REDUCING SEARCH WITH MINIMAL CLAUSE TREES

by

J. D. Horton
Bruce Spencer

TR95-099, November 1995

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
E-mail: fcs@unb.ca

Reducing Search with Minimal Clause Trees

J. D. Horton and Bruce Spencer
Faculty of Computer Science
University of New Brunswick
P.O. Box 4400, Fredericton, New Brunswick
E3B 5A3
Phone 506-453-4566
Fax 506-453-3566
email {jdh,bspencer}@unb.ca

Abstract

The smallest and most efficient resolution based proof of a theorem is represented by a minimal clause tree, proposed by Horton and Spencer. A clause tree T is minimal if and only if it contains no legal tautology paths and no legal unchosen merge paths. A characterization of minimal clause trees is given in terms of derivations. A subsumption relationship between resolution based procedures is defined and all such procedures are shown to be subsumed by clause tree procedures. Three classes of procedures are proposed which produce only minimal clause trees. The first performs surgery on any clause tree that is produced to reduce it to a minimal clause tree. The second avoids any resolution that produces a clause tree on which surgery can be performed, but leaf to internal node paths are allowed to be chosen. The third, and most restrictive, avoids any resolution that produces a non-minimal clause tree.

Key words: theorem proving, reasoning (resolution), search.

Reducing Search with Minimal Clause Trees

1. An intuitive exposition of clause trees

When one uses binary resolution, one starts with a set of ***clauses***, each of which is the disjunction of a set of literals, and applies resolution to them until the clause that one wants is found. This clause is the empty clause if one is looking for a contradiction. Usually a clause is represented as a set of literals, but in this paper a clause is represented by a tree in graph theory terms. An input clause is represented by a ***clause node*** connected to ***atom nodes*** each of which is labeled by an atom. A + (–) sign labels the edge joining the atom node to the clause node if the atom appears positively (negatively) in the clause. Figure 1(A) shows the tree representing the clause {a,b,~c,~d}. Such a tree is called a ***clause tree.***

Clauses can be combined using resolution. For example, the clause *{a,b,~c,~d}* can resolve with the clause *{~b, ~d, e, ~g}* to form the new clause *{a, ~c, ~d, e, ~g}*. Clause trees are resolved by identifying the nodes that represent complementary literals from two different clauses, as shown in Figure 1(B). The leaves of the resulting tree are the literals of the resulting clause. But two of the leaves are labeled by *~d*. The merging of the two literals *~d* that occurs when the union of two sets occurs, is not handled automatically by the clause trees. Instead the two atom nodes that correspond to the same literal can be joined with a ***merge path*** as in Figure 1(C1). The literal at the tail of a merge path is no longer considered to be a literal of the corresponding clause. Finally a resolution between Figure 1(C1) with the clause tree for the clause *{d}* is done, resulting in the clause tree in Figure 1(D) whose clause is *{a, ~c, ~g}*. The operation of resolving two clause trees followed by the insertion of all leaf-to-leaf merge paths is analogous to a resolution operation.

With clause trees, the merge path does not need to be added between open leaves; it can also be added from an open leaf to an internal node (under the conditions to be given in Section 2.) Suppose that the clause tree in Figure 1(B) were resolved against the clause tree for *{d} before* the merge path for *d* were inserted. This result is shown in Figure 1(C2). The clause for this clause tree is *{a, ~c, ~d, e, ~g}*. Inserting the merge path for *d* yields the clause tree in Figure 1(D). Thus clause tree allow us to use the internal node in the tree to remove a literal from the clause.

If the clause tree were much bigger, with many internal nodes, more literals might be removed from the clause. This shows one advantage of using resolution with clause trees over resolution with clauses.

