Logical Methods in Computer Science
Vol. 2 (5:3) 2006, pp. 1–7
www.lmcs-online.org

Submitted Jun. 9, 2006
Published Nov. 07, 2006

THE COMPLETENESS OF PROPOSITIONAL RESOLUTION A SIMPLE AND CONSTRUCTIVE PROOF

JEAN GALLIER

CIS Department, University of Pennsylvania, Philadelphia, PA 19104, USA
e-mail address: jean@cis.upenn.edu

Abstract. It is well known that the resolution method (for propositional logic) is complete. However, completeness proofs found in the literature use an argument by contradiction showing that if a set of clauses is unsatisfiable, then it must have a resolution refutation. As a consequence, none of these proofs actually gives an algorithm for producing a resolution refutation from an unsatisfiable set of clauses. In this note, we give a simple and constructive proof of the completeness of propositional resolution which consists of an algorithm together with a proof of its correctness.

1. Introduction

The resolution method for (propositional) logic due to J.A. Robinson [4] (1965) is well-known to be a sound and complete procedure for checking the unsatisfiability of a set of clauses. However, it appears that the completeness proofs that can be found in the literature (for instance, Chang and Lee [1], Lewis and Papadimitriou [3], Robinson [5]) are existence proofs that proceed by contradiction to show that if a set of clauses is unsatisfiable, then it must have a resolution refutation because otherwise a satisfying assignment can be obtained. In particular, none of these proofs yields (directly) an algorithm producing a resolution refutation from an unsatisfiable set of clauses. In that sense, these proofs are nonconstructive. In Gallier [2] (1986), we gave a completeness proof based on an algorithm for converting a Gentzen-like proof (using sequents) into a resolution DAG (see Chapter 4). Such a method is more constructive than the others but, we found later on that it is possible to give a simple and constructive proof of the completeness of propositional resolution which consists of an algorithm together with a proof of its correctness. This algorithm and its correctness are the object of this note.

It should be noted that Judith Underwood gave other constructive proof procedures in her Ph.D. thesis, notably for the intuitionistic propositional calculus [6].

2000 ACM Subject Classification: F.4.1, I.2 .
Key words and phrases: Resolution method, Unsatisfiability, Completeness, Constructive proof.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-2 (5:3) 2006

2. Review of Propositional Resolution

Recall that a *literal*, L, is either a propositional letter, P, or the negation, $\neg P$, of a propositional letter. A *clause* is a finite set of literals, $\{L_1, \ldots, L_k\}$, interpreted as the disjunction $L_1 \vee \cdots \vee L_k$ (when $k = 0$, this is the empty clause denoted $\square$). A set of clauses, $\Gamma = \{C_1, \ldots, C_n\}$, is interpreted as the conjunction $C_1 \wedge \cdots \wedge C_n$. For short, we write $\Gamma = C_1, \ldots, C_n$.

The *resolution method* (J.A. Robinson [4]) is a procedure for checking whether a set of clauses, Γ, is unsatisfiable. The resolution method consists in building a certain kind of labeled DAG whose leaves are labeled with clauses in Γ and whose interior nodes are labeled according to the *resolution rule*. Given two clauses $C = A \cup \{P\}$ and $C' = B \cup \{\neg P\}$ (where P is a propositional letter, $P \notin A$ and $\neg P \notin B$), the *resolvent of C and C'* is the clause

$$R = A \cup B$$

obtained by cancelling out P and $\neg P$. A *resolution DAG for* Γ is a DAG whose leaves are labeled with clauses from Γ and such that every interior node n has exactly two predecessors, n_1 and n_2 so that n is labeled with the resolvent of the clauses labeling n_1 and n_2. In a resolution step involving the nodes, n_1, n_2 and n, as above, we say that the two clauses C and C' labeling the nodes n_1 and n_2 are the *parent clauses* of the resolvent clause, R, labeling the node n. In a resolution DAG, D, a clause, C' is said to be a *descendant* of a clause, C, iff there is a (directed) path from some node labeled with C to a node labeled with C'. A *resolution refutation for* Γ is a resolution DAG with a single root whose label is the empty clause. (For more details on the resolution method, resolution DAGs, etc., one may consult Gallier [2], Chapter 4, or any of the books cited in Section 1.)

