The Resolution Proof System in Propositional Logic

While the Hilbert proof system has the property that it reflects the methods by which humans commonly do mathematics, it is ill suited for use in automated reasoning, because of the infinite number of axiom which may be generated from an axiom schema.

The resolution proof system is the most successful attempt to provide an alternate system which is more suited to the task of automated reasoning by computer.

The number of semantically distinct formulas:

Call two wff's φ_1 and φ_2 semantically distinct if $\models (\neg(\varphi_1 \equiv \varphi_2))$ holds.

Observation: Let L be a propositional logic with a finite number n of distinct proposition names $P = \{A_1, A_2, .., A_n\}$. Then there are 2^{2^n} semantically distinct wff's over L.
Proof: The truth table for such a wff has 2^n rows, and the entry in for each row may be 0 or 1. □

Despite this, observe that WF(L) is an infinite set, even if there is only one proposition name.

- If we want a proof system which admits algorithmic construction of proofs, it would be a great advantage to have just a finite number of formulas. (In view of the above result, only 2^{2^n} are needed.)

Clauses:

Definition: A clause is a wff which is a disjunction of literals.

Examples:
- $(A_1 \vee \neg A_2 \vee A_3 \vee \neg A_4)$ is a clause.
- $(\neg A_1 \vee \neg A_3 \vee A_3 \vee \neg A_4)$ is a clause which is always true.
- $\perp$ is taken to be a clause of with no literals, and is always false.
  Motivation: A clause is true iff at least one of its literals is true. Since $\perp$ contains no literals, it must always be false.

Observation: Let L be a propositional logic having a finite number n of distinct proposition names. Then the number of distinct clauses over L is finite, the exact number being 4^n.
Proof: For each proposition name A, there are four possibilities for a clause φ:
- A is a disjunct of φ.
- $(\neg A)$ is a disjunct of φ.
- Both A and $(\neg A)$ are disjuncts of φ.
- Neither A nor $(\neg A)$ are disjuncts of φ.

From this observation, the result follows immediately. □

From a practical point of view, a clause which contains a complementary pair of literals is uninteresting, because it is always true.

Call a clause *trivial* if it contains a complementary pair of literals. Clearly, a trivial clause is true in any interpretation, and thus all such clauses are semantically equivalent.

Observation: Let L be a propositional logic a finite number n of distinct proposition names. Then the number of distinct nontrivial clauses over L is finite, the exact number being 3^n.
Proof: Similar to the above with the possibility that both A and $(\neg A)$ being disjuncts of the same clause excluded. □

Remark: It is probably useful to keep one clause around which is always true, in which case the number of interesting clauses will be $3^n + 1$.

Resolution:
Definition: Resolution is the following proof rule:

$$\frac{(\alpha_1 \vee p),\ (\alpha_2 \vee \neg p)}{(\alpha_1 \vee \alpha_2)}$$

In this rule, p is bound to a proposition name, and α_1 and α_2 are to be bound to clauses.

Note that the result of resolution is also a clause, called the *resolvent* of the components from which it is built. More formally, if $\varphi_1 = \psi_1 \vee \ell_1$ and $\varphi_2 = \psi_2 \vee \ell_2$ are clauses with $\{\ell_1, \ell_2\}$ a complementary pair of literals, then $\psi_1 \vee \psi_2$ is called the resolvent of (φ_1, φ_2) with respect to (ℓ_1, ℓ_2). The set of all resolvents of $\{\varphi_1, \varphi_2\}$ is denoted $\text{Res}(\varphi_1, \varphi_2)$.

Example: Let $\varphi_1 = A \vee \neg B$ and $\varphi_2 = \neg A \vee B$. Then $\text{Res}(\varphi_1, \varphi_2) = \{ A \vee \neg A, B \vee \neg B\}$. Note that only one pair of complementary literals may be deleted. $\perp \notin \text{Res}(\varphi_1, \varphi_2)$.

Example: Let $\varphi_1 = A \vee \neg B$ and $\varphi_2 = \neg A$. Then $\text{Res}(\varphi_1, \varphi_2) = \{\neg B \}$.

Example: Let $\varphi_1 = A$ and $\varphi_2 = \neg A$. Then $\text{Res}(\varphi_1, \varphi_2) = \{\perp\}$.

Example: Let $\varphi_1 = A \vee \neg B$ and $\varphi_2 = A \vee \neg C$. Then $\text{Res}(\varphi_1, \varphi_2) = \varnothing$; there are no resolvents. Note carefully that $\text{Res}(\varphi_1, \varphi_2) = \{\perp\}$ and $\text{Res}(\varphi_1, \varphi_2) = \varnothing$ do not mean the same thing.

Example: Let $\varphi_1 = A \vee \neg B \vee C$ and $\varphi_2 = \neg A \vee \neg B \vee C$. Then $\text{Res}(\varphi_1, \varphi_2) = \{\neg B \vee C\}$.

Proposition: Resolution is sound. That is, if φ_1 and φ_2 are clauses, and $\varphi \in \text{Res}(\varphi_1, \varphi_2)$, then $\{\varphi_1, \varphi_2\} \models \varphi$.
Proof: Without loss of generality, suppose that $\varphi_1 = \psi_1 \vee A$ and $\varphi_2 = \psi_2 \vee \neg A$, with A a proposition name. Let $v \in \text{Mod}(\{\varphi_1, \varphi_2\})$. Then either $v \in \text{Mod}(\psi_1)$ or else $v \in \text{Mod}(\psi_2)$, since $\tilde{v}$ cannot be true on both A and $\neg A$. Thus, $\tilde{v} \in \text{Mod}(\psi_1 \vee \psi_2)$, as was to be proved. □

Resolution also has a completeness property, but discusion of it will be deferred until use of this powerful tool has been examined more completely.

