Classical Satisfiability Algorithms

IA085: Satisfiability and Automated Reasoning

Martin Jonáš

FI MUNI, Spring 2025

Last Time

- basic logical notions (entailment, equivalence, satisfiability, ...)
- applications of satisfiability,
- conversion of a formula to equisatisfiable CNF of linear size

Today, we assume that all formulas are in CNF.

Our Goal

An algorithm that can decide satisfiability of formulas with thousands of variables and millions of clauses.

Exhaustive search

Exhaustive search

```
ExhaustiveSearch(formula Φ) {
  foreach truth assignment μ to Atoms(Φ)
      res ← evaluate ϕ under μ
      if res == ⊤
          return SAT
  return UNSAT
}
```

Exhaustive search in practice

- virtually never used in practice
- for unsatisfiable instances always needs $2^{|Atoms(\varphi)|}$ steps
- for satisfiable instances can easily need exponential number of steps

Exhaustive search in practice

- virtually never used in practice
- for unsatisfiable instances always needs $2^{|Atoms(\varphi)|}$ steps
- for satisfiable instances can easily need exponential number of steps

Just buy a big powerful GPU?

- atoms on Earth $\sim 10^{50} \sim$ number of truth assignments to 166 variables
- atoms in the universe $\sim 10^{80} \sim$ number of truth assignments to 266 variables

Propositional resolution

Resolution rule

Rule for deriving new clauses from existing ones

$$\frac{\{A, l_1, \ldots, l_n\} \quad \{\neg A, l'_1, \ldots, l'_m\}}{\{l_1, \ldots, l_n, l'_1, \ldots, l'_m\}}$$

In general form

$$\frac{A \vee \varphi \quad \neg A \vee \psi}{\varphi \vee \psi}$$

Notation and terminology

- $\mathit{Resolve}(x, C_1, C_2)$ returns the resulting formula
- $\mathit{Resolve}(x, C_1, C_2)$ is called resolvent of C_1 and C_2 on x

Correctness

$$C_1 \wedge C_2 \quad \models \quad \mathit{Resolve}(x, C_1, C_2)$$

Resolution rule: notable instances

$$\frac{A \quad \neg A \vee B}{B}$$

Resolution rule: notable instances

$$\frac{A \quad \neg A \vee B}{B} \quad = \quad \frac{A \quad A \to B}{B} \quad = \quad \text{modus ponens}$$

Resolution rule: notable instances

$$\frac{A \quad \neg A \vee B}{B} \quad = \quad \frac{A \quad A \to B}{B} \quad = \quad \text{modus ponens}$$

$$\frac{\neg B \quad \neg A \vee B}{\neg A} \quad = \quad \frac{\neg B \quad A \to B}{\neg A} \quad = \quad \text{modus tollens}$$

Resolution rule: notable instances

$$\frac{A \quad \neg A \lor B}{B} \quad = \quad \frac{A \quad A \to B}{B} \quad = \quad \text{modus ponens}$$

$$\frac{\neg B \quad \neg A \lor B}{\neg A} \quad = \quad \frac{\neg B \quad A \to B}{\neg A} \quad = \quad \text{modus tollens}$$

$$\frac{\neg A \lor B \quad \neg B \lor C}{\neg A \lor C}$$

Resolution rule: notable instances

$$\frac{A \quad \neg A \vee B}{B} \quad = \quad \frac{A \quad A \to B}{B} \quad = \quad \text{modus ponens}$$

$$\frac{\neg B \quad \neg A \vee B}{\neg A} \quad = \quad \frac{\neg B \quad A \to B}{\neg A} \quad = \quad \text{modus tollens}$$

$$\frac{\neg A \vee B \quad \neg B \vee C}{\neg A \vee C} \quad = \quad \frac{A \to B \quad B \to C}{A \to C} \quad = \quad \text{transitivity}$$

Proving unsatisfiability by resolution

Observations

- if $C_1, C_2 \in \Phi$ and R is a resolvent of C_1 and C_2, then $\Phi \models R$
- therefore $\Phi \equiv \Phi \cup \{R\}$

