(Rule-based) Artificial Intelligence - Resolution for Propositional Logic

Lila Kari

University of Waterloo
The major interest of computer scientists in propositional and predicate calculus has been to exploit its expressive power to prove theorems:

**Theorem:** Prem.1, Prem.2, ..., Prem.\(n\) \(\vdash\) Conclusion

In the field of **Artificial Intelligence**, there have been many attempts to construct programs that could prove theorems automatically.

Given a set of **axioms** and a **technique** for deriving new theorems from old theorems and axioms, would such a program be able to prove a particular theorem?
Automated theorem proving

- Early attempts faltered because there seemed to be no efficient technique for deriving new theorems.
- 1965: J.A. Robinson at Syracuse University discovered the technique called resolution.

John Allan Robinson, born 1928
Resolution

Resolution theorem proving is a method of formal derivation (deduction) that has the following features:

- The only formulas allowed in resolution theorem proving are disjunctions of literals, such as \((P \lor Q \lor \neg R)\).
- Such a disjunction of literals is called a (disjunctive) clause. Hence, all formulas involved in resolution theorem proving must be (disjunctive) clauses.
- There is essentially only one rule of formal deduction, resolution.
How does resolution work?

- To prove that the argument \( A_1, A_2, \ldots, A_n \vdash C \) is valid, one shows that the set

\[ \{ A_1, A_2, \ldots, A_n, \neg C \} \]

is not satisfiable.

- We show that the set of formulas \( \{ A_1, A_2, \ldots, A_n, \neg C \} \) is not satisfiable by proving that, for some formula \( B \), one can derive both \( B \) and \( \neg B \) (a contradiction).

- By “derive”, we mean by repeated applications of resolution (our only formal deduction rule).
In general, one can convert any formula into one or more (disjunctive) clauses.

To do this, one first converts the formula into a conjunction of disjunctions; that is, one converts the formula into conjunctive normal form.

Each term of the conjunction is then made into a clause of its own.
Example: Convert $P \rightarrow (Q \land R)$ into clauses.

Solution.
We first eliminate the $\rightarrow$ by writing $\neg P \lor (Q \land R)$.

We then apply the distributive law to obtain

$$P \rightarrow (Q \land R) \models (\neg P \lor Q) \land (\neg P \lor R).$$

This yields the two clauses $\neg P \lor Q$ and $\neg P \lor R$. 
A single rule of formal deduction: resolution

- Two clauses (called parent clauses) can be resolved iff they contain two complementary literals, say $P$ and $\neg P$.
- If the complementary literals are $P$ and $\neg P$, one says that we resolve over $P$, or that resolution is on $P$.
- The result of resolution on $P$ is the resolvent, which is the disjunction of all literals of the parent clauses, except that $P$ and $\neg P$ are omitted.
- NOTE: If the two parent clauses are $P$ and $\neg P$, their resolvent is called the empty clause and is denoted by $\{\}$.  
- NOTE: By definition, the empty clause is not satisfiable.
Example

Find the resolvent of $P \lor \neg Q \lor R$ and $\neg S \lor Q$.

Solution. The two parent clauses $P \lor \neg Q \lor R$ and $\neg S \lor Q$ can be resolved over $Q$ because $Q$ is negative in the first clause and positive in the second.

The resolvent is the disjunction of $P \lor R$ with $\neg S$, which yields $P \lor R \lor \neg S$. 

Soundness of resolution-based formal deduction

Theorem: The resolvent is logically implied by its parent clauses, which makes resolution a sound rule of formal deduction.

To see this, let $P$ be a propositional variable, and let $A$ and $B$ be (possibly empty) clauses. One has

$$P \lor A, \neg P \lor B \models A \lor B$$

This is valid for the following reasons.
Soundness of resolution

- If $P$ is false, then $A$ must be true, because otherwise $P \lor A$ is false.
- Similarly, if $P$ is true, then $B$ must be true, because otherwise $\neg P \lor B$ is false.
- Since $P$ must be true or false, either $A$ or $B$ must be true, and the result follows.
- Of course, $A \lor B$ is the resolvent of the parent clauses $P \lor A$ and $\neg P \lor B$ on $P$, which proves the soundness of resolution.
- In the particular case $P, \neg P \models \{\}$, we note that $\{\}$ is the empty clause, which is a notation for a contradiction (0), and that the above tautological consequence still holds, since the premises are contradictory.
A common mistake in using resolution is to apply it to more than one variable. This is not correct.