Proofs using resolution:

The resolution proof rule defines a proof system **Res** in which there are no axiom schemata, and only one proof rule, resolution. Since the resolution proof rule operates only on clauses, the whole proof system operates only on them.

Since this system is so important, it is worth writing out the definition of a proof in detail.

A *proof* in **Res** of the clause φ from the set of clauses Φ is a sequence $\varphi_1, \varphi_2, .., \varphi_n$ of clauses, with the following properties.
(a) Each φ_i is either:
  (i) A member of Φ;
  (ii) A member of $\text{Res}(\varphi_j, \varphi_k)$, where $1 \le j, k < i$.
(b) $\varphi_n = \varphi$.

$\Phi \vdash_{\text{Res}} \varphi$ denotes that φ is provable from Φ in **Res**.

Notice how much simpler this definition is than the corresponding one for the Hilbert system.

Example: Let
$\varphi_1 = A \vee B \vee \neg D$
$\varphi_2 = A \vee B \vee C \vee D$
$\varphi_3 = \neg B \vee C$
$\varphi_4 = \neg A$
$\varphi = C$

The task is to show that $\{\varphi_1, \varphi_2, \varphi_3, \varphi_4\} \vdash_{\text{Res}} \varphi$.

1. $A \vee B \vee \neg D$ [hypothesis φ_1]

2. $A \vee B \vee C \vee D$ [hypothesis φ_2]

3. $A \vee B \vee C$ [res. on 1, 2 with $\neg D$, D]

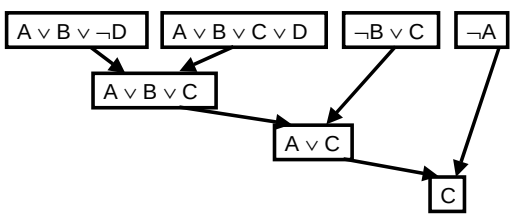
4. $\neg B \vee C$ [hypothesis φ_3]

5. $A \vee C$ [res. on 3, 4 with B, $\neg B$]

6. $\neg A$ [hypothesis φ_4]

7. C [res. on 5, 6 with A, $\neg A$]

This proof may also be represented graphically:



For direct inference, resolution is not complete, even when the goal is a simple clause.

Example: Let $\varphi_1 = A$ and let $\varphi = A \vee B$. Then it is clear that $\{\varphi_1\} \models \varphi$, yet $\varphi \notin Res(\varphi_1, \varphi_2)$.

However, resolution is complete for refutation, *i.e.,* proofs of $\perp$. To solve the above example, negate the goal and add it to the hypotheses. The negated goal is $\neg(A \vee B)$, which is not a clause. However, a simple application de Morgan's identity yields

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

$(\neg A \wedge \neg B)$ is not a clause, but it may be decomposed into two clauses, $\neg A$ and $\neg B$. The problem of establishing that

$$\{A\} \models A \vee B$$

is thus reduced to the problem of establishing that

$$\{A, \neg A, \neg B\} \models \perp.$$

The latter is a trivial resolution.

1. A [hypothesis]

2. $\neg A$ [hypothesis]

3. $\perp$ [res. 1, 2]

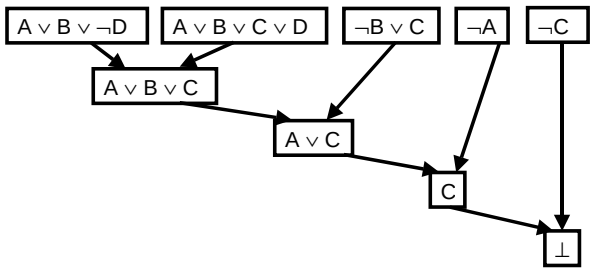
The previous proof may be converted similarly.

Example: Let

$\varphi_1 = A \vee B \vee \neg D$
$\varphi_2 = A \vee B \vee C \vee D$
$\varphi_3 = \neg B \vee C$
$\varphi_4 = \neg A$
$\varphi = C$

The task is show that $\{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \neg\varphi\} \vdash_{Res} \perp$

1. $A \vee B \vee \neg D$	[hypothesis φ_1]
2. $A \vee B \vee C \vee D$	[hypothesis φ_2]
3. $A \vee B \vee C$	[res. on 1, 2 with $\neg D$, D]
4. $\neg B \vee C$	[hypothesis φ_3]
5. $A \vee C$	[res. on 3, 4 with B, $\neg B$]
6. $\neg A$	[hypothesis φ_4]
7. C	[res. on 5, 6 with A, $\neg A$]
8. $\neg C$	[hypothesis $\neg\varphi$]
9. $\perp$	[res. on 7, 8]



Adequacy of clauses and CNF:

- In problem solving, it will not always be the case that the hypotheses and the conclusion will be clauses.
- For resolution to be viewed as a general procedure, it must be possible to begin with hypotheses and goals expressed as arbitrary wff's.
- The solution is to use a normal form called conjunctive normal form (CNF). Every wff may be converted to one in CNF.

It is not the case that every wff is equivalent to a clause. For example, there is no clause which is equivalent to the wff $(A \wedge B)$. However, there is a result which turns out to be just as useful.

Definition: A wff φ is said to be in *conjunctive normal form* (CNF) if it is of the form $\varphi_1 \wedge \varphi_2 \wedge .. \wedge \varphi_k$, with each φ_i a clause.

Example: The formula

$$\varphi = (A_1 \vee \neg A_2 \vee A_3 \vee \neg A_4) \wedge (\neg A_1 \vee \neg A_3 \vee A_3 \vee \neg A_4)$$

is in CNF.

Fact: Every wff is semantically equivalent to a formula in CNF. In fact, there is an algorithm which will convert an arbitrary wff into one in CNF.
Proof: The proof is very similar to that for conversion to DNF. The key difference is that, instead of using the distributive identity which distributes $\wedge$ over $\vee$, we use the dual, which distributes $\vee$ over $\wedge$. Formally, the following steps need to be taken.

