

CS:4980 Topics in Computer Science II
Introduction to Automated Reasoning

Decision Procedures for Satisfiability
in Propositional Logic

Cesare Tinelli

Spring 2024



Credits

These slides are based on slides originally developed by **Cesare Tinelli** at the University of Iowa, and by **Clark Barrett, Caroline Trippel**, and **Andrew (Haoze) Wu** at Stanford University. Adapted by permission.

Decision procedures for propositional logic

From now on, instead of wffs, we **consider only their clausal form** (clause sets)

Decision procedures for propositional logic

Observe:

- Each clause $l_1 \vee \dots \vee l_n$ can be itself regarded as a set, of literals: $\{l_1, \dots, l_n\}$
- A set of clauses is satisfiable iff there is an interpretation of its variables that satisfies at least one literal in each clause

Decision procedures for propositional logic

Observe:

- Each clause $l_1 \vee \dots \vee l_n$ can be itself regarded as a set, of literals: $\{l_1, \dots, l_n\}$
- A set of clauses is satisfiable iff there is an interpretation of its variables that satisfies at least **one literal in each clause**

Decision procedures for propositional logic

Observe:

- Each clause $l_1 \vee \dots \vee l_n$ can be itself regarded as a set, of literals: $\{l_1, \dots, l_n\}$
- A set of clauses is satisfiable iff there is an interpretation of its variables that satisfies at least **one literal in each clause**

Example:

- The clause set $\Delta := \{p_1 \vee p_3, \neg p_1 \vee p_2 \vee \neg p_3\}$ can be represented as $\{\{p_1, p_3\}, \{\neg p_1, p_2, \neg p_3\}\}$
- $v := \{p_1 \mapsto \text{true}, p_2 \mapsto \text{true}, p_3 \mapsto \text{false}\}$ is a satisfying assignment for Δ

Decision procedures for propositional logic

Observe:

- Each clause $l_1 \vee \dots \vee l_n$ can be itself regarded as a set, of literals: $\{l_1, \dots, l_n\}$
- A set of clauses is satisfiable iff there is an interpretation of its variables that satisfies at least **one literal in each clause**

Observe:

- The **empty clause set** is trivially **satisfiable** (no constraints to satisfy)
- The **empty clause** is trivially **unsatisfiable** (no options to choose)

Decision procedures for propositional logic

Observe:

- Each clause $l_1 \vee \dots \vee l_n$ can be itself regarded as a set, of literals: $\{l_1, \dots, l_n\}$
- A set of clauses is satisfiable iff there is an interpretation of its variables that satisfies at least **one literal in each clause**

Observe:

- The **empty clause set** is trivially **satisfiable** (no constraints to satisfy)
- The **empty clause** is trivially **unsatisfiable** (no options to choose)

SAT Solver Overview: features

Automated reasoners for the satisfiability problem in PL are called *SAT solvers*

1. Backtracking search solvers

- Traversing and backtracking on a binary tree
- Sound, complete and terminating

2. Stochastic search solvers

- Solver guesses a full assignment v
- If the set is falsified by v , starts to flip values of variables according to some (greedy) heuristic
- Sound but neither complete nor terminating
- Nevertheless, quite effective in certain applications

SAT Solver Overview: features

There are **two main categories of modern SAT solvers**, both working with clause sets:

1. Backtracking search solvers

- Traversing and backtracking on a binary tree
- Sound, complete and terminating

2. Stochastic search solvers

- Solver guesses a full assignment v
- If the set is falsified by v , starts to flip values of variables according to some (greedy) heuristic
- Sound but neither complete nor terminating
- Nevertheless, quite effective in certain applications

SAT Solver Overview: features

There are **two main categories of modern SAT solvers**, both working with clause sets:

1. Backtracking search solvers

- **Traversing** and **backtracking** on a binary tree
- **Sound**, **complete** and **terminating**

2. Stochastic search solvers

- Solver guesses a full assignment v
- If the set is falsified by v , starts to flip values of variables according to some (greedy) heuristic
- **Sound** but **neither complete nor terminating**
- Nevertheless, quite effective in certain applications

SAT Solver Overview: features

There are **two main categories of modern SAT solvers**, both working with clause sets:

1. Backtracking search solvers

- **Traversing** and **backtracking** on a binary tree
- **Sound**, **complete** and **terminating**

2. Stochastic search solvers

- Solver guesses a full assignment v
- If the set is falsified by v , starts to flip values of variables according to some (greedy) heuristic
- **Sound** but **neither complete nor terminating**
- Nevertheless, quite effective in certain applications

SAT Solver Overview: features

There are **two main categories of modern SAT solvers**, both working with clause sets:

1. Backtracking search solvers

- **Traversing** and **backtracking** on a binary tree
- **Sound, complete and terminating**

2. Stochastic

We focus on backtracking solvers in this course

- Solver guesses a full assignment v
- If the set is falsified by v , starts to flip values of variables according to some (greedy) heuristic
- **Sound** but **neither complete nor terminating**
- Nevertheless, quite effective in certain applications

SAT Solver Overview: performance

The SAT problem is **hard** (NP-complete). How well do SAT solvers do in practice?

- Modern SAT solvers can solve many real-life CNF formulas with **hundreds of thousands or even millions of variables** in a reasonable amount of time
- There are also instances of problems two orders of magnitude smaller that the same tools cannot solve
- In general, it is very **hard to predict** which instance is going to be hard to solve, without actually attempting to solve it

SAT portfolio solvers: use machine-learning techniques to extract features of CNF formulas in order to select the most suitable SAT solver for the job

SAT Solver Overview: performance

The SAT problem is **hard** (NP-complete). How well do SAT solvers do in practice?

- Modern SAT solvers can solve many real-life CNF formulas with **hundreds of thousands or even millions of variables** in a reasonable amount of time
- There are also instances of problems two orders of magnitude smaller that the same tools cannot solve
- In general, it is very **hard to predict** which instance is going to be hard to solve, without actually attempting to solve it

SAT portfolio solvers: use machine-learning techniques to extract features of CNF formulas in order to select the most suitable SAT solver for the job

SAT Solver Overview: performance

The SAT problem is **hard** (NP-complete). How well do SAT solvers do in practice?