For example, the following is an incorrect use of resolution:

1. \( P \lor \neg Q \)
2. \( \neg P \lor Q \)
3. \( \{\} \) (from 1, 2 resolving over \( P \) and \( Q \))

This disagrees with the **Soundness of Resolution Theorem** since the argument

\[
P \lor \neg Q, \quad \neg P \lor Q \models \{\}
\]

is not valid: we can satisfy the premises by setting \( P \) and \( Q \) equal to 1, but cannot satisfy the conclusion \( \{\} \), which signifies 0, and which is by definition not satisfiable.

This is not resolution.
Resolution is particularly effective when one of the parent clause is a unit clause, that is, a clause that contains only one literal.

Example: The resolution of $\neg P \lor Q \lor R$ and $\neg R$. The resolvent is $\neg P \lor Q$.

Example: The resolution of $P \lor Q$ with $\neg P$ yields $Q$, which agrees with the disjunctive syllogism.
Prove Modus Ponens by resolution

\[ P, P \rightarrow Q \vdash Q \]

1. \( P \)  \hspace{1cm} \text{Premise}
2. \( \neg P \lor Q \)  \hspace{1cm} \text{Premise}
3. \( \neg Q \)  \hspace{1cm} \text{Negation of conclusion}
4. \( Q \)  \hspace{1cm} \text{Resolvent of 1, 2}
5. \( \{ \} \)  \hspace{1cm} \text{Resolvent of 3, 4}
Prove Hypothetical Syllogism by resolution

\[ P \rightarrow Q, \ Q \rightarrow R \vdash P \rightarrow R \]

1. \( \neg P \lor Q \) Premise
2. \( \neg Q \lor R \) Premise
3. \( P \) Derived from negation of conclusion
4. \( \neg R \) Derived from the negation of conclusion
5. \( Q \) Resolvent of 1, 3
6. \( \neg Q \) Resolvent of 2, 4
7. \( \{\} \) Resolvent of 5, 6

Resolution for propositional logic
Resolution strategies

When doing resolution automatically, one has to decide in which order to resolve the clauses. This order can greatly affect the time needed to find a contradiction. Strategies include:

- **Unit resolution**: all resolutions involve at least one unit clause. Moreover, preference is given to clauses that have not been used yet.
- The “Set-of-Support Strategy”
- The Davis-Putnam Procedure (DPP)
Example of unit resolution

Prove $P_4$ from $P_1 \rightarrow P_2, \neg P_2, \neg P_1 \rightarrow P_3 \lor P_4, P_3 \rightarrow P_5, P_6 \rightarrow \neg P_5$ and $P_6$.

1. $\neg P_1 \lor P_2$ Premise
2. $\neg P_2$ Premise
3. $P_1 \lor P_3 \lor P_4$ Premise
4. $\neg P_3 \lor P_5$ Premise
5. $\neg P_6 \lor \neg P_5$ Premise
6. $P_6$ Premise
7. $\neg P_4$ Negation of conclusion
8. $\neg P_1$ Resolvent of 1, 2
9. $\neg P_5$ Resolvent of 5, 6
10. $P_1 \lor P_3$ Resolvent of 3, 7
11. $\neg P_3$ Resolvent of 4, 9
12. $P_3$ Resolvent of 8, 10
13. $\{\}$ Resolvent of 11, 12
• The unit resolution is not complete.
• This is demonstrated by the following example.
• The argument

\[ Q \lor R, \ Q \lor \neg R, \ \neg Q \lor R \models Q \land R \]

is a valid argument.
• However, we cannot prove its validity by unit resolution.
• This is because, in this case there is no unit clause, which makes unit resolution impossible.
Set-of-Support Strategy

- One partitions all clauses into two sets, the set of support and the auxiliary set.
- The auxiliary set is formed in such a way that the formulas in it are not contradictory.
- For instance, the premises are usually not contradictory. The contradiction will only arise after one adds the negation of the conclusion.
- One often uses the premises as the initial auxiliary set and the negation of the conclusion as the initial set of support.
Set-of-Support Strategy