Step 1: Eliminate $\equiv$.
Step 2: Eliminate $\rightarrow$.
Step 3: Use de Morgan's laws to move $\neg$'s in to atoms.
Step 4: Eliminate double negatives (implicit).
Step 5: Distribute $\vee$ over $\wedge$. $\square$

Example:
$\neg((A_1 \equiv A_2) \wedge \neg(A_3 \vee \neg\ (A_4 \rightarrow A_1)))$

Steps 1 and 2:
$\neg(((A_1 \wedge A_2) \vee (\neg A_1 \wedge \neg A_2))) \wedge$
$\neg(A_3 \vee \neg(\neg A_4 \vee A_1)))$

Note: Rather than substituting
$((A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1))$
for $(A_1 \equiv A_2)$, it is more productive to use
$((A_1 \wedge A_2) \vee (\neg A_1 \wedge \neg A_2))$.

Steps 3 and 4:
$\neg((A_1 \wedge A_2) \vee (\neg A_1 \wedge \neg A_2)) \vee (\neg A_3 \wedge (\neg A_4 \vee A_1))$

$(\neg(A_1 \wedge A_2) \wedge \neg(\neg A_1 \wedge \neg A_2)) \vee (A_3 \vee\ (A_4 \wedge \neg A_1))$

$((\neg A_1 \vee \neg A_2) \wedge (A_1 \vee A_2) \vee (A_3 \vee\ (A_4 \wedge \neg A_1))$

Step 5:
$((\neg A_1 \vee \neg A_2) \wedge (A_1 \vee A_2)) \vee ((A_3 \vee A_4) \wedge (A_3 \vee \neg A_1))$

$(\neg A_1 \vee \neg A_2 \vee A_3 \vee A_4) \wedge (\neg A_1 \vee \neg A_2 \vee A_3 \vee \neg A_1) \wedge$
$(A_1 \vee A_2 \vee A_3 \vee A_4) \wedge (A_1 \vee A_2 \vee A_3 \vee \neg A_1)$

Simplify:
$(\neg A_1 \vee \neg A_2 \vee A_3 \vee A_4) \wedge (\neg A_1 \vee \neg A_2 \vee A_3) \wedge$
$(A_1 \vee A_2 \vee A_3 \vee A_4)$

Problem solving using resolution:

Recall that, for direct inference, resolution is not complete, even when the goal is a simple clause.

Example: Let $\varphi_1 = A$ and $\varphi_2 = B$. Let $\varphi = A \vee B$. Then it is clear that $\{\varphi_1, \varphi_2\} \models \varphi$, yet $\varphi \notin \text{Res}(\varphi_1, \varphi_2)$.

However, resolution is complete when the goal is the empty clause $\perp$. If $\{\varphi_1, \varphi_2, .., \varphi_n\}$ is a finite set of clauses, then $\{\varphi_1, \varphi_2, .., \varphi_n\} \models \perp$ iff there is a sequence of resolutions which may be applied to $\{\varphi_1, \varphi_2, .., \varphi_n\}$ to yield the empty clause. This will be made more precise, and the result proved, later. For now we look at how some examples are solve.

Example: Recall that a proof of $\models (\varphi \rightarrow \varphi)$ within the Hilbert system was surprisingly difficult. In the resolution system, it is trivial. $(\varphi \rightarrow \varphi)$ is equivalent to $(\neg\varphi \vee \varphi)$. To prove the validity of this formula, convert its negation to CNF: $\neg\ (\neg\varphi \vee \varphi)$ reduces to $(\varphi \wedge \neg\varphi)$, which is represented by the set $\{\varphi, \neg\varphi\}$ of clauses. It is trivial to resolve this set to $\perp$.

Example: We return to the party example, which is Example 2.4 of the textbook. The premises of the problem consist of the following three statements.
$\Phi = \{(J \vee Y), (Y \rightarrow (\neg S \rightarrow C)), (\neg J \rightarrow S)\}$.
The conclusion is the single proposition C.

The first step is to convert the elements of Φ to clauses. This is quite easy.
- $(J \vee Y)$ is already a clause.
- $(Y \rightarrow (\neg S \rightarrow C))$ is equivalent to the the clause $(\neg Y \vee S \vee C)$.
- $(\neg J \rightarrow S)$ is equivalent to the clause $(J \vee S)$.
- The negation of the goal is the clause $\neg C$.

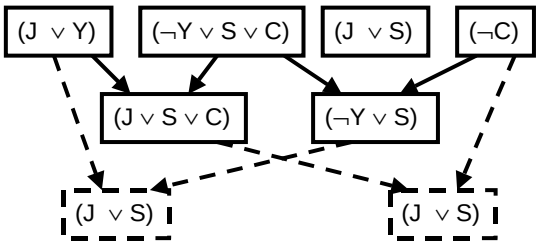
Thus, the set of clauses to work with is

$\psi = \{(J \vee Y), (\neg Y \vee S \vee C), (J \vee S), (\neg C)\}$.

From examination of this problem under other proof methods, we know that the conclusion does not follow from the hypotheses. In this case, what can resolution tell us?

Under these finite circumstances, the number of resolvents is finite. Thus, once no new resolvents can be discovered, the process has completed.

Here is the resolution graph for this problem. Duplicates (which are discarded) are shown with dashed boxes and arrows.



Since an exhaustive search does not yield the empty clause, ψ must be satisfiable.

Since the number of clauses over a finite number of proposition letters is finite, the process of generating resolvents until none further exist is guraranteed to terminate.

A blocks world example:

This example uses the domain of the simple blocks world, introduced on the introductory slides.

