Satisfiability Modulo Theories

Lecture 2: SAT Solving

Lydia Kondylidou

WS 2025/26

Decision procedures for propositional logic

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

Observe:

Each clause $l_1 \vee \cdots \vee l_n$ can be itself regarded as a set, of literals: $\{ l_1, \ldots, l_n \}$

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

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

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

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

We focus on backtracking solvers in this course

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

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

- 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

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

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**!

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

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 $\{C_1, C_2\} \models C$ so $\Delta \models \Delta \cup \{C\}$

Proofs by resolution example

Prove that the following clause set is unsatisfiable

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

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

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

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

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

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** and add three more rules

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

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

$$\textbf{Unsat}\ \frac{\{\} \in \Delta}{\text{UNSAT}} \qquad \textbf{Sat}\ \frac{\text{No other rules apply}}{\text{SAT}}$$

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

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

$$\textbf{Unit Resolve} \quad \frac{C_1, C_2 \in \Delta \quad C_1 = \{l\} \quad C_2 = \{\bar{l}\} \cup D}{\Delta \cup \{D\}}$$

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

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

Davis-Putnam (DP) procedure

A decision procedure for the SAT problem

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** Repeatedly applying these tranformations, 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 $\overline{l}$ 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)* $\overline{l}$ cannot be used to satisfy any clause, and *(ii)* any clause containing l is satisfied and can be ignored

DP procedure: unit propagation

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

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

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: 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 $P \times N$

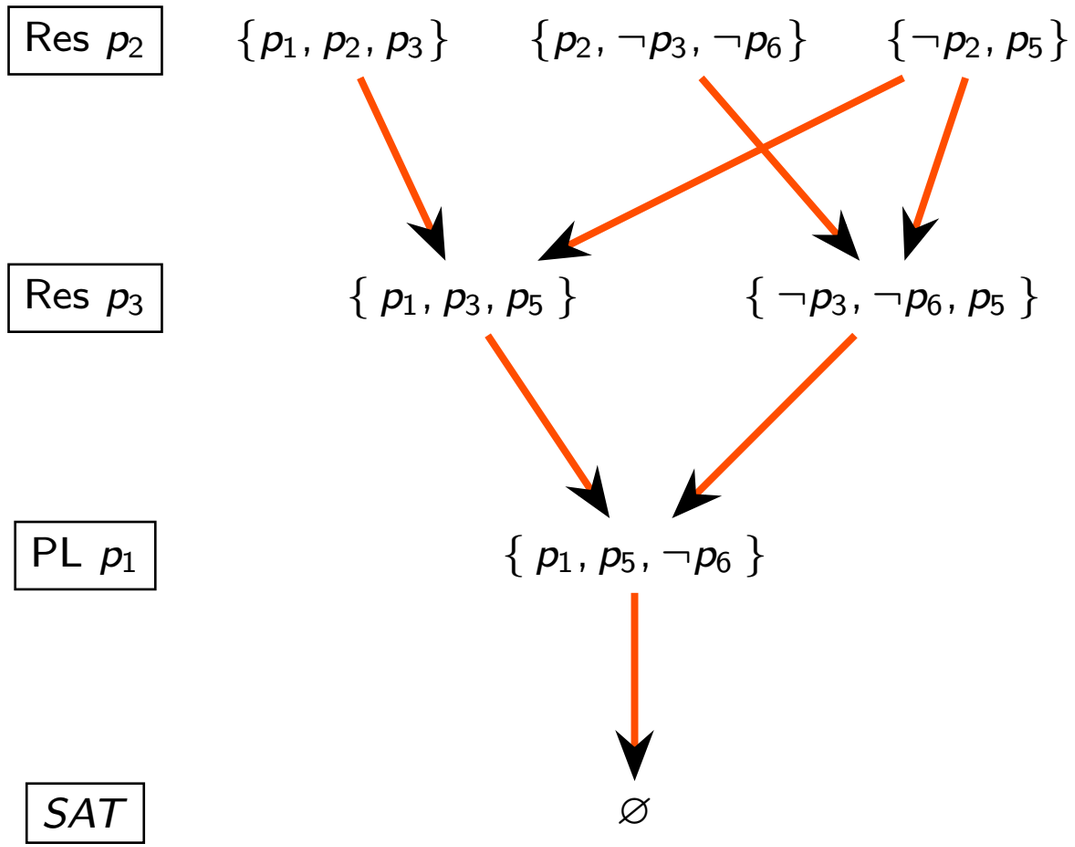
Example:

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

$\Delta_1 := \{ \{ p_2, p_3 \}, \{ p_2, \neg p_3, p_4 \}, \{ p_2, \neg p_4 \} \}$ (*resolution on* p_1)

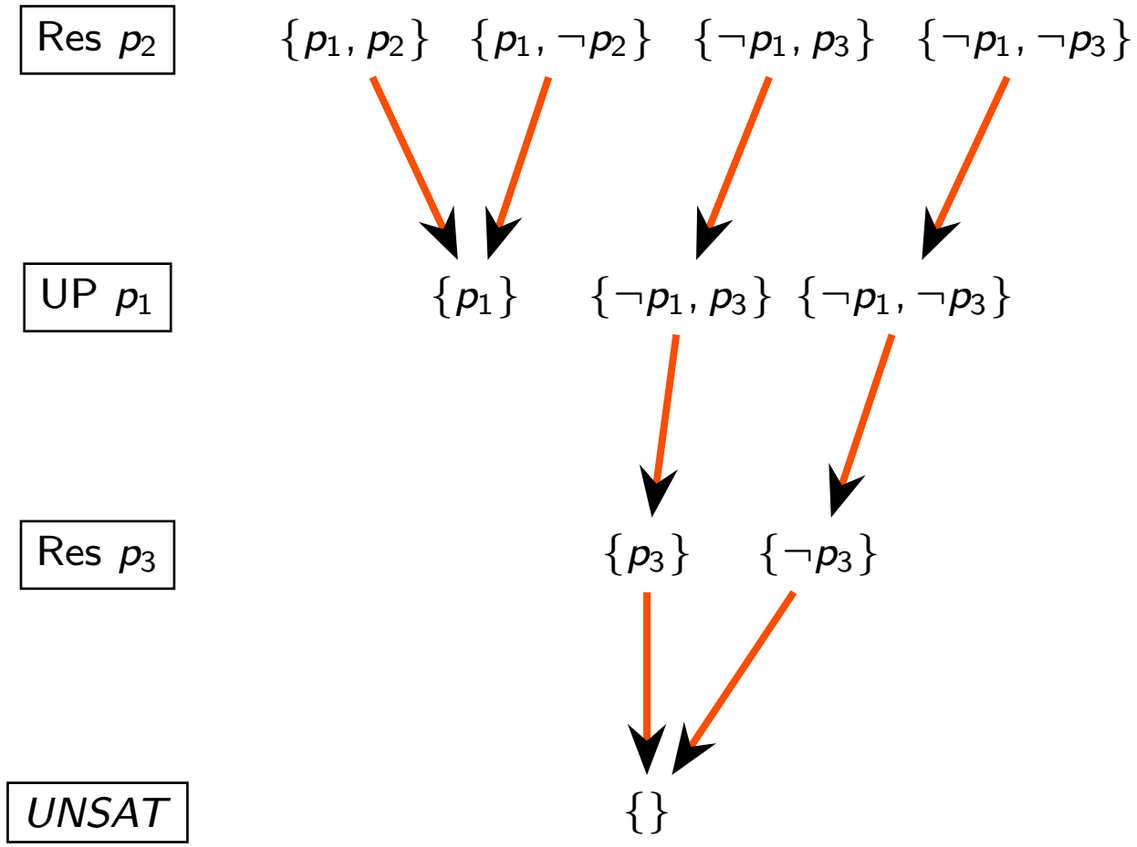
DP Example 1

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



DP Example 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?

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*:

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

The Original DPLL Procedure

Modern SAT solvers are based on an extension of the **DPLL procedure**

DPLL tries to **build** incrementally a satisfying assignment M for a clause set Δ

M is grown by

- **deducing** the truth value of a literal from M and Δ, or
- **guessing** a truth value

If a wrong guess for a literal leads to an inconsistency,
the procedure **backtracks** and tries the opposite value

DPLL as a Proof System

To facilitate a deeper look at DPLL, we present it as a proof system: *Abstract DPLL*

The proof system is a re-elaboration of those in [1,2]

[1] Nieuwenhuis et al, "Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T).", Journal of the ACM, 53(6).

