Chapter 4

Resolution In Propositional Logic

4.1 Introduction

In Chapter 3, a procedure for showing whether or not a given proposition is valid was given. This procedure, which uses a Gentzen system, yields a formal proof in the Gentzen system when the input proposition is valid. In this chapter, another method for deciding whether a proposition is valid is presented. This method due to J. Robinson (Robinson, 1965) and called *resolution*, has become quite popular in automatic theorem proving, because it is simple to implement. The essence of the method is to prove the validity of a proposition by establishing that the negation of this proposition is unsatisfiable.

The main attractive feature of the resolution method is that it has a single inference rule (the resolution rule). However, there is a price to pay: The resolution method applies to a proposition in conjunctive normal form.

In this chapter, the resolution method will be presented as a variant of a special kind of Gentzen-like system. A similar approach is followed in Robinson, 1969, but the technical details differ. We shall justify the completeness of the resolution method by showing that Gentzen proofs can be recursively transformed into resolution proofs, and conversely. Such transformations are interesting not only because they show constructively the equivalence of the two proof systems, but because they also show a relationship between the complexity of Gentzen proofs and resolution refutation proofs. Indeed, it will be shown that every proof tree T in the system $GCNF'$ can be converted to

a resolution refutation D whose number of steps is at most the number of leaves of the proof tree T.

In the original edition of this book, it was claimed that the number of axioms in a Gentzen proof tree was linearly related to the number of resolution steps in the corresponding input resolution DAG, but the proof was wrong.

The essence of the connection between Gentzen proofs and resolution proofs is the following: Given a proposition A, if B is a conjunctive normal form of $\neg A$, A is valid iff B is unsatisfiable, iff the sequent $B \rightarrow$ is valid.

We will show how a Gentzen proof tree for $B \rightarrow$ can be transformed into a resolution DAG for B (and conversely). The first step is to design an efficient Gentzen system for proving sequents of the form $B \rightarrow$, where B is in conjunctive normal form.

4.2 A Special Gentzen System

The system $GCNF'$ is obtained by restricting the system G' to sequents of the form $A \rightarrow$, where A is a proposition in conjunctive normal form.

4.2.1 Definition of the System $GCNF'$

Recall from definition 3.4.7 that a proposition A is in conjunctive normal form iff it is a conjunction $C_1 \wedge ... \wedge C_m$ of disjunctions $C_i = B_{i,1} \vee ... \vee B_{i,n_i}$, where each $B_{i,j}$ is either a propositional symbol P or the negation $\neg P$ of a propositional symbol.

Definition 4.2.1 A *literal* is either a propositional symbol or its negation. Given a literal L, its *conjugate* $\overline{L}$ is defined as follows: If $L = P$ then $\overline{L} = \neg P$, and if $L = \neg P$, then $\overline{L} = P$. A proposition of the form

$$C_i = B_{i,1} \vee ... \vee B_{i,n_i},$$

where each $B_{i,j}$ is a literal is called a *clause*.

By lemma 3.3.6, since both $\wedge$ and $\vee$ are commutative, associative and idempotent, it can be assumed that the clauses C_i are distinct, and each clause can be viewed as the set of its distinct literals. Also, since the semantics of sequents implies that a sequent

$$C_1 \wedge ... \wedge C_n \rightarrow \quad \text{is valid iff}$$
$$\text{the sequent } C_1, ..., C_n \rightarrow \text{ is valid,}$$

it will be assumed in this section that we are dealing with sequents of the form $C_1, ..., C_n \rightarrow$, where $\{C_1, ..., C_n\}$ is a *set* of clauses, and each clause C_i is a set of literals. We will also view a sequent $C_1, ..., C_n \rightarrow$ as the set

of clauses $\{C_1, ..., C_n\}$. For simplicity, the clause $\{L\}$ consisting of a single literal will often be denoted by L. If Γ and Δ denote sets of propositions, then Γ, Δ denotes the set $\Gamma \cup \Delta$. Similarly, if Γ is a set of propositions and A is a proposition, then Γ, A denotes the set $\Gamma \cup \{A\}$. Consequently, $\Gamma, A_1, ..., A_m \rightarrow$ denotes the sequent $\Gamma \cup \{A_1, ..., A_m\} \rightarrow$.

It should be noted that when a sequent $C_1, ..., C_n \rightarrow$ is viewed as the set of clauses $\{C_1, ..., C_n\}$, the comma is interpreted as the connective *and* ($\wedge$), but that when we consider a single clause $C_i = \{L_1, ..., L_m\}$ as a set of literals, the comma is interpreted as the connective *or* ($\vee$).

For sequents of the above form, note that only the $\vee : left$ rule and the $\neg : left$ rule are applicable. Indeed, since each proposition in each C_i is a literal, for every application of the $\neg : left$ rule resulting in a literal of the form $\neg P$ in the antecedent of the conclusion of the rule, the propositional letter P belongs to the right-hand side of the sequent which is the premise of the rule. Since the $\vee : left$ rule does not add propositions to the right-hand side of sequents, only propositional letters can appear on the right-hand side of sequents. Hence, only the $\vee : left$ rule and the $\neg : left$ rule are applicable to sequents of the form defined above. But then, by redefining the axioms to be sequents of the form

$$\Gamma, P, \neg P \rightarrow ,$$

we obtain a proof system in which the only inference rule is $\vee : left$.

A further improvement is achieved by allowing several $\vee : left$ rules to be performed in a single step. If a sequent of the form

$$\Gamma, (A_1 \vee B), ..., (A_m \vee B) \rightarrow$$

is provable, the proof may contain m distinct applications of the $\vee : left$ rule to $(A_1 \vee B)$,...,$(A_m \vee B)$, and the proof may contain $2^m - 1$ copies of the proof tree for the sequent $\Gamma, B \rightarrow$. We can avoid such redundancies if we introduce a rule of the form

$$\frac{\Gamma, A_1, ..., A_m \rightarrow \qquad \Gamma, B \rightarrow}{\Gamma, (A_1 \vee B), ..., (A_m \vee B) \rightarrow}$$

We obtain a proof system in which proofs are even more economical if we introduce a rule of the form

$$\frac{\Gamma_1, C_1, ..., C_k \rightarrow \qquad \Gamma_2, B \rightarrow}{\Gamma, (A_1 \vee B), ..., (A_m \vee B) \rightarrow}$$

where Γ_1 and Γ_2 are arbitrary subsets of $\Gamma \cup \{(A_1 \vee B), ..., (A_m \vee B)\}$ (not necessarily disjoint), and $\{C_1, ..., C_k\}$ is any *nonempty* subset of $\{A_1,...,A_m\}$. In this fashion, we obtain a system similar to LK' with implicit weakenings,

in which every nontrivial sequent (see definition below) has a proof in which the axioms are of the form $P, \neg P \rightarrow$.

Definition 4.2.2 A sequent of the form $C_1, ..., C_n \rightarrow$ is *trivial* if $C_i = P$ and $C_j = \neg P$ for some letter P and some i, j, $1 \leq i, j \leq n$. Otherwise, we say that the sequent is *nontrivial*.

The system $GCNF'$ is defined as follows.

Definition 4.2.3 (The system $GCNF'$). Let $\Gamma, \Gamma_1, \Gamma_2$ denote sets of propositions, A_1,...,A_m denote propositions, B denote a literal, and P denote a propositional letter.

Axioms: All trivial sequents, that is, sequents of the form

$$\Gamma, P, \neg P \rightarrow$$

Inference Rule: All instances of the rule

$$\frac{\Gamma_1, C_1, ..., C_k \rightarrow \qquad \Gamma_2, B \rightarrow}{\Gamma, (A_1 \vee B), ..., (A_m \vee B) \rightarrow}$$

where Γ_1 and Γ_2 are arbitrary subsets of $\Gamma \cup \{(A_1 \vee B), ..., (A_m \vee B)\}$ (possibly empty and not necessarily disjoint), $\{C_1, ..., C_k\}$ is any *nonempty* subset of $\{A_1, ..., A_m\}$, and B is a *literal*.

Remark: In the above rule, B is a *literal* and not an arbitrary *proposition*. This is not a restriction because the system $GCNF'$ is complete. We could allow arbitrary propositions, but this would complicate the transformation of a proof tree into a resolution refutation (see the problems). The system obtained by restricting Γ_1 and Γ_2 to be subsets of Γ is also complete. However, if such a rule was used, this would complicate the transformation of a resolution refutation into a proof tree.