The goal is to prove that if B2 is atop B1, then either P1 or else P2 must be on the table. In the associated propositional logic, this is expressed as follows:

On[B2,B1] → (On_table[P1] ∨ On_table[P2])

This is the goal of the problem. In a resolution solution, it must be negated. This negation yields three atomic clauses:

(1) On[B2,B1]
(2) ¬On_table[P1]
(3) ¬On_table[P2]}

To convert all of the first-order sentences describing the constraints on the blocks world to propositional wff's would be a huge task. So, we will select just the ones that we need.

No object can rest atop a pyramid.
(∀x)(∀y)(¬(Is_pyramid(x) ∧ On(y,x)))

We need:
(4) ¬On(P1,P1)
(5) ¬On(P1,P2)
(6) ¬On(P2,P1)
(7) ¬On(P2,P2)

At most one object can rest atop another object.
(∀x)(∀y)(∀z) ((On(y,x) ∧ On(z,x)) → y=z))

¬(On[B2,B1] ∧ On[P1,B1])
¬(On[B2,B1] ∧ On[P2,B1])
¬(On[P1,B2] ∧ On[P2,B2])

which are represented by the following clauses:

(8) (¬On[B2,B1] ∨ ¬On[P1,B1])
(9) (¬On[B2,B1] ∨ ¬On[P2,B1])
(10) (¬On[P1,B2] ∨ ¬On[P2,B2])

Every object is either on the table or else atop another object.
(∀x)(∃y)(On_table(x) ∨ On(x,y)))

(11) (Ontable[P1] ∨ On[P1,B1] ∨ On[P1,B2] ∨ On[P1,P1] ∨ On[P1,P2])

(12) (Ontable[P2] ∨ On[P2,B1] ∨ On[P2,B2] ∨ On[P2,P1] ∨ On[P2,P2])

So, here is the database of clauses necessary to perform the deduction:

(1) On[B2,B1]

(2) ¬On_table[P1],

(3) ¬On_table[P2]

(4) ¬On(P1,P1)

(5) ¬On(P1,P2)

(6) ¬On(P2,P1)

(7) ¬On(P2,P2)

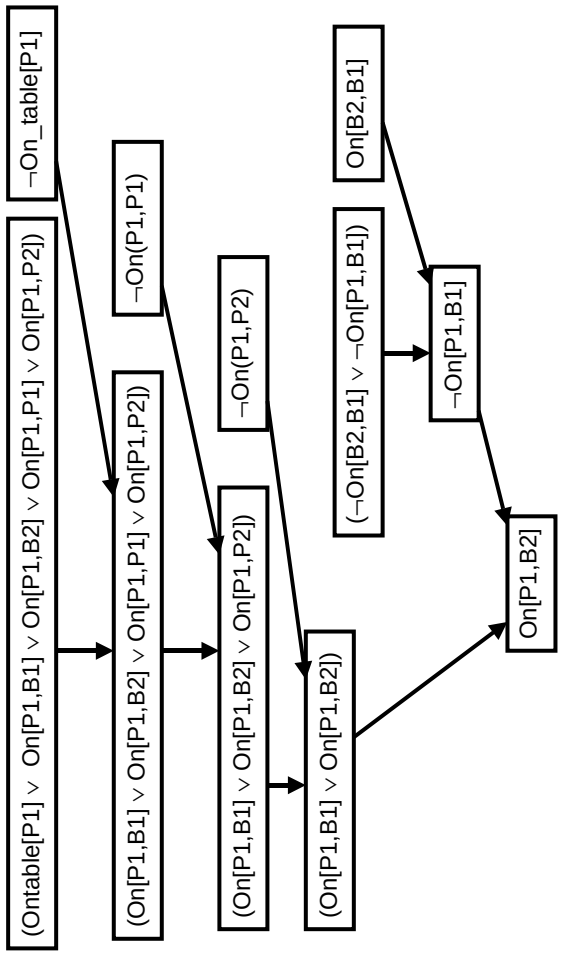
(8) (¬On[B2,B1] ∨ ¬On[P1,B1])

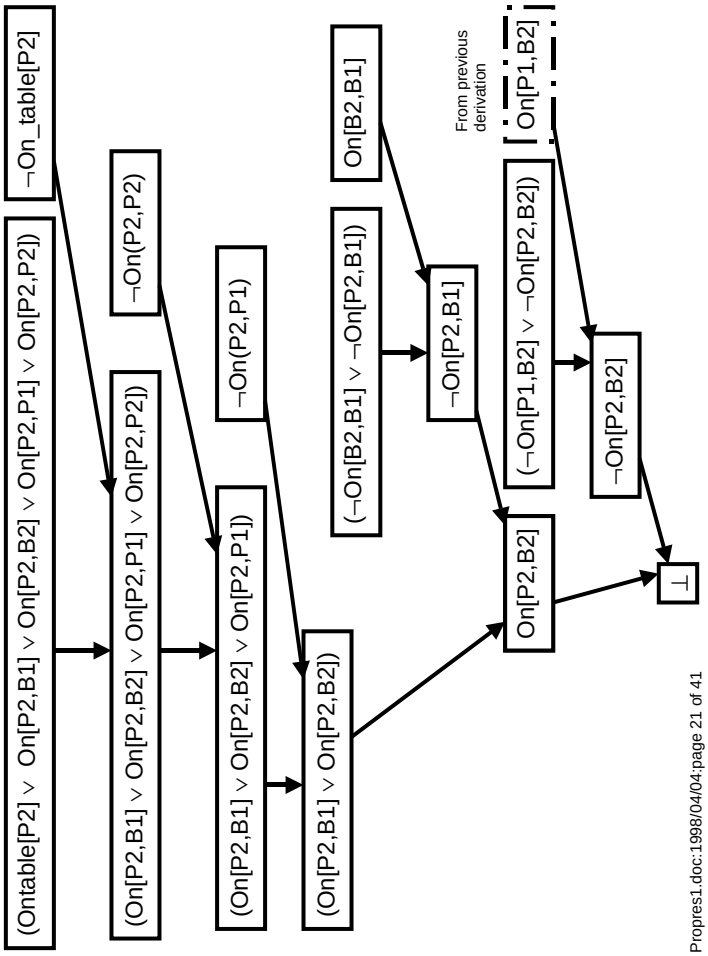
(9) (¬On[B2,B1] ∨ ¬On[P2,B1])