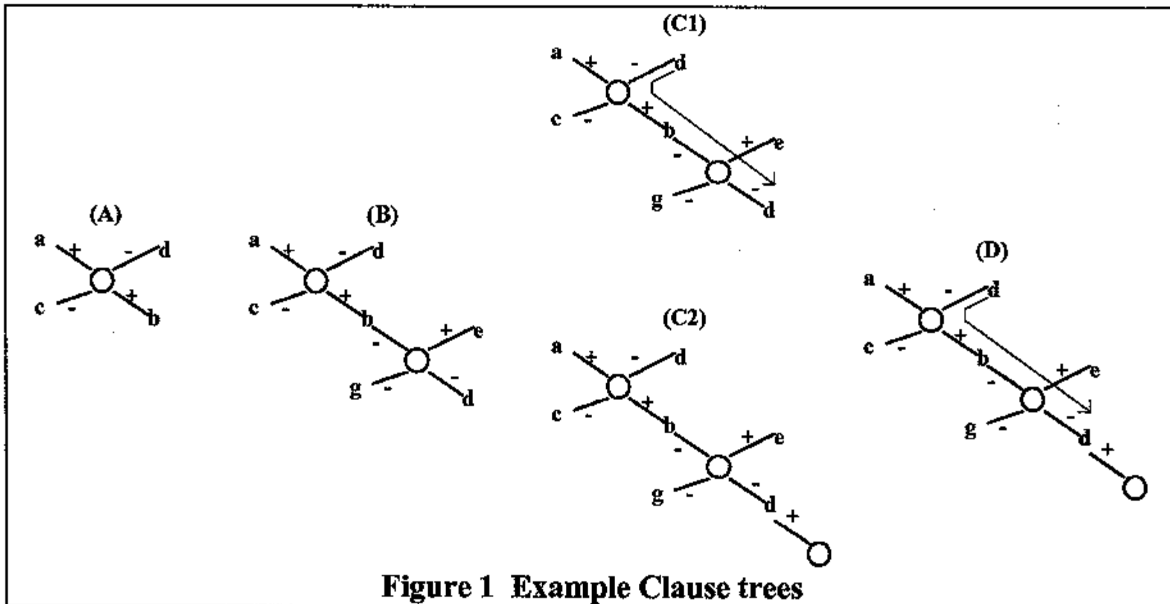

Figure 1 Example Clause trees

Figure 1 also illustrates how clause trees distinguish between the order in which the resolutions must be performed in a binary resolution proof, and the order in which the clause tree operations are performed. The two sequences of operation in Figure 1 are (A,B,C1,D) and (A,B,C2,D). The first sequence is also a binary resolution sequence since it corresponds to binary resolution steps. The second sequence is not a binary resolution sequence since the third step does not correspond to any binary resolution step. The final clause tree could be built using the second sequence of operations, but is justified by the first sequence.

Thus if clause trees are used, it may not matter in which order resolutions are done while searching for proofs, as the resulting clause tree can be made the same as the best result. A proof procedure that uses clause trees instead of sets to represent clauses can take advantage of this redundancy and use fewer resolutions to search for proofs than other resolution-based procedures. Moreover, since a clause tree really represents several distinct proofs, we can detect if any one of those proofs contains a clause that contains a literal and its negation. Such a clause is a tautology and we know that it cannot be useful. We can safely discontinue work on this clause tree, and hence we can cut this branch of the search. This is one of the restrictions used below.

2. Definitions

The definition of clause tree in this paper differs from that in [1]. There the definition is procedural, in that operations that construct clause trees are given. Here, as in [2] the definition is structural.

A ***path*** *path*(v_0, v_n) in a graph from v_0 to v_n is an alternating sequence $\langle v_0\, e_1\, v_1 \ldots e_n\, v_n \rangle$ where each v_i is a node and each e_j is an edge. The first node v_0 is the ***tail*** of the path, and the final node v_n is the ***head.***

Definition 1 *$T=\langle N,E,L,M\rangle$ is a **clause tree** on a set S of input clauses if:*

(a) $\langle N,E\rangle$ is a(n unrooted) tree.

*(b) L is a labeling of the nodes and edges of the tree. $L{:}N \cup E \rightarrow S^* \cup A \cup \{+,-\}$, where S^* is the set of instances of clauses in S and 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 –.*

(c) No atom node is incident with two edges labeled the same.

