St. Francis Xavier University
Department of Computer Science

CSCI 544: Computational Logic
Lecture 5: Propositional Logic IV—Resolution
Fall 2025

1 Testing (Un-)Satisfiability

In this lecture, we focus on another method for establishing the satisfiability or unsatisfiability of formulas in propositional logic. This method, known as *resolution*, is relatively recent compared to the other material we cover in this course: the first iteration of the method was introduced by the American mathematicians Martin Davis and Hilary Putnam in 1960, while a more refined and efficient version of the method was demonstrated by the English logician John Alan Robinson in 1965.

Resolution in propositional logic is useful as a decision procedure to test the unsatisfiability of a formula. Later, we will find that we're able to generalize resolution to a rather powerful and efficient algorithm (though, unfortunately, not a decision procedure) when we revisit the method in our study of predicate logic. Resolution is indeed such a powerful method that it's used as the basis of many logic programming languages and automated theorem provers.

2 Special Forms of Formulas

The method of resolution is particular in terms of the input it expects when trying to determine the satisfiability of a formula. A formula given as input must be in a certain form. This restriction is common when it comes to decision problems involving logical formulas. For example, if you've taken a course in theoretical computer science, you've likely heard of the Boolean satisfiability problem, which determines whether there exists a satisfying assignment of truth values to variables in a given Boolean formula. Like resolution, the Boolean satisfiability problem requires the input formula to be in a certain form.

Here, we will define the special forms of formulas that we will need in order to use the method of resolution, and we will show that we can convert any arbitrary formula into an equivalent formula in that special form.

2.1 Conjunctive Normal Form

Our first special form, the *conjunctive normal form*, is a restructuring of a formula such that the formula is divided into subformulas, with each subformula consisting of some number of literals joined by disjunction ($\vee$) connectives, and all subformulas are joined by conjunction ($\wedge$) connectives.

Definition 1 (Conjunctive normal form)**.** A formula is in conjunctive normal form, or CNF, if it consists of a conjunction of disjunctions of literals.

Example 2. The following formula is in conjunctive normal form:

$$(p \vee q \vee \neg r) \wedge (s \vee t) \wedge (\neg u)$$

In contrast, the following formulas are *not* in conjunctive normal form:

$$\begin{aligned} (p \wedge q) \vee r \quad & \text{(disjunction of conjunctions)} \\ p \wedge (q \vee (r \wedge s)) \quad & \text{(conjunction within clause)} \\ \neg(p \vee q) \wedge (r \vee s) \quad & \text{(negation outside of clause)} \end{aligned}$$

Fortunately, we need not worry if we have a formula that isn't in conjunctive normal form. As the following result will establish, it's possible to transform any arbitrary formula into an equivalent formula in conjunctive normal form.

Theorem 3. *Every propositional formula can be converted to an equivalent propositional formula in conjunctive normal form.*

Proof. We prove this theorem by constructing a method to transform formulas into their conjunctive normal form. This method makes use of the logical equivalences we established in an earlier lecture.

Given an arbitrary formula, apply the following steps in any appropriate order:

1. Remove all logical connectives from the formula except for negation, disjunction, and conjunction. For this, we use the following logical equivalences:

$$A \Rightarrow B \equiv \neg A \vee B$$
$$A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$$

2. Remove double negations. For this, we use the following identity:

$$\neg\neg A \equiv A$$

3. Move all negations inside clauses. For this, we use De Morgan's laws:

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

At this point, our formula now consists solely of a mixture of disjunctions and conjunctions of literals. To convert this formula into one consisting of conjunctions *of* disjunctions of literals, apply this final step.

4. Apply the distributive laws to the formula:

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$
$$(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$$

At this point, our formula is now in conjunctive normal form. ☐

Example 4. Consider the formula $A = \neg((\neg p \Rightarrow \neg q) \wedge \neg r)$. We can convert A to an equivalent formula in conjunctive normal form in the following way.

