If $\mathcal{T}$ is a(n extended) claw $K_{n,1}$, then T itself is a linear binary tree, with every resolution being between an input unit clause and the "central" clause. The resolutions can be done in any order, and so the number of equivalent brts is $n!$. See Figure 3.

If $\mathcal{T}$ is a path, see Figure 4, then the corresponding brt T can be shaped like any binary tree with n internal nodes. Let $f(n)$ be the number of brts corresponding to a clause tree which is a path, containing n internal atom nodes. Assume that the k^{th} internal atom node corresponds to the last resolution. Removing the atom node breaks the path into two clause trees which are paths themselves,

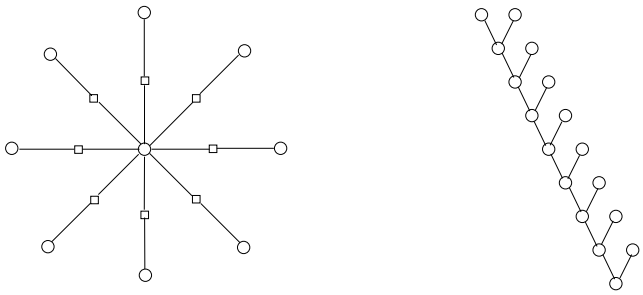

Fig. 3. A clause tree maximizing the number of equivalent brts, with shape of corresponding brt.

Fig. 4. A clause tree minimizing the number of equivalent brts.

one with $k-1$ internal atom nodes and the other with $n-k$. The number of brts then is $f(k-1)f(n-k)$. Summing over all the choices of internal atom nodes, any of which can be the last resolution, $f(n) = \sum_{k=1}^{n} f(k-1)f(n-k)$. The solution to this recurrence is well-known to be $f(n) = C_n$. Therefore the lower bound is tight.

Lastly we prove the lower bound. Assume that $c(\mathcal{T}^*) \geq C_k$ if $\mathcal{T}^*$ has $k < n$ internal atom nodes. Let the internal atom nodes of $\mathcal{T}$ be $\{a_1, a_2, \ldots, a_n\}$. For each atom node a_i, let the subtrees determined by breaking $\mathcal{T}$ at the atom node a_i be $\mathcal{T}_i$ and $\mathcal{T}_i'$. Assume that $\mathcal{T}_i$ has no more internal atom nodes than $\mathcal{T}_i'$, and that this number is b_i. Thus the number of internal atom nodes in $\mathcal{T}_i'$ is $n-1-b_i \geq b_i$. Then

$$c(\mathcal{T}) \geq \sum_{i=1}^{n} C_{b_i} C_{n-1-b_i} \equiv f(\mathcal{T}) \tag{1}$$

The function f defined in equation (1) is a lower bound on $c(\mathcal{T})$ which can be calculated from the shape of $\mathcal{T}$. We show that if $\mathcal{T}$ has a node of degree 3, then we can find another clause tree with n internal atom nodes for which this lower bound f is smaller.

Note that

$$\frac{C_k}{C_{k-1}} = \frac{(2k)!}{k!(k+1)!}\frac{(k-1)!k!}{(2k-2)!} = \frac{(2k)(2k-1)}{k(k+1)} = 4 - 6/(k+1) \tag{2}$$

This ratio increases with k. But the product $C_k C_{n-1-k}$ decreases as k increases for a fixed n, as long as $2k < n$, since

$$\frac{C_{k-1}C_{n-k}}{C_k C_{n-1-k}} = \frac{(4 - 6/(n-k+1))}{(4-6/(k+1))} > 1 \tag{3}$$

It follows that the product $C_k C_{n-1-k}$ is minimized when $k = \lfloor (n-1)/2 \rfloor$.

Suppose that $\mathcal{T}$ has a node c of degree 3 or more. Rename the atom nodes of T such that a_1 and a_2 are the two atom nodes adjacent to c which make b_1 and b_2 are as small as possible, with $b_1 \leq b_2$. Then b_1 is the number of internal atom nodes in the subtree obtained by deleting a_1 from $\mathcal{T}$ and does not contain the clause node c. Similarly b_2 is the number of internal atom nodes in the subtree obtained by deleting a_2 from $\mathcal{T}$ and does not contain the clause node c. Thus $b_1 \leq b_2 < n - 2 - b_1 - b_2$.