*(d) Each edge $e=\{a,c\}$ joins an atom node a and a clause node c; it is **associated** with the literal $L(e)L(a)$.*

*(e) For each clause node c, $L(c) = \{L(\{a,c\})L(a) \mid \{a,c\} \in E\}$. A path $\langle v_0\, e_1\, v_1 \ldots e_n\, v_n\rangle$ where $0 \le i \le n$, $v_i \in N$ and $e_j \in E$ where $1 \le j \le n$ is a **merge path** if $L(e_1)L(v_0) = L(e_n)L(v_n)$. Path $\langle v_0 \ldots v_n\rangle$ **precedes** ($\prec$) path $\langle w_0 \ldots w_m\rangle$ if $v_n = w_i$ for some $i=1,\ldots,m-1$.*

*(f) M is the set of merge paths called **chosen merge paths** such that:*

*(i) the tail of each is a leaf (called a **closed leaf**),*

(ii) the tails are all distinct and different from the heads, and

(iii) the relation $\prec$ on M can be extended to a partial order.

A set M of paths in a clause tree is ***legal*** if the $\prec$ relation on M can be extended to a partial order. A path P is ***legal*** in $T = \langle N,E,L,M\rangle$ if $M \cup P$ is legal. If the path joining t to h is legal in T, we say that h is ***visible*** from t.

A path $\langle v_0\, e_1\, v_1 \ldots e_n\, v_n\rangle$ where $v_i \in N$ and $e_j \in E$ is a ***tautology path*** if $L(v_0) = L(v_n)$ and $L(e_1) \neq L(e_n)$. A path is a ***unifiable tautology path*** if $L(e_1) \neq L(e_n)$ and there exists a substitution θ such that $L(v_0)\theta = L(v_n)\theta$. A path is a ***unifiable merge path*** if there exists a substitution θ such that $L(e_1)L(v_0)\theta = L(e_n)L(v_n)\theta$.

A clause tree with a single clause node is said to be ***elementary***. 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 T is called the ***clause*** of T, ***cl(T)***.

There are various operations on clause trees: creating an elementary clause tree from an input clause, resolving two clause trees, adding a merge path to the set of chosen paths and instantiation. Each of these operations results in a clause tree.

Operation 1. Creating an elementary clause tree from an input clause

*Given a clause C in S and a substitution θ for variables in C, the **elementary clause tree** $T= \langle N,E,L,\phi\rangle$ representing $C\theta = \{a_1,\ldots,a_n\}$ satisfies the following:*

1) N consists of a clause node and n atom nodes, where L labels the atom nodes with $a_1,\ldots,a_n$ and labels the clause node with $C\theta$.

2) E consists of n undirected edges, each of which joins the clause node to one of the atom nodes and is labeled by L positively or negatively according to whether the atom is positive or negative in the clause.

Operation 2. Resolving two mergeless clause trees
Let $T_1 = \langle N_1,E_1,L_1,M_1\rangle$ *and* $T_2 = \langle N_2,E_2,L_2,M_2\rangle$ *be two clause trees with no nodes in common such that* n_1 *is an atom node leaf of* T_1 *and* n_2 *is an atom node leaf of* T_2*. No variable may occur in a label of both an atom node in* T_1 *and an atom node in* T_2*. Let* L_1 *label* n_1 *with some atom* a_1 *and label the edge* $\{n_1, m_1\}$ *negatively, and* L_2 *label* n_2 *with the atom* a_2 *but label the edge* $\{n_2, m_2\}$ *positively. Further let* a_1 *and* a_2 *be unifiable with a substitution* θ*. Let* $N = N_1 \cup N_2 - \{n_1\}$*. Let* $E = E_1 \cup E_2 - \{\{n_1,m_1\}\} \cup \{\{n_2,m_1\}\}$ *where* $\{n_2,m_1\}$ *is a new edge. Let L be a new labeling relation that results from two modifications to* $L_1 \cup L_2$*: the new edge* $\{n_2,m_1\}$ *is labeled negatively, and* θ *is applied to the label of each atom node. Let M be the set of merge paths that results from* $M_1 \cup M_2$ *by replacing each occurrence of* n_1 *in each path of* M_1 *with* n_2*. Then* $T = \langle N,E,L,M\rangle$ *is a clause tree.*

We write T_1 *res* T_2 to refer to the clause tree that results from Operation 2. We use a similar notation for resolving two clauses together.