Proving unsatisfiability by resolution

Observations

- if $C_1, C_2 \in \Phi$ and R is a resolvent of C_1 and C_2, then $\Phi \models R$
- therefore $\Phi \equiv \Phi \cup \{R\}$

Resolution method

- extend Φ with all possible resolvents of clauses from Φ
- if $\emptyset \in \Phi$ at some point, return UNSAT
- if no more clauses can be derived and $\emptyset \notin \Phi$, return SAT

Proving unsatisfiability by resolution

$$
\begin{aligned}
\{&\{A, B\},\\
&\{\neg B, C\},\\
&\{\neg B, \neg C\},\\
&\{\neg A, \neg B, \neg D\},\\
&\{\neg A, B, \neg D\},\\
&\{\neg A, B, D\}
\end{aligned}
$$

Proving unsatisfiability by resolution

$$\{\{A, B\},$$
$$\{\neg B, C\},$$
$$\{\neg B, \neg C\},$$
$$\{\neg A, \neg B, \neg D\},$$
$$\{\neg A, B, \neg D\},$$
$$\{\neg A, B, D\}$$

Proving unsatisfiability by resolution

$$
\begin{aligned}
\{&\{A, B\},\\
&\{\neg B, C\},\\
&\{\neg B, \neg C\},\\
&\{\neg A, \neg B, \neg D\},\\
&\{\neg A, B, \neg D\},\\
&\{\neg A, B, D\}\\
&\{\neg B\},
\end{aligned}
$$

Proving unsatisfiability by resolution

$$
\begin{aligned}
\{&\{A, B\},\\
&\{\neg B, C\},\\
&\{\neg B, \neg C\},\\
&\{\neg A, \neg B, \neg D\},\\
&\{\neg A, B, \neg D\},\\
&\{\neg A, B, D\}\\
&\{\neg B\},
\end{aligned}
$$

Proving unsatisfiability by resolution

$$
\begin{aligned}
\{&\{A, B\},\\
&\{\neg B, C\},\\
&\{\neg B, \neg C\},\\
&\{\neg A, \neg B, \neg D\},\\
&\{\neg A, B, \neg D\},\\
&\{\neg A, B, D\}\\
&\{\neg B\},\\
&\{A\},
\end{aligned}
$$

Proving unsatisfiability by resolution

$$\{\{A, B\},$$
$$\{\neg B, C\},$$
$$\{\neg B, \neg C\},$$
$$\{\neg A, \neg B, \neg D\},$$
$$\{\neg A, B, \neg D\},$$
$$\{\neg A, B, D\}$$
$$\{\neg B\},$$
$$\{A\},$$

Proving unsatisfiability by resolution

$$\begin{aligned}
\{&\{A, B\},\\
&\{\neg B, C\},\\
&\{\neg B, \neg C\},\\
&\{\neg A, \neg B, \neg D\},\\
&\{\neg A, B, \neg D\},\\
&\{\neg A, B, D\}\\
&\{\neg B\},\\
&\{A\},\\
&\{\neg A, B\},
\end{aligned}$$

Proving unsatisfiability by resolution

$$
\begin{aligned}
\{ & \{A, B\}, \\
& \{\neg B, C\}, \\
& \{\neg B, \neg C\}, \\
& \{\neg A, \neg B, \neg D\}, \\
& \{\neg A, B, \neg D\}, \\
& \{\neg A, B, D\} \\
& \{\neg B\}, \\
& \{A\}, \\
& \{\neg A, B\},
\end{aligned}
$$

Proving unsatisfiability by resolution

$$
\begin{aligned}
&\{\{A, B\},\\
&\ \{\neg B, C\},\\
&\ \{\neg B, \neg C\},\\
&\ \{\neg A, \neg B, \neg D\},\\
&\ \{\neg A, B, \neg D\},\\
&\ \{\neg A, B, D\}\\
&\ \{\neg B\},\\
&\ \{A\},\\
&\ \{\neg A, B\},\\
&\ \{B\},
\end{aligned}
$$