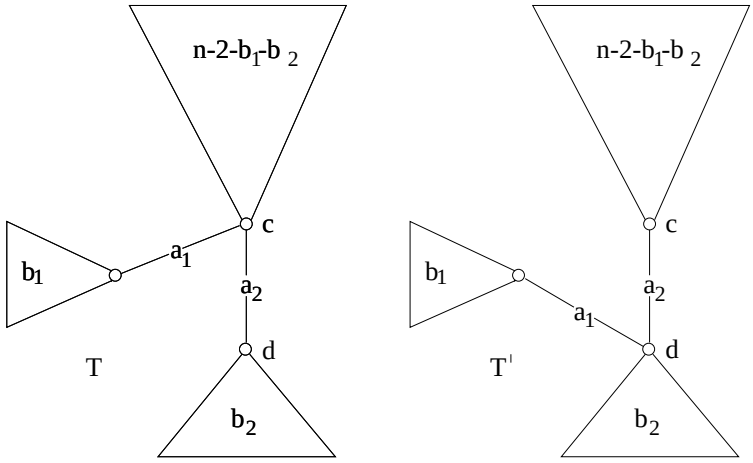

Fig. 5. The construction to show the lower bound

We modify $\mathcal{T}$ to make a new clause tree $\mathcal{T}'$ by detaching a_1 from c, and re-attaching it to the other clause node d adjacent to a_2. See Figure 5. Let b_i' be defined for $\mathcal{T}'$ in the same way as b_i is defined for $\mathcal{T}$. Thus b_i' is the number of internal atom nodes in the smaller clause tree obtained by deleting a_i from $\mathcal{T}'$. Then $b_i = b_i'$ except for $i = 2$. For this case $b_2' = min\{b_1+b_2+1, n-2-b_1-b_2\} > b_2$. By equation (3), $f(\mathcal{T}') < f(\mathcal{T})$. Thus f is minimized only for clause trees with all nodes of degree less than 3, that is, only for paths. Since for a path the value of f from equation (1) is exactly $C_n = \sum C_i C_{n-1-i}$, the lower bound is proved.□

4 A polynomial algorithm to count mergeless proofs

Given a mergeless clause tree $\mathcal{T}$, it is possible to count in polynomial time the exact number of corresponding brts. The algorithm uses dynamic programming. Given a specific clause node c of $\mathcal{T}$, let $f(c, h, \mathcal{T})$ be the number of brt's corresponding to $\mathcal{T}$ in which the leaf node corresponding to c occurs at height h. Once $f(c, h, \mathcal{T})$ is known for any specific c and all $h = 1, \ldots, n$, one can calculate $c(\mathcal{T})$ by summing $f(c, h, \mathcal{T})$ over all values for h. If $\mathcal{T}$ contains zero atom nodes and one clause node c, then $f(c, 0, \mathcal{T}) = 1$ and $f(c, k, \mathcal{T}) = 0$ for $k > 0$.

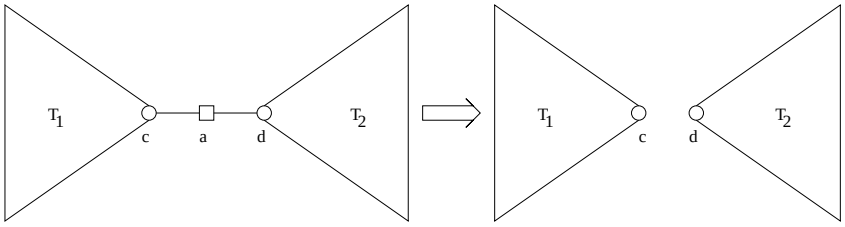

Fig. 6. Breaking the clause tree

Next we need a recursive formula for $f(c, h, \mathcal{T})$. Let a be an atom node adjacent to c, with d being the other clause node adjacent to a. Deleting a, and its incident edges, breaks $\mathcal{T}$ into two smaller clause trees $\mathcal{T}_1$ containing c and $\mathcal{T}_2$ containing d. See Figure (6).