- Modern SAT solvers can solve many real-life CNF formulas with **hundreds of thousands or even millions of variables** in a reasonable amount of time
- There are also instances of problems two orders of magnitude smaller that the same tools cannot solve
- In general, it is very **hard to predict** which instance is going to be hard to solve, without actually attempting to solve it

SAT portfolio solvers: use machine-learning techniques to extract features of CNF formulas in order to select the most suitable SAT solver for the job

SAT Solver Overview: performance

The SAT problem is **hard** (NP-complete). How well do SAT solvers do in practice?

- Modern SAT solvers can solve many real-life CNF formulas with **hundreds of thousands or even millions of variables** in a reasonable amount of time
- There are also instances of problems two orders of magnitude smaller that the same tools cannot solve
- In general, it is very **hard to predict** which instance is going to be hard to solve, without actually attempting to solve it

SAT portfolio solvers: use machine-learning techniques to extract features of CNF formulas in order to select the most suitable SAT solver for the job

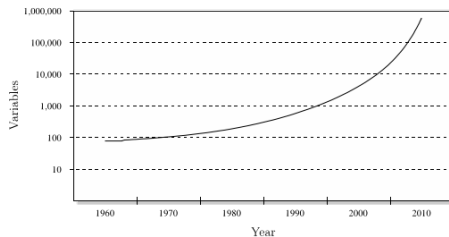
SAT Solver Overview: performance

The SAT problem is **hard** (NP-complete). How well do SAT solvers do in practice?

- Modern SAT solvers can solve many real-life CNF formulas with **hundreds of thousands or even millions of variables** in a reasonable amount of time
- There are also instances of problems two orders of magnitude smaller that the same tools cannot solve
- In general, it is very **hard to predict** which instance is going to be hard to solve, without actually attempting to solve it

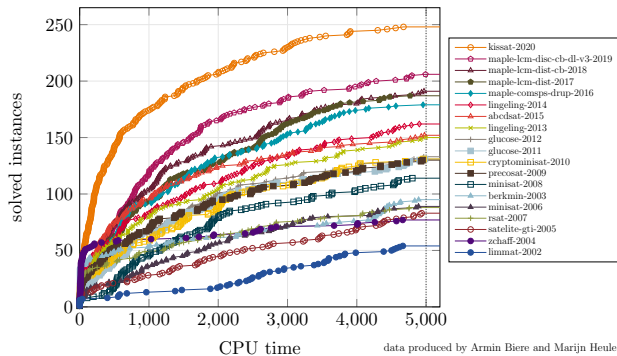
SAT portfolio solvers: use machine-learning techniques to extract features of CNF formulas in order to select the most suitable SAT solver for the job

SAT Solver Overview: performance



Left: Size of industrial clause sets (y-axis) regularly solved by solvers in a few hours each year (x-axis). Instances come from realistic problems like planning or hardware verification

SAT Competition Winners on the SC2020 Benchmark Suite



Right: Top contenders in SAT solver competitions from 2002 to 2020; each point shows number of solved instances (y-axis) per unit of time (x-axis). Note that no. of instances solved within 20 minutes more than doubled in less than a decade

SAT Solver Overview: performance

Success of SAT solvers can largely be attributed to their ability to:

- Learn from failed assignments
- Prune large parts of the search spaces quickly
- Focus first on important variables

SAT Solver Overview: performance

Success of SAT solvers can largely be attributed to their ability to:

- Learn from failed assignments
- Prune large parts of the search spaces quickly
- Focus first on important variables

SAT Solver Overview: performance

Success of SAT solvers can largely be attributed to their ability to:

- Learn from failed assignments
- Prune large parts of the search spaces quickly
- Focus first on important variables

The DIMACS format

A standard format for clause sets accepted by most modern SAT solvers

The DIMACS format

- Comment lines: Start with a lower-case letter *c*
- Problem line: *p cnf* *<#variables > <#clauses >*
- Clause lines:
 - Each variable is assigned a unique index *i* greater than 0
 - A positive literal is represented by an index
 - A negative literal is represented by the negation of its complement's index
 - A clause is represented as a list of literals separated by white space
 - Value 0 is used to mark the end of a clause

The DIMACS format

- Comment lines: Start with a lower-case letter *c*
- Problem line: *p cnf* *<#variables>* *<#clauses>*
- Clause lines:
 - Each variable is assigned a unique index *i* greater than 0
 - A positive literal is represented by an index
 - A negative literal is represented by the negation of its complement's index
 - A clause is represented as a list of literals separated by white space
 - Value 0 is used to mark the end of a clause

Example:

$\{p_1 \vee \neg p_3, p_2 \vee p_3 \vee \neg p_1\}$



```
c example.cnf
p cnf 3 2
1 -3 0
2 3 -1 0
```

Basic SAT solvers

- 1960: *Davis-Putnam (DP)* algorithm
- 1961: *Davis-Putnam-Logemann-Loveland (DPLL)* algorithm
- 1996: Modern SAT solver based on *Conflict-Driven Clause Learning (CDCL)*
derived from DP and DPLL

Basic SAT solvers

- 1960: *Davis-Putnam (DP)* algorithm
- 1961: *Davis-Putnam-Logemann-Loveland (DPLL)* algorithm
- 1996: Modern SAT solver based on *Conflict-Driven Clause Learning (CDCL)* derived from DP and DPLL

A proof system for clause sets: resolution

There is a **refutation sound and complete** proof system for clause sets Δ that consists of just **one proof rule!**

$$\text{RESOLVE} \frac{C_1, C_2 \in \Delta \quad p \in C_1 \quad \neg p \in C_2 \quad C = (C_1 \setminus \{p\}) \cup (C_2 \setminus \{\neg p\}) \quad C \notin \Delta}{\Delta \cup \{C\}}$$

Clause C is a *(p-)resolvent* of C_1 and C_2 , and p is the *pivot*

Example: $\Delta := \{ \{p_1, p_3\}, \{p_2, \neg p_3\} \}$ has a p_3 -resolvent: $\{p_1, p_2\}$

A proof system for clause sets: resolution

There is a **refutation sound and complete** proof system for clause sets Δ that consists of just **one proof rule!**

$$\text{RESOLVE} \frac{C_1, C_2 \in \Delta \quad p \in C_1 \quad \neg p \in C_2 \quad C = (C_1 \setminus \{p\}) \cup (C_2 \setminus \{\neg p\}) \quad C \notin \Delta}{\Delta \cup \{C\}}$$