Proving unsatisfiability by resolution

$$
\begin{aligned}
\{ & \{A, B\}, \\
& \{\neg B, C\}, \\
& \{\neg B, \neg C\}, \\
& \{\neg A, \neg B, \neg D\}, \\
& \{\neg A, B, \neg D\}, \\
& \{\neg A, B, D\} \\
& \{\neg B\}, \\
& \{A\}, \\
& \{\neg A, B\}, \\
& \{B\},
\end{aligned}
$$

Proving unsatisfiability by resolution

$$\begin{aligned}
\{&\{A, B\},\\
&\{\neg B, C\},\\
&\{\neg B, \neg C\},\\
&\{\neg A, \neg B, \neg D\},\\
&\{\neg A, B, \neg D\},\\
&\{\neg A, B, D\}\\
&\{\neg B\},\\
&\{A\},\\
&\{\neg A, B\},\\
&\{B\},\\
&\emptyset \quad \}
\end{aligned}$$

Resolution Method: Properties

Theorem (Soundness)
If the resolution method returns UNSAT, the formula Φ is unsatisfiable.

Theorem (Completeness)
If the formula is unsatisfiable, the resolution method returns UNSAT.

Resolution Method: Properties

Resolution method is not used in practice

- the size of Φ never decreases
- the size of Φ grows quickly (often exponentially)
- as presented, the algorithm is not deterministic

Davis-Putnam algorithm

Davis-Putnam algorithm (1960)

- eagerly apply simple resolution cases first – unit resolution (unit propagation)
- fix an order of variables in which to resolve
- for a variable x, use resolution on all clauses that can be resolved on x at once and remove the original clauses

Davis-Putnam algorithm: Unit propagation

Variable assignment

- for example

$$\left\{\{A, B\}, \{C, \neg D\}, \{\neg A, D\}\right\}\Big|_A = \{\{C, \neg D\}, \{D\}\}$$

- $\Phi\big|_v = \{C \setminus \{\neg v\} \mid C \in \Phi$ and $v \notin C\}$
- similarly for $\Phi\big|_{\neg v}$

Unit propagation

- if Φ contains a unit clause ($\{l\} \in \Phi$), we can directly assign its value
- for example

$$\{\{A, \neg B\}, \{B\}, \{B, C\}, \{C, \neg D, A\}\} \rightsquigarrow \{\{A\}, \{C, \neg D, A\}\}$$

Davis-Putnam algorithm: Variable elimination

- divide $\Phi = \Psi \cup \Psi_x \cup \Psi_{\neg x}$ where clauses in Ψ do not contain x, clauses in Ψ_x contain x positively, and $\Psi_{\neg x}$ contain x negatively
- $\mathit{EliminateVar}(x, \Phi) = \Psi \cup \{\mathit{Resolve}(x, C_1, C_2) \mid C_1 \in \Psi_x, C_2 \in \Psi_{\neg x}\}$ without tautological clauses

$$\Phi = \{\{A, B\}, \{\neg B, C\}, \{\neg B, \neg C\}, \{\neg A, \neg B, \neg D\}, \{\neg A, B, \neg D\}, \{\neg A, B, D\}\}$$

$$\begin{aligned} \mathit{EliminateVar}(A, \Phi) = \{&\{\neg B, C\}, \{\neg B, \neg C\}, \\ &\{B, \neg B, \neg D\}, \\ &\{B, \neg D\}, \\ &\{B, D\}\} \end{aligned}$$