[2] Krstić and Goel, "Architecting Solvers for SAT Modulo Theories: Nelson-Oppen with DPLL.", FroCos 2007.

Abstract DPLL: A Proof System for DPLL

States:

$$\textsc{unsat} \qquad\qquad \langle M, \Delta \rangle$$

where

M is a *sequence of literals* and *decision points* $\bullet$
denoting a **partial variable assignment**

Δ is a *set of clauses* denoting a CNF formula

Note: When convenient, we treat M as a set Provided M contains no complementary literals it determines the assignment

$$v_M(p) = \begin{cases} \text{true} & \text{if } p \in M \\ \text{false} & \text{if } \overline{p} \in M \\ \text{undef} & \text{otherwise} \end{cases}$$

Abstract DPLL: A Proof System for DPLL

Notation: If $M = M_0 \bullet M_1 \bullet \cdots \bullet M_n$ where each M_i contains no decision points

M_i is *decision level* i of M

$M^{[i]}$ denotes the subsequence $M_0 \bullet \cdots \bullet M_i$, from decision level 0 to decision level i

Initial state:

$\langle \epsilon, \Delta_0 \rangle$, where ϵ is the empty assignment and Δ_0 is to be checked for satisfiability

Expected final states:

UNSAT if Δ_0 is **unsatisfiable**

$\langle M, \Delta_n \rangle$ otherwise, where Δ_n is **equisatisfiable** with Δ_0 and **satisfied** by M

Some clause terminology

Notation $\overline{l}$ denotes the *complement* of l, that is, $\neg l$ if l is a variable, and p if l is $\neg p$

Given a partial assignment: $v := \{\, p_1 \mapsto \text{true},\ p_2 \mapsto \text{false},\ p_4 \mapsto \text{true} \,\}$

- clause $\{p_1, p_3, \overline{p}_4\}$ is *satisfied* by v
- clause $\{\overline{p}_1, p_2\}$ is *conflicting* with v
- clause $\{\overline{p}_1, p_3, \overline{p}_4\}$ is *unit* in v
- clause $\{\overline{p}_1, p_3, p_5\}$ is *unresolved* by v
- variable p_1 is *assigned* in v
- variable p_3 is *unassigned* in v

Abstract DPLL proof rules: extending the assignment

$$\textbf{Propagate}\ \frac{\{l_1, \ldots, l_n, l\} \in \Delta \qquad \bar{l}_1, \ldots, \bar{l}_n \in \mathrm{M} \qquad l, \bar{l} \notin \mathrm{M}}{\mathrm{M} := \mathrm{M}\ l}$$

Deduce the value of unassigned literal in unit clauses

The clause $\{l_1, \cdots, l_n, l\}$ is the *antecedent clause* of l, denoted by Antecedent(l)

$$\textbf{Pure}\ \frac{l \text{ literal of } \Delta \qquad \bar{l} \text{ not literal of } \Delta \qquad l, \bar{l} \notin \mathrm{M}}{\mathrm{M} := \mathrm{M}\ l}$$

Make a **pure literal** true

Abstract DPLL proof rules: extending the assignment

$$\textbf{Decide}\ \frac{l \in \mathcal{L}\Delta \qquad l, \bar{l} \notin \mathrm{M}}{\mathrm{M} := \mathrm{M} \bullet l}$$

Guess a truth value for an **unassigned** literal

Notation: $\mathcal{L}\Delta := \{l \mid l \text{ literal of } \Delta\} \cup \{\bar{l} \mid l \text{ literal of } \Delta\}$

l is a *decision literal* of the new M

Abstract DPLL proof rules: repairing the assignment

$$\textbf{Backtrack}\ \frac{\{l_1, \ldots, l_n\} \in \Delta \qquad \bar{l}_1, \ldots, \bar{l}_n \in \mathrm{M} \qquad \mathrm{M} = M_1 \bullet l\ M_2 \qquad \bullet \notin M_2}{\mathrm{M} := M_1\ \bar{l}}$$

