Finding a Tree Structure in a Resolution Proof is NP-complete

Jan Hoffmann*
Institut für Informatik
Ludwig-Maximilians Universität
D-80538 München, Germany
jan.hoffmann@ifi.lmu.de

January 22, 2009

Abstract

The resolution tree problem is to decide whether a given sequence-like resolution refutation admits a tree structure. This paper shows the NP-completeness of both the resolution tree problem and a natural generalization of the resolution tree problem that does not involve resolution.

1 Introduction

In the context of propositional proof complexity, a *proof system* is a polynomial-time decidable relation R such that a propositional formula ϕ has a proof π with $R(\phi, \pi)$ if and only if ϕ is unsatisfiable. A proof system R is polynomially bounded if there is a polynomial p such that every unsatisfiable formula ϕ has a proof π with $|\pi| < p(|\phi|)$ and $R(\phi, \pi)$.

It is an open question whether a polynomially bounded propositional proof system exists, but it is not hard to see that such a proof system exists if and only if the complexity classes NP and coNP are equal, which is widely believed to be unlikely among experts. This relation to computational complexity was in fact a main motivation of Cook and Reckhow, who introduced the concept of a proof system [8], and it is the origin of a research enterprise to separate NP and coNP that is often called *Cook's program*. It asks to find lower bounds for more and more particular proof systems until the developed techniques can be applied to prove a lower bound for all proof systems and therefore NP $\neq$ coNP.

One of the most extensively studied propositional proof systems is *resolution* that is based on the resolution inference rule. It is common to define a resolution refutation for a set of clauses Γ as a sequence of clauses such that each clause

*Supported by the DFG Graduiertenkolleg 1480 (PUMA).

is a member of Γ or can be derived from two earlier clauses by a resolution inference.

Alternatively, a resolution refutation can be viewed as a directed acyclic graph (dag) in which every node has either two or zero predecessors. Many well-known resolution refinements refer to the dag-description of a refutation, including unit resolution, regular resolution, linear resolution, input resolution and tree-like resolution. A refutation is, for example, tree-like if its dag is a tree.

There has been a lot of work on proving superpolynomial lower bounds on the sizes of resolution refutations and its refinements, as well as on comparing the strength of the refinements in terms of proof sizes [12, 9, 5, 4, 2]. Moreover, different conditional lower bounds for *finding* a resolution proof have been given [1, 3, 10]. These results became increasingly popular since resolution refinements are closely related to proof search procedures that are used in SAT solvers that are highly successful in practice [7].

This paper proves a result that reveals an inaccuracy in some papers that define a tree-like resolution proof to be a sequence of clauses. It is based on the observation that it is not well-defined to call a sequence-like refutation tree-like if *its* underlying dag is a tree since the underlying dag of a sequence-like resolution refutation is not unique. We show that it is in fact NP-complete to decide whether a sequence-like resolution proof admits *some* dag that is a tree. The NP-hardness of the problem is shown by a non-trivial reduction from the exact cover problem.

A consequence of this result is that there is no polynomial time algorithm that decides for a given set of clauses Γ and a sequence of clauses whether the sequence is a tree-like refutation for Γ unless P = NP. Hence it is doubtful whether some notions of tree-like resolution that appear in the literature define a proof system in the sense of Cook and Reckhow. Although none of the existing results on tree-like resolution are affected by the issue, authors should nevertheless be aware of the problem and define tree-like resolution in an appropriate way.

In a previous work we showed some related results: On the one hand it is also NP-complete to decide whether a sequence-like refutation admits a regular dag [6]. On the other hand it is decidable in polynomial time for a given sequence-like refutation whether it admits a dag that is a unit resolution, a linear resolution or an input resolution [6].

The present paper also describes a generalized purely-combinatorial version of the tree-structure problem that does not involve resolution. This problem is also shown to be NP-complete.

2 Preliminaries

Resolution Refutations By $x, y, z, \ldots$ we denote variables that range over the values *True* and *False*. An *assignment* is a mapping from the variables to $\{$*True*, *False*$\}$. A literal l is a variable x or a negated variable $\overline{x}$. An assignment α satisfies x if x $\alpha(x) =$ *True*. α satisfies $\overline{x}$ if $\alpha(x) =$ *False*. A *clause* C is a

(possibly empty) set of literals. The empty clause is denoted by $\Box$. C is satisfied by an assignment α if α satisfies at least one literal of C.