Davis-Putnam algorithm: Variable elimination

- divide $\Phi = \Psi \cup \Psi_x \cup \Psi_{\neg x}$ where clauses in Ψ do not contain x, clauses in Ψ_x contain x positively, and $\Psi_{\neg x}$ contain x negatively
- $\mathit{EliminateVar}(x, \Phi) = \Psi \cup \{\mathit{Resolve}(x, C_1, C_2) \mid C_1 \in \Psi_x, C_2 \in \Psi_{\neg x}\}$ without tautological clauses

$$\Phi = \{\{A, B\}, \{\neg B, C\}, \{\neg B, \neg C\}, \{\neg A, \neg B, \neg D\}, \{\neg A, B, \neg D\}, \{\neg A, B, D\}\}$$

$$\begin{aligned} \mathit{EliminateVar}(A, \Phi) = \{&\{\neg B, C\}, \{\neg B, \neg C\}, \\ &\{B, \neg B, \neg D\}, \\ &\{B, \neg D\}, \\ &\{B, D\}\} \end{aligned}$$

```
DP(formula Φ):
    while Φ contains unit clause {l}:
        Φ ← Φ|_l

    if Φ = ∅ return SAT
    if ∅ ∈ Φ return UNSAT

    v ← PickVariable(Φ)
    Φ ← EliminateVar(v, Φ)
    return DP(Φ)
```

Davis-Putnam algorithm: Properties

Theorem (Soundness)
If DP(Φ) *returns UNSAT, the formula* Φ *is unsatisfiable.*

Theorem (Completeness)
If the formula Φ *is unsatisfiable,* DPLL(Φ) *returns UNSAT.*

Proof idea.
Invariant: at every step, the formula Φ is equisatisfiable with the original.

- Unit propagation is satisfiability preserving.
- Variable elimination is satisfiability preserving. □

Theorem (Soundness)
If DP(Φ) *returns UNSAT, the formula* Φ *is unsatisfiable.*

Theorem (Completeness)
If the formula Φ *is unsatisfiable,* DPLL(Φ) *returns UNSAT.*

Proof idea.
Invariant: at every step, the formula Φ is equisatisfiable with the original.

- Unit propagation is satisfiability preserving.
- Variable elimination is satisfiability preserving. □

Corollary (Complexity)

Davis-Putnam algorithm: Properties

Theorem (Soundness)
If DP(Φ) *returns UNSAT, the formula* Φ *is unsatisfiable.*

Theorem (Completeness)
If the formula Φ *is unsatisfiable,* DPLL(Φ) *returns UNSAT.*

Proof idea.
Invariant: at every step, the formula Φ is equisatisfiable with the original.

- Unit propagation is satisfiability preserving.
- Variable elimination is satisfiability preserving. □

Corollary (Complexity)
Unless P = NP, *the procedure DP does not run in polynomial time.*

Pigeonnhole formula PHP_n

- Can $n+1$ pigeons be assigned to n boxes such that there is at most one pigeon in one box?
- variables $x_{i,j}$ – pigeon i is in the box j
- for each $1 \leq i \leq n+1$ a clause $\bigvee_{1 \leq j \leq n} x_{i,j}$
- for each $1 \leq j \leq n$ and $1 \leq i < i' \leq n+1$ a clause $\neg x_{i,j} \vee \neg x_{i',j}$
- obviously unsatisfiable

Resolution lower bounds

Pigeonnhole formula PHP_n

- Can $n+1$ pigeons be assigned to n boxes such that there is at most one pigeon in one box?
- variables $x_{i,j}$ – pigeon i is in the box j
- for each $1 \leq i \leq n+1$ a clause $\bigvee_{1 \leq j \leq n} x_{i,j}$
- for each $1 \leq j \leq n$ and $1 \leq i < i' \leq n+1$ a clause $\neg x_{i,j} \vee \neg x_{i',j}$
- obviously unsatisfiable

Theorem (Haken, 1985)
Every resolution proof of $\mathrm{PHP_n}$ *has size* $2^{\Omega(n)}$.