Operation 3. Adding a leaf-to-leaf unifiable merge path
Let $T = \langle N,E,L,M\rangle$ *and let* n_1 *and* n_2 *be two open leaves in T such that* $P=path(n_1,n_2)$ *is a unifiable merge path of* $\langle N,E,L,\phi\rangle$*, with* n_2 *not being the tail of any chosen merge path in M and* n_1 *not being the head or tail of any chosen merge path. Let* θ *be a substitution such that* $L(n_1)\theta=L(n_2)\theta$*. Let* $L\theta$ *be the labeling relation that results from applying* θ *to the label of each atom node, and otherwise leaving L the same. Then* $T_1 = \langle N,E,L\theta,M \cup \{P\}\rangle$ *is a clause tree.*

Operation 4. Instance of a clause tree
A clause tree $T' = \langle N,E,L',M\rangle$ *is an* ***instance*** *of a clause tree* $T = \langle N,E,L,M\rangle$ *if L' and L are identical on the clause nodes, atom nodes and edges, and there is a substitution* θ *such that for each atom node n,* $L'(n) = (L(n))\theta$*.*

Theorem 1. Closure of Clause Tree Operations
Each of Operation 1, Operation 2, Operation 3 and Operation 4 applied to a clause tree(s) generates a clause tree.

3. Clause tree derivations

Definition 2. Derivation of a clause tree

Given a set S of clauses, a ***derivation*** of T_n from S is a sequence $\langle T_1,\ldots,T_n\rangle$ of clause trees such that each T_i for $i = 1, \ldots, n$ (exactly) one of A1, A2, A3 or A4 holds and B holds.

A1. T_i is an elementary clause tree from an input clause C in S

A2. T_i is the result of resolving T_i and T_j where $j<i$ and $k<i$. In this case T_i ***depends on*** T_j and T_k.

A3. T_i is the result of choosing a leaf-to-leaf merge path in T_j where $j<i$. In this case T_i ***depends on*** T_j.

A4. T_i is an instance of T_j for $j<i$. In this case T_i ***depends on*** T_j.

B. T_n transitively depends on T_i for $i = 1, \ldots, n$.

Because T_n depends on all previous T_i a derivation must not have unused clause trees. This condition does not change the essential nature of clause trees, but is added in order to make certain conditions easier to ensure.

Definition 3 *A derivation $\langle T_1,\ldots,T_n\rangle$ where $T_i = \langle N_i, E_i, L_i, M_i\rangle$ is **admissible** if for each $P\in M_n$, such that P is a path in T_i but $P\notin M_i$ then $P\in M_j$ for some $j > i$ and for all $i < k \leq j$, T_k depends on T_{k-1} and T_k is the not the result of resolving clause trees.*

Thus a derivation is admissible if all leaf-to-leaf paths that appear in T_n are chosen as soon as possible in the derivation and before the next resolution step.

Theorem 2. *Every clause tree has an admissible derivation.*

Proof. Let $T = \langle N,E,L,M\rangle$ be a clause tree with n_a atom nodes, n_c clause nodes, n_m merge paths and n_o open leaf atom nodes. The constructed admissible derivation has $n_c+ n_a + n_m - n_o$ clause trees. For each clause node c in T, construct an elementary clause tree with atom nodes labeled the same as the atom nodes adjacent to c in T. Place these as the first n_c clause trees in the derivation. Thus all remaining clause trees in the derivation will be the result of applications of Operation 2 or Operation 3. Extend $\prec$ to a total order on M. Process the paths $\langle P_1,\ldots\rangle$ in this order. For each internal atom node of the path P_i not already considered, construct the clause tree corresponding to the resolution step involving this atom node on the two clause trees already in the derivation, and insert the result next into the derivation. After every step if any path P_j for $j\geq i$ is in the current clause tree then insert a clause tree with that path chosen into the derivation before the next resolution step. Note that P_i is a path in some tree in the derivation and that no resolution steps have been done since this tree was constructed, and that all operations done since have depended on this tree so the derivation so far is admissible. Continue until all paths are processed and then do all remaining resolutions. The resulting derivation is admissible. □

Theorem 3. [1]***Soundness and completeness of clause trees***

$S \models C$ iff there is a clause tree T on S such that $cl(T) \subseteq C$.