There is a **conflicting clause** and a decision point to backtrack to

Backtrack over last decision point and add **complement** of decision literal

Note: Premise $\bullet \notin N$ enforces **chronological** backtracking

$$\textbf{Fail}\ \frac{\{l_1, \ldots, l_n\} \in \Delta \qquad \bar{l}_1, \ldots, \bar{l}_n \in \mathrm{M} \qquad \bullet \notin \mathrm{M}}{\mathrm{UNSAT}}$$

There is a **conflicting clause** and no decision points to backtrack to

Conclude that clause set is unsatisfiable

The DPLL proof system

$$\textbf{Propagate}\ \frac{\{l_1, \ldots, l_n, l\} \in \Delta \qquad \bar{l}_1, \ldots, \bar{l}_n \in \mathrm{M} \qquad l, \bar{l} \notin \mathrm{M}}{\mathrm{M} := \mathrm{M}\ l}$$

$$\textbf{Pure}\ \frac{l \text{ literal of } \Delta \qquad \bar{l} \text{ not literal of } \Delta \qquad l, \bar{l} \notin \mathrm{M}}{\mathrm{M} := \mathrm{M}\ l}$$

$$\textbf{Decide}\ \frac{l \text{ or } \bar{l} \text{ occurs in } \Delta \qquad l, \bar{l} \notin \mathrm{M}}{\mathrm{M} := \mathrm{M} \bullet l}$$

$$\textbf{Backtrack}\ \frac{\{l_1, \ldots, l_n\} \in \Delta \qquad \bar{l}_1, \ldots, \bar{l}_n \in \mathrm{M} \qquad \mathrm{M} = M_1 \bullet l\ M_2 \qquad \bullet \notin M_2}{\mathrm{M} := M_1\ \bar{l}}$$

$$\textbf{Fail}\ \frac{\{l_1, \ldots, l_n\} \in \Delta \qquad \bar{l}_1, \ldots, \bar{l}_n \in \mathrm{M} \qquad \bullet \notin \mathrm{M}}{\mathrm{UNSAT}}$$

Transforming DPLL to Resolution

The search procedure of DPLL can be reduced a posteriori to a resolution proof

(a sequence of applications of resolution rules)

DPLL Shortcomings

OK for randomly generated CNFs, but not for practical ones. Why?

No learning: throws away all work performed to conclude that current assignment is bad

Revisits bad partial assignments leading to conflicts due to the same root cause

Chronological backtracking: backtracks only one level, even if it can be concluded that the current partial assignment became doomed at a lower level

Naïve decisions: picks an arbitrary variable to branch on

Fails to consider the state of the search to make heuristically better decisions

Conflict-Driven Clause Learning (CDCL)

Learning: Δ is augmented with a **conflict clause** that summarizes the root cause of the conflict

Non-chronological backtracking: can backtrack **several levels**, based on the cause of the conflict (*conflict-driven backjumping*)

Decision heuristics: chooses the next literal to add to the current assignment based on the current state of the search

From DPLL to CDCL Solvers

To model conflict-driven backjumping and learning, we add a third component C to states whose value is either $\mathtt{no}$ or a clause C, the conflict clause

States:

$$\textsc{unsat} \qquad\qquad \langle \mathsf{M}, \Delta, \mathsf{C} \rangle$$

Initial state:

$\langle \epsilon, \Delta_0, \mathtt{no} \rangle$, where Δ_0 is to be checked for satisfiability

Expected final states:

$\textsc{unsat}$ if Δ_0 is unsatisfiable

$\langle \mathsf{M}, \Delta_n, \mathtt{no} \rangle$ otherwise, where Δ_n is equisatisfiable with Δ_0 and satisfied by M

From DPLL to CDCL Solvers

Replace **Backtrack** with three rules:

$$\textbf{Conflict}\ \frac{\mathsf{C} = \mathtt{no} \qquad \{l_1, \ldots, l_n\} \in \Delta \qquad \bar{l}_1, \ldots, \bar{l}_n \in \mathsf{M}}{\mathsf{C} := \{l_1, \ldots, l_n\}}$$