Here is an example of a resolution refutation for the set of clauses

$$\Gamma = \{\{P, Q\}, \{P, \neg Q\}, \{\neg P, Q\}, \{\neg P, \neg Q\}\}$$

shown in Figure 1:

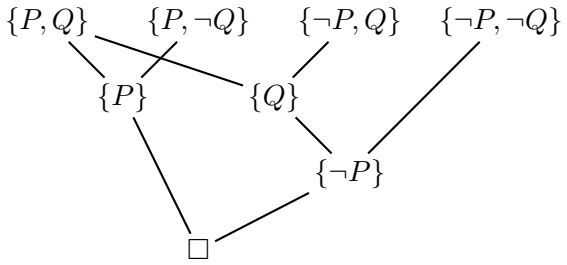

Figure 1: A Resolution Refutation

3. Completeness of Propositional Resolution: An Algorithm and its Correctness

Let Γ be a set of clauses. Thus, Γ is either the empty clause, $\square$, or it is a conjunction of clauses, $\Gamma = C_1, \ldots, C_n$. We define the *complexity*, $c(C)$, of a clause, C, as the number of disjunction symbols in C; i.e., if C consists of a single literal (i.e., $C = \{L\}$, for some literal, L), then $c(C) = 0$, else if $C = \{L_1, \ldots, L_m\}$ (with $m \geq 2$) where the L_i's are literals,

then $c(C) = m - 1$ (we also set $c(\square) = 0$). If Γ is a conjunction of clauses, $\Gamma = C_1, \ldots, C_n$, then we set

$$c(\Gamma) = c(C_1) + \cdots + c(C_n).$$

We now give a recursive algorithm, `buildresol`, for constructing a resolution DAG from any set of clauses and then prove its correctness, namely, that if the input set of clauses is unsatisfiable, then the output resolution DAG is a resolution refutation. This establishes the completeness of propositional resolution constructively.

Our algorithm makes use of two functions, `percolate`, and `graft`.

1. The function `percolate`(D, A, L)

The inputs are: a resolution DAG, D, some selected leaf of D labeled with a clause, A, and some literal, L. This function adds the literal L to the clause A to form the clause $A \cup \{L\}$ and then "percolates" L down to the root of D. More precisely, we construct the resolution DAG, D', whose underlying unlabeled DAG is identical to D, as follows: Since D and D' have the same unlabeled DAG we refer to two nodes of D of D' as *corresponding nodes* if they are identical in the underlying unlabeled DAG. Consider any resolution step of D. If both parent clauses are not descendants of the premise A, then the corresponding resolution step of D' is the same. If the parent clauses in D are C and C' where C' is a descendant of the premise A (resp. C is a descendant of the premise A) and if R is the resolvent ot C and C' in D, then the corresponding parent nodes in D' are labeled with C and $C' \cup \{L\}$ and their resolvent node with $R \cup \{L\}$ (resp. the corresponding parent nodes in D' are labeled with $C \cup \{L\}$ and C' and their resolvent node wih $R \cup \{L\}$). If both parent clauses C and C' in D are descendant of the premise A, then the corresponding parent nodes in D' are labeled with $C \cup \{L\}$ and $C' \cup \{L\}$ and their resolvent node with $R \cup \{L\}$.

Observe that if $\Delta \cup \{A\}$ is the set of premises of D, then $\Gamma = \Delta \cup \{A \cup \{L\}\}$ is the set of premises of `percolate`(D, A, L).

For example, if D is the resolution DAG shown in Figure 2 (in fact, a resolution refutation)

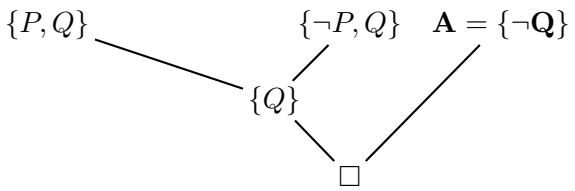

Figure 2: Resolution DAG D

then adding $L = \neg P$ to $A = \{\neg Q\}$ in D yields the resolution DAG D' produced by `percolate`(D, A, L) shown in Figure 3:

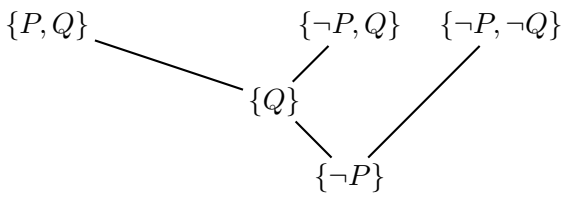

Figure 3: Resolution DAG $D' =$ `percolate`(D, A, L)

2. The function `graft`(D_1, D_2)

Its inputs are two resolution DAGs, D_1 and D_2, where the clause, C, labeling the root of D_1 is identical to one of the premises of D_2. Then, this function combines D_1 and D_2 by connecting the links to the premise labeled C in D_2 to the root of D_1, also labeled C, obtaining the resolution DAG $\texttt{graft}(D_1, D_2)$.