(10) (¬On[P1,B2] ∨ ¬On[P2,B2])

(11) (Ontable[P1] ∨ On[P1,B1] ∨ On[P1,B2] ∨ On[P1,P1] ∨ On[P1,P2])

(12) (Ontable[P2] ∨ On[P2,B1] ∨ On[P2,B2] ∨ On[P2,P1] ∨ On[P2,P2])

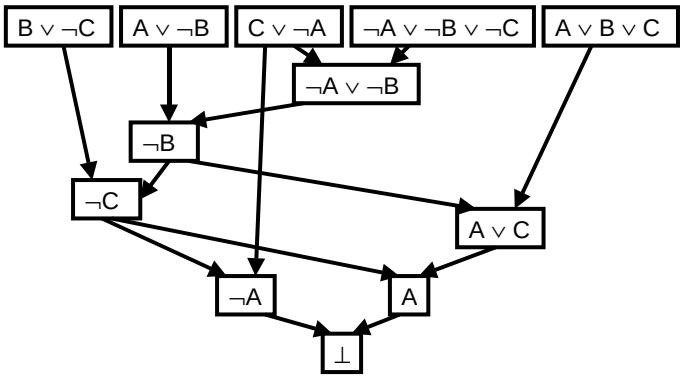




Simplification Strategies:

In general, when a resolution is performed, the clauses which were resolved may **not** be discarded, since they may be needed again in a subsequent resolution.

Example: Let $\Phi = \{ A \vee \neg B,\ B \vee \neg C,\ C \vee \neg A,\ \neg A \vee \neg B \vee \neg C,\ A \vee B \vee C \}$.
It is easy to see that Φ is unsatisfiable, as illustrated by the following resolution proof graph.



However, note that there are three clauses which are used in each of two distinct resolutions. In general, this multiple use of clauses cannot be avoided.

Definition: A clause φ_1 is a *subclause* of clause φ_2 if every literal of φ_1 is also a literal of φ_2. If, in addition, φ_2 contains some literal which is not in φ_1, then φ_1 is a *proper subclause* of clause φ_2.

Examples:

- $A \vee \neg C$ is a proper subclause of $A \vee B \vee \neg C$
- $A \vee \neg C$ is a subclause of itself, but not proper.
- $\bot$ is a subclause of every clause, and a proper subclause of every clause except itself.
- $A \vee C$ is not a subclause of $A \vee B \vee \neg C$. It is the literals, and not just the associated atoms, which must satisfy the containment condition.

Observation: Let φ_1 be a subclause of clause φ_2. Then

$$\{\varphi_1\} \models \varphi_2. \ \square$$

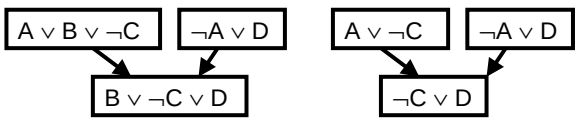
This observation may be translated into a practical simplification strategy for the resolution process.

Terminology: If φ_1 is a (proper) subclause of clause φ_2, then φ_2 is a (proper) superclause of φ_1.

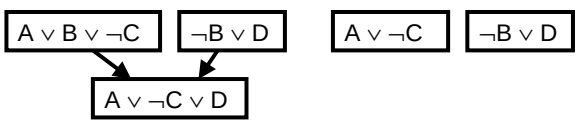
Simplification strategy: Suppose that Φ is a set of clauses. Then each clause which is a superclause of some other clause of Φ may be removed without changing whether or not resolution will find a refutation of Φ.

Example: Suppose that
$A \vee \neg C,\ A \vee B \vee \neg C,\ \neg A \vee D,\ \neg B \vee D \in \Phi$.
We will illustrate why $A \vee B \vee \neg C$ (which is a superclause of $A \vee \neg C$), may be removed without compromising the resolution process.

First consider the following two resolutions, in which the first "resolvee" of the pair on the left is a superclause of the corresponding one on the right. Note further that the literal of resolution is contained in both clauses. In this case, the result of the first resolution is a superclause of the result of the second.



Now suppose that the literal of resolution is contained only in the superclause. No resolution is possible for the right pair, but the result of the left resolution is a superclause of the first clause on the right.



In both cases, the resolution involving the superclause adds nothing useful. The clause $A \vee B \vee \neg C$ may be discarded!

Here is a more formal statement.

Let Φ be a set of clauses. Define Base(Φ) to be the subset of Φ obtained by deleting each clause which is a superclause of some other member of Φ.