Clause C is a *(p-)resolvent* of C_1 and C_2 , and p is the *pivot*

Example: $\Delta := \{ \{p_1, p_3\}, \{p_2, \neg p_3\} \}$ has a p_3 -resolvent: $\{p_1, p_2\}$

A proof system for clause sets: resolution

There is a **refutation sound and complete** proof system for clause sets Δ that consists of just **one proof rule!**

$$\text{RESOLVE} \frac{C_1, C_2 \in \Delta \quad p \in C_1 \quad \neg p \in C_2 \quad C = (C_1 \setminus \{p\}) \cup (C_2 \setminus \{\neg p\}) \quad C \notin \Delta}{\Delta \cup \{C\}}$$

Clause C is a $(p\text{-})$ resolvent of C_1 and C_2 , and p is the *pivot*

Example: $\Delta := \{ \{p_1, p_3\}, \{p_2, \neg p_3\} \}$ has a p_3 -resolvent: $\{p_1, p_2\}$

A proof system for clause sets: resolution

There is a **refutation sound and complete** proof system for clause sets Δ that consists of just **one proof rule!**

$$\text{RESOLVE} \frac{C_1, C_2 \in \Delta \quad p \in C_1 \quad \neg p \in C_2 \quad C = (C_1 \setminus \{p\}) \cup (C_2 \setminus \{\neg p\}) \quad C \notin \Delta}{\Delta \cup \{C\}}$$

Clause C is a (p) -*resolvent* of C_1 and C_2 , and p is the *pivot*

Example: $\Delta := \{ \{p_1, p_3\}, \{p_2, \neg p_3\} \}$ has a p_3 -resolvent: $\{p_1, p_2\}$

A proof system for clause sets: resolution

There is a **refutation sound and complete** proof system for clause sets Δ that consists of just **one proof rule!**

$$\text{RESOLVE} \frac{C_1, C_2 \in \Delta \quad p \in C_1 \quad \neg p \in C_2 \quad C = (C_1 \setminus \{p\}) \cup (C_2 \setminus \{\neg p\}) \quad C \notin \Delta}{\Delta \cup \{C\}}$$

Clause C is a (p) -*resolvent* of C_1 and C_2 , and p is the *pivot*

Example: $\Delta := \{ \{p_1, p_3\}, \{p_2, \neg p_3\} \}$ has a p_3 -resolvent: $\{p_1, p_2\}$

Note: if C is a resolvent of $C_1, C_2 \in \Delta$ then $\{C_1, C_2\} \models C$

A proof system for clause sets: resolution

There is a **refutation sound and complete** proof system for clause sets Δ that consists of just **one proof rule**!

$$\text{RESOLVE} \frac{C_1, C_2 \in \Delta \quad p \in C_1 \quad \neg p \in C_2 \quad C = (C_1 \setminus \{p\}) \cup (C_2 \setminus \{\neg p\}) \quad C \notin \Delta}{\Delta \cup \{C\}}$$

Clause C is a (p) -*resolvent* of C_1 and C_2 , and p is the *pivot*

Example: $\Delta := \{ \{p_1, p_3\}, \{p_2, \neg p_3\} \}$ has a p_3 -resolvent: $\{p_1, p_2\}$

Note: if C is a resolvent of $C_1, C_2 \in \Delta$ then $\{C_1, C_2\} \models C$ and so $\Delta \models \Delta \cup \{C\}$

Proofs by resolution example

Prove that the following clause set is unsatisfiable

$$\begin{array}{l} \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\}, \{ \} \} \end{array}$$

- The last clause set is unsatisfiable since it contains the empty clause $\{ \}$
- Since every clause set entails the next, it must be that the first one is unsatisfiable

Proofs by resolution example

Prove that the following clause set is unsatisfiable

$$\begin{array}{l} \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\}, \{ \} \} \end{array}$$

- The last clause set is unsatisfiable since it contains the empty clause $\{ \}$
- Since every clause set entails the next, it must be that the first one is unsatisfiable

Proofs by resolution example

Prove that the following clause set is unsatisfiable

$$\begin{array}{l} \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\}, \{ \} \} \end{array}$$

- The last clause set is unsatisfiable since it contains the empty clause $\{ \}$
- Since every clause set entails the next, it must be that the first one is unsatisfiable

Proofs by resolution example

Prove that the following clause set is unsatisfiable

$$\begin{array}{l} \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\}, \{ \} \} \end{array}$$

- The last clause set is unsatisfiable since it contains the empty clause $\{ \}$
- Since every clause set entails the next, it must be that the first one is unsatisfiable

Proofs by resolution example

Prove that the following clause set is unsatisfiable

$$\begin{array}{l} \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\}, \{ \} \} \end{array}$$

- The last clause set is unsatisfiable since it contains the empty clause $\{ \}$
- Since every clause set entails the next, it must be that the first one is unsatisfiable

Proofs by resolution example

Prove that the following clause set is unsatisfiable

$$\begin{array}{l} \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\}, \{\} \} \end{array}$$

- The last clause set is unsatisfiable since it contains the empty clause $\{\}$
- Since every clause set entails the next, it must be that the first one is unsatisfiable

Proofs by resolution example

Prove that the following clause set is unsatisfiable

$$\begin{array}{l} \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\}, \{\} \} \end{array}$$

- The last clause set is unsatisfiable since it contains the empty clause $\{\}$
- Since every clause set entails the next, it must be that the first one is unsatisfiable

Proofs by resolution example

Prove that the following clause set is unsatisfiable

$$\begin{array}{l} \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\}, \{ \} \} \end{array}$$

- The last clause set is unsatisfiable since it contains the empty clause $\{ \}$
- Since every clause set entails the next, it must be that the first one is unsatisfiable

Proofs by resolution example

Prove that the following clause set is unsatisfiable

$$\begin{array}{l} \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\}, \{\} \} \end{array}$$

- The last clause set is unsatisfiable since it contains the empty clause $\{\}$
- Since every clause set entails the next, it must be that the first one is unsatisfiable

Proofs by resolution example

Prove that the following clause set is unsatisfiable

$$\begin{array}{l} \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\}, \{\} \} \end{array}$$