- Since one cannot derive any contradiction by resolving clauses within the auxiliary set, one avoids such resolutions.
- Stated positively, when using the Set-of-Support Strategy, each resolution takes at least one clause from the set of support.
- The resolvent is then added to the set of support.
- Theorem. Resolution with the set of support strategy is complete. (the proof is omitted)
Example

Prove $P_4$ from $P_1 \rightarrow P_2, \neg P_2, \neg P_1 \rightarrow P_3 \lor P_4, P_3 \rightarrow P_5, P_6 \rightarrow \neg P_5$ and $P_6$, by using the set of support strategy.

Initially the set of support is given by $\neg P_4$, the negation of the conclusion.

One then does all the possible resolutions involving $\neg P_4$, then all possible resolutions involving the resulting resolvents, and so on.

If the initial 7 clauses are omitted, this yields the following derivation:

8. $P_1 \lor P_3$  Resolvent of 7, 3
9. $P_2 \lor P_3$  Resolvent of 1, 8
10. $P_3$  Resolvent of 2, 9
11. $P_5$  Resolvent of 4, 10
12. $\neg P_6$  Resolvent of 5, 11
13. $\{\}$  Resolvent of 6, 12
The Pigeonhole Principle

- The Pigeonhole Principle $\mathcal{P}_n$ says that one cannot put $n + 1$ objects into $n$ slots, with distinct objects going into distinct slots.

- **Example:** Among any group of 367 people there must be at least two with the same birthday.
Pigeonhole Principle and Resolution

- Formulate the Pigeonhole Principle as a conjunction of logic formulas.
- Choose propositional variables $P_{ij}$ for $1 \leq i \leq n+1$, $1 \leq j \leq n$
- $P_{ij}$ is true iff the $i$th pigeon goes into the $j$th slot.
- Construct clauses for:
  1. Each pigeon $i$ for $1 \leq i \leq n+1$ goes into some slot $k$, for $1 \leq k \leq n$
     $$P_{i1} \lor P_{i2} \ldots \lor P_{in} \quad \text{for } 1 \leq i \leq n+1.$$
  2. Distinct pigeons $i$ and $j$ (between 1 and $n+1$) cannot go into the same slot $k$
     $$\neg P_{ik} \lor \neg P_{jk} \quad \text{for } 1 \leq i < j \leq n+1, \text{ and } 1 \leq k \leq n$$
- Any value assignment that satisfies the conditions (the conjunction of all the above clauses) would map $n+1$ pigeons one-to-one into $n$ slots.
- Of course, this cannot be done, so the set of clauses must be unsatisfiable.
Example

What is the pigeonhole $P_2$ (3 pigeons and 2 slots) stated as a resolution problem?

- Every pigeon in at least one slot: $P_{11} \lor P_{12}$, $P_{21} \lor P_{22}$, $P_{31} \lor P_{32}$
- No two pigeons per slot:

  Slot 1: $\neg P_{11} \lor \neg P_{21}$, $\neg P_{11} \lor \neg P_{31}$, $P_{21} \lor \neg P_{31}$

  Slot 2: $\neg P_{12} \lor \neg P_{22}$, $\neg P_{12} \lor \neg P_{32}$, $\neg P_{22} \lor \neg P_{32}$

Note: We do not need all possible pairs $(i, j)$ for every slot $k$ because, e.g., $(P_{31} \rightarrow \neg P_{11}) \models (P_{11} \rightarrow \neg P_{31}) \models (\neg P_{11} \lor \neg P_{31})$

Since the set of the 9 clauses is not satisfiable (due to the Pigeonhole Principle), one should be able to derive the empty clause from it.

Exercise: Give a resolution derivation of the empty clause for the pigeonhole principle $P_2$ above.
And NUH is the letter I use to spell Nutches
Who live in small caves, known as Nitches, for hutches,
These Nutches have troubles, the biggest of which is
the fact there are many more Nutches than Nitches.
Each Nutch in a Nitch knows that some other Nutch
Would like to move into his Nitch very much.
So each Nutch in a Nitch has to watch that small Nitch
Or Nutches who haven’t got Nitches will snitch.