There is no conflict clause but a clause of Δ is falsified by M

So we set C to be that clause

From DPLL to CDCL Solvers

$$\textbf{Explain}\ \frac{\mathsf{C} = \{l\} \cup C' \quad \{l_1, \ldots, l_n, \bar{l}\} \in \Delta \quad \bar{l}_1, \ldots, \bar{l}_n, \bar{l} \in \mathsf{M} \quad \bar{l}_1, \ldots, \bar{l}_n \prec_{\mathsf{M}} \bar{l}}{\mathsf{C} := \{l_1, \ldots, l_n\} \cup C'}$$

$l \prec_M l'$ iff l occurs before l' in M

Δ contains a clause $D = \{l_1, \ldots, l_n, \bar{l}\}$ such that

1. l is in the conflict clause and is falsified by M
2. $l_1, \cdots, l_n$ are all falsified by M before l

We derive a new conflict clause by **resolution** of C and D

From DPLL to CDCL Solvers

$$\textbf{Backjump}\ \frac{\mathsf{C} = D \quad D = \{l_1, \ldots, l_n, l\} \quad \mathtt{lev}(\bar{l}_1), \ldots, \mathtt{lev}(\bar{l}_n) \leq\ i < \mathtt{lev}(\bar{l})}{\mathsf{M} := \mathsf{M}^{[i]} l \qquad \mathsf{C} := \mathtt{no} \qquad \Delta := \Delta \cup \{D\}}$$

To compute the level to backjump to:

1. find the literal $\bar{l} \in D$ that was assigned last
2. choose a level i smaller than $\mathtt{lev}(\bar{l})$ but not smaller than the levels of the other literals in D

Then learn conflict clause D, reset C, backtrack to level i and add l to it

Note: $\mathtt{lev}(l) = n$ iff l occurs in decision level n of M

Note: The rules maintain the **invariant**: $\Delta \models \mathsf{C}$ and $\mathsf{M} \models \neg\mathsf{C}$ when $\mathsf{C} \neq \mathtt{no}$

From DPLL to CDCL Solvers

Modify **Fail** to

$$\textbf{Fail}\ \frac{\mathsf{C} \neq \mathtt{no} \qquad \bullet \notin \mathsf{M}}{\text{UNSAT}}$$

C contains a **conflict clause** but there are no decision points to backjump over

Conclude that Δ is **unsatisfiable**

From DPLL to full CDCL Solvers

Also add

$$\textbf{Learn}\ \frac{D \text{ is a clause} \qquad \Delta \models D \qquad D \notin \Delta}{\Delta := \Delta \cup \{D\}}$$

Can be applied to any clause entailed by Δ

In particular, to any conflict clause $\mathsf{C} \neq \texttt{no}$ (because then $\Delta \models \mathsf{C}$)

The learned clause D is called a *lemma*

$$\textbf{Forget}\ \frac{\mathsf{C} = \texttt{no} \qquad \Delta = \Delta' \cup \{C\} \qquad \Delta' \models C}{\Delta := \Delta'}$$

From DPLL to full CDCL Solvers

Learning can quickly add millions of clauses to Δ

So it is useful to be able to delete **redundant** clauses that might not be useful anymore

$$\textbf{Restart}\ \frac{}{M := M^{[0]} \quad C := \texttt{no}}$$

If we are stuck in a hopeless area of the search space it may be better to just restart

Note: Restart is not from scratch since propagations at level 0 are maintained, together with all the learned lemmas not eliminated by **Forget**

Non-chronological vs. chronological backtracking

Note: Backjumping is **not always better** than chronological backtracking

See, e.g.,

"Chronological Backtracking" by Nadel and Ryvchin, SAT 2018.

"Lazy Reimplication in Chronological Backtracking" by Coutelier et al., SAT 2024.