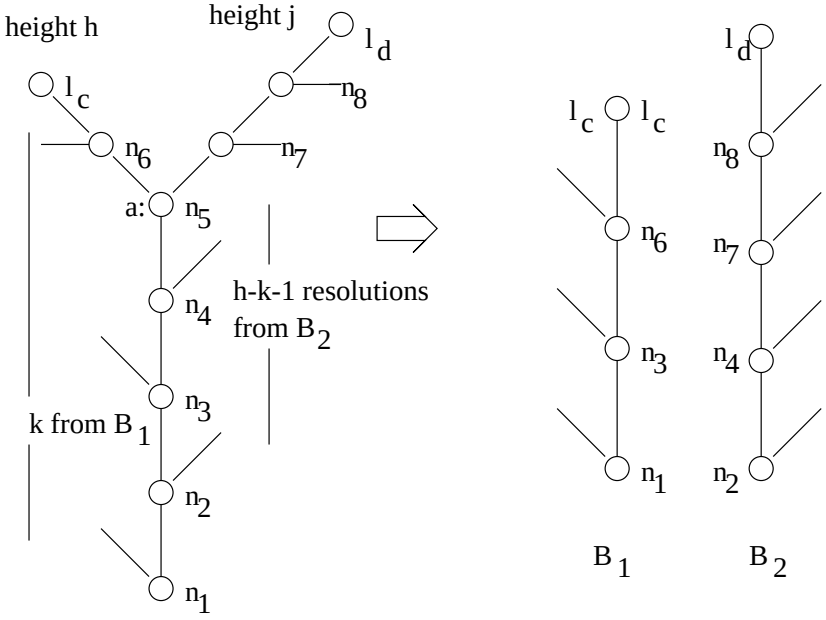

Fig. 7. Breaking the brt

Consider any brt B corresponding to $\mathcal{T}$. Let the leaf l_c corresponding to c be at height h in B. Consider the path from this leaf to the root. It must contain the node at which the resolution corresponding to a is done. Suppose that it contains k nodes at which other resolutions from $\mathcal{T}_1$ are done. Then it has $h - k - 1$ nodes at which resolutions from $\mathcal{T}_2$ are done, excluding the resolution corresponding to a. If one were to remove the nodes corresponding to atom nodes

of $\mathcal{T}_2$ and reconnect this path from l_c to the root, then the result would be a brt B_1 corresponding to $\mathcal{T}_1$ in which the leaf corresponding to c would be at height k. Similarly removing the nodes correponding to atom nodes of $\mathcal{T}_1$ would leave a brt B_2 corresponding to $\mathcal{T}_2$, with the leaf corresponding to d being at height $j \geq h-k-1$. See Figure 7. Looking at the path from l_c to the root in B again, the nodes from B_1 can be inserted anywhere on this path, as long as they are below the node corresponding to a. The number of ways in which the nodes from $\mathcal{T}_1$ can be placed is $\binom{h}{k}$. Summing over the possible values of k,

$$f(c,h,\mathcal{T}) = \sum_{k=0}^{h-1} (\binom{h}{k} f(c,k,\mathcal{T}_1) \sum_{j \geq h-k-1} f(d,j,\mathcal{T}_2))$$

The values of f can now be calculated using dynamic programming. Consider the clause tree to be a tree rooted at some clause node, complete with child/parent and ancestor/descendant relationships. Suppose that we evaluate f for each subtree rooted at any clause node c. To evaluate f for c, first all the values of f for all immediate descendant clause nodes must be known, for the subtree rooted at that clause node. Then the recursion formula must be applied for each of the atom node children of c, so that the formula, for each possible value of h, is applied up to $degree(c)$ times at c. The recursion formula must be applied once for each internal atom node of $\mathcal{T}$, for values of h up to the number of internal atom nodes below the parent clause node of the atom node in the rooted $\mathcal{T}$.

The internal summation for $\mathcal{T}_2$ can be found for all the values of h in linear time. Thus the values of f can be calculated in time quadratic in h for any given c and T, assuming that the values of F are known for the subtrees and the nodes c and d. The set of recursions need to be calculated once for each atom node of $\mathcal{T}$. Hence the whole calculation can be done in time cubic in the number of atom nodes.

5 Proofs with merges

If the proof includes merging or factoring of literals, then the above arguments are not valid. Each merge requires that some of the resolutions be performed before the resolution of the merged literal. Hence the number of equivalent proofs is smaller for the same number of resolution steps. A single merge can decrease the number of equivalent brt's by as much as a factor of the number of resolutions. In the case of a single merge in a proof, the factor is exactly the number of internal atom nodes on the merge path in the clause tree. If the merge were just within an input clause, it does not change the number of equivalent brt's at all. It has been assumed that merges always occur between literals from different input clauses, specifying that different occurrences of a given input clause are considered to be different input clauses. Moreover we assume that no two distinct literals from the same occurrence of a clause are merged with the same literal of another clause. In a clause tree this means that two merge paths with

the same head cannot have their tails being adjacent to the same clause node, because this implies hat the two literals of one input clause are factored.

The following theorem defines a lower bound on $c(\mathcal{T})$, denoted by $lb(n,m)$, which is a product of Catalan numbers.