(Dr. Seuss, On Beyond Zebra)
Davis-Putnam Procedure (DPP) - Preprocessing the Input

- Any clause corresponds to a set of literals, that is, the literals contained within the clause.
- For instance, the clause \( P \lor \neg Q \lor R \) corresponds to the set \( \{P, \neg Q, R\} \) and \( \neg S \lor Q \) corresponds to the set \( \{\neg S, Q\} \).
- Since the order of the literals in a disjunction is irrelevant, and since the same is true for the multiplicity in which the terms occur, the set associated with the clause completely determines the clause.
- For this reason, one frequently treats clauses as sets, which allows one to speak of the union of two clauses.
DPP: Resolution as operation between sets

- If clauses are represented as sets, one can write the **resolvent** of two clauses $A$ and $B$ on $P$ as follows:

  $$C = (A \cup B) \setminus \{P, \neg P\}.$$  

- In words, the resolvent is the union of all literals of $A$ and $B$ except that the two literals involving $P$ are omitted.

- Note that, by convention, resolving $\{P\}$ with $\{\neg P\}$ results in the empty clause $\{\}$. 

The Davis-Putnam Procedure (DPP)

Given as input a nonempty set of clauses in the propositional variables $P_1, P_2, \ldots P_n$, the Davis Putnam Procedure (DPP) repeats the following steps until there are no variables left:

- Discard all clauses that have both a literal $L$ and its complement $\neg L$ in them (they will never lead to a contradiction).
- Choose a variable $P$ appearing in one of the clauses.
- Add all possible resolvents using resolution on $P$ to the set of clauses.
- Discard all clauses with $P$ or $\neg P$ in them.
DPP - Eliminating a variable

- We refer to the preceding sequence of steps as eliminating the variable $P$.
- If in some step one resolves $\{P\}$ and $\{\neg P\}$ then one obtains the empty clause, and it will be the only clause at the end of the procedure.
- If one never has a pair $\{P\}$ and $\{\neg P\}$ to resolve, then all the clauses will be thrown out and the output will be no clauses.
- So the output of DPP either the empty clause or no clauses.
- This may seem rather subtle but just think of the difference between arriving to the library with an empty backpack (empty clause), as opposed to no backpack (empty set).


**Davis-Putnam algorithm**

- Let \( S_1 = S \).
- Let \( i = 1 \).
- **LOOP** until \( i = n + 1 \).
- Discard members of \( S_i \) in which a literal and its complement appear, to obtain \( S'_i \).
- Let \( T_i \) be the set of **parent clauses** in \( S'_i \) in which \( P_i \) or \( \neg P_i \) appears.
- Let \( U_i \) be the set of **resolvent clauses** obtained by resolving (over \( P_i \)) every pair of clauses \( C \cup \{ P_i \} \) and \( D \cup \{ \neg P_i \} \) in \( T_i \).
- Set \( S_{i+1} \) equal to \((S'_i \setminus T_i) \cup U_i\). (Eliminate \( P_i \)).
- Let \( i \) be increased by 1.
- **ENDLOOP**.
- Output \( S_{n+1} \).
Example

Let us apply the Davis-Putnam procedure to the clauses

\[ \{\neg P, Q\}, \{\neg Q, \neg R, S\}, \{P\}, \{R\}, \{\neg S\} \]

- Eliminating \( P \) gives \( \{Q\}, \{\neg Q, \neg R, S\}, \{R\}, \{\neg S\} \) (This is \( S_2 \) and \( S'_2 \)).
- Eliminating \( Q \) gives \( \{\neg R, S\}, \{R\}, \{\neg S\} \). (This is \( S_3 \) and \( S'_3 \).)
- Eliminating \( R \) gives \( \{S\}, \{\neg S\} \). (This is \( S_4 \) and \( S'_4 \).)
- Eliminating \( S \) gives \( \{\} \). (This is \( S_5 \).)