$$\begin{aligned}
\neg((\neg p \Rightarrow \neg q) \wedge \neg r) &\equiv \neg((\neg\neg p \vee \neg q) \wedge \neg r) && \text{(remove implication)} \\
&\equiv \neg((p \vee \neg q) \wedge \neg r) && \text{(remove double negation)} \\
&\equiv \neg(p \vee \neg q) \vee \neg\neg r && \text{(move negation inside clause)} \\
&\equiv \neg(p \vee \neg q) \vee r && \text{(remove double negation)} \\
&\equiv (\neg p \wedge \neg\neg q) \vee r && \text{(move negation inside clause)} \\
&\equiv (\neg p \wedge q) \vee r && \text{(remove double negation)} \\
&\equiv (\neg p \vee r) \wedge (q \vee r) && \text{(distributive law)}
\end{aligned}$$

2.2 Clausal Form

Having introduced the notion of conjunctive normal form, we can move on to discussing the *clausal form* of a formula, which is essentially nothing more than a variant of the conjunctive normal form. A formula in conjunctive normal form is a conjunction of disjunctions of literals. In the clausal form, our focus is placed on the *clauses* of the formula; that is, the sets corresponding to the disjunctions of literals.

To illustrate the correspondence between conjunctive normal form and clausal form, consider our earlier example formula, $(p \vee q \vee \neg r) \wedge (s \vee t) \wedge (\neg u)$. This formula contains three clauses: $\{p, q, \neg r\}$, $\{s, t\}$, and $\{\neg u\}$. We've seen clauses before when we discussed semantic tableaux, but there we just referred to them as "sets of literals".

If we represent a formula in terms of the clauses that compose it, then we say that the formula is in clausal form.

Definition 5 (Clausal form)**.** A formula is in clausal form if it consists of a set of clauses.

Observe that one "side effect" of converting from conjunctive normal form to clausal form is that we move from being able to represent formulas as trees to representing formulas as sets. This means that, in clausal form, we no longer need to concern ourselves with duplicate or identical literals, since any literal can only appear at most once in a set.

Example 6. Consider the formula $C = (p \vee q \vee r) \wedge (p \vee q \vee \neg r) \wedge (\neg p \vee q \vee r) \wedge (\neg p \vee \neg q \vee r)$ in conjunctive normal form. The equivalent formula written in clausal form is

$$\{\{p, q, r\}, \{p, q, \neg r\}, \{\neg p, q, r\}, \{\neg p, \neg q, r\}\}.$$

To save space, we can condense the way we write a clausal form by concatenating each literal together and denoting the negation of a literal using a bar:

$$\{pqr, pq\bar{r}, \bar{p}qr, \bar{p}\bar{q}r\}.$$

As before, we can show that it's possible to transform any arbitrary formula into an equivalent formula in clausal form.

Theorem 7. *Every propositional formula can be converted to an equivalent propositional formula in clausal form.*

Proof. The first step of the conversion process is to convert the given formula into an equivalent formula in conjunctive normal form. To do this, we use the process outlined in the proof of Theorem 3.

Using the equivalent formula in conjunctive normal form, transform each disjunction of literals into a clause and then transform the overall formula into a set of clauses. If a particular literal appears more than once in a given disjunction, then the same literal will appear exactly once in the corresponding clause as a consequence of the identity $A \vee A \equiv A$. Similarly, if a particular disjunction appears more than once in the formula, then the corresponding clause will appear exactly once in the corresponding set of clauses as a consequence of the identity $A \wedge A \equiv A$. □

Example 8. Consider the formula $D = (p \vee q) \wedge (\neg p \vee q \vee p) \wedge (q \vee p) \wedge (p \vee \neg p \vee q \vee p \vee q)$. The equivalent formula written in clausal form is

$$\{\{p, q\}, \{p, \neg p, q\}\}$$

or, alternatively,

$$\{pq, p\bar{p}q\}.$$

Observe that the first and third disjunctions in D are equivalent, so they only appear once in the clausal form as the set $\{p, q\}$. Similarly, the second and fourth disjunctions contain the same literals, so they only appear once in the clausal form as the set $\{p, \neg p, q\}$.

There are a couple of particular types of clauses for which we can establish some useful properties. One of these clause types appeared in the previous example: the clause $\{p, \neg p, q\}$. A clause with a pair of complementary literals, like p and $\neg p$, is called a *trivial clause.*

By one of our identities, $p \vee \neg p \equiv \mathrm{T}$, we know that every trivial clause is valid. Therefore, we can safely remove trivial clauses from our set of clauses without changing the truth value of the corresponding formula.