Definition 4 *A derivation* $\langle T_1,\ldots,T_n\rangle$ *is* ***minimal*** *if, for i = 1,...,n,* T_i *has two leaves with the same label then a merge path joining these leaves is among the chosen paths in* T_n, *and for any other pair of nodes of* T_i *labeled the same, neither node is a leaf.*

Definition 5 *A clause tree* $\langle N,E,L,M\rangle$ *is* ***minimal*** *if it contains no legal merge path not in M and no legal tautology path.*

A clause tree that is not minimal can be made minimal by applying surgery on all legal tautology and legal unchosen merge paths. Surgery is an operation that involves cutting out parts of the tree, if necessary, and rearranging the remainder, possibly adding a new merge path, so that the resulting structure is a clause tree. Surgery will not be discussed much in this paper. See [1].

Theorem 4. *A clause tree is minimal iff all admissible derivations of it are minimal.*

Proof. Let clause tree T_n have a derivation $\langle T_1,\ldots,T_n\rangle$ which is admissible but not minimal, where $T_i = \langle N_i, E_i, L_i, M_i\rangle$. Then there is a clause tree T_i which has two identically labeled leaves n_1 and n_2 but the path P joining these leaves is not in M_n. Let $T_i' = \langle N_i, E_i, L_i, M_i \cup \{P\}\rangle$, and let $T_k' = \langle N_k, E_k, L_k, M_k \cup \{P\}\rangle$ for any T_k which depends on T_i, $T_k' = T_k$ otherwise. Then the sequence $\langle T_1,\ldots,T_i, T_i',\ldots,T_n'\rangle$ is a derivation. But T_n' has $M_n \cup \{P\}$ as its set of chosen paths, which implies that P is legal in T_n. Hence T_n is not minimal.

Conversely, assume that $T = \langle N, E, L, M\rangle$ is non-minimal. Then there is a legal path $P \notin M$ between atom nodes in T. Let $P_1,\ldots,P_i$, P, $P_{i+1},\ldots,P_n$ be an ordering of $M \cup \{P\}$ that is an extension of the precedes relation. Then construct an admissible derivation as follows. Build a derivation of T as in the construction in Theorem 1 according to this sequence except do not insert the path P. Note that the tree in the derivation in which P could be first added as a merge path breaks the minimality condition on the admissible derivation.□

4. Comparing resolution-based procedures

Consider any resolution-based procedure, A, working with a set I of input clauses. A produces in order the clauses $\mathcal{C} = \langle C_1, C_2,\ldots,C_i,\ldots\rangle$ where

a) $C_i \in I$,
b) $C_i = C_j \ res\ C_k$ where $j<i$ and $k<i$, or
c) $C_j\theta \subseteq C_i$ where $j<i$, and θ is a substitution.

This sequence is called the ***generated clause sequence*** of A on input I. A refutation procedure A stops with success if $C_i = \phi$ for some i; otherwise it fails. Note that the generated sequences determine the procedure if they are known for all inputs.

Let the generated clause sequence of the procedure B on input I be $\mathcal{B} = \langle D_1, D_2,\ldots,D_i,\ldots\rangle$. B is said to ***subsume*** A on I if for all i there is an $f(i)$, $f(i) \le i$ and $D_{f(i)}$ subsumes C_i. Procedure B ***strictly subsumes*** procedure A if B subsumes A and A does not subsume B.

Lemma 1. *Let* $\mathcal{C} = \langle C_1, C_2, \ldots, C_m, \ldots \rangle$ *be a generated clause sequence. If* $\mathcal{D} = \langle D_1, D_2, \ldots, D_n \rangle$ *is a generated clause sequence of finite length,* $n \leq m$, *and* $\mathcal{D}$ *subsumes* $\langle C_1, C_2, \ldots, C_m \rangle$, *then* $\mathcal{D}$ *can be extended to a sequence which subsumes* $\mathcal{C}$, *and this extension is either an infinite sequence or contains* ϕ.

Proof. Suppose $D_{f(i)}$ subsumes C_i, $f(i) \leq i$. Then for each C_k, $k > m$, we define $D_{f(k)}$ where $f(k)$ is defined to be the next available index if the clause is not already in the sequence:

a) if C_k is an input clause, $D_{f(k)} = C_k$;
b) if $C_k = C_j\theta$, $j < k$, C_k is subsumed by $D_{f(j)}$, that is $f(k) = f(j)$;
c) if $C_k = C_i \; res \; C_j$, then C_k is subsumed by one of $D_{f(i)}$, $D_{f(j)}$ or $D_{f(i)} \; res \; D_{f(j)}$ where the resolution is on the instance of the literal resolved upon in $C_i \; res \; C_j$.