Pigeonnhole formula PHP_n

- Can $n+1$ pigeons be assigned to n boxes such that there is at most one pigeon in one box?
- variables $x_{i,j}$ – pigeon i is in the box j
- for each $1 \leq i \leq n+1$ a clause $\bigvee_{1 \leq j \leq n} x_{i,j}$
- for each $1 \leq j \leq n$ and $1 \leq i < i' \leq n+1$ a clause $\neg x_{i,j} \vee \neg x_{i',j}$
- obviously unsatisfiable

Theorem (Haken, 1985)
Every resolution proof of $\mathrm{PHP_n}$ has size $2^{\Omega(n)}$.

Corollary (Complexity)
The procedure DP does not run in polynomial time.

Davis-Putnam-Logemann-Loveland algorithm (DPLL)

DPLL

Davis-Putnam-Logemann-Loveland algorithm (1962)

- replace the resolution step in DP by variable assignment
- assign one value; if UNSAT, backtrack and try the opposite value
- eagerly apply unit propagation whenever possible

```
DPLL(formula Φ):
    while Φ contains unit clause {l}:
        Φ ← Φ|_l

    if Φ = ∅ return SAT
    if ∅ ∈ Φ return UNSAT

    v ← PickVariable(Φ)
    if DPLL(Φ|_v) == SAT:
        return SAT
    return DPLL(Φ|_¬v)
```

DPLL: Example

$$\{\{A, B\}, \{\neg A, B\}, \{\neg A, C, \neg B\}, \{\neg A, \neg C, \neg B\}\}$$

DPLL: Example

$$\{\{A, B\}, \{\neg A, B\}, \{\neg A, C, \neg B\}, \{\neg A, \neg C, \neg B\}\}$$

A

$$\{\{B\}, \{C, \neg B\}, \{\neg C, \neg B\}\}$$

DPLL: Example

$$\{\{A, B\}, \{\neg A, B\}, \{\neg A, C, \neg B\}, \{\neg A, \neg C, \neg B\}\}$$

A

$$\{\{B\}, \{C, \neg B\}, \{\neg C, \neg B\}\}$$

B

$$\{\{C\}, \{\neg C\}\}$$

DPLL: Example

$$\{\{A, B\}, \{\neg A, B\}, \{\neg A, C, \neg B\}, \{\neg A, \neg C, \neg B\}\}$$

A

$$\{\{B\}, \{C, \neg B\}, \{\neg C, \neg B\}\}$$

B

$$\{\{C\}, \{\neg C\}\}$$

C

$$\{\emptyset\}$$

DPLL: Example

$$\{\{A, B\}, \{\neg A, B\}, \{\neg A, C, \neg B\}, \{\neg A, \neg C, \neg B\}\}$$