Theorem 9. *Let S be a set of clauses, and let $C \in S$ be a trivial clause. Then $S \setminus \{C\}$ is logically equivalent to S.*

Proof. Let $\mathscr{I}$ be an arbitrary interpretation for the set of clauses $S \setminus \{C\}$. The truth value $v_{\mathscr{I}}(S \setminus \{C\})$ is not changed by adding the clause C back to S, since $v_{\mathscr{I}}(C) = \mathrm{T}$ and $A \wedge \mathrm{T} \equiv A$ by one of our identities. Therefore, $v_{\mathscr{I}}(S \setminus \{C\}) = v_{\mathscr{I}}(S)$. Since $\mathscr{I}$ is arbitrary, we conclude that $S \setminus \{C\} \equiv S$. $\square$

Let's now move on to considering what it means for clauses and sets of clauses to be *empty*. An empty clause, denoted by $\square$, is simply a clause containing no literals. Similarly, an empty set of clauses, denoted $\emptyset$, is a formula containing no clauses.

Remark. Just like with empty words and empty languages in formal language theory, be careful not to confuse the notions of empty clauses and empty sets of clauses. In formal language theory, $\emptyset$ is the empty language, while $\{\epsilon\}$ is the nonempty language containing the empty word. Likewise, $\emptyset$ is the empty set of clauses, while $\{\square\}$ is the nonempty set of clauses containing the empty clause.

We can draw the following conclusions about empty clauses and empty sets of clauses.

Theorem 10. *The empty clause $\square$ is unsatisfiable.*

Proof. Let $\mathscr{I}$ be an arbitrary interpretation. A clause is satisfiable if and only if there exists some interpretation where at least one literal in the clause is true. Since the empty clause $\square$ contains no literals, there exist no literals that evaluate to true under the interpretation $\mathscr{I}$. Therefore, $\square$ is unsatisfiable. $\square$

Theorem 11. *The empty set of clauses $\emptyset$ is valid.*

Proof. A set of clauses is valid if and only if every clause in the set evaluates to true under any interpretation. However, the empty set of clauses $\emptyset$ contains no clauses, and so there are no clauses that need to evaluate to true. Therefore, $\emptyset$ is valid. $\square$

Theorem 10 in particular will be of great use when we define our method of resolution in the following section.

Lastly, there are a number of nice properties that allow us to simplify a formula in clausal form. We summarize these properties in the following proposition, and give examples of each property in use.

Proposition 12. *Let ℓ be a literal, and denote the complement of ℓ by ℓ^C. Suppose S is a formula in clausal form. Then:*

1. *If ℓ appears in some clause of S, but ℓ^C does not appear in any clause, then the new clausal form S' obtained by removing all clauses in S containing ℓ is satisfiable if and only if S is satisfiable;*
2. *If $C = \{\ell\}$ is a unit clause, then the new clausal form S' obtained by removing C and all clauses containing ℓ^C from S is satisfiable if and only if S is satisfiable;*
3. *If S contains two clauses C and C' where $C \subseteq C'$, then the new clausal form S' obtained by removing C' from S is satisfiable if and only if S is satisfiable; and*
4. *If a clause C in S contains both ℓ and ℓ^C, then the new clausal form S' obtained by removing C from S is satisfiable if and only if S is satisfiable.*

Example 13. We consider examples of each of the properties from Proposition 12 in use.

1. Let $S = \{pq\bar{r}, p\bar{q}, \bar{p}q\}$. The literal r appears in the first clause, but $\bar{r}$ appears nowhere in S. Therefore, $S' = \{p\bar{q}, \bar{p}q\}$ is an equivalent clausal form.
2. Let $S = \{p, \bar{p}q\bar{r}, \bar{q}q\bar{r}, \bar{p}q\}$. The literal p forms a unit clause, so we can remove the unit clause p and the literal $\bar{p}$ from all other clauses to get $S' = \{q\bar{r}, \bar{q}q\bar{r}, q\}$, which is an equivalent clausal form.

3. Let $S = \{p, \bar{p}q\bar{r}, \bar{q}q\bar{r}, \bar{p}q\}$. The clause $\bar{p}q$ is "contained in" the clause $\bar{p}q\bar{r}$, so we can remove the latter clause to get $S' = \{p, \bar{q}q\bar{r}, \bar{p}q\}$, which is an equivalent clausal form.
4. Let $S = \{p, \bar{p}q\bar{r}, \bar{q}q\bar{r}, \bar{p}q\}$. The clause $\bar{q}q\bar{r}$ contains the complementary literals q and $\bar{q}$, so we can remove that clause to get $S' = \{p, \bar{p}q\bar{r}, \bar{p}q\}$, which is an equivalent clausal form.