Deduction trees and proof trees are defined in the obvious way as in definition 3.4.5.

EXAMPLE 4.2.1

The following is a proof tree in the system $GCNF'$.

$$\dfrac{\neg \mathbf{S}, S \rightarrow \qquad \dfrac{S, \neg S \rightarrow \qquad \neg Q, Q \rightarrow}{\neg \mathbf{Q}, S, \{\neg S, Q\} \rightarrow}}{\{P, \neg \mathbf{Q}\}, \{\neg \mathbf{S}, \neg \mathbf{Q}\}, S, \{\neg S, Q\} \rightarrow}$$

In the above proof, the $\vee : left$ rule is applied to $\{P, \neg \mathbf{Q}\}, \{\neg \mathbf{S}, \neg \mathbf{Q}\}$. The literal $\neg \mathbf{Q}$, which occurs in both clauses, goes into the right premise of the lowest inference, and from the set $\{P, \neg S\}$, only $\neg S$ goes into

the left premise. The above proof in $GCNF'$ is more concise than the following proof in G'. This is because rules of $GCNF'$ can apply to several propositions in a single step, avoiding the duplication of subtrees.

$$\frac{T_1 \qquad T_2}{\{P, \neg Q\}, \{\neg S, \neg Q\}, S, \{\neg S, Q\} \rightarrow}$$

where T_1 is the tree

$$\dfrac{P, \neg S, S, \{\neg S, Q\} \rightarrow \qquad \dfrac{P, \neg Q, S, \neg S \rightarrow \qquad P, \neg Q, S, Q \rightarrow}{P, \neg Q, S, \{\neg S, Q\} \rightarrow}}{P, \{\neg S, \neg Q\}, S, \{\neg S, Q\} \rightarrow}$$

and T_2 is the tree

$$\dfrac{\neg Q, \neg S, S, \{\neg S, Q\} \rightarrow \qquad \dfrac{\neg Q, S, \neg S \rightarrow \qquad \neg Q, S, Q \rightarrow}{\neg Q, S, \{\neg S, Q\} \rightarrow}}{\neg Q, \{\neg S, \neg Q\}, S, \{\neg S, Q\} \rightarrow}$$

In order to prove the soundness and completeness of the System $GCNF'$, we need the following lemmas.

4.2.2 Soundness of the System $GCNF'$

First, we prove the following lemma.

Lemma 4.2.1 Every axiom is valid. For every valuation v, if v satisfies both premises of a rule of $GCNF'$, then v satisfies the conclusion.

Proof: It is obvious that every axiom is valid. For the second part, it is equivalent to show that if v falsifies $\Gamma, (A_1 \vee B), ..., (A_m \vee B) \rightarrow$, then either v falsifies $\Gamma_1, C_1, ..., C_k \rightarrow$, or v falsifies $\Gamma_2, B \rightarrow$. But v falsifies $\Gamma, (A_1 \vee B), ..., (A_m \vee B) \rightarrow$ if v satisfies all propositions in Γ and v satisfies all of $(A_1 \vee B)$,...,$(A_m \vee B)$. If v satisfies B, then v satisfies Γ and B, and so v falsifies $\Gamma_2, B \rightarrow$ for every subset Γ_2 of Γ,$(A_1 \vee B)$,...,$(A_m \vee B)$. If v does not satisfy B, then it must satisfy all of $A_1, ..., A_m$. But then, v satisfies Γ_1 and $C_1, ..., C_k$ for any subset Γ_1 of Γ,$(A_1 \vee B)$,...,$(A_m \vee B)$ and nonempty subset $\{C_1, ..., C_k\}$ of $\{A_1, ..., A_m\}$. This concludes the proof. □

Using induction on proof trees as in lemma 3.4.3, we obtain the following corollary:

Corollary (Soundness of $GCNF'$) Every sequent provable in $GCNF'$ is valid.

In order to prove the completeness of $GCNF'$, we will also need the following normal form lemma.

Lemma 4.2.2 Every proof in G' for a sequent of the form $C_1, ..., C_m \rightarrow$, where each C_i is a clause is equivalent to a G'-proof in which all instances of the $\neg : left$ rule precede all instances of the $\vee : left$ rule, in the sense that on every path from a leaf to the root, all $\neg : left$ rules precede all $\vee : left$ rules.

Proof: First, observe that every proof tree in G' of the form

Subtree of type (1)

$$\dfrac{\dfrac{\dfrac{T_1}{\Gamma, A \rightarrow P} \qquad \dfrac{T_2}{\Gamma, B \rightarrow P}}{\Gamma, (A \vee B) \rightarrow P}}{\Gamma, (A \vee B), \neg P \rightarrow}$$

can be transformed to a proof tree of the same depth of the form

Subtree of type (2)

$$\dfrac{\dfrac{\dfrac{T_1}{\Gamma, A \rightarrow P}}{\Gamma, A, \neg P \rightarrow} \qquad \dfrac{\dfrac{T_2}{\Gamma, B \rightarrow P}}{\Gamma, B, \neg P \rightarrow}}{\Gamma, (A \vee B), \neg P \rightarrow}$$

Next, we prove the lemma by induction on proof trees. Let T be a proof tree. If T is a one-node tree, it is an axiom and the lemma holds trivially. Otherwise, either the inference applied at the root is the $\vee : left$ rule, or it is the $\neg : left$ rule. If it is the $\vee : left$ rule, T has two subtrees T_1 and T_2 with root S_1 and S_2, and by the induction hypothesis, there are proof trees T_1' and T_2' with root S_1 and S_2 satisfying the conditions of the lemma. The tree obtained from T by replacing T_1 by T_1' and T_2 by T_2' satisfies the conditions of the lemma. Otherwise, the inference applied at the root of T is the $\neg : left$ rule. Let T_1 be the subtree of T having $\Gamma, (A \vee B) \rightarrow P$ as its root. If T_1 is an axiom, the lemma holds trivially. If the rule applied at the root of T_1 is a $\neg : left$ rule, by the induction hypothesis, there is a tree T_1' with the same depth as T_1 satisfying the condition of the lemma. If T_1' contains only $\neg : left$ rules, the lemma holds since for some letter Q, both Q and $\neg Q$ must belong to $\Gamma, (A \vee B), \neg P \rightarrow$. Otherwise, the tree T' obtained by replacing T_1 by T_1' in T is a tree of type (1), and $depth(T') = depth(T)$. By the the remark at the beginning of the proof, this tree can be replaced by a tree of type (2) having the same depth as T' (and T). We conclude by applying the induction hypothesis to the two subtrees of T'. Finally, if the rule applied at the root

of the subtree T_1 is a $\vee : left$ rule, T is a tree of type (1). We conclude as in the previous case. □

4.2.3 Completeness of the System $GCNF'$

Having this normal form, we can prove the completeness of $GCNF'$.

Theorem 4.2.1 (Completeness of $GCNF'$). If a sequent

$$C_1, ..., C_m \rightarrow$$

is valid, then it is provable in $GCNF'$. Equivalently, if

$$C_1 \wedge ... \wedge C_m$$

is unsatisfiable, the sequent

$$C_1, ..., C_m \rightarrow$$

is provable in $GCNF'$. Furthermore, every nontrivial valid sequent has a proof in $GCNF'$ in which the axioms are of the form $P, \neg P \rightarrow$.

Proof: Since $\vee$ is associative, commutative and idempotent, a clause $\{L_1, ..., L_k\}$ is equivalent to the formula

$$(((...(L_1 \vee L_2) \vee ...) \vee L_{k-1}) \vee L_k).$$

By the completeness of G' (theorem 3.4.1), the sequent C_1,...,$C_m \rightarrow$ has a proof in which the $\vee : left$ rules are instances of the rule of definition 4.2.3. From lemma 4.2.2, the sequent $C_1, ..., C_m \rightarrow$ has a G'-proof in which all the $\neg : left$ rules precede all the $\vee : left$ rules. We prove that every nontrivial valid sequent $C_1, ..., C_m \rightarrow$ has a proof in $GCNF'$ by induction on proof trees in G'.