For example, if D_1 and D_2 are the resolution refutation DAGs shown in Figure 4 and Figure 5

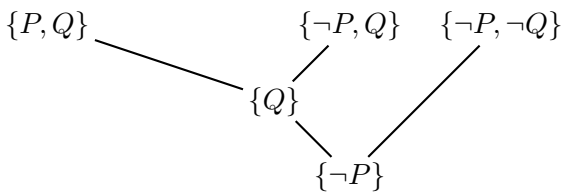

Figure 4: Resolution DAG D_1

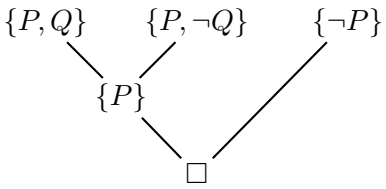

Figure 5: Resolution DAG D_2

we obtain the resolution DAG in Figure 6

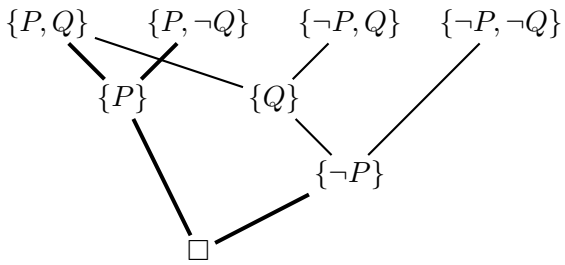

Figure 6: Resolution DAG $\texttt{graft}(D_1, D_2)$

where the edges coming from D_2 are indicated with thicker lines. The algorithm `buildresol` is shown below.

3. The algorithm `buildresol`(Γ)

The input to `buildresol` is a set of clauses, Γ.

function `buildresol`(Γ)
begin
if all clauses in Γ are literals **then**
  if Γ contains complementary literals L and $\neg L$,
  then return a resolution refutation with leaves L and $\neg L$
  else abort
  endif

else select any nonliteral clause, C, in Γ and select any literal, L, in C;
let $C = A \cup \{L\}$; let $\Gamma = \Delta \cup \{C\}$;
$D_1 = \texttt{buildresol}(\Delta \cup \{A\})$; $D_2 = \texttt{buildresol}(\Delta \cup \{L\})$; $D_1' = \texttt{percolate}(D_1, A, L)$;
  if D_1' is a resolution DAG
  then return D_1'
  else $D = \texttt{graft}(D_1', D_2)$; return D
  endif
endif
end

Finally, we prove the correctness of our recursive algorithm `buildresol`.

Theorem 3.1. *For every conjunction of clauses, Γ, if Γ is unsatisfiable, then the algorithm* `builresol` *outputs a resolution refutation for Γ. Therefore, propositional resolution is complete.*

Proof. We prove the correctness of the algorithm `buildresol` by induction on $c(\Gamma)$. Let $\Gamma = C_1, \ldots, C_n$. We may assume $\Gamma \neq \square$, since the case $\Gamma = \square$ is trivial. We proceed by induction on $c(\Gamma)$.

If $c(\Gamma) = 0$, then every clause, C_i, contains a single literal and if Γ is unsatisfiable, then there must be two complementary clauses, $C_i = \{P\}$ and $C_j = \{\neg P\}$, in Γ. Thus, we instantly get a resolution refutation by applying the resolution rule to $\{P\}$ and $\{\neg P\}$.

Otherwise, $c(\Gamma) > 0$, so there is some clause in Γ that contains at least two literals. Pick any such clause, C, and pick any literal, L, in C. Write $C = A \cup \{L\}$ with $A \neq \square$ and write $\Gamma = \Delta, C$ (Δ can't be empty since Γ is unsatisfiable). As $\Gamma = \Delta, A \cup \{L\}$ is unsatisfiable, *both* Δ, A and Δ, L must be unsatisfiable. However, observe that

$$c(\Delta, A) < c(\Gamma) \quad \text{and} \quad c(\Delta, L) < c(\Gamma).$$

Therefore, by the induction hypothesis, the algorithm `buildresol` produces two resolution refutations, D_1 and D_2, with sets of premises Δ, A and Δ, L, respectively. Now, consider the resolution DAG, $D_1' = \texttt{percolate}(D_1, A, L)$, obtained from D_1 by adding L to the clause A and letting L percolate down to the root.