- The last clause set is unsatisfiable since it contains the empty clause $\{\}$
 - Since every clause set entails the next, it must be that the first one is unsatisfiable

Proofs by resolution example

Prove that the following clause set is unsatisfiable

$$\begin{array}{l} \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\} \} \\ \hline \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\}, \{p_1\}, \{p_3\}, \{\neg p_3\}, \{\} \} \end{array}$$

- The last clause set is unsatisfiable since it contains the empty clause $\{\}$
- Since every clause set entails the next, it must be that the first one is unsatisfiable

A resolution-based satisfiability proof system

- In addition to the SAT and UNSAT states, we consider states of the form

$$\langle \Delta, \Phi \rangle$$

with Δ and Φ clause sets

- Initial states have the form

$$\langle \Delta_0, \{\} \rangle$$

where Δ_0 is the clause set to be checked for satisfiability

A resolution-based satisfiability proof system

We modify the resolution rule **RESOLVE** as highlighted below and add three more rules

$$\mathbf{RESOLVE} \frac{C_1, C_2 \in \Delta \quad p \in C_1 \quad \neg p \in C_2 \quad C = (C_1 \setminus \{p\}) \cup (C_2 \setminus \{\neg p\}) \quad C \notin \Delta \cup \Phi}{\Delta := \Delta \cup \{C\}}$$

$$\mathbf{CLASH} \frac{C \in \Delta \quad p, \neg p \in C}{\Delta := \Delta \setminus \{C\} \quad \Phi := \Phi \cup \{C\}}$$

$$\mathbf{UNSAT} \frac{\{\} \in \Delta}{\mathbf{UNSAT}}$$

$$\mathbf{SAT} \frac{\text{No other rules apply}}{\mathbf{SAT}}$$

This proof system is sound, complete and terminating

A resolution-based satisfiability proof system

We modify the resolution rule **RESOLVE** as highlighted below and add three more rules

$$\text{RESOLVE} \frac{C_1, C_2 \in \Delta \quad p \in C_1 \quad \neg p \in C_2 \quad C = (C_1 \setminus \{p\}) \cup (C_2 \setminus \{\neg p\}) \quad C \notin \Delta \cup \Phi}{\Delta := \Delta \cup \{C\}}$$

$$\text{CLASH} \frac{C \in \Delta \quad p, \neg p \in C}{\Delta := \Delta \setminus \{C\} \quad \Phi := \Phi \cup \{C\}}$$

$$\text{UNSAT} \frac{\{\} \in \Delta}{\text{UNSAT}}$$

$$\text{SAT} \frac{\text{No other rules apply}}{\text{SAT}}$$

This proof system is **sound**, **complete** and **terminating**

A resolution-based decision procedure

Given a clause set Δ , apply **CLASH** or **RESOLVE** until either

1. an empty clause is derived (return **UNSAT**)
2. neither applies (return **SAT**)

This procedure is terminating and decides the SAT problem

A resolution-based decision procedure

Given a clause set Δ , apply **CLASH** or **RESOLVE** until either

1. an empty clause is derived (return **UNSAT**)
2. neither applies (return **SAT**)

This procedure is terminating and decides the SAT problem

Unit resolution

Notation

If l is a literal and p is its variable, $\bar{l} = \begin{cases} \neg p & \text{if } l = p \\ p & \text{if } l = \neg p \end{cases}$

The *unit resolution rule* is a special case of resolution where one of the resolving clauses is a *unit clause*, i.e., a clause with only one literal

$$\text{UNIT RESOLVE } \frac{C_1, C_2 \in \Delta \quad C_1 = \{l\} \quad C_2 = \{\bar{l}\} \cup D}{\Delta \cup \{D\}}$$

Unit resolution

Notation

If l is a literal and p is its variable, $\bar{l} = \begin{cases} \neg p & \text{if } l = p \\ p & \text{if } l = \neg p \end{cases}$

The *unit resolution rule* is a special case of resolution where one of the resolving clauses is a *unit clause*, i.e., a clause with only one literal

$$\text{UNIT RESOLVE } \frac{C_1, C_2 \in \Delta \quad C_1 = \{l\} \quad C_2 = \{\bar{l}\} \cup D}{\Delta \cup \{D\}}$$

Unit resolution

Notation

If l is a literal and p is its variable, $\bar{l} = \begin{cases} \neg p & \text{if } l = p \\ p & \text{if } l = \neg p \end{cases}$

The *unit resolution rule* is a special case of resolution where one of the resolving clauses is a *unit clause*, i.e., a clause with only one literal

$$\text{UNIT RESOLVE } \frac{C_1, C_2 \in \Delta \quad C_1 = \{l\} \quad C_2 = \{\bar{l}\} \cup D}{\Delta \cup \{D\}}$$

A proof system with unit resolution alone is **not refutation-complete**
(consider an unsat Δ with no unit clauses)

Unit resolution

Notation

If l is a literal and p is its variable, $\bar{l} = \begin{cases} \neg p & \text{if } l = p \\ p & \text{if } l = \neg p \end{cases}$

The *unit resolution rule* is a special case of resolution where one of the resolving clauses is a *unit clause*, i.e., a clause with only one literal

$$\text{UNIT RESOLVE } \frac{C_1, C_2 \in \Delta \quad C_1 = \{l\} \quad C_2 = \{\bar{l}\} \cup D}{\Delta \cup \{D\}}$$

Modern SAT solvers use **unit resolution** **plus backtracking search** for deciding SAT

Davis-Putnam (DP) procedure

A decision procedure for the SAT problem

First procedure to implement something more sophisticated than truth tables

DP leverages 4 satisfiability-preserving transformations:

Davis-Putnam (DP) procedure

A decision procedure for the SAT problem

First procedure to implement something more sophisticated than truth tables

DP leverages 4 satisfiability-preserving transformations:

Davis-Putnam (DP) procedure

A decision procedure for the SAT problem

First procedure to implement something more sophisticated than truth tables

DP leverages 4 **satisfiability-preserving** transformations:

- Unit propagation
- Pure literal elimination
- Tautology elimination
- Exhaustive resolution

Davis-Putnam (DP) procedure

A decision procedure for the SAT problem

First procedure to implement something more sophisticated than truth tables

DP leverages 4 **satisfiability-preserving** transformations:

- Unit propagation
- Pure literal elimination
- Tautology elimination
- Exhaustive resolution

Davis-Putnam (DP) procedure

A decision procedure for the SAT problem

First procedure to implement something more sophisticated than truth tables

DP leverages 4 **satisfiability-preserving** transformations:

- Unit propagation
- Pure literal elimination
- Tautology elimination
- Exhaustive resolution

Davis-Putnam (DP) procedure

A decision procedure for the SAT problem

First procedure to implement something more sophisticated than truth tables

DP leverages 4 **satisfiability-preserving** transformations:

- Unit propagation
- Pure literal elimination
- Tautology elimination
- Exhaustive resolution

Davis-Putnam (DP) procedure

A decision procedure for the SAT problem

First procedure to implement something more sophisticated than truth tables

DP leverages 4 **satisfiability-preserving** transformations:

- Unit propagation
- Pure literal elimination
- Tautology elimination
- Exhaustive resolution

Davis-Putnam (DP) procedure

A decision procedure for the SAT problem

First procedure to implement something more sophisticated than truth tables

DP leverages 4 **satisfiability-preserving** transformations:

- Unit propagation
- Pure literal elimination
- Tautology elimination
- Exhaustive resolution

The **first two** transformations **reduce** the total number of **literals** in the clause set

Davis-Putnam (DP) procedure

A decision procedure for the SAT problem

First procedure to implement something more sophisticated than truth tables

DP leverages 4 **satisfiability-preserving** transformations:

- Unit propagation
- Pure literal elimination
- Tautology elimination
- Exhaustive resolution

The **first two** transformations **reduce** the total number of **literals** in the clause set

The **third** transformation **reduces** the number of **clauses**

Davis-Putnam (DP) procedure

A decision procedure for the SAT problem

First procedure to implement something more sophisticated than truth tables

DP leverages 4 **satisfiability-preserving** transformations:

- Unit propagation
- Pure literal elimination
- Tautology elimination
- Exhaustive resolution

Repeatedly applying these transformations, eventually leads to an **empty clause** (indicating **unsatisfiability**) or an **empty clause set** (indicating **satisfiability**)

DP procedure: unit propagation

Also called the *1-literal rule*

Premise: The clause set Δ contains a unit clause $C = \{l\}$

Conclusion:

- Remove all occurrences of \bar{l} from clauses in Δ
- Remove all clauses containing l (including C)

DP procedure: unit propagation

Also called the *1-literal rule*

Premise: The clause set Δ contains a **unit clause** $C = \{l\}$

Conclusion:

- Remove all occurrences of \bar{l} from clauses in Δ
- Remove all clauses containing l (including C)

DP procedure: unit propagation

Also called the *1-literal rule*

Premise: The clause set Δ contains a **unit clause** $C = \{l\}$

Conclusion:

- Remove all occurrences of \bar{l} from clauses in Δ
- Remove all clauses containing l (including C)

DP procedure: unit propagation

Also called the *1-literal rule*

Premise: The clause set Δ contains a **unit clause** $C = \{l\}$

Conclusion:

- Remove all occurrences of \bar{l} from clauses in Δ
- Remove all clauses containing l (including C)

Justification: The only way to satisfy C is to make l true; thus, (i) \bar{l} cannot be used to satisfy any clause, and (ii) any clause containing l is satisfied and can be ignored

DP procedure: unit propagation

Also called the *1-literal rule*

Premise: The clause set Δ contains a **unit clause** $C = \{l\}$

Conclusion:

- Remove all occurrences of \bar{l} from clauses in Δ
- Remove all clauses containing l (including C)

Example:

$$\Delta_0 := \{ \{p_1\}, \{p_1, p_4\}, \{p_2, p_3, \neg p_1\} \}$$

$$\Delta_1 := \{ \{p_4\}, \{p_2, p_3\} \}$$

(unit propagation on p_1)

$$\Delta_2 := \{ \{p_2, p_3\} \}$$

(unit propagation on p_4)

DP procedure: unit propagation

Also called the *1-literal rule*

Premise: The clause set Δ contains a **unit clause** $C = \{l\}$

Conclusion:

- Remove all occurrences of \bar{l} from clauses in Δ
- Remove all clauses containing l (including C)

Example:

$$\Delta_0 := \{ \{p_1\}, \{p_1, p_4\}, \{p_2, p_3, \neg p_1\} \}$$

$$\Delta_1 := \{ \{p_4\}, \{p_2, p_3\} \}$$

$$\Delta_2 := \{ \{p_2, p_3\} \}$$

(unit propagation on p_1)

(unit propagation on p_4)

DP procedure: unit propagation

Also called the *1-literal rule*

Premise: The clause set Δ contains a **unit clause** $C = \{l\}$

Conclusion:

- Remove all occurrences of \bar{l} from clauses in Δ
- Remove all clauses containing l (including C)

Example:

$$\Delta_0 := \{ \{p_1\}, \{p_1, p_4\}, \{p_2, p_3, \neg p_1\} \}$$

$$\Delta_1 := \{ \{p_4\}, \{p_2, p_3\} \}$$

(unit propagation on p_1)

$$\Delta_2 := \{ \{p_2, p_3\} \}$$

(unit propagation on p_4)

DP procedure: pure literal elimination

Also called the *affirmation-negation rule*

Premise: A literal l occurs in Δ but \bar{l} does not

Conclusion: Delete all clauses containing l

DP procedure: pure literal elimination

Also called the *affirmation-negation rule*

Premise: A literal l occurs in Δ but \bar{l} does not

Conclusion: Delete all clauses containing l

DP procedure: pure literal elimination

Also called the *affirmation-negation rule*

Premise: A literal l occurs in Δ but \bar{l} does not

Conclusion: Delete all clauses containing l

DP procedure: pure literal elimination

Also called the *affirmation-negation rule*

Premise: A literal l occurs in Δ but \bar{l} does not

Conclusion: Delete all clauses containing l

Justification: For every assignment that satisfies Δ there is one that satisfies both Δ and l ; thus, all clauses containing l can be deleted since they can always be satisfied