A *propositional formula in conjunctive normal form (CNF)* is a set of clauses. A CNF Γ is satisfied by an assignment α if α satisfies every clause in Γ. If there is an assignment that satisfies Γ, we say Γ is *satisfiable*. Otherwise Γ is *unsatisfiable*.

A *resolution inference* is an inference of the form

$$\frac{C_1 \cup \{x\} \qquad C_2 \cup \{\overline{x}\}}{C_1 \cup C_2}$$

We say that the clause $C = C_1 \cup C_2$ is derived by resolution on x from $D_1 = C_1 \cup \{x\}$ and $D_2 = C_2 \cup \{\overline{x}\}$ and write $D_1, D_2 \vdash C$.

A *(sequence-like) resolution proof* of C_m from a CNF Γ is a sequence S of clauses $C_1, \ldots, C_m$ such that for every $C_k \notin \Gamma$ there are $1 \leq i < j < k$ with $C_i, C_j \vdash C_k$. If $C_m = \Box$ is the empty clause then S is called a *resolution refutation* of Γ. Clauses $C_k \in \Gamma$ are called *input clauses*. A resolution refutation for Γ shows that Γ is unsatisfiable.

We always assume that input clauses in a resolution refutation are marked. Sometimes we define resolution refutations without defining a CNF Γ that is proved to be unsatisfiable by the refutation. In that case Γ is implicitly given through the set of the marked input clauses in the refutation.

Exact Cover Let $U = \{u_1, \ldots, u_m\}$ be a finite set and and let $\mathcal{X} = (X_k : k = 1, \ldots, n)$ be a family of subsets of U. The problem Exact Cover for $(U, \mathcal{X})$ is to decide whether there is a subset $A \subseteq \{1, \ldots, n\}$ such that the subfamily $(X_k : k \in A)$ is a partition of U, i.e., $\bigcup_{k \in A} X_k = U$ and $X_k \cap X_i = \emptyset$ for all distinct $i, k \in A$. Such an A is called an *exact cover* for $(U, \mathcal{X})$.

Exact set cover is one of Karp's 21 NP-complete problems [11].

Theorem 1 (Karp 1972)**.** Exact Cover *is NP-complete.*

Sequences For two sequences $S_1 = s_1, \ldots, s_n$ and $S_2 = t_1, \ldots l_m$ we write S_1, S_2 to denote the sequence $s_1, \ldots, s_n, t_1, \ldots, t_m$. Similarly, S_1, t_1 denotes the sequence $s_1, \ldots, s_n, t_1$.

3 The Resolution Tree Problem

This section gives a proof for the NP-completeness of the problem of finding a tree structure for a sequence-like resolution refutation. The main part of the proof is a reduction from Exact Cover and it turned out that it is easier to reduce Exact Cover to a more general problem that we call Resolution Forest. A reduction from Resolution Forest is then used to show the NP-hardness of the original problem.

Let $S = C_1, \ldots, C_m$ be a resolution proof of C_m and let $D_1, \ldots, D_n$ be a subsequence of S. We say that S *admits a forest with roots* $D_1, \ldots, D_n$ if there exist disjoint binary trees $T_1, \ldots, T_n$ with nodes $C_1, \ldots, C_m$ such that

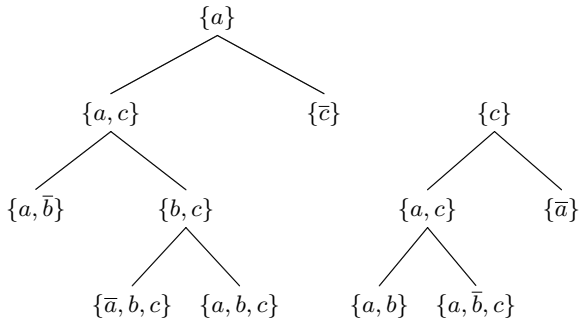

Figure 1: A resolution forest with roots $\{a\}, \{c\}$ for the resolution proof S.

- D_i is the root of T_i for $i \in \{1, \ldots, n\}$,
- if C_k is a leaf of some T_i then C_k is marked as an input clause in S and
- if C_k is an inner node of some T_l with children C_i, C_j then $i < j < k$ and $C_i, C_j \vdash C_k$.

We say that S *admits a tree structure* if S admits a forest with a single root $D_1 = C_m$.