Let T be a G'-proof of $C_1, ..., C_m \rightarrow$ in G'. Since it is assumed that $C_1, ..., C_m \rightarrow$ is nontrivial, some C_i, say C_1, is of the form $(A \vee B)$, and the bottom inference must be the $\vee : left$ rule. So, T is of the form

$$\dfrac{\dfrac{T_1}{A, C_2, ..., C_m \rightarrow} \qquad \dfrac{T_2}{B, C_2, ..., C_m \rightarrow}}{(A \vee B), C_2, ..., C_m \rightarrow}$$

If both T_1 and T_2 only contain applications of the $\neg : left$ rule, A must be a literal and some C_i is its conjugate, since otherwise, $C_2, ..., C_m$ would be trivial. Assume that C_i is the conjugate of A. Similarly, B is a literal, and some C_j is its conjugate.

Then, we have the following proof in $GCNF'$:

$$\frac{A, \overline{A} \rightarrow \qquad B, \overline{B} \rightarrow}{(A \vee B), C_2, ..., C_m \rightarrow}$$

In this proof, $\Gamma_1 = \{\overline{A}\} = C_i$, and $\Gamma_2 = \{\overline{B}\} = C_j$.

Otherwise, either $A, C_2, ..., C_m$ or $B, C_2, ..., C_m$ is nontrivial. If both are nontrivial, by the induction hypothesis, they have proofs S_1 and S_2 in $GCNF'$, and by one more application of the $\vee : left$ rule, we obtain the following proof in $GCNF'$:

$$\frac{\dfrac{S_1}{A, C_2, ..., C_m \rightarrow} \qquad \dfrac{S_2}{B, C_2, ..., C_m \rightarrow}}{(A \vee B), C_2, ..., C_m \rightarrow}$$

If $A, C_2, ..., C_m$ is trivial, then A must be a literal and some C_i is its conjugate, since otherwise, $C_1, ..., C_m$ would be trivial. Assume that $C_i = \overline{A}$. Since $B, C_2, ..., C_m$ is nontrivial, by the induction hypothesis it has a proof S_2 in $GCNF'$. Then, the following is a proof of $C_1, ..., C_m \rightarrow$ in $GCNF'$:

$$\frac{A, \overline{A} \rightarrow \qquad \dfrac{S_2}{B, C_2, ..., C_m \rightarrow}}{(A \vee B), C_2, ..., C_m \rightarrow}$$

The case in which $A, C_2, ..., C_m$ is nontrivial and $B, C_2, ..., C_m$ is trivial is symmetric. This concludes the proof of the theorem. □

The completeness of the system $GCNF'$ has been established by showing that it can simulate the system G'. However, the system $GCNF'$ is implicitly more nondeterministic than the system G'. This is because for a given sequent of the form

$$\Gamma, (A_1 \vee B), ..., (A_m \vee B) \rightarrow ,$$

one has the choice to pick the number of propositions $(A_1 \vee B)$,...,$(A_m \vee B)$, and to pick the subsets Γ_1, Γ_2 and $C_1, ..., C_k$. A consequence of this nondeterminism is that smaller proofs can be obtained, but that the process of finding proofs is not easier in $GCNF'$ than it is in G'.

PROBLEMS

4.2.1. Show that the set of clauses

$$\{\{A, B, \neg C\}, \{A, B, C\}, \{A, \neg B\}, \{\neg A\}\}$$

is unsatisfiable.

4.2.2. Show that the following sets of clauses are unsatisfiable:

(a) $\{\{A, \neg B, C\}, \{B, C\}, \{\neg A, C\}, \{B, \neg C\}, \{\neg B\}\}$

(b) $\{\{A, \neg B\}, \{A, C\}, \{\neg B, C\}, \{\neg A, B\}, \{B, \neg C\}, \{\neg A, \neg C\}\}$

4.2.3. Which of the following sets of clauses are satisfiable? Give satisfying valuations for those that are satisfiable, and otherwise, give a proof tree in $GCNF'$:

(a) $\{\{A, B\}, \{\neg A, \neg B\}, \{\neg A, B\}\}$

(b) $\{\{\neg A\}, \{A, \neg B\}, \{B\}\}$

(c) $\{\{A, B\}, \bot\}$

4.2.4. Consider the following algorithm for converting a proposition A into a proposition B in conjunctive normal form, such that A is satisfiable if and only if B is satisfiable. First, express A in terms of the connectives $\vee$, $\wedge$ and $\neg$. Then, let $A_1, ..., A_n$ be all the subformulae of A, with $A_n = A$. Let $P_1, ..., P_n$ be distinct propositional letters. The proposition B is the conjunction of P_n with the conjunctive normal forms of the following propositions:

$$\begin{array}{rll} ((P_i \vee P_j) \equiv P_k) & \text{whenever} & A_k \text{ is } (A_i \vee A_j) \\ ((P_i \wedge P_j) \equiv P_k) & \text{whenever} & A_k \text{ is } (A_i \wedge A_j) \\ (\neg P_i \equiv P_k) & \text{whenever} & A_k \text{ is } \neg A_i. \end{array}$$

(a) Prove that A is satisfiable if and only if B is satisfiable, by showing how to obtain a valuation satisfying B from a valuation satisfying A and vice versa.

(b) Prove that the above algorithm runs in polynomial time in the length of the proposition A.

Hint: Use problem 3.2.4.

4.2.5. Prove that the system $GCNF'$ is still complete if we require all leaf nodes of a proof tree to contain only literals.

4.2.6. Design a new system $GDNF'$ with a single inference rule $\wedge : right$, for sequents of the form

$$\rightarrow D_1, ..., D_n,$$

where $D_1 \vee ... \vee D_n$ is in disjunctive normal form. Prove the completeness of such a system.

4.2.7. Prove that the system $GCNF'$ is complete for infinite sets of clauses.

4.2.8. Write a computer program for testing whether a set of clauses is unsatisfiable, using the system $GCNF'$.

4.3 The Resolution Method for Propositional Logic

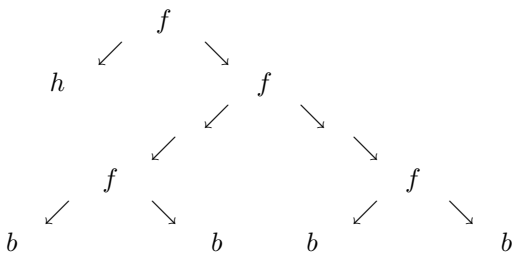
We will now present the resolution method, and show how a proof in the system $GCNF'$ can be transformed into a proof by resolution (and vice versa). In the resolution method, it is convenient to introduce a notation for the clause $\{\bot\}$ consisting of the constant *false* ($\bot$). This clause is denoted by $\square$ and is called the *empty clause*. Then, one attempts to establish the unsatisfiability of a set $\{C_1, ..., C_m\}$ of clauses by constructing a kind of tree whose root is the empty clause. Such trees usually contain copies of identical subtrees, and the resolution method represents them as collapsed trees. Technically, they are represented by directed acyclic graphs (DAGs).

Proof trees in $GCNF'$ can also be represented as DAGs and in this way, more concise proofs can be obtained. We could describe the algorithms for converting a proof in $GCNF'$ into a proof by resolution and vice versa in terms of DAGs, but this is not as clear and simple for the system $GCNF'$ using DAGs as it is for the standard system $GCNF'$ using trees. Hence, for clarity and simplicity, we shall present our results in terms of proof trees and resolution DAGs. Some remarks on proof DAGs will be made at the end of this section.

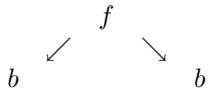
4.3.1 Resolution DAGs

For our purposes, it is convenient to define DAGs as pairs (t, R) where t is a tree and R is an equivalence relation on the set of tree addresses of t satisfying certain conditions. Roughly speaking, R specifies how common subtrees are shared (collapsed into the same DAG).

EXAMPLE 4.3.1



The above tree t contains two copies of the subtree



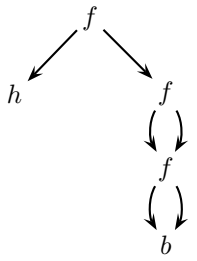
which itself contains two copies of the one-node subtree b.