3 Resolution

Given a formula in clausal form, we can finally define the method of resolution to determine whether the formula is unsatisfiable. The method of resolution is known as a *refutation procedure*, meaning that it is a procedure that, given a contradictory set of clauses, is able to derive a contradiction. Refutation procedures are akin to proofs by contradiction, in a sense: if we wish to prove $A \vdash B$, for example, then this is equivalent to proving $A, \neg B \vdash \bot$, and we can use resolution on the set $\{A, \neg B\}$ to establish our desired result.

The method of resolution relies on one principle known as the *resolution rule*, stated as follows.

Resolution rule. Suppose C_1 and C_2 are clauses where $\ell \in C_1$ and $\ell^{\mathrm{C}} \in C_2$ for some literal ℓ. In this case, C_1 and C_2 are said to clash on the complementary pair of literals ℓ and ℓ^{C}. From this, we can determine the resolvent of C_1 and C_2,

$$\mathrm{Res}(C_1, C_2) = (C_1 \setminus \{\ell\}) \cup (C_2 \setminus \{\ell^{\mathrm{C}}\}).$$

Where it is clear, we simply refer to $\mathrm{Res}(C_1, C_2)$ as C.

What the resolution rule tells us is that, if some pair of clauses together contains a complementary pair of literals, we can compute the resolvent of those clauses by taking the union of all literals in those clauses excluding the complementary pair. We're able to remove the complementary pair because of the fact that this pair forms a trivial clause, and by Theorem 9, removing a trivial clause from a set of clauses S produces a logically equivalent set of clauses S'.

Example 14. Suppose we have the clauses $C_1 = \bar{p}q\bar{r}$ and $C_2 = p\bar{q}$. These two clauses together contain the complementary pair of literals p and $\bar{p}$. We compute the resolvent of C_1 and C_2 to be

$$\mathrm{Res}(C_1, C_2) = q\bar{r} \cup \bar{q} = q\bar{r}\bar{q}.$$

Since the clauses C_1 and C_2 similarly contain the complementary set of literals q and $\bar{q}$, we could have alternatively computed the resolvent to be

$$\mathrm{Res}(C_1, C_2)' = \bar{p}\bar{r} \cup p = \bar{p}\bar{r}p.$$

Observe that the resolution rule maintains the satisfiability of a formula: if a formula is satisfiable before an application of the resolution rule, it will remain satisfiable after applying the resolution rule.

Theorem 15. *Clauses C_1 and C_2 are satisfiable if and only if the resolvent $C = \mathrm{Res}(C_1, C_2)$ is satisfiable.*

Proof. ($\Rightarrow$): Suppose that clauses C_1 and C_2 are both satisfiable, and suppose $\mathscr{I}$ is an interpretation where this is the case. Suppose ℓ and ℓ^{C} are the pair of complementary literals in C_1 and C_2. Then we must have that either

- $v_{\mathscr{I}}(\ell) = \mathrm{T}$ and $v_{\mathscr{I}}(\ell^{\mathrm{C}}) = \mathrm{F}$; or
- $v_{\mathscr{I}}(\ell) = \mathrm{F}$ and $v_{\mathscr{I}}(\ell^{\mathrm{C}}) = \mathrm{T}$.

Suppose the first case is true. Then $v_{\mathscr{I}}(\ell) = \mathrm{T}$, and C_2 can be satisfied only if $v_{\mathscr{I}}(\ell') = \mathrm{T}$ for some other literal $\ell' \neq \ell^{\mathrm{C}}$. Since this literal ℓ' still appears in $C = \mathrm{Res}(C_1, C_2)$, the resolvent clause will be satisfied. The second case is proved in a similar manner.