Example: Let
$\Psi = \{ A \vee \neg B,\ B \vee \neg C,\ C \vee \neg A,\ \neg A \vee \neg B \vee \neg C,$
$A \vee B \vee C,\ A \vee C,\ \neg A \vee \neg B \}.$
Then
Base(Ψ) = $\{ A \vee \neg B,\ B \vee \neg C,\ C \vee \neg A,$
$A \vee C,\ \neg A \vee \neg B \}.$

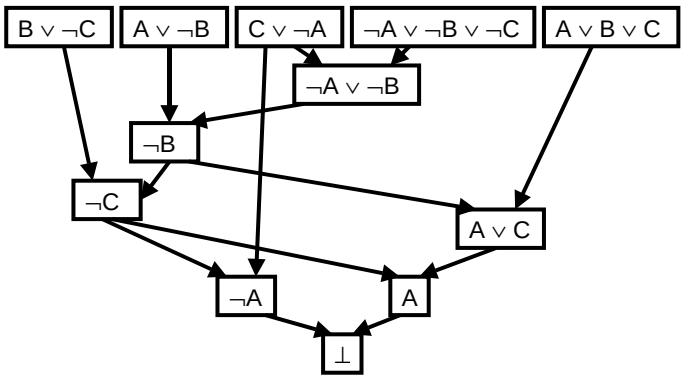
Observation: Let Φ be a set of clauses. Then
$\Phi \models \bot$ iff Base(Φ) $\models \bot$. $\square$

We already know that resolution is sound, and we will soon see that resolution is complete, so that
$\Phi \models \bot$ iff $\Phi \vdash_{Res} \bot$.

Thus: Let Φ be a set of clauses. Then
$\Phi \vdash_{Res} \bot$ iff Base(Φ) $\vdash_{Res} \bot$. $\square$

The process of removing superclauses from the database of clauses may even be performed on the fly.

For example, in the proof on slide 22, which is reproduced here:



- $A \vee B \vee C$ may be deleted from the clause base when as $A \vee C$ is derived.
- $\neg A \vee \neg B \vee \neg C$ may be deleted from the clause base when $\neg A \vee \neg B$ is derived.
- $A \vee \neg B$ and $\neg A \vee \neg B$ may be deleted from the clause base when $\neg B$ is derived.
- $B \vee \neg C$ may be deleted from the clause base when $\neg C$ is derived.
- $C \vee \neg A$ may be deleted from the clause base when $\neg A$ is derived.
- $A \vee C$ may be deleted from the clause base when A is derived.

This will simplify the search process for new resolvents greatly!

Unit and Input Resolution:

A *unit clause* is one which consists of a single literal.

A *unit resolution* is one in which one of the clauses which are resolved is a unit clause.

A *unit refutation* of a set Φ of clauses is a refutation proof $\Phi \vdash_{Res} \bot$ in which each resolution is a unit resolution.

Example: The previous blocks-world proof is a unit refutation.

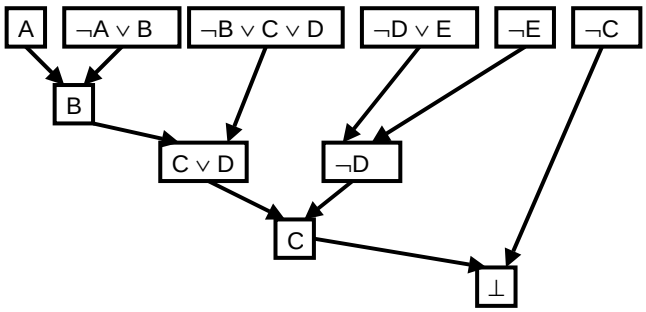
Given a set Φ of clauses, an *input resolution* with respect to Φ is a resolution in which one of the clauses to be resolved is an element of Φ.

An *input refutation* of a set Φ of clauses is a refutation proof $\Phi \vdash_{Res} \bot$ in which each resolution is an input resolution.

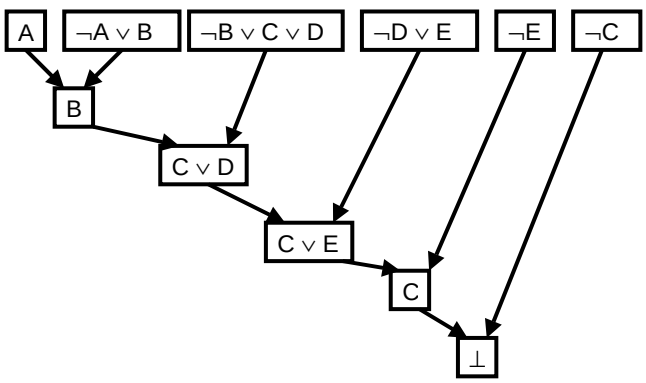
Example: Let
$\Phi = \{A, \neg A \vee B, \neg B \vee C \vee D, \neg D \vee E, \neg E, \neg C\}.$

Here are three distinct refutations of Φ, with distinct properties.

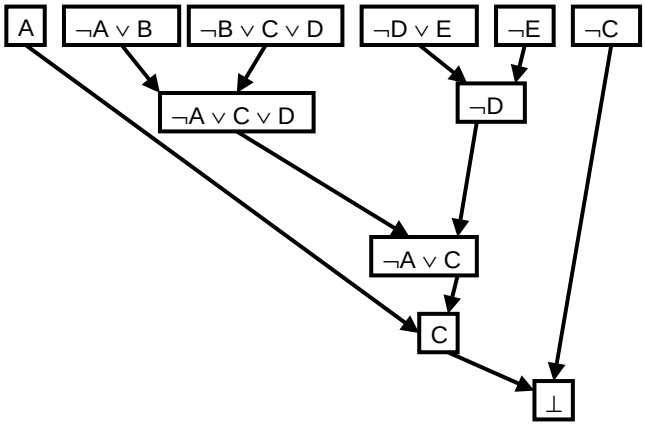
A unit refutation which is not an input refutation:



An input refutation which is not a unit refutation:



A refutation which is neither unit nor input:



Despite the differences, the following remarkable result holds.

Theorem: A set Φ of clauses has a unit refutation iff it has a input refutation. □

Q: Why are unit resolution and input resolution important?

A: For these forms of resolution, very efficient algorithms exist. Although they are not complete, they can solve a very large class of interesting subproblems. Sometimes, a tradeoff of completeness for efficiency is warranted.

Important: There are sets of clauses which are inconsistent, yet which admit no input or unit refutation.