If we define the equivalence relation R on the domain $dom(t) = \{e, 1, 2, 21, 22, 211, 212,\ 221, 222\}$ as the least equivalence relation containing the set of pairs

$$\{(21, 22), (211, 221), (212, 222), (211, 212)\},$$

the equivalence classes modulo R are the nodes of a graph representing the tree t as a collapsed tree. This graph can be represented as follows:

EXAMPLE 4.3.2



The equivalence relation R is such that two equivalent tree addresses must be the roots of identical subtrees. If two addresses u, v are equivalent, then they must be independent (as defined in Subsection 2.2.2), so that there are no cycles in the graph.

The formal definition of a DAG is as follows.

Definition 4.3.1 A *directed acyclic graph* (for short, a *DAG*) is a pair (t, R), where t is a tree labeled with symbols from some alphabet Δ, and R is an equivalence relation on $dom(t)$ satisfying the following conditions:

For any pair of tree addresses $u, v \in dom(t)$, if $(u, v) \in R$, then either $u = v$ or u and v are independent (as defined in Subsection 2.2.2) and, for all $i > 0$, we have:

(1) $ui \in dom(t)$ iff $vi \in dom(t)$;

(2) If $ui \in dom(t)$, then $(ui, vi) \in R$;

(3) $t(u) = t(v)$.

The tree t is called the *underlying tree* of the DAG. The equivalence classes modulo R are called the *nodes* of the DAG. Given any two equivalence classes S and T, an *edge* from S to T is any pair $([u], [ui])$ of equivalence classes of tree addresses modulo R, such that $u \in S$, and $ui \in T$. By conditions (1)-(3), the definition of an edge makes sense.

The concept of root, descendant, ancestor, and path are also well defined for DAGs: They are defined on the underlying tree.

This definition only allows connected rooted DAGs, but this is enough for our purpose. Indeed, DAGs arising in the resolution method are sets of DAGs as defined above.

We now describe the resolution method.

4.3.2 Definition of the Resolution Method for Propositional Logic

The resolution method rests on the fact that the proposition

$$((A \vee P) \wedge (B \vee \neg P)) \equiv ((A \vee P) \wedge (B \vee \neg P) \wedge (A \vee B)) \qquad (*)$$

is a tautology. Indeed, since the above is a tautology, the set of clauses

$$\{C_1, ..., C_m, \{A, P\}, \{B, \neg P\}\}$$

is logically equivalent to the set

$$\{C_1, ..., C_m, \{A, P\}, \{B, \neg P\}, \{A, B\}\}$$

obtained by adding the clause $\{A, B\}$. Consequently, the set

$$\{C_1, ..., C_m, \{A, P\}, \{B, \neg P\}\}$$

is unsatisfiable if and only if the set

$$\{C_1, ..., C_m, \{A, P\}, \{B, \neg P\}, \{A, B\}\}$$

is unsatisfiable.

The clause $\{A, B\}$ is a *resolvent* of the clauses $\{A, P\}$ and $\{B, \neg P\}$. The resolvent of the clauses $\{P\}$ and $\{\neg P\}$ is the empty clause $\square$. The process of adding a resolvent of two clauses from a set S to S is called a *resolution step*. The resolution method attempts to build a sequence of sets of clauses obtained by successive resolution steps, and ending with a set containing the empty clause. When this happens, we know that the original clause is unsatisfiable, since resolution steps preserve unsatisfiability, and a set of clauses containing the empty clause is obviously unsatisfiable.

There are several ways of recording the resolution steps. A convenient and space-efficient way to do so is to represent a sequence of resolution steps as a DAG. First, we show that the proposition $(*)$ defined above is a tautology.

Lemma 4.3.1 The proposition

$$((A \vee P) \wedge (B \vee \neg P)) \equiv ((A \vee P) \wedge (B \vee \neg P) \wedge (A \vee B))$$

is a tautology, even when either A or B is empty. (When A is empty, $(A \vee P)$ reduces to P and $(A \vee B)$ to B, and similarly when B is empty. When both A and B are empty, we have the tautology $(P \wedge \neg P) \equiv (P \wedge \neg P)$.)

Proof: We prove that

$$((A \vee P) \wedge (B \vee \neg P)) \supset ((A \vee P) \wedge (B \vee \neg P) \wedge (A \vee B))$$

is valid and that

$$((A \vee P) \wedge (B \vee \neg P) \wedge (A \vee B)) \supset ((A \vee P) \wedge (B \vee \neg P))$$

is valid. The validity of the second proposition is immediate. For the first one, it is sufficient to show that

$$((A \vee P) \wedge (B \vee \neg P)) \supset (A \vee B)$$

is valid. We have the following proof tree (in G'):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A, (B \vee \neg P) \rightarrow A, B \qquad \dfrac{P, B \rightarrow A, B \qquad \dfrac{P \rightarrow P, A, B}{P, \neg P \rightarrow A, B}}{P, (B \vee \neg P) \rightarrow A, B}}{(A \vee P), (B \vee \neg P) \rightarrow A, B}}{(A \vee P), (B \vee \neg P) \rightarrow (A \vee B)}}{(A \vee P) \wedge (B \vee \neg P) \rightarrow (A \vee B)}}{\rightarrow ((A \vee P) \wedge (B \vee \neg P)) \supset (A \vee B)}$$

□

Definition 4.3.2 Given two clauses C_1 and C_2, a clause C is a *resolvent* of C_1 and C_2 iff, for some literal L, $L \in C_1$, $\overline{L} \in C_2$, and

$$C = (C_1 - \{L\}) \cup (C_2 - \{\overline{L}\}).$$

In other words, a resolvent of two clauses is any clause obtained by striking out a literal and its conjugate, one from each, and merging the remaining literals into a single clause.

EXAMPLE 4.3.3

The clauses $\{A, B\}$ and $\{\neg A, \neg B\}$ have the two resolvents $\{A, \neg A\}$ and $\{B, \neg B\}$. The clauses $\{P\}$ and $\{\neg P\}$ have the empty clause as their resolvent.

Observe that by lemma 4.3.1, any set S of clauses is logically equivalent to the set $S \cup \{C\}$ obtained by adding any resolvent of clauses in S. However, it is not true that the set

$$(S \cup \{C\}) - \{C_1, C_2\}$$

obtained by adding a resolvent of two clauses C_1 and C_2, and deleting C_1 and C_2 from S is equivalent to S. If $S = \{\{P, Q\}, \{\neg Q\}\}$, then $\{P\}$ is a resolvent of $\{P, Q\}$ and $\{\neg Q\}$, but S is not equivalent to $\{\{P\}\}$. Indeed, the valuation v which satisfies both P and Q satisfies $\{\{P\}\}$ but does not satisfy S since it does not satisfy $\{\neg Q\}$.

We now define resolution DAGs. We believe that it is more appropriate to call resolution DAGs having the empty clause at their root *resolution refutations* rather than *resolutions proofs*, since they are used to show the unsatisfiability of a set of clauses.

Definition 4.3.3 Given a set $S = \{C_1, ..., C_n\}$ of clauses, a *resolution DAG* for S is any finite set

$$D = \{(t_1, R_1), ..., (t_m, R_m)\}$$

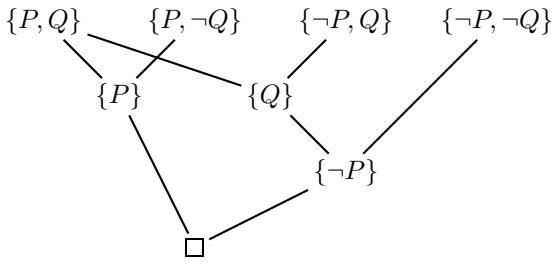
of distinct DAGs labeled with clauses and such that:

(1) The leaf nodes of each underlying tree t_i are labeled with clauses in S.

(2) For every DAG (t_i, R_i), for every non-leaf node u in t_i, u has exactly two successors $u1$ and $u2$, and if $u1$ is labeled with a clause C_1 and $u2$ is labeled with a clause C_2 (not necessarily distinct from C_1), then u is labeled with a resolvent of C_1 and C_2.

A resolution DAG is a *resolution refutation* iff it consists of a single DAG (t, R) whose root is labeled with the empty clause. The nodes of a DAG which are not leaves are also called *resolution steps*.

EXAMPLE 4.3.4



The DAG of example 4.3.4 is a resolution refutation.