Theorem 3. *Let T is a brt with n internal nodes and m merges, so that it corresponds to a clause tree $\mathcal{T}$ with m merge paths. Also let $n = \sum_{i=0}^{m} n_i$ such that $n_i = \lfloor n/(m+1) \rfloor$ or $n_i = \lfloor n/(m+1) \rfloor + 1$. Then*

$$c(\mathcal{T}) \geq \prod_{i=0}^{m} C_{n_i} \equiv lb(n,m)$$

Moreover this bound is tight.

Proof. First we demonstrate the extreme case. Let $n = k(m+1)$. Let brt T correspond to a clause tree $\mathcal{T}$ whose internal atom nodes $a_1, a_2, \ldots, a_n$ all lie in order on a single path. Thus all the nodes of $\mathcal{T}$ form a path except possibly for atom nodes which are leaves. Let $\mathcal{T}$ have m merge paths $P_1, P_2, \ldots, P_m$. Let the heads of the paths be spaced out almost equally along the path, with the head of path P_i at atom node a_{ki+1}, and the tail of P_i adjacent to the clause node adjacent to a_1 and not between a_1 and a_2. See Figure 8.

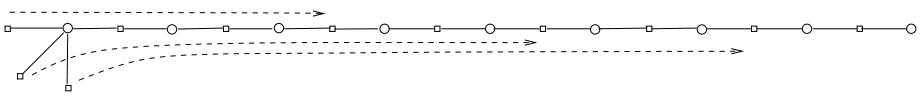

Fig. 8. An example of the lower bound, with $c(\mathcal{T}) = (C_2)^4$.

Let n_i be the interior node of the brt T corresponding to the interior atom node a_i of the clause tree $\mathcal{T}$. The resolutions corresponding to $a_1, \ldots, a_k$ must all occur before the resolution of a_{k+1}, and so $n_1, \ldots, n_k$ form a sub-brt $T_1$of T. Because $\mathcal{T}$ is effectively a path, the root of T_1 is a parent of n_{k+1}. By the results of the previous section, there are C_k brt's rotationally equivalent to T_1. Similarly the resolutions corresponding to $a_{k+1}, \ldots, a_{2k}$ must all occur before the resolution of a_{2k+1}, and again these resolutions can be organised in C_k ways. The nodes $n_1, \ldots, n_{2k}$ must form a sub-brt T_2 of T, and the root of T_2 must be a parent of n_{2k+1}. The number of brt's rotationally equivalent to T_2 equals the product of the number of ways that the nodes $n_{k+1}, \ldots n_{2k}$ can be organised and the number of brt's rotationally equivalent to T_1, that is $(C_k)^2$. Continuing in this way, for $j = 1, \ldots, m$, nodes $n_1, \ldots, n_{kj}$ form a sub-brt T_j which is rotationally equivalent to $(C_k)^j$ brt's. Then $T_m = T$, and is rotationally equivalent to $(C_k)^m$ brt's.

Next we prove the lower bound. Consider the set A of interior atom nodes of $\mathcal{T}$ that are not interior nodes of any merge path. They may be heads of merge paths. Consider $\mathcal{C} = \mathcal{T} - A$, $\mathcal{T}$ with the nodes of A and their incident edges

deleted.. $\mathcal{C}$ consists of a set of $k \leq m$ component clause trees which are sub-trees of $\mathcal{T}$. Some of these trees may consist of a single isolated clause node. Let the nontrivial component clause trees be $\mathcal{T}_i$ with n_i internal atom nodes and m_i merge paths. Note that for each non-trivial component clause tree $\mathcal{T}_i$, the head of at least one merge path with internal atom nodes in it has as its head an atom node from A. Exclude these merge paths from being chosen merge paths of the $\mathcal{T}_i$. Also consider the clause tree $\mathcal{D}$ obtained from $\mathcal{T}$ by contracting all the edges of $\mathcal{C}$. $\mathcal{D}$ is a mergeless clause tree with, say, n_0 internal atom nodes. Note that $\sum_{i=0}^{k} n_i = n$ and $\sum_{i=1}^{k} m_i \leq m - k$.