There is no need to ever consider a sequence $\langle C_1, C_2, \ldots, C_i, \ldots \rangle$ in which step (c) of the definition is used, since omitting C_i from the sequence leaves a sequence which can be extended to a sequence which subsumes the original sequence. Similarly it is pointless ever to do a resolution that does not use a most general unifier of the literals being resolved. From now on we only consider procedures that use most general unifiers and do not generate clause sequences using part (c) of the definition.

Any resolution based procedure can be considered to be a clause tree based procedure, by substituting for each resolution of clauses, the equivalent resolution of clause trees. Thus we can define a clause tree based procedure on the set of input clauses I as a sequence of clause trees $\langle T_1, T_2, \ldots, T_i, \ldots \rangle$ in which

a) T_i is an elementary clause tree from an input clause of I; or
b) $T_i = T_j \; res \; T_k$, where $j < i$ and $k < i$.

where $\langle cl(T_1), cl(T_2), \ldots, cl(T_i), \ldots \rangle$ is an equivalent resolution based procedure.

Clause trees also allow the operations of surgery, including inserting merge paths. Thus we can replace b) in the definition of the generated clause sequence with:

b') T_i = the result of surgery on $T_j \; res \; T_k$, where $j < i$ and $k < i$, and T_i is a minimal clause tree.

Generated clause sequences using steps a) and b') define our first class of minimal clause tree procedures. If the surgery in b') is performed and it removes an open leaf, then the resulting sequence of clauses strictly subsumes the original sequence of clauses. However the sequence of clauses may no longer be a resolution based sequence, because the clause resulting from surgery may not be derivable from the preceding sequence in one step.

Theorem 5. *Any resolution based procedure can be subsumed by a clause tree based procedure. Moreover the subsumption can be made strict unless the equivalent clause trees are all minimal.*

We have shown how to compare resolution based procedures, using subsumption. A subsuming sequence is never longer but leads to the same goal. Theorem 5 concludes that

there is no point in considering a generated sequence that does not come from clause trees since some generated sequence of clause trees subsumes it. We have defined a class of clause tree procedures which generate only minimal clause trees by performing surgery whenever possible. In the following section we propose some further restrictions on generated sequences of clause trees.

5. Bottom up procedures

Most resolution based procedures do not allow just any two clauses to be resolved. For instance top down procedures generally try either to extend the current clause with an input clause, or, when backtracking, to extend a clause on which the current clause depends. No other resolutions are allowed. Let us say that a clause C_i is ***retained*** by a procedure if it can later be resolved with another clause to form C_j, $j > I$; otherwise C_i is ***rejected***. Generally a bottom up procedure retains all clauses that it produces, unless they are subsumed by another clause in the sequence. Consider a generated sequence $\langle C_1, C_2, \ldots, C_i, \ldots\rangle$. If C_i is subsumed by C_j, $j < i$, then C_i is rejected by forward subsumption; if $i < j$ then C_i is rejected by backward subsumption.

Suppose a bottom up procedure ordered the clause trees produced so that all subtrees of the clause tree being produced were produced first. Then there would never be a need to apply surgery because the result of the surgery would have been produced first. However merge paths from leaves to internal nodes can still be chosen. Hence the resolution of two clause trees T_1 and T_2 which would produce a clause tree to which surgery could be applied, other than the chosing of a merge path, need never be done. This restricts the number of resolutions that need to be considered at any given step of the procedure, and hence decreases the search required to maintain a compete procedure. In effect, a forward subsumption check can be avoided in this way.

The above gives us our second class of minimal clause tree procedures. Two minimal clause trees can be resolved if the result is minimal, or can be made minimal by choosing merge paths whose tails are open leaves, which must be done. All other resolutions are forbidden.