4.3.3 Soundness of the Resolution Method

The soundness of the resolution method is given by the following lemma.

Lemma 4.3.2 If a set S of clauses has a resolution refutation DAG, then S is unsatisfiable.

Proof: Let (t_1, R_1) be a resolution refutation for S. The set S' of clauses labeling the leaves of t_1 is a subset of S.

First, we prove by induction on the number of nodes in a DAG (t, R) that the set of clauses S' labeling the leaves of t is equivalent to the set of clauses labeling all nodes in t.

If t has one node, the property holds trivially. If t has more than one node, let l and r be the left subtree and right subtree of t respectively. By the induction hypothesis, the set S_1 of clauses labeling the nodes of l is equivalent to the set L_1 of clauses labeling the leaves of l, and the set S_2 of clauses labeling the nodes of r is equivalent to the set L_2 of clauses labeling the leaves of r. Let C_1 and C_2 be the clauses labeling the root of l and r respectively. By definition 4.3.3, the root of t is a resolvent R of C_1 and C_2. By lemma 4.3.1, the set $S_1 \cup S_2$ is equivalent to the set $S_1 \cup S_2 \cup \{R\}$. But then, $S_1 \cup S_2 \cup \{R\}$ is equivalent to $S' = L_1 \cup L_2$, since $S_1 \cup S_2$ is equivalent to $L_1 \cup L_2$. This concludes the induction proof.

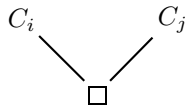
Applying the above property to t_1, S' is equivalent to the set of clauses labeling all nodes in t_1. Since the resolvent D labeling the root of t_1 is the empty clause, the set S' of clauses labeling the leaves of t_1 is unsatisfiable, which implies that S is unsatisfiable since S' is a subset of S. □

4.3.4 Converting $GCNF'$-proofs into Resolution Refutations and Completeness

The completeness of the resolution method is proved by showing how to transform a proof in the system $GCNF'$ into a resolution refutation.

Theorem 4.3.1 There is an algorithm for constructing a resolution refutation D from a proof tree T in the system $GCNF'$. Furthermore, the number of resolution steps in D (nonleaf nodes) is less than or equal to the number of axioms in T.

Proof: We give a construction and prove its correctness by induction on proof trees in $GCNF'$. Let $S = \{C_1, ..., C_m\}$. If the proof tree of the sequent $C_1, ..., C_m \rightarrow$ is a one-node tree labeled with an axiom, there must a literal L such that some $C_i = L$ and some $C_j = \overline{L}$. Hence, we have the resolution refutation consisting of the DAG



This resolution DAG has one resolution step, so the base step of the induction holds.

Otherwise, the proof tree is of the form

$$\dfrac{\dfrac{T_1}{\Gamma_1, F_1, ..., F_k \rightarrow} \qquad \dfrac{T_2}{\Gamma_2, B \rightarrow}}{\Gamma, \{A_1, B\}, ..., \{A_n, B\} \rightarrow}$$

where $\Gamma_1, \Gamma_2 \subseteq \Gamma \cup \{\{A_1, B\}, ..., \{A_n, B\}\}$, $\{F_1, ..., F_k\} \subseteq \{A_1, ...A_n\}$, and $\Gamma \cup \{\{A_1, B\}, ..., \{A_n, B\}\} = S$. By the induction hypothesis, there is a resolution refutation D_1 obtained from the tree T_1 with root $\Gamma_1, F_1, ..., F_k \rightarrow$, and a resolution refutation D_2 obtained from the tree T_2 with root $\Gamma_2, B \rightarrow$. The leaves of D_1 are labeled with clauses in $\Gamma_1 \cup \{F_1, ..., F_k\}$, and the leaves of D_2 are labeled with clauses in $\Gamma_2 \cup \{B\}$. Let $\{F'_1, ..., F'_p\}$ be the subset of clauses in $\{F_1, ..., F_k\}$ that label leaves of D_1.

If $\{F'_1, ..., F'_p\} \subseteq S$, then D_1 is also resolution refutation for S. Similarly, if the set of clauses labeling the leaves of D_2 is a subset of S, D_2 is a resolution refutation for S. In both cases, by the induction hypothesis, the number of resolution steps in D_1 (resp. D_2) is less than or equal to the number of leaves of T_1 (resp. T_2), and so, it is less than the number of leaves of T.

Otherwise, let $\{F''_1, ..., F''_q\} \subseteq \{F_1, ..., F_k\}$ be the set of clauses *not* in S labeling the leaves of D_1. Also, $B \notin S$ and B labels some leaf of D_2, since otherwise the clauses labeling the leaves of D_2 would be in S. Let D'_1 be the resolution DAG obtained from D_1 by replacing $F''_1, ..., F''_q$ by $\{F''_1, B\}, ..., \{F''_q, B\}$, and applying the same resolution steps in D'_1 as the resolution steps applied in D_1. We obtain a resolution DAG that may be a resolution refutation, or in which the clause B is the label of the root node.

If D'_1 is a resolution refutation, we are done.

Otherwise, since $B \notin S$ and D'_1 is not a resolution refutation, the root of D'_1 is labeled with B and no leaf of D'_1 is labeled with B. By the induction hypothesis, the number of resolution steps n_1 in D_1 (and D'_1) is less than or equal to the number of axioms in T_1, and the number n_2 of resolution steps in D_2 is less than or equal to the number of axioms in T_2. We construct a resolution refutation for a subset of $\Gamma, \{A_1, B\}, ..., \{A_n, B\}$ by combining D'_1 and D_2 as follows: Identify the root labeled with B in D'_1 with the leaf labeled with B in D_2, and identify all leaf nodes u, v with $u \in D'_1$ and $v \in D_2$, iff u and v are labeled with the same clause. (Since $B \notin S$, B cannot be such a clause.) The number of resolution steps in D is $n_1 + n_2$, which is less than or

equal to the number of axioms in T. This concludes the construction and the proof. □

Theorem 4.3.2 (Completeness of the resolution method) Given a finite set S of clauses, S is unsatisfiable if and only if there is a resolution refutation for S.

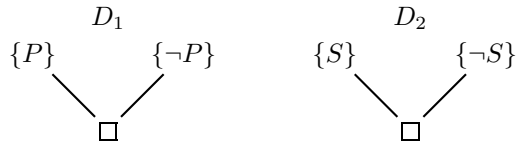
Proof: By lemma 4.3.2, if S has a resolution refutation, S is unsatisfiable. If S is unsatifiable, then the sequent $S \rightarrow$ is valid. Since the system $GCNF'$ is complete (theorem 4.2.1), there is a proof tree T for $S \rightarrow$. Finally, by theorem 4.3.1, a resolution refutation D is obtained from T. □

EXAMPLE 4.3.5

Consider the following proof tree in $GCNF'$:

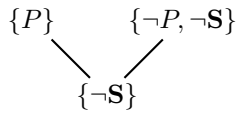
$$\dfrac{\dfrac{P, \neg P \rightarrow \qquad S, \neg S \rightarrow}{P, S, \{\neg P, \neg S\} \rightarrow} \qquad \neg Q, Q \rightarrow}{\{P, \neg Q\}, \{S, \neg Q\}, Q, \{\neg P, \neg S\} \rightarrow}$$

The leaves $P, \neg P \rightarrow$ and $S, \neg S \rightarrow$ are mapped into the following DAGs:



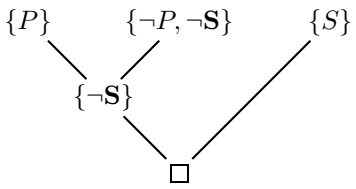
Let D_3 be the DAG obtained from D_1 by adding $\neg S$ to $\{\neg P\}$:

D_3

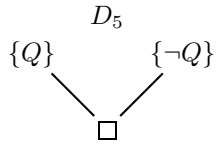


We now combine D_3 and D_2 to obtain D_4:

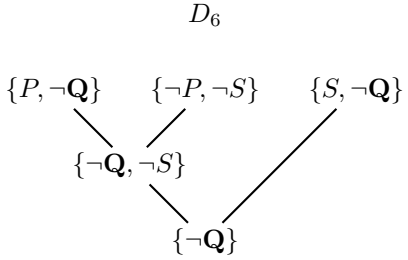
D_4



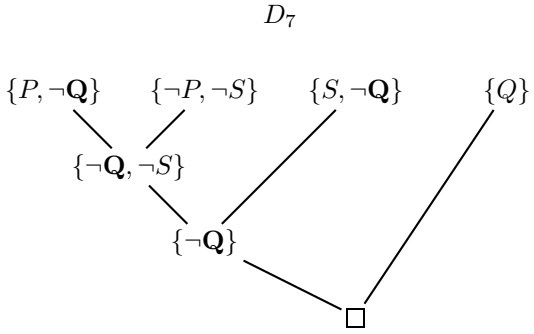
The leaf $Q, \neg Q \rightarrow$ is mapped into the DAG D_5:



Let D_6 be the DAG obtained from D_4 by adding $\neg Q$ to $\{P\}$ and $\{S\}$:



Finally, let D_7 be obtained by combining D_6 and D_5:



The theorem actually provides an algorithm for constructing a resolution refutation. Since the number of resolution steps in D is less than or equal to the number of axioms in T, the number of nodes in the DAG D cannot be substantially larger than the number of nodes in T. In fact, the DAG D has at most two more nodes than the tree, as shown by the following lemma.