Modeling Modern SAT Solvers

At the core, current CDCL SAT solvers are implementations of the proof system with rules

Propagate, **Pure**, **Decide**,

Conflict, **Explain**, **Backjump**, **Fail**

Learn, **Forget**, **Restart**

$$\textit{Basic CDCL} \quad := \quad \{\ \textbf{Propagate}, \textbf{Pure}, \textbf{Decide}, \textbf{Conflict}, \textbf{Explain}, \textbf{Backjump}, \textbf{Fail}\ \}$$

$$\textit{CDCL} \quad := \quad \text{Basic CDCL} + \{\ \textbf{Learn}, \textbf{Forget}, \textbf{Restart}\ \}$$

The CDCL System – Strategies

To **ensure termination** for the full system,

1. apply at least one Basic CDCL rule between each two **Learn** applications
2. apply **Restart** less and less often

The CDCL System – Strategies

A **common basic strategy** applies the rules with the following priorities, using a bound n initially set to 0, until an irreducible state is reached:

1. If $n > 0$ conflicts have been found so far, increase n and apply **Restart**
2. If M falsifies a clause and has no decision points, apply **Fail** and stop
3. If M falsifies a clause, apply **Conflict**
   (a) Apply **Explain** repeatedly
   (b) Apply **Learn** to the current conflict clause
   (c) Apply **Backjump**
4. Apply **Propagate** to completion
5. Apply **Decide**

The CDCL System – Strategies

Steps 3.1–3.3 achieve a form of *conflict analysis* and involve some **heuristic choices**:

1. When to stop applying **Explain** to a conflict?
2. Which level to **Backjump** to?

The CDCL proof system

$$\textbf{Propagate}\ \frac{\{l_1, \ldots, l_n, l\} \in \Delta \qquad \bar{l}_1, \ldots, \bar{l}_n \in \mathrm{M} \qquad l, \bar{l} \notin \mathrm{M}}{\mathrm{M} := \mathrm{M}\ l}$$

$$\textbf{Pure}\ \frac{l \text{ literal of } \Delta \qquad \bar{l} \text{ not literal of } \Delta \qquad l, \bar{l} \notin \mathrm{M}}{\mathrm{M} := \mathrm{M}\ l}$$

$$\textbf{Decide}\ \frac{l \text{ or } \bar{l} \text{ occurs in } \Delta \qquad l, \bar{l} \notin \mathrm{M}}{\mathrm{M} := \mathrm{M} \bullet l}$$

The CDCL proof system (continued)

$$\textbf{Conflict}\ \frac{\mathsf{C} = \mathtt{no} \qquad \{l_1, \ldots, l_n\} \in \Delta \qquad \bar{l}_1, \ldots, \bar{l}_n \in \mathsf{M}}{\mathsf{C} := \{l_1, \ldots, l_n\}}$$

$$\textbf{Explain}\ \frac{\mathsf{C} = \{l\} \cup C' \quad \{l_1, \ldots, l_n, \bar{l}\} \in \Delta \quad \bar{l}_1, \ldots, \bar{l}_n, \bar{l} \in \mathsf{M} \quad \bar{l}_1, \ldots, \bar{l}_n \prec_{\mathsf{M}} \bar{l}}{\mathsf{C} := \{l_1, \ldots, l_n\} \cup C'}$$

$$\textbf{Backjump}\ \frac{\mathsf{C} = D \quad D = \{l_1, \ldots, l_n, l\} \quad \mathtt{lev}(\bar{l}_1), \ldots, \mathtt{lev}(\bar{l}_n) \leq\ i < \mathtt{lev}(\bar{l})}{\mathsf{M} := \mathsf{M}^{[i]} l \qquad \mathsf{C} := \mathtt{no} \qquad \Delta := \Delta \cup \{D\}}$$

$$\textbf{Fail}\ \frac{\mathsf{C} \neq \mathtt{no} \qquad \bullet \notin \mathsf{M}}{\mathrm{UNSAT}}$$

The CDCL proof system (continued)

$$\textbf{Learn}\ \frac{D \text{ is a clause} \qquad \Delta \models D \qquad D \notin \Delta}{\Delta := \Delta \cup \{D\}}$$

$$\textbf{Forget}\ \frac{\mathsf{C} = \mathtt{no} \qquad \Delta = \Delta' \cup \{C\} \qquad \Delta' \models C}{\Delta := \Delta'}$$

$$\textbf{Restart}\ \frac{}{\mathsf{M} := \mathsf{M}^{[0]} \quad C := \mathtt{no}}$$