Example: {A ∨ B, ¬A ∨ B, A ∨ ¬B, ¬A ∨ ¬B}.

Clearly, this set cannot have a unit resolution (since it has no unit clauses), so by the above result, it cannot have an input refutation either.

Let us now examine the complexity of unit resolution more closely.

The Complexity of Unit Refutation:

Observation: Let φ_1 be any clause, let φ_2 be a unit clause, and suppose that φ_1 and φ_2 are resolvable. Then the (unique) resolvent of φ_1 and φ_2 is a subclause of φ_1. □

Example: Let $\varphi_1 = A \vee \neg B \vee \neg C$;
$\varphi_2 = C$.
The resolvent is $A \vee \neg B$, which is a subclause of φ_1.

Definition:
(a) Let φ be a clause. Define Length(φ) to be the number of literals in φ.
(b) Let Φ be a set of clauses. Define Length(Φ) to be the sum of the lengths of its elements.

Example: Let
$\Phi = \{A, \neg A \vee B, \neg B \vee C \vee D, \neg D \vee E, \neg E, \neg C\}$.
Then Length(Φ) = 10.

Observation: Let Φ be a set of clauses. If superclauses are discarded at each step, then the total number of unit resolutions which may be applied to Φ is bounded by Length(Φ). □

The number of (unrestricted) resolutions which may be applied to Φ is $\Theta(2^{\text{Length}(\Phi)})$ in the worst case, so the number of possible unit resolutions is much smaller than the total number of possible resolutions.

A Comparison to Semantic Tableaux:

Semantic Tableaux:

- The basic input unit is a conjunction of literals.
- A general input must be in DNF.
- Satisfiability is established when one of the conjunctions of literals is found to be satisfiable.
- Unsatisfiability is established only when all conjunctions of literals are found to be unsatisfiable.
- Most of the complexity is emodied in converting the problem to a DNF representation.

Resolution:

- The basic input unit is a disjunction of literals (clause).
- A general input must be in CNF.
- Unsatisfiability is established when a proof of the empty clause is found.
- Satisfiability is established only when all resolvents are computed, and the empty clause is not amongst them.

Many problem statements are naturally in CNF, or something close. The complexity lies in the resolution process.

Some implementation issues surrounding resolution:

The implementation of a resolution-based theorem prover is a course in itself. However, there are some simple points which should be emphasized.

Data structures which capture the following information must be maintained:

- The pairs of clauses, with the associated literals, which have already been resolved. (Note that a pair of clauses may have more than one resolvent.)
- The collection of all hypotheses and resolvents which have been found.

These structures are necessary in order that it may be determined:

- that no more resolvents may be obtained.
- that a given pair of clauses has already been resolved on a given literal pair.
- That resolution of a given pair of clauses on a given pair of literals will generate a clause which has already been found.

The Completeness of Resolution:

A thorough, formal proof of the completeness of the resolution principle will now be presented. Note that such a proof is not presented in the textbook.

Recall that resolution is
(a) a clause-based procedure, and
(b) a refutation procedure.

Thus, the result which is to be established is the following.

Theorem: Let $\Phi = \{\varphi_1, \varphi_2, .., \varphi_n\}$ be a finite set of clauses. Then

$$\Phi \models \bot \text{ implies } \Phi \vdash_{Res} \bot.$$

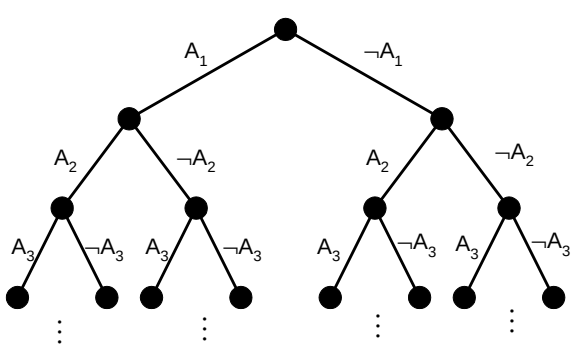
Convention: Throughout this proof, it is assumed that all clauses are taken over a fixed finite propositonal logic $L = (P, C, A)$, in which $P = \{A_1, A_2, .., A_n\}$. Furthermore, it will be assumed that these proposition letters have an order imposed upon them

$$\theta = (A_1, A_2, .., A_n)$$

The actual ordering is of no consequence; any will do.

Definition: The *semantic tree* corresponding to θ is the full binary tree T_θ whose edges are labelled as follows:
(i) The left edge from any node at level r is labelled with A_r.
(ii) The right edge from any node at level r is labelled with $\neg A_r$.

In a picture:



Definition: A partial function $f: X \to Y$ is a function which may be defined only on some of the elements of X. An ordinary function, which is defined on all elements of X, is a special case of a partial function.

Definition: Let N be a node in T_θ. Define the partial function

$$\omega(N,\theta)(A_j) = \begin{cases} 1 & \text{if } A_j \text{ is on the path from the root to N.} \\ 0 & \text{if } \neg A_j \text{ is on the path from the root to N.} \\ & \text{undefined otherwise.} \end{cases}$$

Definition: A partial function

$$w: P \to \{0,1\}$$

Is called a *partial interpretation* (of L).

Definition: Let w be a partial interpretation, and let $\Phi \subseteq WF(L)$.

(a) If w has an extension $\hat{w} : P \to \{0,1\} \in Mod(\Phi)$, then w is called a *partial model* of Φ.
(b) If, for each $\varphi \in \Phi$, w has an extension $\hat{w}_\varphi : P \to \{0,1\} \in Mod(\Phi)$, then w is called a *weak partial model* of Φ.