Lemma 4.3.3 If T is a proof tree in $GCNF'$ and T has m nodes, the number n of nodes in the DAG D given by theorem 4.3.1 is such that, $r{+}1 \leq n \leq m{+}2$, where r is the number of resolution steps in D.

Proof: Assume that T has k leaves. Since this tree is binary branching, it has $m = 2k - 1$ nodes. The number r of resolution steps in D is less than or equal to the number k of leaves in T. But the number n of nodes in the DAG is at least $r + 1$ and at most $2r + 1$, since every node has at most two successors. Hence, $n \leq 2r + 1 \leq 2k + 1 = m + 2$. $\square$

In example 4.3.5, the number of resolution steps in the DAG is equal to the number of leaves in the tree. The following example shows that the number of resolution steps can be strictly smaller.

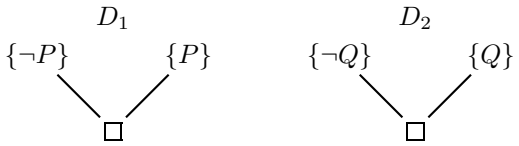
EXAMPLE 4.3.6

Consider the following proof tree T:

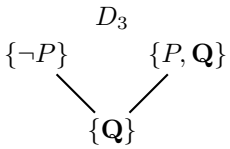
$$\dfrac{\dfrac{\dfrac{}{\neg P, P \rightarrow} \quad \dfrac{}{\neg Q, Q \rightarrow}}{\neg Q, \neg P, \{P, Q\} \rightarrow}}{\neg R, R \rightarrow} \qquad \dfrac{}{\neg R, R \rightarrow}$$

$$\dfrac{\neg R, \neg Q, \{R, \neg P\}, \{P, Q\} \rightarrow \qquad \neg R, R \rightarrow}{\neg R, \neg Q, \{R, \neg P\}, \{P, Q, R\} \rightarrow}$$

The result of recursively applying the algorithm to T yields the two DAGs D_1 and D_2:



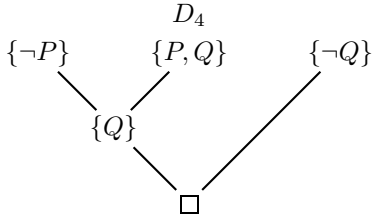
corresponding to the axioms $\neg P, P \rightarrow$ and $\neg Q, Q \rightarrow$. Adding $\mathbf{Q}$ to P in the DAG D_1, we obtain the DAG D_3:



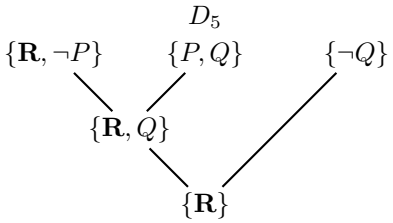
Identifying the root of DAG D_3 with the leaf labeled Q of DAG D_2, we obtain the DAG D_4 corresponding to the subtree

$$\dfrac{\dfrac{}{\neg P, P \rightarrow} \quad \dfrac{}{\neg Q, Q \rightarrow}}{\neg Q, \neg P, \{P, Q\} \rightarrow}$$

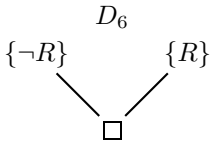
of T:



Adding $\mathbf{R}$ to $\neg P$ in D_4, we obtain the DAG D_5:



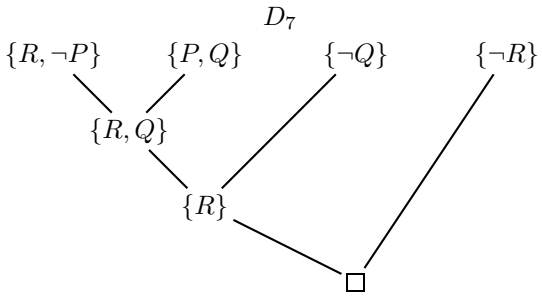
The axiom $\neg R, R \rightarrow$ yields the DAG D_6:



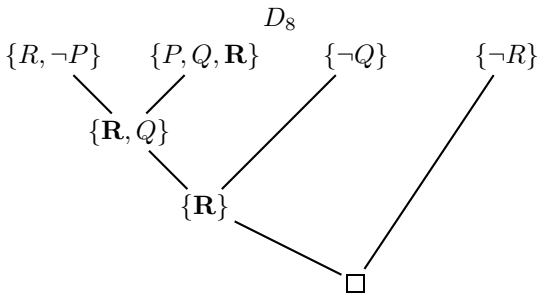
and merging the root of D_5 with the leaf labeled R in D_6, we obtain the DAG D_7 corresponding to the left subtree

$$\dfrac{\dfrac{\neg R, R \rightarrow}{\neg R, \neg Q, \{R, \neg P\}, \{P, Q\} \rightarrow} \qquad \dfrac{\neg P, P \rightarrow \qquad \neg Q, Q \rightarrow}{\neg Q, \neg P, \{P, Q\} \rightarrow}}{}$$

of the original proof tree:



At this stage, since the right subtree of the proof tree T is the axiom $\neg R, R \rightarrow$, we add $\mathbf{R}$ to $\{P, Q\}$ in D_7. However, since $\mathbf{R}$ is also in $\{R, \neg P\}$, the resulting DAG D_8 is a resolution refutation:



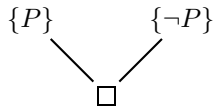
The DAG D_8 has three resolution steps, and the original proof tree T has four axioms. This is because the original proof tree is not optimal. If we had applied the $\vee : rule$ to $\{\neg P, R\}$ and $\{P, Q, R\}$, we would obtain a proof tree with three axioms, which yields a DAG with three resolution steps.

4.3.5 From Resolution Refutations to $GCNF'$-proofs

We will now prove that every resolution refutation can be transformed into a proof tree in $GCNF'$. Unfortunately, contrary to what was claimed in the original edition of this book, the size of the resulting tree is not necessarily polynomial in the size of the DAG. This result only holds if the resolution refutation is a tree.

Theorem 4.3.3 There is an algorithm which transforms any resolution refutation D into a proof tree T in the system $GCNF'$.

Proof: Let $S = \{C_1, ..., C_m\}$. We construct the tree and prove the theorem by induction on the number of resolution steps in D. If D contains a single resolution step, D is a DAG of the form



Hence, for some clauses C_i and C_j in the set of clauses $\{C_1, ..., C_m\}$, $C_i = \{P\}$ and $C_j = \{\neg P\}$. Hence, the one-node tree labeled with the axiom $C_1, ..., C_m \rightarrow$ is a proof tree.

If D has more than one resolution step, it is a DAG (t, R) whose root is labeled with the empty clause. The descendants n_1 and n_2 of the root r of (t, R) are labeled with clauses $\{P\}$ and $\{\neg P\}$ for some propositional letter P. There are two cases: Either one of n_1, n_2 is a leaf, or neither is a leaf.