A

$$\{\{B\}, \{C, \neg B\}, \{\neg C, \neg B\}\}$$

B

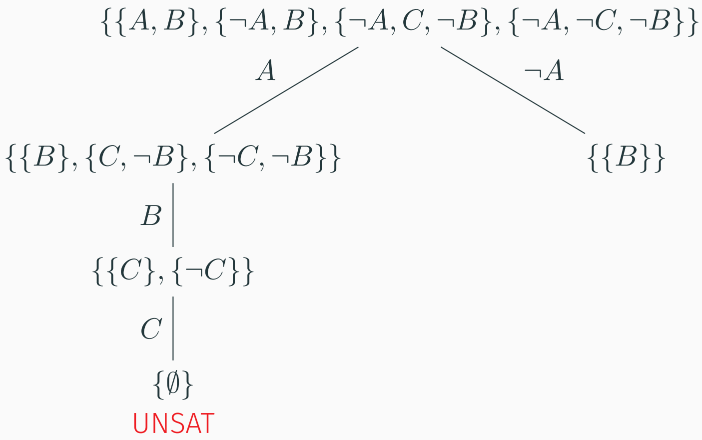
$$\{\{C\}, \{\neg C\}\}$$

C

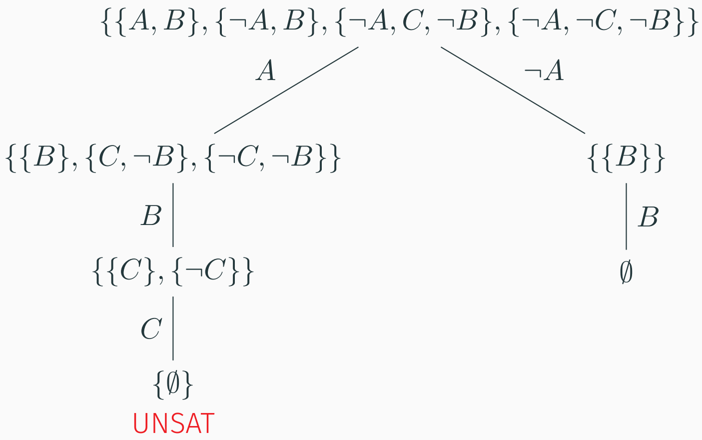
$$\{\emptyset\}$$

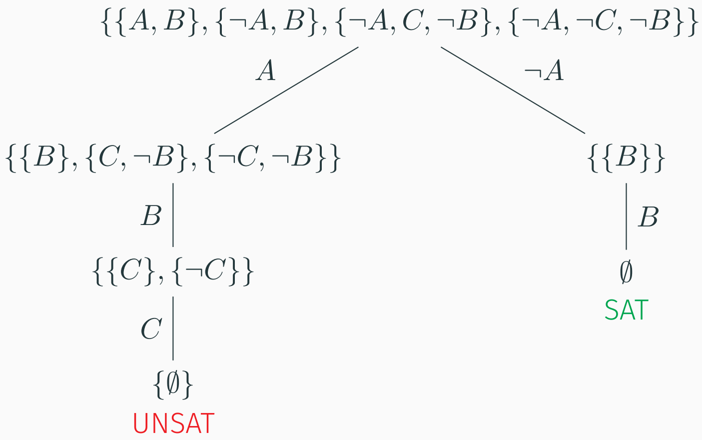
UNSAT

DPLL: Example



DPLL: Example




$\{\{A, B\}, \{\neg A, B\}, \{\neg A, C, \neg B\}, \{\neg A, \neg C, \neg B\}\}$
A
$\neg A$
$\{\{B\}, \{C, \neg B\}, \{\neg C, \neg B\}\}$
$\{\{B\}\}$
B
B
$\{\{C\}, \{\neg C\}\}$
$\emptyset$
SAT
C
$\{\emptyset\}$
UNSAT

DPLL: Properties

Theorem (Soundness)
If `DPLL(`Φ`)` *returns SAT, the formula* Φ *is satisfiable.*

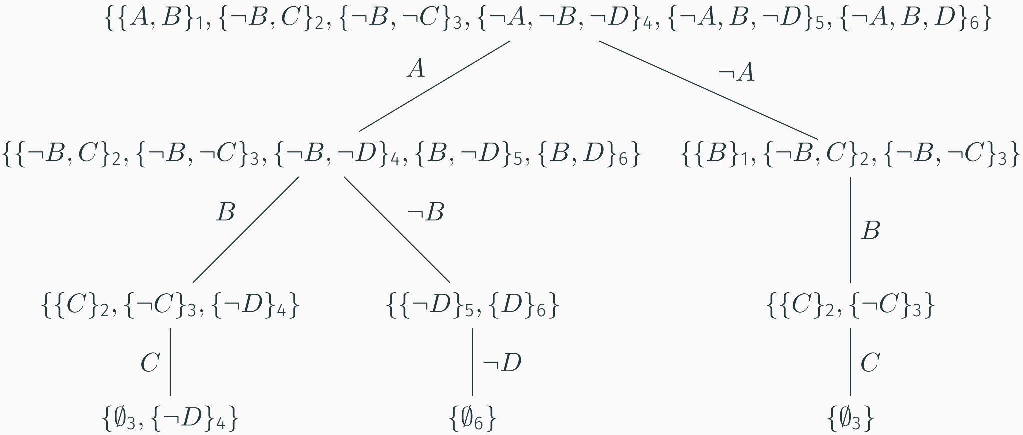
Theorem (Completeness)
If the formula Φ *is satisfiable,* `DPLL(`Φ`)` *returns SAT.*