Many a bottom up resolution based procedure is subsumed by such a clause tree procedure. A bottom up procedure generally proceeds in stages, in which all clauses of one type are resolved with all possible retained clauses, avoiding any resolution step that has been previously done. If the clauses are processed so that a clause C, whose derivation is smaller than a clause D, is processed before D, then the equivalent clause tree procedure is possible, and will subsume the bottom up resolution based procedure.
For example, one such procedure could produce all clauses whose derivations use k clauses, and hence clause trees with k clause nodes, in the kth stage. As all subtrees of a clause tree have fewer clause nodes than it has, no clause tree that admits surgery needs to be produced.

Theorem 6. *Any minimal clause tree based procedure of the first class that retains all the subtrees of the tree being produced, is subsumed by a minimal clause tree procedure of the second class.*

6. A still more restrictive procedure

However merge paths to internal nodes can still be chosen in a new clause tree in this second class of procedures that avoid surgery. Since we need to produce only minimal clause trees, which can be produced using minimal derivations (Theorem 4), we do not need to allow a resolution that produces a non-minimal clause tree. This means not only clause trees to which surgery can be applied, but also clause trees which have an unchosen merge path from a leaf to an internal node.

When two minimal clause trees T_1 and T_2 are resolved, the resulting tree T can only be non-minimal if there is a merge/tautology path going from T_1 to T_2, or from T_2 to T_1. The only allowable merge/tautology paths in this third class of procedures are leaf to leaf merge paths. Any tautology path causes T to be non-minimal, as does any internal node to leaf, or any internal node to internal node merge paths. Such trees are not retained by these procedures.

To implement such a procedure, one needs to be able to calculate, for any clause tree T:
1. The literals at the open leaves, $cl(T)$, and the corresponding set of atoms, $atom(T)$.
2. The atoms labeling internal nodes, $int(T)$.
3. The atoms visible from (outside) a given open leaf L, $vis(T,L)$.

For T_1 and T_2 to be resolved on open leaves L_1 of T_1 and L_2 of T_2 of each, it is necessary that the following sets be empty:
1. $(atom(T_1) \cup vis(T_1,L_1)) \cap int(T_2)$;
2. $(atom(T_2) \cup vis(T_2,L_2)) \cap int(T_1)$;
3. $cl(T_1) \cap \sim cl(T_2)$.

The following procedure based on these ideas suggests itself. In stage k, all minimal clause trees containing exactly k clause nodes are produced:
1. In stage *1*, all most general factors of the input clauses are produced.
2. In stage k, for all $i = 1,...,k/2$, all clause trees produced in stage i are resolved in all ways possible with the clause trees produced in stage $k-i$, but only minimal clause trees are produced.

It is not yet clear the best way to use subsumption with this class of procedures. For example in the above procedure, maybe one should not remove a smaller clause tree that is subsumed by a larger clause tree. The smaller clause tree may be part of a derivation of the smallest clause tree that proves the goal, in which case the goal may not be proved until a later stage.

7. Conclusions

One way to make progress in automated reasoning is to define restrictions. In [1] the restricted space of minimal clause trees is identified. This paper proposes restrictions on resolution based procedures that explore this space.

We have defined three new classes of resolution based procedures based on clause trees, each more restrictive than the previous. The first allows any clause tree to be produced, but performs surgery on it if possible. Any resolution based procedure is subsumed by some procedure in this category. The second allows any clause tree to be produced as long as surgery cannot be applied to it. However trees requiring merge paths from open leaves to internal nodes can still be retained, once these paths are chosen. A large class of bottom up procedures are subsumed by this class of clause tree procedures. The third class does not allow any non-minimal clause tree to be produced.

We are continuing the implementation and experimental evaluation of these procedures, especially the third class which is the most restrictive.

References

[1] J. D. Horton and B. Spencer, Clause trees: a tool for doing and understanding automated reasoning, TR95-095, Fac. Comp. Sc., Univ. New Brunswick, June 1995, available at http://www.cs.unb.ca/profs/bspencer/htm/clause_trees/TR95-095.ps.Z.

[2] J. D. Horton and Bruce Spencer, A top-down algorithm to find only minimal clause trees, Proceedings of CPL-95, held in conjunction with KI-95, Bielefeld, Germany, Sept.11-13, 1995, 77-78, available at http://www.cs.unb.ca/profs/bspencer/htm/clause_trees/ki95pr.ps.Z.

[3] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle *Journal of the ACM,* 12 (1965), 23–41.