Case 1: Assume that n_1 is a leaf, so that $C_1 = \{P\}$, the other case being similar. Let

$$\{A_1, \neg P\}, ..., \{A_n, \neg P\}$$

be the terminal nodes of all paths from the node n_2, such that every node in each of these paths contains $\neg P$. Let D_2 be the resolution DAG obtained by deleting r and n_1 (as well as the edges having these nodes as source) and by deleting $\neg P$ from every node in every path from n_2 such that every node in such a path contains $\neg P$. The root of D_2 is labeled with the empty clause. By the induction hypothesis, there is a proof tree T_2 in $GCNF'$ for

$$\Gamma, A_1, ..., A_n \rightarrow, \quad \text{where}$$
$$\Gamma = \{C_2, ..., C_m\} - \{\{A_1, \neg P\}, ..., \{A_n, \neg P\}\}.$$

If $\{A_1, ..., A_n\}$ is a subset of Γ, then the tree obtained from T_2 by replacing Γ by $\{C_1, ..., C_m\}$ is also a proof tree. By the induction hypothesis the number of leaves k in T_2 is less than or equal to the number of resolution steps in D_2, which is less than the number of resolution steps in D.

If $\{A_1, ..., A_n\}$ is not a subset of Γ, then, the following is a proof tree T in $GCNF'$ for $C_1, ..., C_m \rightarrow$.

$$\dfrac{\dfrac{T_2}{\Gamma, A_1, ..., A_n \rightarrow} \qquad P, \neg P \rightarrow}{P, \Gamma, \{A_1, \neg P\}, ..., \{A_n, \neg P\} \rightarrow}$$

The number of axioms in T is $k-1+1 = k$, which is less than or equal to the number of resolution steps in D. In the special case where $\Gamma, A_1, ..., A_n \rightarrow$ is an axiom, since $C_1, ..., C_m \rightarrow$ is not a trivial sequent (the case of a trivial sequent being covered by the base case of a DAG with a single resolution step), there is some proposition in $\Gamma, A_1, ..., A_n$ which is the conjugate of one of the A_i. Then, T_2 is simply the axiom $A_i, \overline{A}_i \rightarrow$.

Case 2: Neither n_1 nor n_2 is a leaf. Let

$$\{A_1, P\}, ..., \{A_n, P\}$$

be the set of terminal nodes of all paths from the node n_1, such that every node in each of these paths contains P. Let D_1 be the resolution DAG consisting of the descendant nodes of n_1 (and n_1 itself), and obtained by deleting P from every node in every path from n_1 such that every node in such a path contains P. The root of D_1 is labeled with the empty clause. By the induction hypothesis, there is a proof tree T_1 in $GCNF'$ for $\Gamma_1, A_1, ..., A_n \rightarrow$, where Γ_1 is some subset of $\{C_1, ...C_m\}$. By the induction hypothesis the number of axioms in T_1 is less than or equal to the number m_1 of resolution steps in D_1. Similarly, let D_2 be the DAG obtained by deleting all descendants of n_1 that are not descendants of n_2, and deleting all edges having these nodes as source, including the edges from n_1, so that n_1 is now a leaf. By the induction hypothesis, there is a proof tree T_2 in $GCNF'$ with root labeled with $\Gamma_2, P \rightarrow$, where Γ_2 is some subset of $\{C_1, ..., C_m\}$.

If $\{A_1, ..., A_n\}$ is a subset of Γ_1, then the tree T_1 is also a proof tree for $\{C_1,...,C_m\}$. Similarly, if P belongs to Γ_2, the tree T_2 is a proof tree for $\{C_1,...,C_m\}$. Otherwise, we have the following proof tree T in $GCNF'$:

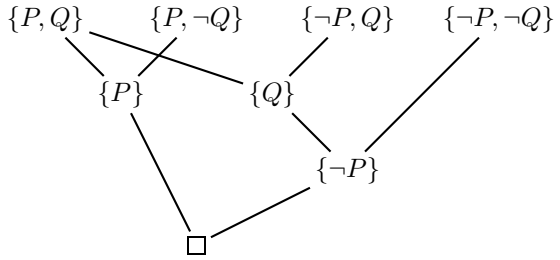
$$\frac{T_1 \qquad\qquad T_2}{\dfrac{\Gamma_1, A_1, ..., A_n \rightarrow \qquad\qquad \Gamma_2, P \rightarrow}{\Gamma, \{A_1, P\}, ..., \{A_n, P\} \rightarrow}}$$

The special cases in which either $\Gamma_1, A_1, ..., A_n \rightarrow$ or $\Gamma_2, P \rightarrow$ is an axiom is handled as in case 1. The details are left as an exercise. This completes the proof. □

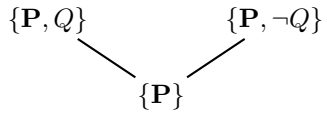
Remark. Although it is tempting to claim that the number of resolution steps in D is $m_1 + m_2$, this is unfortunately **false**! The problem is that because a resolution refutation is a DAG, certain nodes can be shared.

EXAMPLE 4.3.7

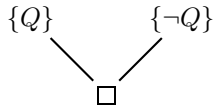
Let D be the following resolution refutation:



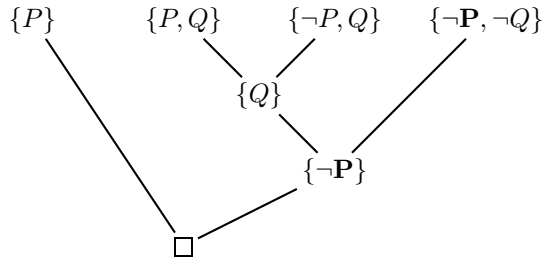
Let D_1 be the set of descendants of $\{P\}$:



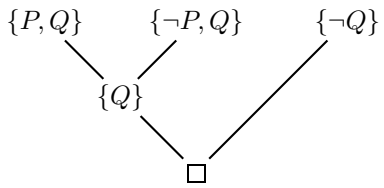
Let D_2 be the resolution refutation obtained by deleting P:



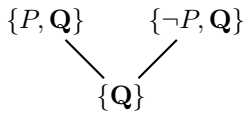
Let D_3 be the DAG obtained from D by deleting the descendants of $\{P\}$ that are not descendants of $\{\neg P\}$. Note that since $\{P, Q\}$ is a descendant of $\{\neg P\}$, it is retained. However, $\{P, \neg Q\}$ is deleted.



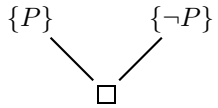
Let D_4 be the resolution refutation obtained by deleting $\neg P$ from all paths from $\{\neg P\}$ containing it:



Let D_5 be the set of descendants of $\{Q\}$:



Let D_6 be the resolution refutation obtained from D_5 by deleting Q:



The method of theorem 4.3.3 yields the following proof trees: The tree T_1:

$$Q, \neg Q \rightarrow$$

corresponds to D_2;

The tree T_2:

$$\frac{\mathbf{Q}, \neg Q \rightarrow \qquad \mathbf{P}, \neg \mathbf{P} \rightarrow}{\{\mathbf{P}, \mathbf{Q}\}, \{\neg \mathbf{P}, \mathbf{Q}\}, \neg Q \rightarrow}$$

corresponds to D_4;

The tree T_3:

$$\frac{\dfrac{Q, \neg Q \rightarrow \qquad P, \neg P \rightarrow}{\{P, Q\}, \{\neg P, Q\}, \neg \mathbf{Q} \rightarrow} \qquad P, \neg \mathbf{P} \rightarrow}{P, \{P, Q\}, \{\neg P, Q\}, \{\neg \mathbf{P}, \neg \mathbf{Q}\} \rightarrow}$$

corresponds to D_3;

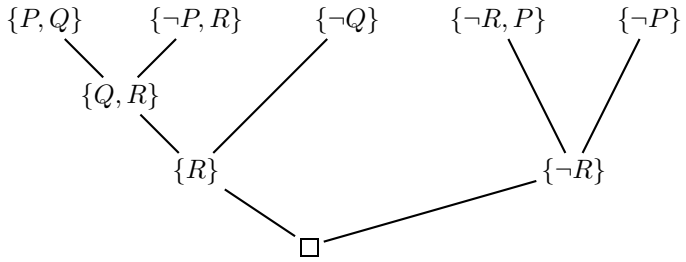
The tree T_4:

$$\dfrac{\dfrac{\dfrac{Q, \neg Q \rightarrow \qquad P, \neg P \rightarrow}{\{P, Q\}, \{\neg P, Q\}, \neg Q \rightarrow} \qquad P, \neg P \rightarrow}{\mathbf{P}, \{P, Q\}, \{\neg P, Q\}, \{\neg P, \neg Q\} \rightarrow} \qquad \mathbf{Q}, \neg \mathbf{Q} \rightarrow}{\{\mathbf{P}, \mathbf{Q}\}, \{\mathbf{P}, \neg \mathbf{Q}\}, \{\neg P, Q\}, \{\neg P, \neg Q\} \rightarrow}$$

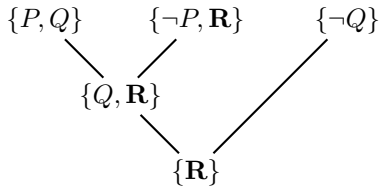
corresponds to D. The number of axioms of T_4 is four, which is the number of resolution steps in D. The next example shows that it is possible that the proof tree constructed from a resolution DAG has fewer leaves than the number of resolution steps.