Note that $T_1, \ldots, T_n$ have exactly m nodes, i.e., each C_k occurs exactly once in the forest.

The problem Resolution Forest is to decide for a given resolution proof $S = C_1, \ldots, C_m$ with $C_k \neq \square$ for all $1 \leq k \leq m$ and a subsequence R of S whether S admits a forest with roots R.

The problem Resolution Tree is to decide for a given resolution refutation whether it admits a tree structure.

Before we prove the hardness of Resolution Forest we give a short example of the problem. Consider the sequence of clauses

$$S_1 = \{\overline{a}, b, c\}, \{a, b, c\}, \{a, b\}, \{a, \overline{b}, c\}, \{a, \overline{b}\}, \{\overline{a}\}, \{\overline{c}\}$$

and let

$$S = S_1, \{b, c\}, \{a, c\}, \{a, c\}, \{a\}, \{c\}$$

be a resolution proof in which exactly the clauses in S_1 are input clauses. Figure 1 shows that S admits a resolution forest with roots $\{a\}, \{c\}$. To demonstrate that a resolution forest for a sequence is not unique Figure 2 shows that S also admits a different resolution forest.

Note that S contains the clause $\{a, c\}$ twice. The resolution proof $S' = S_1, \{b, c\}, \{a, c\}, \{a\}, \{c\}$ that contains $\{a, c\}$ only once does not admit a resolution forest with roots $\{a\}, \{c\}$ since the number of nodes of two binary trees must be even but the number of clauses in S' is odd. The following lemma shows that counting does not always help to decide Resolution Forest.

Lemma 2. Resolution Forest *is NP-hard.*

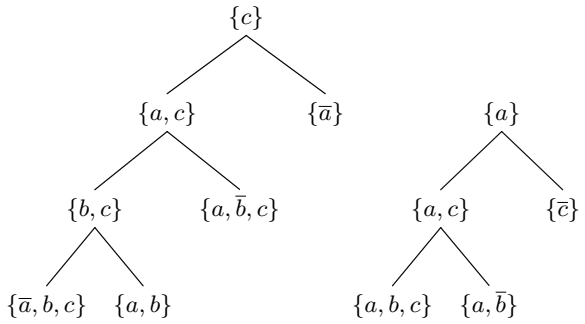

Figure 2: Another resolution forest with roots $\{a\}, \{c\}$ for the proof S.

Proof. We give a polynomial-time reduction from Exact Cover. Let therefore $(U, \mathcal{X})$ be an instance of Exact Cover. We construct a resolution proof $S = S_1, S_2, S_3, S_4, S_5$ such that S admits a resolution forest with roots S_2, S_4, S_5 if and only if $(U, \mathcal{X})$ admits an exact cover. $S = S_1, \ldots, S_5$ will be defined step by step starting with S_1 and S_2.

For $k \in \{1, \ldots, n\}$ let $X_k = \{u_{k,1}, \ldots, u_{k,m_k}\} \subseteq U$. The clauses in S will then use the following variables.

$$u_1, \ldots, u_m, x_{1,1}, \ldots, x_{1,m_1}, \ldots, x_{n,1}, \ldots, x_{n,m_n}, a, d, b_1, \ldots, b_m$$

We define three sequences for each $X_k \in \mathcal{X}$:

$$\begin{aligned}
\sigma(X_k) =& \{u_{k,1}, \overline{x}_{k,1}\}, \{x_{k,1}, \overline{u}_{k,2}\}, \{u_{k,2}, \overline{x}_{k,2}\}, \ldots, \\
& \{u_{k,m_k}, \overline{x}_{k,m_k}\}, \{x_{k,m_k}, \overline{u}_{k,1}\} && \text{(input clauses)} \\
\upsilon(X_k) =& \{u_{k,1}, \overline{u}_{k,2}\}, \{u_{k,2}, \overline{u}_{k,3}\}, \ldots, \{u_{k,m_k}, \overline{u}_{k,1}\} && \text{(non-input)} \\
\xi(X_k) =& \{x_{k,1}, \overline{x}_{k,2}\}, \{x_{k,2}, \overline{x}_{k,3}\}, \ldots, \{x_{k,m_k}, \overline{x}_{k,1}\} && \text{(non-input)}
\end{aligned}$$