Let C_i be a brt corresponding to $\mathcal{C}_i$, and D a brt corresponding to $\mathcal{D}$. Then build a brt from D by replacing each leaf of D which corresponds to a contracted $\mathcal{C}_i$, by C_i. The resulting brt corresponds to $\mathcal{T}$. Moreover the result is different if any of the component brts are changed. Thus

$$c(\mathcal{T}) \geq c(\mathcal{D}) \prod_{i=1}^{k} c(\mathcal{C}_i) \geq C_{n_0} \prod_{i=1}^{k} lb(n_i, m_i)$$

Each $lb(n_i, m_i)$ is a product of Catalan numbers. By the observation after equation 3 that the product $C_k C_{h-k}$ is minimized when $k = h/2$ or $k = (h-1)/2$ for a fixed h, the product of all the Catalan numbers is minimized when the subscripts are as equal as possible. One can also see that maximizing the number of factors minimizes the product, by considering adding a factor of C_0 to the product if there is not a factor for each merge path. This gives the lower bound in the theorem.□

Because $C_n = (4^n)/n^{\Theta(1)}$, it follows that

$$c(T) \geq 2^{2n - \Theta(m \log(n))}$$

.

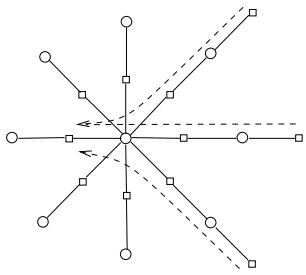

Fig. 9. A clause tree with merges and many equivalent brts.

The upper bound on the number of equivalent brt's when merge paths are allowed does decrease when merge paths are allowed, but not very much. I do not have a proof of an upper bound significantly better than $n!$, but it is no

less than $n!/(m+1)$. Consider the same clause tree as in the mergeless case, the claw if you like, and add m merge paths, all with the same head, h, from tails $t_1, t_2, \ldots, t_m$ attached at m distinct clause nodes, and not the central clause node. The m atom nodes internal to the merge paths must resolve before the atom node at the head. This decreases the number of brt's by a factor of $m+1$. See Figure 9. I believe that this clause tree is the extreme case, but I have no proof.

6 Open Questions

The first open question is the final conjecture of the preceding section. What is the exact upper bound on the number of rotationally equivalent brt's with n internal nodes and m merges?

A somewhat more important question is whether there is a polynomial-time algorithm to count the number of equivalent brt's with n internal nodes and m merges? If we cannot count them, or alternatively show that this is a difficult question, then our understanding of binary resolution must be quite limited.

More important again is asking the equivalent questions for dags. Dags are what binary resolution theorem provers usually produce as proofs. Can one count the number of rotationally equivalent binary resolution dags, with or without merges? Can one put bounds on the number of such dags? Maybe this is a very difficult question.

Perhaps the most interesting question is: Given a brt, what is the smallest equivalent dag? Goerdt [6] has shown that there are unsatisfiable sets of clauses such that the smallest non-regular (an atom label does not occur twice on one branch of the brt) refutation is exponentially bigger that the smallest refutation. When combined with the result that the smallest refutation brt is regular, indeed surgically minimal, for any unsatisfiable set of clauses [12], the smallest brt must be exponentially larger than the smallest dag. I do not know whether the smallest rotationally equivalent dag is typically, commonly or only rarely much smaller than the smallest brt.

The previous question leads to another. Is there some other reasonable definition of equivalence for dags? Using clause trees, one can do more than just path reversals to get more proofs which are almost equivalent. One has the operations of surgery and supplanting. Surgery is not always reversable, so this is not a strict equivalence relation, yet a refutation always transforms to a refutation, and even non-refutations may transform to a refutation.

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. R. S. Boyer. *Locking: a Restriction of Resolution.* PhD thesis, University of Texas at Austin, 1971.

3. Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York and London, 1973.
4. Hans de Nivelle. Resolution games and non-liftable resolution orderings. *Collegium Logicum, Annals of the Kurt Gödel Society*, 2:1–20, 1996.
5. E.Ben-Sasson and A. Wigderson. Short proofs are narrow — resolution made simple. *J. ACM*, 48:149–169, 2001.
6. Andreas Goerdt. Reguar resolution versus unrestricted resolution. *SIAM Journal on Computing*, 22:661–683, 1993.
7. J. D. Horton and B. Spencer. Clause trees: a tool for understanding and implementing resolution in automated reasoning. *Artificial Intelligence*, 92:25–89, 1997.
8. 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, Lindau, Germany, July 1998. Springer.
9. J. Reynolds. Seminar notes. Stanford University, Palo Alto, California, 1965.
10. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12:23–41, 1965.
11. Bruce Spencer. Avoiding duplicate proofs with the foothold refinement. *Annals of Mathematics and Artificial Intelligence*, 12:117–140, 1994.
12. 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.