($\Leftarrow$): Suppose that the resolvent $C = \mathrm{Res}(C_1, C_2)$ is satisfiable, and suppose $\mathscr{I}$ is an interpretation where this is the case. Then we have that $v_{\mathscr{I}}(\ell') = \mathrm{T}$ for at least one literal $\ell' \in C$. By the resolution rule, ℓ' must have appeared in either C_1 or C_2 (or both). If $\ell' \in C_1$, then $v_{\mathscr{I}}(C_1) = \mathrm{T}$, and we can extend the interpretation $\mathscr{I}$ to admit the truth value $v_{\mathscr{I}}(\ell^{\mathrm{C}}) = \mathrm{T}$ and ensure that $v_{\mathscr{I}}(C_2) = \mathrm{T}$. The case where $\ell' \in C_2$ is analogous. $\square$

If we iteratively apply the resolution rule to a given formula in clausal form, then we will end up in one of two situations: either we will obtain a clausal form to which we cannot apply the resolution rule anymore, or we will obtain the empty clause $\square$. Since we know that the empty clause is unsatisfiable by Theorem 10, if we end up in the latter situation, we can conclude that the original formula must have also been unsatisfiable. In this case, we say that the derivation of the empty clause $\square$ from the original set of clauses S is a *resolution refutation* of S.

Our technique for performing the method of resolution, given an input formula S in clausal form, is summarized in the following algorithm.

Algorithm 1: Method of resolution

```
$S_0 \leftarrow S$
$i \leftarrow 0$
**while** $S_i$ contains a pair of clashing clauses **do**        ▷ I.e., while we can apply the resolution rule
    $C_1, C_2 \leftarrow$ some pair of clashing clauses in $S_i$
    $C \leftarrow \text{Res}(C_1, C_2)$
    **if** $C = \square$ **then**
        **return** $S$ is not satisfiable
    $S_{i+1} \leftarrow S_i \cup \{C\}$
    **if** $S_{i+1} = S_i$ **then**
        **return** $S$ is satisfiable
    $i \leftarrow i + 1$
```

Example 16. Let's use our algorithm to determine whether the set of clauses $S = \{p, r, \bar{s}, \bar{p}q, \bar{q}\bar{r}s\}$ is satisfiable.

On the first iteration, we set $S_0 = S = \{p, r, \bar{s}, \bar{p}q, \bar{q}\bar{r}s\}$. This set contains the pair of clashing clauses p and $\bar{p}q$, so we apply the resolution rule to get $C = \{q\}$. Then, we compute $S_1 = S_0 \cup C = \{p, r, \bar{s}, \bar{p}q, \bar{q}\bar{r}s, q\}$.

Next, S_1 contains the pair of clashing clauses q and $\bar{q}\bar{r}s$, so we apply the resolution rule to get $C = \{\bar{r}s\}$. Then, we compute $S_2 = S_1 \cup C = \{p, r, \bar{s}, \bar{p}q, \bar{q}\bar{r}s, q, \bar{r}s\}$.

Again, S_2 contains the pair of clashing clauses r and $\bar{r}s$, so we apply the resolution rule to get $C = \{s\}$. Then, we compute $S_3 = S_2 \cup C = \{p, r, \bar{s}, \bar{p}q, \bar{q}\bar{r}s, q, \bar{r}s, s\}$.

Once again, S_3 contains the pair of clashing clauses s and $\bar{s}$, so we apply the resolution rule to get $C = \square$. Since this resolvent is the empty clause, we conclude that the original set of clauses is not satisfiable.

Example 17. We will use the method of resolution to show that the formula $(p \Rightarrow q) \Rightarrow (\neg q \Rightarrow \neg p)$ is valid. To establish the validity of our original formula, we will show that the negation of the formula, $\neg((p \Rightarrow q) \Rightarrow (\neg q \Rightarrow \neg p))$, is unsatisfiable.

First, we must convert the negated formula into an equivalent formula in conjunctive normal form. We do so in the following way:

$$\begin{aligned}
\neg((p \Rightarrow q) \Rightarrow (\neg q \Rightarrow \neg p)) &\equiv (p \Rightarrow q) \wedge \neg(\neg q \Rightarrow \neg p) && \text{(remove implication)} \\
&\equiv (\neg p \vee q) \wedge \neg(q \vee \neg p) && \text{(remove implication)} \\
&\equiv (\neg p \vee q) \wedge \neg q \wedge \neg\neg p && \text{(move negation inside clause)} \\
&\equiv (\neg p \vee q) \wedge \neg q \wedge p && \text{(remove double negation)}
\end{aligned}$$

Next, we use the method of resolution to show that the negated formula is unsatisfiable. The negated formula, in clausal form, is $S = \{\bar{p}q, \bar{q}, p\}$.