Observe that in D_1', every clause that is a descendant of the premise $A \cup \{L\}$ is of the form $C \cup \{L\}$, where C is the corresponding clause in D_1. Therefore, the root of the new DAG D_1' obtained from D_1 is either labeled $\square$ (this may happen when the other clause in a resolution step involving a descendent of the clause A already contains L) or L. In the first case, D_1' is already a resolution refutation for Γ and we are done. In the second case, we can combine D_1' and D_2 using $\texttt{graft}(D_1', D_2)$ since the root of D_1' is also labeled L, one of the premises of D_2. Clearly, we obtain a resolution refutation for Γ. $\square$

As an illustration of our algorithm, consider the set of clauses

$$\Gamma = \{\{P, Q\}, \{P, \neg Q\}, \{\neg P, Q\}, \{\neg P, \neg Q\}\}$$

as above and pick $C = \{\neg P, \neg Q\}$, $L = \neg P$ and $A = \{\neg Q\}$. After the two calls $\texttt{buildresol}(\Delta \cup \{A\})$ and $\texttt{buildresol}(\Delta \cup \{L\})$, we get the resolution refutations D_1 shown in Figure 7:

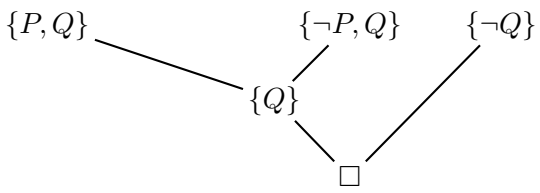

Figure 7: Resolution DAG $D_1 = \texttt{buildresol}(\Delta \cup \{A\})$

and D_2 shown in Figure 8:

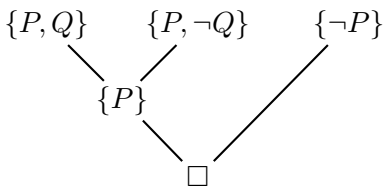

Figure 8: Resolution DAG $D_2 = \texttt{buildresol}(\Delta \cup \{L\})$

When we add $L = \neg P$ to $A = \{\neg Q\}$ in D_1, we get the resolution DAG $D_1' = \texttt{percolate}(D_1, A, L)$ shown in Figure 9:

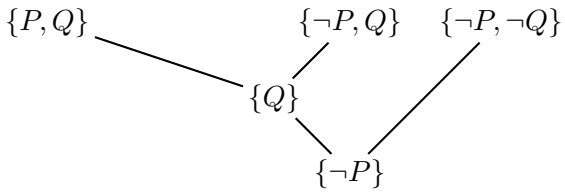

Figure 9: Resolution DAG $D_1' = \texttt{percolate}(D_1, A, L)$

Finally, we construct the resolution refutation $D = \texttt{graft}(D_1', D_2)$ shown in Figure 10:

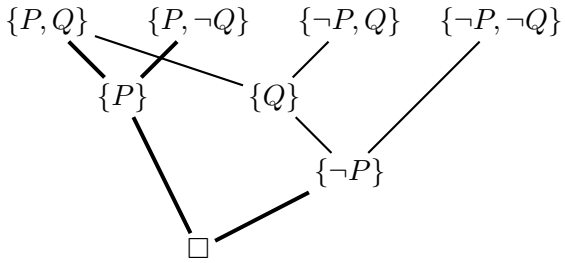

Figure 10: Resolution DAG $D = \texttt{graft}(D_1', D_2)$

where the edges coming from D_2 are indicated with thicker lines.

Observe that the proof of Theorem 3.1 proves that if Γ is unsatisfiable, then our algorithm succeeds no matter which clause containing at least two literals is chosen and no matter which literal is picked in such a clause.

Furthermore, as pointed out by one of the referees, although the proof of completeness is constructive in the sense that it shows an algorithm is correct, it does not explicitly use constructive logic. Nevertheless the logical proof can be recovered from the algorithm and it is constructive.

Acknowledgement

The author wishes to thank Robert Constable and the referees for very helpful comments.

References

[1] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, first edition, 1973.
[2] Jean H. Gallier. *Logic For Computer Science*. Wiley, first edition, 1986.
[3] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, first edition, 1981.
[4] J.A. Robinson. A machine oriented logic based on the resolution principle. *J.ACM*, 12(1):23–41, 1965.
[5] J.A. Robinson. *Logic: Form and Function*. North-Holland, first edition, 1979.
[6] Judith Underwood. The tableau algorithm for intuitionistic propositional calculus as a constructive completeness proof. In Basin D., Fronhofer B., Hahnle R., Posegga J., and Schwind C., editors, *Second Workshop on Theorem Proving with Analytic Tableaux and Related Methods, Marseille, France*, pages 245–248. Max–Planck–Institut fur Informatik, Saarbrucken, Germany, 1993.