Note that if we define a clause like $\{u_{k,1}, u_{k,2}\}$ then this is the clause $\{u_i, u_j\} \subseteq U$ such that $u_{k,1} = u_i$ and $u_{k,2} = u_j$. Thus sometimes $u_{k,i}$ and $u_{k',j}$ denote the same variable for $k \neq k'$. On the other hand the $x_{k,i}$ and $x_{k',j}$ are always distinct variables for $k' \neq k$. The three sequences allow the following two types of inferences (here $i+1$ denotes 1 if $i = m_k$).

$$\frac{\{u_{k,i}, \overline{x}_{k,i}\} \qquad \{x_{k,i}, \overline{u}_{k,i+1}\}}{\{u_{k,i}, \overline{u}_{k,i+1}\}} \qquad \frac{\{x_{k,i}, \overline{u}_{k,i+1}\} \qquad \{u_{k,i+1}, \overline{x}_{k,i+1}\}}{\{x_{k,i}, \overline{x}_{k,i+1}\}}$$

We will define S in such a way that from the clauses in $\sigma(X_k)$, the only derivable clauses are in $\xi(X_k)$ or are identical to one of the clauses in $\upsilon(X_k)$ (recall that a clause can occur several times a sequence and here the clauses in $\upsilon(X_k)$ can also occur in a $\upsilon(X_{k'})$). Thus, a resolution forest for S must either use all C in $\sigma(X_k)$ to derive all clauses in $\xi(X_k)$ or it must use all C to derive clauses that are identical to the clauses in $\upsilon(X_k)$. The intuition is that deriving $\upsilon(X_k)$ means to use X_k for a partition of U and deriving $\xi(X_k)$ means not to use X_k.

Since we can not derive both $\xi(X_k)$ and $\upsilon(X_k)$ from $\sigma(X_k)$ in a tree structure, we have to provide additional input clauses that can be picked in a forest. We set

$$\begin{aligned}
\alpha(X_k) =& \{a, \overline{u}_{k,1}\}, \{\overline{a}, u_{k,1}\}, \ldots, \{a, \overline{u}_{k,m_k}\}, \{\overline{a}, u_{k,m_k}\} && \text{(input)}\\
\delta(X_k) =& \{d, \overline{x}_{k,1}\}, \{\overline{d}, x_{k,1}\}, \ldots, \{d, \overline{x}_{k,m_k}\}, \{\overline{d}, x_{k,m_k}\} && \text{(input)}\\
S_1 =& \sigma(X_1), \alpha(X_1), \delta(X_1), \ldots, \sigma(X_n), \alpha(X_n), \delta(X_n)\\
S_2 =& \xi(X_1), \upsilon(X_1), \ldots, \xi(X_n), \upsilon(X_n)
\end{aligned}$$

Now we can use the clauses $\alpha(X_k)$ to derive the clauses in $\upsilon(X_k)$ and the clauses in $\delta(X_k)$ to derive the clauses in $\xi(X_k)$:

$$\frac{\{\overline{a}, u_{k,i}\} \qquad \{a, \overline{u}_{k,i+1}\}}{\{u_{k,i}, \overline{u}_{k,i+1}\}} \qquad \frac{\{\overline{d}, x_{k,i}\} \qquad \{d, \overline{x}_{k,i+1}\}}{\{x_{k,i}, \overline{x}_{k,i+1}\}}$$

Hence we have too many input clauses to build a forest. S_3, S_4 and S_5 will therefore provide alternate clauses that can be derived by the ones in $\alpha(X_k)$ and $\delta(X_k)$.

Note that in $\upsilon(X_k)$ there is exactly one clause that contains u_i and exactly one clause that contains $\overline{u}_i$ for each $u_i \in X_k$. Our next goal is to force every tree structure to use the clauses in $\sigma(X_1), \ldots, \sigma(X_n)$ to derive exactly one clause with u_i and one clause with $\bar{u}_i$ for every $u_i \in U$. That would already mean to use the clauses in the same X_k for both the clause with u_i and the clause with $\overline{u}_i$ in tree structure. To achieve this goal, we define S_3 such that for each $u_i \in U$ one $\{a, \overline{u}_i\}$ and one $\{\overline{a}, u_i\}$ has to be used in S_3.