Corollary (Complexity)
Unless $\mathsf{P} = \mathsf{NP}$, *the procedure DPLL does not run in polynomial time.*

UNSAT DPLL $\rightarrow$ Resolution

$$\{\{A, B\}_1, \{\neg B, C\}_2, \{\neg B, \neg C\}_3, \{\neg A, \neg B, \neg D\}_4, \{\neg A, B, \neg D\}_5, \{\neg A, B, D\}_6\}$$

UNSAT DPLL $\rightarrow$ Resolution

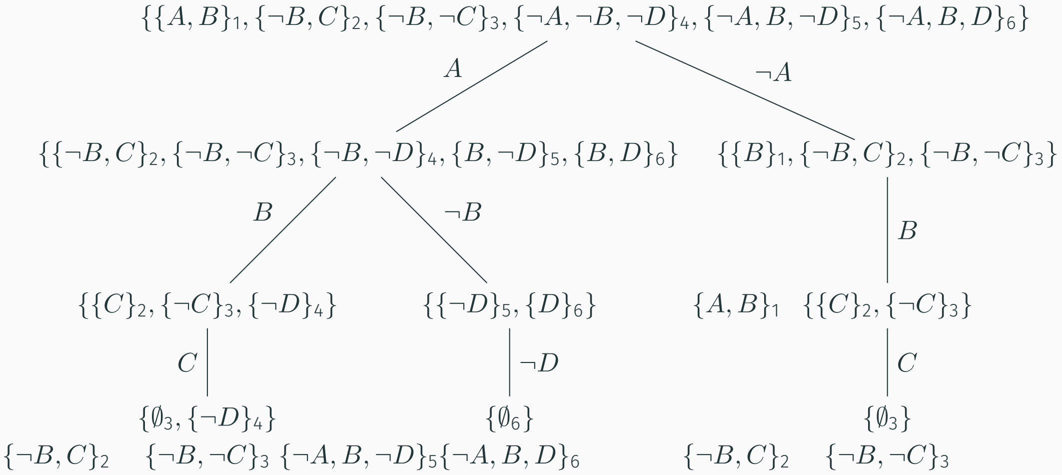


UNSAT DPLL $\rightarrow$ Resolution

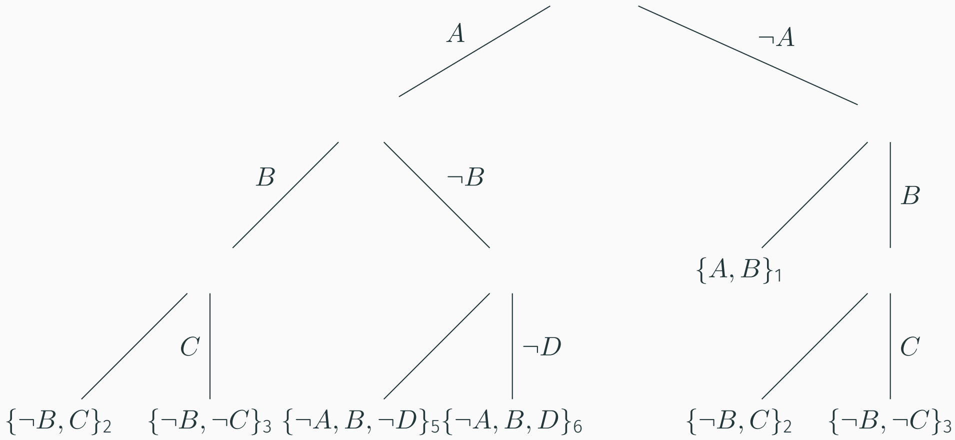
$$\{\{A,B\}_1,\{\neg B,C\}_2,\{\neg B,\neg C\}_3,\{\neg A,\neg B,\neg D\}_4,\{\neg A,B,\neg D\}_5,\{\neg A,B,D\}_6\}$$