DP procedure: pure literal elimination

Also called the *affirmation-negation rule*

Premise: A literal l occurs in Δ but \bar{l} does not

Conclusion: Delete all clauses containing l

Example:

$$\Delta_0 := \{ \{ p_1, p_2, \neg p_3 \}, \{ \neg p_1, p_4 \}, \{ \neg p_3, \neg p_2 \}, \{ \neg p_3, \neg p_4 \} \}$$

$$\Delta_1 := \{ \{ \neg p_1, p_4 \} \}$$

DP procedure: pure literal elimination

Also called the *affirmation-negation rule*

Premise: A literal l occurs in Δ but \bar{l} does not

Conclusion: Delete all clauses containing l

Example:

$$\Delta_0 := \{ \{ p_1, p_2, \neg p_3 \}, \{ \neg p_1, p_4 \}, \{ \neg p_3, \neg p_2 \}, \{ \neg p_3, \neg p_4 \} \}$$

$$\Delta_1 := \{ \{ \neg p_1, p_4 \} \}$$

DP procedure: tautology elimination

Also called the *clashing clause rule*

Premise: a clause $C \in \Delta$ contains both p and $\neg p$

Conclusion: remove C from Δ

Justification: C is satisfied by every variable assignment

DP procedure: tautology elimination

Also called the *clashing clause rule*

Premise: a clause $C \in \Delta$ contains both p and $\neg p$

Conclusion: remove C from Δ

Justification: C is satisfied by every variable assignment

DP procedure: tautology elimination

Also called the *clashing clause rule*

Premise: a clause $C \in \Delta$ contains both p and $\neg p$

Conclusion: remove C from Δ

Justification: C is satisfied by every variable assignment

DP procedure: tautology elimination

Also called the *clashing clause rule*

Premise: a clause $C \in \Delta$ contains both p and $\neg p$

Conclusion: remove C from Δ

Justification: C is satisfied by every variable assignment

DP procedure: resolution

Also called the *rule for eliminating atomic formulas*

Premise: A variable p occurs in a clause of Δ and $\neg p$ occurs in another clause

Conclusion:

DP procedure: resolution

Also called the *rule for eliminating atomic formulas*

Premise: A variable p occurs in a clause of Δ and $\neg p$ occurs in another clause

Conclusion:

DP procedure: resolution

Also called the *rule for eliminating atomic formulas*

Premise: A variable p occurs in a clause of Δ and $\neg p$ occurs in another clause

Conclusion:

- Let P be the set of clauses in Δ where p occurs positively and let N be the set of clauses in Δ where p occurs negatively
- Replace the clauses in P and N with those obtained by resolution on p using all pairs of clauses from $P \times N$

DP procedure: resolution

Also called the *rule for eliminating atomic formulas*

Premise: A variable p occurs in a clause of Δ and $\neg p$ occurs in another clause

Conclusion:

- Let P be the set of clauses in Δ where p occurs positively and let N be the set of clauses in Δ where p occurs negatively
- Replace the clauses in P and N with those obtained by resolution on p using all pairs of clauses from $P \times N$

DP procedure: resolution

Also called the *rule for eliminating atomic formulas*

Premise: A variable p occurs in a clause of Δ and $\neg p$ occurs in another clause

Conclusion:

- Let P be the set of clauses in Δ where p occurs positively and let N be the set of clauses in Δ where p occurs negatively
- Replace the clauses in P and N with those obtained by resolution on p using all pairs of clauses from $P \times N$

Example:

$$\Delta_0 := \{ \{ p_1, p_2 \}, \{ \neg p_1, p_3 \}, \{ \neg p_1, \neg p_3, p_4 \}, \{ p_2, \neg p_4 \} \}$$

$$\Delta_1 := \{ \{ p_2, p_3 \}, \{ p_2, \neg p_3, p_4 \}, \{ p_2, \neg p_4 \} \}$$

(resolution on p_1)

DP procedure: resolution

Also called the *rule for eliminating atomic formulas*

Premise: A variable p occurs in a clause of Δ and $\neg p$ occurs in another clause

Conclusion:

- Let P be the set of clauses in Δ where p occurs positively and let N be the set of clauses in Δ where p occurs negatively
- Replace the clauses in P and N with those obtained by resolution on p using all pairs of clauses from $P \times N$

Example:

$$\Delta_0 := \{ \{ p_1, p_2 \}, \{ \neg p_1, p_3 \}, \{ \neg p_1, \neg p_3, p_4 \}, \{ p_2, \neg p_4 \} \}$$

$$\Delta_1 := \{ \{ p_2, p_3 \}, \{ p_2, \neg p_3, p_4 \}, \{ p_2, \neg p_4 \} \}$$

(resolution on p_1)

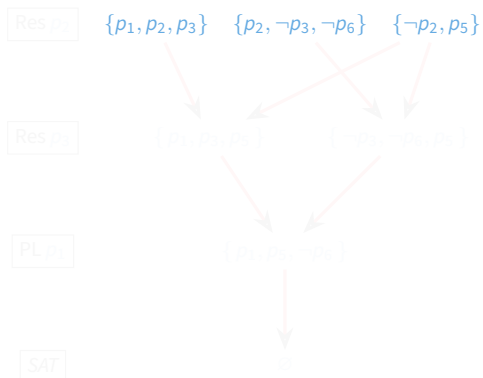
DP Example 1

$$\Delta := \{ \{p_1, p_2, p_3\}, \{p_2, \neg p_3, \neg p_6\}, \{\neg p_2, p_5\} \}$$



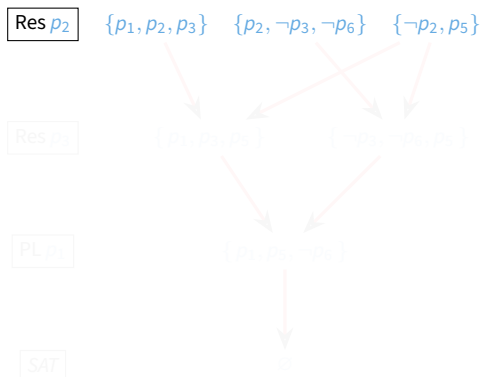
DP Example 1

$$\Delta := \{ \{p_1, p_2, p_3\}, \{p_2, \neg p_3, \neg p_6\}, \{\neg p_2, p_5\} \}$$