$$\begin{aligned}
S_3 =& \{u_1, b_1\}, \{\overline{u}_1, b_1\}, \ldots, \{u_m, b_m\}, \{\overline{u}_m, b_m\}, && \text{(input)}\\
& \{a, b_1\}, \{\overline{a}, b_1\}, \ldots, \{a, b_m\}, \{\overline{a}, b_m\} && \text{(non-input)}\\
S_4 =& \{b_1\}, \ldots, \{b_m\} && \text{(non-input)}
\end{aligned}$$

For every $i \in \{1, \ldots, m\}$ we have the following derivation.

$$\frac{\dfrac{\{\overline{a}, u_i\} \qquad \{\overline{u}_i, b_i\}}{\{\overline{a}, b_i\}} \qquad \dfrac{\{a, \overline{u}_i\} \qquad \{u_i, b_i\}}{\{a, b_i\}}}{\{b_i\}}$$

A forest for S_1, S_2, S_3 can now use all clauses that appear in a $\sigma(X_k)$ or in a $\alpha(X_k)$ as well as $2|U| = 2m$ clauses from the $\delta(X_k)$ (if an exact cover exists then they are used to derive the clauses in $\xi(X_k)$ for the X_k that are part of the partition). It follows that $r = 2 \cdot (\sum_{k=1,\ldots n} m_k - m)$ clauses that are elements of a $\delta(X_k)$ remain unused. We use them to derive the clauses $\{d, \overline{d}\}$ in S_5:

$$\frac{\{d, \overline{x}_{k,i}\} \qquad \{\overline{d}, x_{k,i}\}}{\{d, \overline{d}\}}$$

$$S_5 = \underbrace{\{d, \overline{d}\}, \ldots, \{d, \overline{d}\}}_{r \text{ times}} \qquad \text{(non-input)}$$

This completes the construction of S. To prove its correctness it has to be shown that S admits a tree structure with roots S_2, S_4, S_5 if and only if $(U, \mathcal{X})$ has an exact cover.

To see the implication from right to left let $A \subseteq \{1, \ldots, n\}$ be an exact cover for $(U, \mathcal{X})$. Given the explanations in the construction of S, it is easy to use A to construct a forest F for S. F uses the clauses in $\sigma(X_k)$ to derive the clauses in $\upsilon(X_k)$, i.e., as u-children, if $k \in A$ and it uses the clauses in $\sigma(X_k)$ to derive the clauses in $\xi(X_k)$ otherwise. By following the description of the above construction of S it is straightforward to build the forest F.

To prove the left to right direction let $F = T_1, \ldots, T_t$ be a forest for S. We call the clauses of the form $\{u_i, \overline{u}_j\}$ that occur in the sequences $\upsilon(X_k)$ u-clauses. Children of a u-clause in F are called u-children.

Let $k \in \{1, \ldots, n\}$ and let $\sigma(X_k) = C_1, \ldots, C_{2m_k}$. We first prove that

$$\text{if one clause in } \sigma(X_k) \text{ is a } u\text{-child then all clauses in } \sigma(X_k) \text{ are } u\text{-children.} \tag{1}$$

To prove (1) we show that if C_j is a u-child then C_{j+1} is also a u-child for all $1 \leq j \leq 2m_k$ (let $j+1$ be 1 if $j = 2m_k$). There are two cases. Assume first that $j = 2i - 1$ is odd. Let again $i + 1$ be 1 if $i = m_k$. Then $C_j = \{u_{k,i}, \overline{x}_{k,i}\}$ and it is clear that $C_{j+1} = \{x_{k,i}, \overline{u}_{k,i+1}\}$ is also a u-child since the only way to derive a u-clause with C_j in S is by the inference

$$\frac{\{u_{k,i}, \overline{x}_{k,i}\} \qquad \{x_{k,i}, \overline{u}_{k,i+1}\}}{\{u_{k,i}, \overline{u}_{k,i+1}\}}$$

Assume now that $j = 2i$ is even. Then $C_j = \{x_{k,i}, \overline{u}_{k,i+1}\}$ and we have to show that $C_{j+1} = \{u_{k,i+1}, \overline{x}_{k,i+1}\}$ is a u-clause. If C_{j+1} would not be a u-clause it would be used as a child of a non-u-clause somewhere in F. The only possible inference in S that uses C_{j+1} without inferring a u-clause is

$$\frac{\{x_{k,i}, \overline{u}_{k,i+1}\} \qquad \{\overline{u}_{k,i+1}, x_{k,i+1}\}}{\{x_{k,i}, \overline{x}_{k,i+1}\}}$$

But this inference cannot be used in F since $C_j = \{x_{k,i}, \overline{u}_{k,i+1}\}$ is used already to derive a u-clause and cannot be used twice in F. Thus C_{j+1} must be a u-clause. This completes the proof of (1).