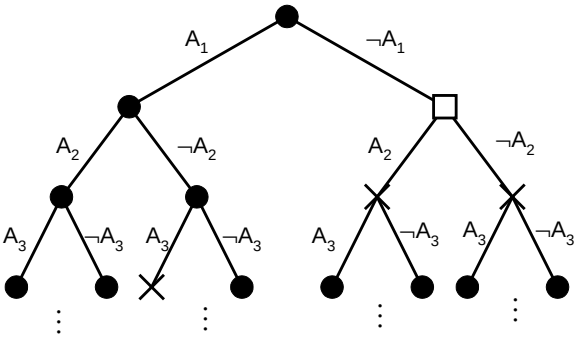
Observation: Clearly, every weak partial model is a partial model. However, the converse is false. Examples: Let $\Phi = \{A \vee B, A \vee \neg B\}$. Then

- $f : P \to \{0,1\}$: $A \mapsto 1$; $B \mapsto$ undefined is a partial model.
- $g : P \to \{0,1\}$: $A \mapsto 0$; $B \mapsto$ undefined is a weak partial model, but not a partial model.

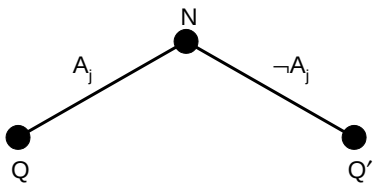
Definition: Let Φ be a finite set of clauses. A node N in T_θ is, with respect to Φ:

(a) a *safe node* if $\omega(N,\theta)$ is a weak partial model of Φ;
(b) an *inference node* if it is a safe node, but no descendant of it is a safe node;
(c) a *failure node* if it is not a safe node, but every ancestor of it is a safe node.

Example: Let $\Phi = \{A_1 \vee A_2, A_1 \vee \neg A_2, A_2 \vee \neg A_3\}$.
The failure nodes are labelled with X's.
The inference nodes are labelled with boxes.



Proposition: Let Φ be a finite set of clauses, and let N be an inference node for Φ in the tree T_θ. Let Q and Q′ be the children of N. Select $\varphi \in \Phi$ such that $\omega(Q,\theta)$ cannot be extended to a model of φ. (Such a φ must exist by the definition of a failure node.) Similarly, select $\varphi' \in \Phi$ with the property that $\omega(Q',\theta)$ cannot be extended to a model of φ'. Let A_j be the proposition name on which Q and Q′ branch, as illustrated in the figure below. Then, $\omega(N,\theta)$ cannot be extended to a model of the resolvent of φ and φ' on $(A_j, \neg A_j)$.



Proof: It is clear that $\varphi = \psi \vee \neg A_j$, and that $\varphi' = \psi' \vee \neg A_j$, for clauses ψ and ψ', since Q and Q′ are failure nodes. Let $\Lambda = \{\ell_1, \ell_2, .., \ell_{j-1}\}$ be the literals along the path from the root to N. Let $\Lambda_\neg = \{\neg\ell_1, \neg\ell_2, .., \neg\ell_{j-1}\}$ be the set of complements of Λ. Then ψ must be a disjunction of some of the elements of $\Lambda_\neg$, else Q could not be failure node. Similarly, ψ' must be a disjunction of elements of $\Lambda_\neg$. But then N must be a failure node for $\psi \vee \psi'$, which is precisely the resolvent of φ and φ' on $(A_j, \neg A_j)$. □

Definition: Let Φ be a finite set of clauses. Define $Res(\Phi)$ to be the closure of Φ under the operation of resolution.

Lemma: Let Φ be a finite set of unsatisfiable clauses. Then every node of T_θ is a failure node with respect to $Res(\Phi)$.
Proof: Suppose, to the contrary, that T_θ contains a safe node N. This node N cannot be a leaf, else $\omega(N,\theta)$ would be a model for Φ. Let M be a safe node which is furthest from the root. Then M must be an inference node, else one of its children would be safe, contradicting the assumption that is a safe node which is furthest from the root. Since it is an inference node, the preceding proposition asserts that it is a failure node for an element of $Res(\Phi)$. This is a contradiction, and so no such M can exist. Hence every node of T_θ is a failure node with respect to $Res(\Phi)$. □

Theorem: Resolution is complete as a refutation procedure: If Φ be a finite set of unsatisfiable clauses, then

$$\Phi \models \perp \text{ implies } \Phi \vdash_{Res} \perp.$$

Proof: In view of the above lemma, the root node of T_θ must be a failure node with respect to $Res(\Phi)$. This can only happen if $\perp \in Res(\Phi)$. □

Summary:

Note that soundness of resolution states that

$$\Phi \models \varphi \text{ implies } \Phi \vdash_{Res} \varphi,$$

which holds in particular when φ is $\perp$. Thus, in view of the preceding theorem, the following combination result holds.

Corollary: Resolution is sound and complete as a refutation procedure: If Φ be a finite set of unsatisfiable clauses, then

$$\Phi \models \perp \text{ iff } \Phi \vdash_{Res} \perp. \ \square$$

It must be emphasized that this result does not hold for general inference on clause. The last line of the corollary **cannot** be replaced by

$\Phi \models \varphi$ iff $\Phi \vdash_{Res} \varphi$.

Finiteness Issues:

Oservation: It is possible to implement $\vdash_{Res}$ as a decision procedure. That is, for the problem

$$\Phi \vdash_{Res} \bot$$

it is possible it implement the inference engine in such as way that either

(a) It establishes that $\Phi \vdash_{Res} \bot$ holds, by finding a proof, from Φ, of the empty clause $\bot$.

(b) It establishes that $\Phi \vdash_{Res} \bot$ does not hold, by generating the resolution closure of Φ and noting that it does not contain the empty clause.

Note that this is not possible with the Hilbert system. Since the axiom schemata of the Hilbert system allow one to generate an infinite number of axioms, it is *always* possible to generate something new, so the analog of condition (b) above cannot be realized.