So the output is the empty clause.
Comments

- If the set of clauses is more complex, before each iteration (elimination of a variable) we should give each clause in $T_i$ a numerical identifier.
- In the next phase (that produces the resolvents in $U_i$ from parent clauses in $T_i$) we should provide, for each resolvent, the identifiers of the two parent clauses that produced it.
- If the output of DPP is the empty clause, $\{\}$, then this indicates that both $P$ and $\neg P$ were produced, that is, the set of clauses that originated from the premises and negation of the conclusion is not satisfiable, that is, the original argument (theorem) is valid.
- If the output of DPP is no clause, no contradiction can be found, and the original argument (theorem) is not valid.
Lemma 1. DPP resolution preserves satisfiability, that is, if a set of clauses $S$ is satisfiable by a value assignment $\nu$, then any resolvent of a pair of clauses from $S$ is also satisfied by $\nu$.

Proof. Suppose $S$ is a set of clauses that is satisfied by $\nu$.

- Let $C_1 \lor L$ and $C_2 \lor \neg L$ be two members of $S$.
- If $\nu(L) = 0$, then since $\nu(C_1 \lor L) = 1$, we have $\nu(C_1) = 1$ and hence $\nu(C_1 \lor C_2) = 1$.
- If $\nu(\neg L) = 0$, then since $\nu(C_2 \lor \neg L) = 1$, we have $\nu(C_2) = 1$ and hence $\nu(C_1 \lor C_2) = 1$.
- In both cases $\nu(C_1 \lor C_2) = 1$, thus the resolvent $C_1 \lor C_2$ of the clauses $C_1 \lor L$ and $C_2 \lor \neg L$ is satisfied by $\nu$. 
Soundness and Completeness of DPP

Theorem [The DPP is sound and complete].

Let $S$ be a finite set of clauses. Then $S$ is not satisfiable iff the output of the Davis-Putnam procedure is the empty clause.

Proof. “$S \vdash \{\}$ by DPP” implies “$S$ not satisfiable” (soundness)

General Idea:

- We can use induction on $i$ to show that if $C$ is any clause in $S_i$ then there is a resolution derivation of $C$ from the initial $S$.
- Since the output to DPP is the empty clause, that is, $\{\} \in S_{n+1}$, it would follow that there is a resolution derivation from $S$ to $\{\}$.
- Since $\{\}$ is not satisfiable and resolution preserves satisfiability (Lemma 1), this would then imply that $S$ is not satisfiable.
How does “\( S \vdash \{\} \) by DPP” implies “\( S \) not satisfiable” prove soundness of DPP?

Take \( S = \Sigma \cup \{\neg A\} \), where \( \Sigma \) is a set of premises and \( A \) is a conclusion.

If \( \Sigma, \neg A \vdash \{\} \) by DPP resolution then, by the theorem we just “proved”, this implies that \( \Sigma \cup \neg A \) is not satisfiable

\( \Sigma \cup \neg A \) not satisfiable further implies \( \Sigma \models A \).

Thus, if we prove the validity of an argument formally, using DPP, then the argument is indeed correct, that is, DPP is sound.
Completeness of DPP resolution

“S not satisfiable” implies “S ⊢ {}” by DPP (completeness)

- How would this (if proved) show completeness of DPP?
- Assume we have a valid argument Σ |= A.
- This implies Σ ∪ {¬A} not satisfiable.
- By the above theorem, taking S = Σ ∪ {¬A}, this implies Σ, ¬A ⊢ {} by DPP resolution.
- This means that every correct argument can be proved valid by DPP resolution, that is, DPP is complete.
“S not satisfiable” implies “S ⊢ {}” by DPP” (completeness)

Proof by contradiction:

- Assume the output of the DPP is not the empty clause {}, but the empty set ∅ (the only other possibility).
- This means $S_{n+1} = ∅$, which is trivially satisfiable.
- We will prove that if $S_{i+1}$ is satisfiable then $S_i$ is satisfiable.
- In other words, satisfiability also propagates “backwards”.
- If proved, this would lead to a contradiction with our assumption that $S = S_1$ was not satisfiable, and complete the “completeness” part of the proof.
$S_{i+1}$ satisfiable implies $S_i$ satisfiable