It follows directly from (1) and from the definition of $\sigma(X_k)$ that if a u_i occurs in a u-clause with children in $\sigma(X_k)$ then $\overline{u}_i$ occurs in a u-clause with children in $\sigma(X_k)$ too. We say that u_i is X_k-derived in that case. We now show for each $i \in \{1, \ldots, m\}$ that

$$u_i \text{ is } X_k\text{-derived for exactly one } X_k. \tag{2}$$

To prove (2), fix a u_i and let $x(i)$ be the number of X_k's that contain u_i. S has exactly $x(i)$ u-clauses that contain u_i ($\overline{u}_i$ respectively) and $x(i)$ clauses $\{\overline{a}, u_i\}$ ($\{a, \overline{u}_i\}$ resp.). The sequence S_3 is defined such that the clause $\{b_i\}$ can only be derived in the way described during the construction. So one $\{a, \overline{u}_i\}$ and one $\{\overline{a}, u_i\}$ have to be used in F to derive the clause $\{b_i\}$. Thus there is exactly one u-clause that contains u_i and one u-clause that contains $\overline{u}_i$ that are not derived by a clause $\{\overline{a}, u_i\}$ or $\{a, \overline{u}_i\}$, respectively. These clauses (maybe it is only one clause) must be X_k-derived for some k.

From (1) and (2) it follows immediately that the family (X_k : there is an i such that u_i is X_k-derived) is a partition of U. □

Theorem 3. Resolution Tree *is NP-complete.*

Proof. To see that Resolution Tree is a member of NP, consider the following NP-algorithm for a given sequence $S = C_1, \ldots, C_n$. First compute the set P_k of pairs of possible children for each C_k. This can be done by a deterministic algorithm in quadratic time. Then non-deterministically guess for every $k \in \{1, \ldots, l\}$ a pair of children $\{C_i, C_j\} \in P_k$ or decide that C_k is a leaf in the tree. Finally verify that every C_k with $k < l$ has been picked exactly once as a child and that every leaf is an input-clause.

Now the NP-hardness of Resolution Tree is shown by a reduction from Resolution Forest. Let $S = C_1, \ldots, C_m$ be a resolution proof with $C_i \neq \square$ for all $1 \leq i \leq m$ and let $D_1, \ldots, D_n$ be a subsequence of S. We will construct a resolution refutation $R = S, R_1, R_2$ such that R admits a tree structure if and only if S admits a forest with roots $D_1, \ldots, D_n$. The idea is that every clause in R_1, R_2 has exactly one pair of possible predecessors (if we identify equal D_k's) and that every D_k occurs exactly once among these predecessors such that it is possible to build a tree structure from the sequence $D_1, \ldots, D_n, R_1, R_2$.

Let $k \in \{1, \ldots n\}$, $D_k = \{l_{k,1}, \ldots, l_{k,m_k}\}$ and let d_k be a new variable. Define

$$\begin{aligned}
\sigma(D_k) =& \{\bar{l}_{k,1}, d_k\}, \ldots, \{\bar{l}_{k,m_k}, d_k\}, && \text{(input)}\\
& \{l_{k,2}, \ldots l_{k,m_k}, d_k\}, \ldots, \{l_{k,m_k}, d_k\}, \{d_k\} && \text{(non-input)}\\
R_1 =& \sigma(D_1), \ldots, \sigma(D_n)
\end{aligned}$$

Since d_k is a new variable $\sigma(D_k)$ can only be used in R as shown in the following inference (recall that $D_k \neq \square$).

$$\dfrac{\dfrac{\dfrac{D_k \qquad \{\bar{l}_{k,1}, d_k\}}{\{l_{k,2}, \ldots l_{k,m_k}, d_k\}} \qquad \{\bar{l}_{k,2}, d_k\}}{\{l_{k,3}, \ldots l_{k,m_k}, d_k\}}}{\begin{array}{c}\vdots\\ \{l_{k,m_k}, d_k\}\end{array}} \qquad \{\bar{l}_{k,m_k}, d_k\}}{\{d_k\}}$$

Now we define R_2 such that it can be used to connect the $\{d_k\}$ to a single tree.