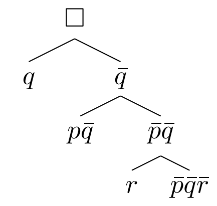
We begin by taking $S_0 = S = \{\bar{p}q, \bar{q}, p\}$. This set contains the pair of clashing clauses $\bar{q}$ and $\bar{p}q$, so we apply the resolution rule to get $C = \{\bar{p}\}$. Then, we compute $S_1 = S_0 \cup C = \{\bar{p}q, \bar{q}, p, \bar{p}\}$.

Next, S_1 contains the pair of clashing clauses p and $\bar{p}$, so we apply the resolution rule to get $C = \square$. Since this resolvent is the empty clause, we conclude that the original set of clauses is not satisfiable.

Altogether then, since the negated formula is unsatisfiable via resolution refutation, we can conclude that the original formula must be valid.

Before we continue, we make one last observation about the method of resolution. Just as we saw with semantic tableaux, we can write a proof using resolution either as a list of sets of clauses or as a tree. The benefit of using a tree structure is that we're able to present the connection between two clauses and their resolvent in terms of parent/child relationships and levels. It is worth noting, of course, that this *resolution tree* is distinct from a semantic tableau, and the two notions are not related apart from both having a tree structure.

Example 18. Consider the set of clauses $S = \{q, p\bar{q}, r, \bar{p}\bar{q}\bar{r}\}$. Following the method of resolution, we will eventually end up deriving $\Box$ and producing a resolution refutation of S. Instead of writing out this resolution refutation as a list, however, we can represent it as the following tree, where the parent of two vertices is the resolvent of the clauses labelling those two vertices:



4 Soundness and Completeness

As before, we conclude by establishing the soundness and completeness of the method of resolution. With our notion of a resolution tree, it's quite straightforward to formulate our soundness result:

Theorem 19 (Soundness of resolution)**.** *If the set of clauses labelling the leaves of some resolution tree is satisfiable, then the clause labelling the root of the resolution tree is satisfiable.*

Proof. Follows by induction using Theorem 15. $\Box$

However, this formulation of the soundness result is phrased in terms of satisfiability, and using this formulation makes the completeness result difficult to prove as the converse of Theorem 19 is not always true.

The method of resolution, being a refutation procedure, is better framed through the lens of unsatisfiability. Thus, for that reason, we will phrase and prove our soundness and completeness results in terms of unsatisfiability.

4.1 Proving Soundness

To establish soundness, we must show that any set of clauses having a resolution refutation must be unsatisfiable. Proving the soundness of the method of resolution is extremely easy. In fact, soundness is an immediate consequence of two of our earlier results!

Theorem 19 (Soundness of resolution—reformulated)**.** *Given a set of clauses S, if there exists a resolution refutation of S, then S is unsatisfiable.*

Proof. Follows from Theorem 10 and from the "satisfiable" formulation of soundness of resolution. $\Box$

4.2 Proving Completeness

With completeness, on the other hand, the usual situation arises where proving the converse direction requires much more work. In this case, the converse direction has us proving that any unsatisfiable set of clauses has a corresponding resolution refutation. Let's begin by formulating our completeness result.

Theorem 20 (Completeness of resolution)**.** *Given a set of clauses S, if S is unsatisfiable, then there exists a resolution refutation of S where the empty clause $\square$ is derived via the method of resolution.*

The essence of our proof of completeness will be to show that, starting with an unsatisfiable set of clauses S, the method of resolution will eventually halt and produce the empty clause $\square$. We know that the method will halt because there can be only a finite number of distinct clauses formed by the finite number of atomic propositions that appear in S.

To build the machinery of our proof, we require a new notion: that of a *semantic tree*. With a semantic tree, we are able to store truth values corresponding to the atomic propositions of a formula as we search for a satisfying interpretation for that formula. Note that, like resolution trees, semantic trees are distinct from semantic tableaux and the two structures should not be confused (despite their similar names).

Definition 21 (Semantic tree)**.** Let S be a set of clauses, and let $P_S = \{p_1, p_2, \ldots, p_n\}$ be the set of atomic propositions appearing in clauses in S. The semantic tree for S, denoted T, is a complete binary tree of depth n where, for all $1 \leq i \leq n$, the left-branching edge from a vertex at depth $(i-1)$ is labelled by p_i and the right-branching edge is labelled by $\bar{p}_i$.