EXAMPLE 4.3.8

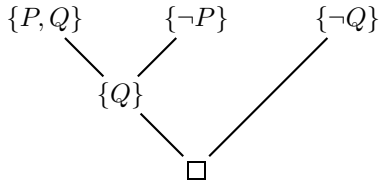
Consider the following resolution refutation D:



Let D_1 be the DAG consisting of the descendants of $\{R\}$:



Let D_2 be the DAG obtained from D_1 by deleting $\mathbf{R}$:



Note that D_2 is a resolution for a subset of the set of clauses

$$\{\{P, Q\}, \{\neg P, R\}, \{\neg Q\}, \{\neg R, P\}, \{\neg P\}\}.$$

The construction recursively applied to D_2 yields the following proof tree:

$$\frac{P, \neg P \rightarrow \qquad Q, \neg Q \rightarrow}{\{P, Q\}, \neg P, \neg Q \rightarrow}$$

From the above tree, we obtain the following proof tree for the original set of clauses:

$$\frac{P, \neg P \rightarrow \qquad Q, \neg Q \rightarrow}{\{P, Q\}, \{\neg P, R\}, \neg Q, \{\neg R, P\}, \neg P \rightarrow}$$

This last tree has two leaves, whereas the DAG D has four resolution steps.

When the resolution DAG D is a tree, we still have the following lemma showing that the Gentzen system $GCNF'$ is basically as efficient as the resolution method.

Lemma 4.3.4 If a resolution refutation D is a tree with m nodes, then the proof tree T constructed by the method of theorem 4.3.3 has a number of nodes n such that, $n \leq 2m - 3$.

Proof: Assume that D has r resolution steps. Since the tree T is binary branching and has $k \leq r$ leaves, it has $n = 2k - 1$ nodes. But the number m of nodes in the DAG D is such that, $r + 1 \leq m \leq 2r + 1$. Hence, $n \leq 2m - 3$. □

In the proof tree of example 4.3.7, there are two leaves labeled with $P, \neg P \rightarrow$, and two leaves labeled with $Q, \neg Q \rightarrow$. Hence, if this tree is represented as a DAG, it will have five nodes instead of seven. The original DAG has eight nodes. This suggests that the system $GCNF'$ using DAGs instead of trees is just as efficient as the resolution method. However, we do not have a proof of this fact at this time.

PROBLEMS

4.3.1. Show that the set of clauses

$$\{\{A, B, \neg C\}, \{A, B, C\}, \{A, \neg B\}, \{\neg A\}\}$$

is unsatisfiable using the resolution method.

4.3.2. Show that the following sets of clauses are unsatisfiable using the resolution method:

(a) $\{\{A, \neg B, C\}, \{B, C\}, \{\neg A, C\}, \{B, \neg C\}, \{\neg B\}\}$

(b) $\{\{A, \neg B\}, \{A, C\}, \{\neg B, C\}, \{\neg A, B\}, \{B, \neg C\}, \{\neg A, \neg C\}\}$

4.3.3. Construct a proof tree in $GCNF'$ for the set of clauses of problem 4.3.1, and convert it into a resolution DAG using the algorithm of theorem 4.3.1.

4.3.4. Construct proof trees in $GCNF'$ for the sets of clauses of problem 4.3.2, and convert them into resolution DAGs using the algorithm of theorem 4.3.1.

4.3.5. Convert the resolution DAG for the clause of problem 4.3.1 into a proof tree using the algorithm of theorem 4.3.3.

4.3.6. Convert the resolution DAGs for the clause of problem 4.3.2 into proof trees using the algorithm of theorem 4.3.3.

4.3.7. Find all resolvents of the following pairs of clauses:

(a) $\{A, B\}, \{\neg A, \neg B\}$

(b) $\{A, \neg B\}, \{B, C, D\}$

(c) $\{\neg A, B, \neg C\}, \{B, C\}$

(d) $\{A, \neg A\}, \{A, \neg A\}$

4.3.8. Construct a resolution refutation for the following set of clauses:

$$\{\{P, Q\}, \{\neg P, Q\}, \{P, \neg Q\}, \{\neg P, \neg Q\}\}.$$

Convert this resolution DAG into a proof tree using the algorithm of theorem 4.3.3

4.3.9. Another way of presenting the resolution method is as follows. Given a (finite) set S of clauses, let

$$R(S) = S \cup \{C \mid C \text{ is a resolvent of two clauses in } S\}.$$

Also, let

$$\begin{aligned} R^0(S) &= S, \\ R^{n+1}(S) &= R(R^n(S)), n \geq 0, \text{ and let} \\ R^*(S) &= \bigcup_{n \geq 0} R^n(S). \end{aligned}$$

(a) Prove that S is unsatisfiable if and only if $R^*(S)$ is unsatisfiable.

(b) Prove that if S is finite, there is some $n \geq 0$ such that

$$R^*(S) = R^n(S).$$

(c) Prove that there is a resolution refutation for S if and only if the empty clause $\square$ is in $R^*(S)$.

(d) Prove that S is unsatisfiable if and only if $\square$ belongs to $R^*(S)$.

4.3.10. Find $R(S)$ for the following sets of clauses:

(a) $\{\{A, \neg B\}, \{A, B\}, \{\neg A\}\}$

(b) $\{\{A, B, C\}, \{\neg B, \neg C\}, \{\neg A, \neg C\}\}$

(c) $\{\{\neg A, \neg B\}, \{B, C\}, \{\neg C, A\}\}$

(d) $\{\{A, B, C\}, \{A\}, \{B\}\}$

4.3.11. Prove that the resolution method is still complete if the resolution rule is restricted to clauses that are not tautologies (that is, clauses not containing both P and $\neg P$ for some propositional letter P.)

4.3.12. We say that a clause C_1 *subsumes* a clause C_2 if C_1 is a proper subset of C_2. In the version of the resolution method described in problem 4.3.8, let

$$R_1(S) = R(S) - \{C \mid C \text{ is subsumed by some clause in } R(S)\}.$$

Let

$$\begin{aligned} R_1^0 &= S, \\ R_1^{n+1}(S) &= R_1(R_1^n(S)) \text{ and} \\ R_1^*(S) &= \bigcup_{n \geq 0} \{R_1^n(S)\}. \end{aligned}$$

Prove that S is unsatisfiable if and only if $\square$ belongs to $R_1^*(S)$.

4.3.13. Prove that the resolution method is also complete for infinite sets of clauses.

4.3.14. Write a computer program implementing the resolution method.

Notes and Suggestions for Further Reading

The resolution method was discovered by J.A. Robinson (Robinson, 1965). An earlier method for testing the unsatisfiability of a set of clauses is the Davis-Putnam procedure (Davis and Putnam, 1960). Improvements due to Prawitz (Prawitz, 1960) and Davis (Davis, 1963) led to the resolution method. An exposition of resolution based on Gentzen sequents has also been given by J. Robinson (Robinson, 1969), but the technical details are different. To the best of our knowledge, the constructive method used in this chapter for proving the completeness of the resolution method by transforming a Gentzen-like proof is original.

With J.A. Robinson, this author believes that a Gentzen-sequent based exposition of resolution is the best from a pedagogical point of view and for theoretical understanding.

The resolution method and many of its refinements are discussed in Robinson, 1969; Loveland, 1978; and in Chang and Lee, 1973.