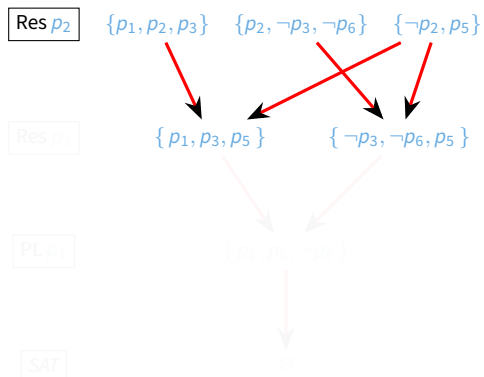
DP Example 1

$$\Delta := \{ \{p_1, p_2, p_3\}, \{p_2, \neg p_3, \neg p_6\}, \{\neg p_2, p_5\} \}$$



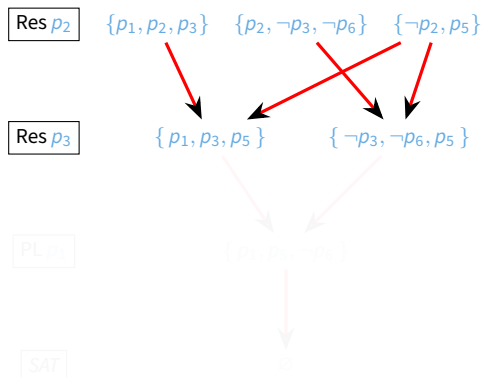
DP Example 1

$$\Delta := \{ \{p_1, p_2, p_3\}, \{p_2, \neg p_3, \neg p_6\}, \{\neg p_2, p_5\} \}$$



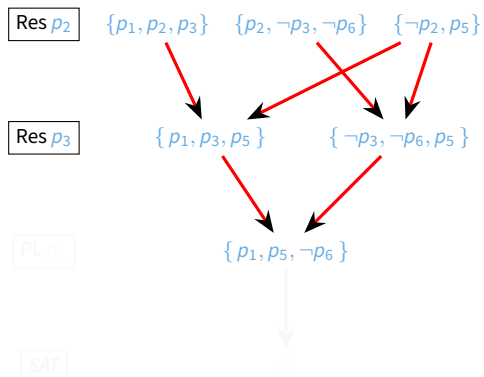
DP Example 1

$$\Delta := \{ \{p_1, p_2, p_3\}, \{p_2, \neg p_3, \neg p_6\}, \{\neg p_2, p_5\} \}$$



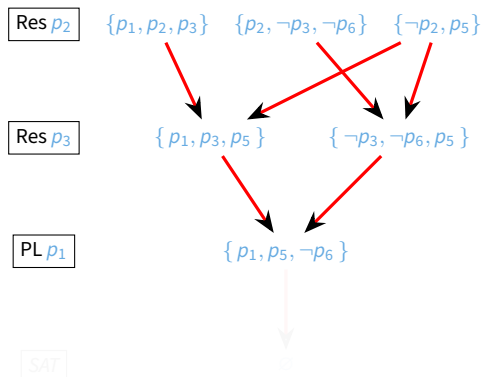
DP Example 1

$$\Delta := \{ \{p_1, p_2, p_3\}, \{p_2, \neg p_3, \neg p_6\}, \{\neg p_2, p_5\} \}$$



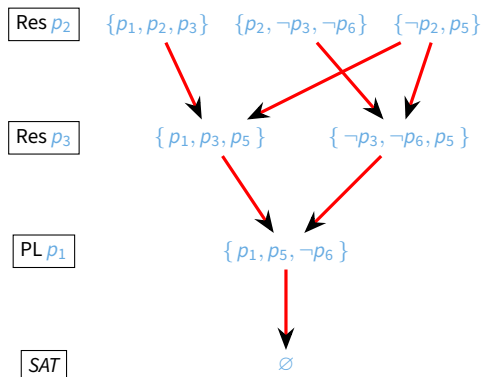
DP Example 1

$$\Delta := \{ \{p_1, p_2, p_3\}, \{p_2, \neg p_3, \neg p_6\}, \{\neg p_2, p_5\} \}$$



DP Example 1

$$\Delta := \{ \{p_1, p_2, p_3\}, \{p_2, \neg p_3, \neg p_6\}, \{\neg p_2, p_5\} \}$$