$$\begin{aligned} R_2 = &\{\overline{d}_1, \ldots, \overline{d}_n\}, && \text{(input)} \\ &\{\overline{d}_2, \ldots, \overline{d}_n\}, \ldots, \{\overline{d}_n\}, \square && \text{(non-input)} \end{aligned}$$

The non-input clauses in R_2 are inferred as follows.

$$\dfrac{\dfrac{\{\overline{d}_1, \ldots, \overline{d}_n\} \qquad \{d_1\}}{\{\overline{d}_2, \ldots, \overline{d}_n\}}}{\begin{array}{c}\vdots \\ \{\overline{d}_n\} \qquad\qquad\qquad \{d_n\}\end{array}}$$
$$\square$$

This completes the construction of R. It is easy to see that R admits a tree structure if and only if S admits a forest with roots $D_1, \ldots, D_n$. □

4 The Tree Structure Problem

In this section we define the abstract combinatorial problem Tree Structure which is a generalization of Resolution Tree that neither involves resolution nor the notion of a proof. It follows from the results in the previous section that Tree Structure is also NP-complete.

The problem Tree Structure is mentioned for two reasons. On the one hand it has been conjectured to be NP-complete in an earlier work [6]. On the other hand it shows that the hardness of finding a tree structure for a sequence is not specific to resolution but may also apply to different proof systems or to problems in a completely different context.

Let $S = s_1, \ldots, s_l$ be a sequence and let for each $k \in \{1, \ldots, l\}$ P_k be a (possibly empty) set of unordered pairs $\{s_i, s_j\}$ with $1 \leq i < j < k$. Let $P = (P_k : k = 1, \ldots l)$.

(S, P) *admits a tree structure* if there is a binary tree T with nodes $\{s_1, \ldots, s_l\}$ such that

- if s_k is a leaf of T then $P_k = \emptyset$ and
- if s_k is an inner node of T with children s_i, s_j then $\{s_i, s_j\} \in P_k$.

The problem Tree Structure is to decide whether (S, P) admits a tree structure.

Theorem 4. Tree Structure *is NP-complete.*

Proof. It easy to see that Tree Structure is a member of NP by adapting the algorithm that is given in the proof of Theorem 3.

But it is also clear that Tree Structure is NP-hard since Resolution Tree is NP-hard and an instance $S = C_1, \ldots, C_m$ of Resolution Tree can be transformed into an instance of Tree Structure by defining $P_k = \{\{C_i, C_j\} \mid C_i, C_j \vdash C_k, i < j < k\}$. □

Acknowledgments I thank Martin Hofmann and Jan Johannsen for helpful discussions and for suggesting many improvements to the presentation of this work. I also appreciate the remarks of the anonymous reviewers.

References

[1] Michael Alekhnovich, Samuel R. Buss, Shlomo Moran, and Toniann Pitassi. Minimum propositional proof length is NP-hard to linearly approximate. *J. Symb. Log.*, 66(1):171–191, 2001.

[2] Michael Alekhnovich, Jan Johannsen, Toniann Pitassi, and Alasdair Urquhart. An exponential separation between regular and general resolution. *Theory of Computing*, 3(1):81–102, 2007.

[3] Michael Alekhnovich and Alexander A. Razborov. Resolution is not automatizable unless W[P] is tractable. In *45th Symposium on Foundations of Computer Science*, pages 210–219, 2001.

[4] Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near optimal separation of tree-like and general resolution. *Combinatorica*, 24(4):585–603, 2004.

[5] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow - resolution made simple. *J. ACM*, 48(2):149–169, 2001.

[6] Samuel R. Buss and Jan Hoffmann. The NP-hardness of finding a directed acyclic graph for regular resolution. *Theor. Comput. Sci.*, 396(1-3):271–276, 2008.

[7] Samuel R. Buss, Jan Hoffmann, and Jan Johannsen. Resolution trees with lemmas: Resolution refinements that characterize DLL algorithms with clause learning. *LMCS-4*, 4:13, 2008.

[8] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *J. Symb. Log.*, 44(1):36–50, 1979.

[9] Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985.

[10] Kazuo Iwama. Complexity of finding short resolution proofs. In Igor Prívara and Peter Ruzicka, editors, *Mathematical Foundations of Computer Science, 22nd International Symposium*, volume 1295 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 1997.

[11] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[12] G.S. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125, 1968.