Following this construction, each branch from the root of a semantic tree T to some leaf corresponds to some sequence $\{\ell_1, \ell_2, \ldots, \ell_n\}$ where $\ell_i \in \{p_i, \bar{p}_i\}$. Moreover, for each branch b of T, we can define an interpretation $\mathscr{I}_b$ by taking

$$\mathscr{I}_b(p_i) = \text{T if } \ell_i = p_i; \text{ and}$$
$$\mathscr{I}_b(p_i) = \text{F if } \ell_i = \bar{p}_i.$$

Then, we say that a branch b is *closed* if $\mathscr{I}_b(S) = \text{F}$, and that b is *open* otherwise. Similarly, the overall semantic tree T is closed if all branches are closed, and T is open otherwise.

With these notions in mind, it is possible to show the following connection between semantic trees and the satisfiability of a set of clauses.

Proposition 22. *The semantic tree T for a set of clauses S is closed if and only if S is unsatisfiable.*

As we traverse a semantic tree, we are able to compute a partial interpretation of truth values assigned to the atomic propositions found along the already-traversed edges. In some cases, the truth value of a clause may be false, which would render the entire set of clauses false in that partial interpretation.

If we construct a semantic tree for a given set of clauses S, then any clauses created during a resolution refutation of S will be located at certain vertices of the tree that we will call *failure vertices*. In essence, a failure vertex corresponds to an assignment of truth values that renders the associated clauses false. In a closed semantic tree, there must exist a failure vertex along every branch of the tree.

Definition 23 (Failure vertex)**.** Let T be a closed semantic tree for a set of clauses S, and consider any branch b in T. The vertex in b closest to the root of T that renders S false is called a failure vertex.

Furthermore, we say that the clause rendered false by some failure vertex is the clause *associated with* that failure vertex. It is possible to have more than one clause associated with the same failure vertex, and we can in fact characterize the clauses that are associated with any failure vertex via the following proposition.

Proposition 24. *Let n be a failure vertex in a semantic tree T. A clause C associated with n consists of a subset of the complements of the atomic propositions appearing along the branch b from the root of T to n.*

Going one step further, if there exists some vertex in a semantic tree whose children are both failure vertices, then we say that vertex is an *inference vertex*.

Definition 25 (Inference vertex)**.** Let n be some vertex in a semantic tree T. Then n is an inference vertex if and only if both children of n are failure vertices.

By this definition, we know that a semantic tree containing a certain number of failure vertices must necessarily contain at least one inference vertex.

Proposition 26. *Let T be a closed semantic tree for a set of clauses S. If there exists at least two failure vertices in T, then it must be the case that there exists at least one inference vertex.*

In the last step we need before proving completeness, we draw a connection between these failure/inference vertices and our earlier notion of a resolvent.

Proposition 27. *Let T be a semantic tree, let n be an inference vertex in T, and let n_1 and n_2 each be failure vertices that are children of n. Furthermore, suppose clauses C_1 and C_2 are associated with n_1 and n_2, respectively. Then the following two statements hold:*

1. *Clauses C_1 and C_2 clash, and the partial interpretation corresponding to the branch from the root of T to n renders $\text{Res}(C_1, C_2)$ false.*
2. *Let $C = \text{Res}(C_1, C_2)$. Then $S \cup \{C\}$ has a failure vertex that is either n itself or some ancestor vertex of n, and C is a clause associated with that failure vertex.*

Finally, we can prove completeness by showing that repeated applications of the resolution rule will lead to the method of resolution eventually halting.

Proof of completeness. If S is an unsatisfiable set of clauses, then there exists a closed semantic tree T for S. If S is unsatisfiable and does not already contain the empty clause $\square$, then there must exist at least two failure vertices in T, and so there necessarily exists at least one inference vertex in T by Proposition 26.

Applying the resolution rule to this inference vertex adds the resolvent $C = \text{Res}(C_1, C_2)$ to the set of clauses, which adds a failure vertex to T and removes two other failure vertices from T by the second statement of Proposition 27. Thus, the total number of failure vertices of T is reduced by one upon each application of the resolution rule. When, eventually, the total number of failure vertices in T is exactly one, this failure vertex must correspond to the root of T, and the root must be associated with the derived empty clause $\square$. $\square$