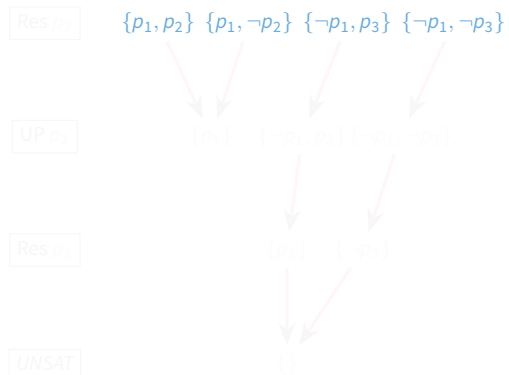
DP Example 2

$$\Delta := \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \}$$



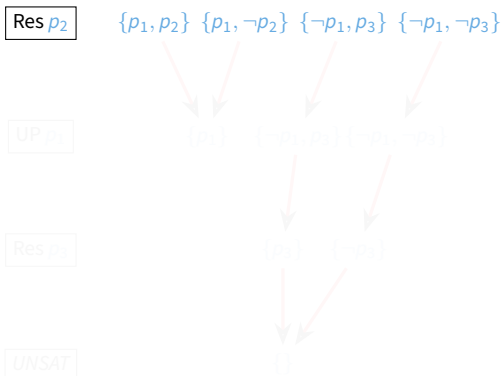
DP Example 2

$$\Delta := \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \}$$



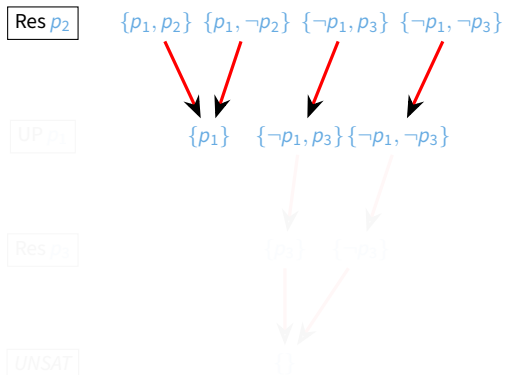
DP Example 2

$$\Delta := \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \}$$



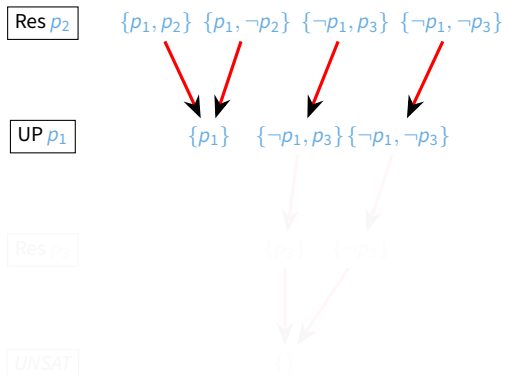
DP Example 2

$$\Delta := \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \}$$



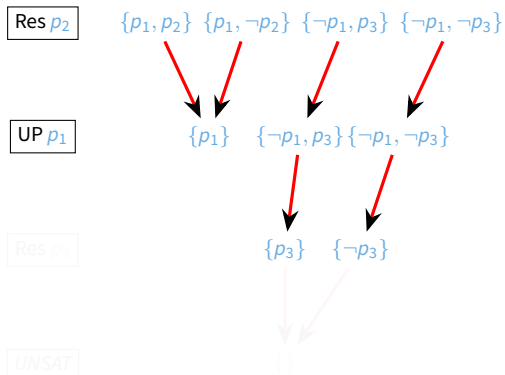
DP Example 2

$$\Delta := \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \}$$



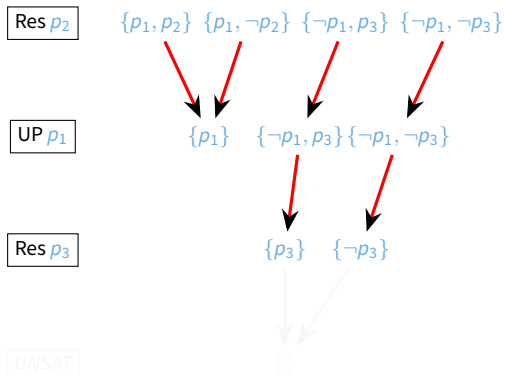
DP Example 2

$$\Delta := \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \}$$



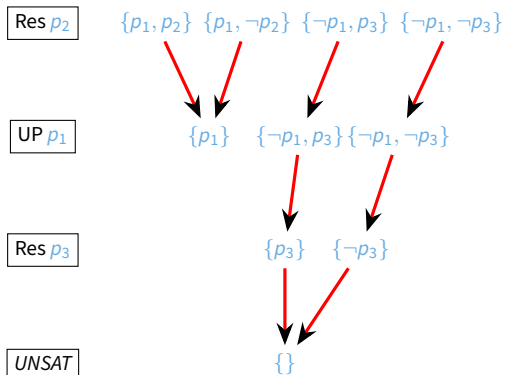
DP Example 2

$$\Delta := \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \}$$



DP Example 2

$$\Delta := \{ \{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_3\}, \{\neg p_1, \neg p_3\} \}$$



From DP to DPLL

The resolution transformation does not increase the number of variables
However, it **may increase** the size of the **clause set**

Question: If a variable appears positively in 3 clauses and negatively in 3 clauses, how many clauses after applying resolution? 9

In the worst case, the **resolution** transformation can cause a **quadratic expansion** each time it is applied

For large enough formulas, this can quickly **exhaust** the available **memory**

From DP to DPLL

The resolution transformation does not increase the number of variables
However, it **may increase** the size of the **clause set**

Question: If a variable appears positively in 3 clauses and negatively in 3 clauses, how many clauses after applying resolution? 9

In the worst case, the resolution transformation can cause a quadratic expansion each time it is applied

For large enough formulas, this can quickly exhaust the available memory

From DP to DPLL

The resolution transformation does not increase the number of variables
However, it **may increase** the size of the **clause set**

Question: If a variable appears positively in 3 clauses and negatively in 3 clauses,
how many clauses after applying resolution? 9

In the worst case, the resolution transformation can cause a quadratic expansion
each time it is applied

For large enough formulas, this can quickly exhaust the available memory

From DP to DPLL

The resolution transformation does not increase the number of variables
However, it **may increase** the size of the **clause set**

Question: If a variable appears positively in 3 clauses and negatively in 3 clauses,
how many clauses after applying resolution? 9

In the worst case, the **resolution** transformation can cause a **quadratic expansion**
each time it is applied

For large enough formulas, this can quickly **exhaust** the available **memory**

From DP to DPLL

The resolution transformation does not increase the number of variables
However, it **may increase** the size of the **clause set**

Question: If a variable appears positively in 3 clauses and negatively in 3 clauses,
how many clauses after applying resolution? 9

In the worst case, the **resolution** transformation can cause a **quadratic expansion**
each time it is applied

For large enough formulas, this can quickly **exhaust** the available **memory**

From DP to DPLL

The DPLL procedure improves on DP by replacing resolution with *splitting*:

- Let Δ be the input clause set
- Arbitrarily choose a literal l occurring in Δ
- Recursively check the satisfiability of $\Delta \cup \{ \{l\} \}$
 - If result is SAT, return SAT
 - Otherwise, recursively check the satisfiability of $\Delta \cup \{ \{\neg l\} \}$ and return that result

We will discuss DPLL in more detail later

From DP to DPLL

The DPLL procedure improves on DP by replacing resolution with *splitting*:

- Let Δ be the input clause set
- Arbitrarily choose a literal l occurring in Δ
- Recursively check the satisfiability of $\Delta \cup \{ \{l\} \}$
 - If result is SAT, return SAT
 - Otherwise, recursively check the satisfiability of $\Delta \cup \{ \{\neg l\} \}$ and return that result

We will discuss DPLL in more detail later

From DP to DPLL

The DPLL procedure improves on DP by replacing resolution with *splitting*:

- Let Δ be the input clause set
- Arbitrarily choose a literal l occurring in Δ
- Recursively check the satisfiability of $\Delta \cup \{ \{l\} \}$
 - If result is SAT, return SAT
 - Otherwise, recursively check the satisfiability of $\Delta \cup \{ \{\neg l\} \}$ and return that result

We will discuss DPLL in more detail later