- $S_{i+1}$ has variables $P_{i+1}, \ldots, P_n$
- $S_i$ has variables $P_i, P_{i+1}, \ldots, P_n$ (one extra variable, $P_i$, which is eliminated in iteration $i$ of DPP, that constructs $S_{i+1}$ from $S_i$)
- Clearly, $S_i$ is satisfiable iff $S'_i$ is satisfiable (all the deleted clauses contain complementary literals and are thus satisfiable)
- Since
  \[ S_{i+1} = (S'_i \setminus T_i) \cup U_i, \]
and $S_{i+1}$ is satisfiable, to show that $S_i$ is also satisfiable, it suffices to show that $T_i$ is satisfiable by an extension of the value assignment that satisfies $S_{i+1}$ to a value assignment that agrees with the one that satisfies $S_{i+1}$ on the variables $P_{i+1}, \ldots, P_n$, and which assigns a value also to $P_i$ (details on next slide).
Show that $T_i$ (parents with $P_i$) is satisfiable

- Assume $S_{i+1} = (S'_i \setminus T_i) \cup U_i$ is satisfied by some value assignment $v_{i+1}$ that assigns some values to $P_{i+1}, \ldots, P_n$.
- If neither of the assignments below satisfy $T_i$
  - $v_0$: agrees with $v_{i+1}$ for variables $P_{i+1}, \ldots P_n$, and $v_0(P_i) = 0$
  - $v_1$: agrees with $v_{i+1}$ for variables $P_{i+1}, \ldots P_n$, and $v_1(P_i) = 1$
- ...then since $v_0$ makes all formulas in $T_i$ that contain $\neg P_i$ true, it must falsify some clause $D \cup \{P_i\}$ in $T_i$.
- Thus, $D \cup \{P_i\}$ is not satisfied by $v_0$, which means $D$ is not satisfied by $v_{i+1} = v_0$ ($D$ does not contain variable $P_i$).
- ...also, since $v_1$ makes all formulas in $T_i$ that contain $P_i$ true, it must falsify some clause $E \cup \{\neg P_i\}$.
- Thus, $E \cup \{\neg P_i\}$ is not satisfied by $v_1$, which means $E$ is not satisfied by $v_{i+1} = v_1$ ($E$ does not contain the variable $P_i$).
- This further implies that $v_{i+1}$ does not satisfy $D \cup E$, but this is a contradiction, since $D \cup E \subseteq S_{i+1}$ and $v_{i+1}$ satisfies $S_{i+1}$.
- Thus, our assumption was false, and one of $v_0$ or $v_1$ satisfies $T_i$. 

Resolution for propositional logic

CS245, Logic and Computation 39 / 41
Conclusion of Completeness of DPP

- Since we assumed
  \[ S_{i+1} = (S'_i \setminus T_i) \cup U_i \]
  was satisfiable, and we proved that \( T_i \) is satisfied by an extension of one of the assignments that satisfies \( S_{i+1} \), we have that \( S_i \) is also satisfiable by that assignment.

- Recall that we had assumed (for the sake of contradiction) that \( S_{n+1} = \emptyset \), which is trivially satisfiable.

- Working backwards, this implies that \( S_1 = S \) is satisfiable, which contradicts the hypothesis of the theorem we have to prove, namely

  \[ \text{“} S \text{ not satisfiable” implies “} S \vdash \{\} \text{” by DPP”} \]

- Since we reached a contradiction, our assumption that \( “S \vdash \emptyset” \) by DPP was incorrect, and we have \( “S \vdash \{\} \text{” by DPP”} \) q.e.d.
Exercise

Use the **Davis-Putnam Procedure (DPP)** to show that the set of 12 clauses below is **not satisfiable**. (If the clauses originated from the premises and the negation of conclusion of an argument in propositional calculus, the set of clauses being not satisfiable implies that the argument is **valid**.) Eliminate the variables in the order $P, Q, R, S, T$.

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

$\{Q, \neg R, T\}$

$\{Q, R, \neg T\} \ {\neg Q, \neg R, \neg T}$

$\{\neg R, S\} \ {R, \neg S}$

$\{\neg P, S, T\} \ {P, \neg S, T}$

$\{P, S, \neg T\} \ {\neg P, \neg S, \neg T}$