- A: $\{\{\neg B,C\}_2,\{\neg B,\neg C\}_3,\{\neg B,\neg D\}_4,\{B,\neg D\}_5,\{B,D\}_6\}$
  - B: $\{\{C\}_2,\{\neg C\}_3,\{\neg D\}_4\}$
    - C: $\{\emptyset_3,\{\neg D\}_4\}$
      - $\{\neg B,\neg C\}_3$
  - $\neg B$: $\{\{\neg D\}_5,\{D\}_6\}$
    - $\neg D$: $\{\emptyset_6\}$
      - $\{\neg A,B,D\}_6$
- $\neg A$: $\{\{B\}_1,\{\neg B,C\}_2,\{\neg B,\neg C\}_3\}$
  - B: $\{\{C\}_2,\{\neg C\}_3\}$
    - C: $\{\emptyset_3\}$
      - $\{\neg B,\neg C\}_3$

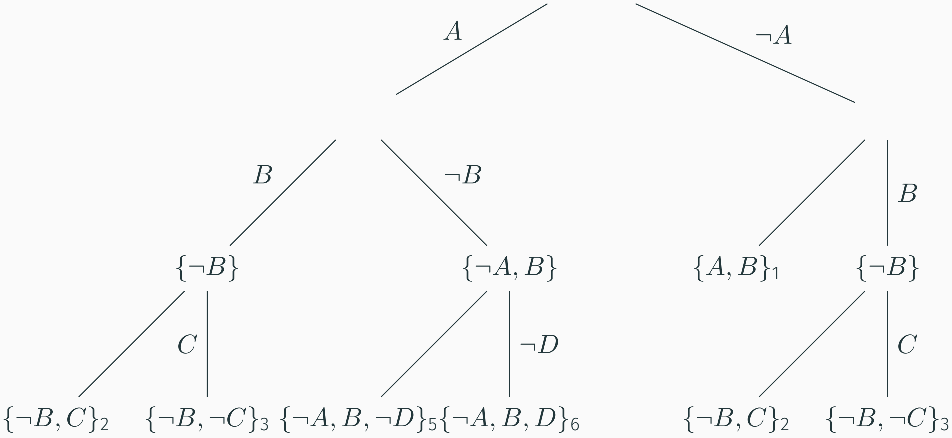
UNSAT DPLL → Resolution



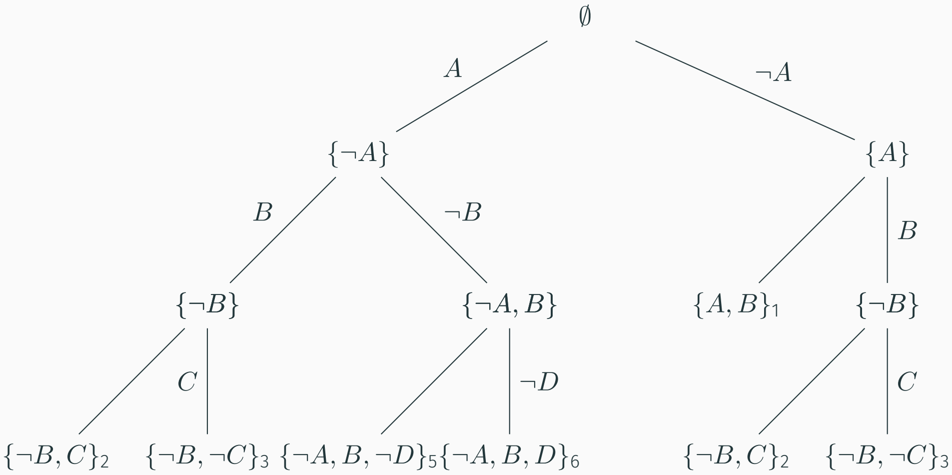
UNSAT DPLL $\rightarrow$ Resolution



UNSAT DPLL → Resolution



UNSAT DPLL $\rightarrow$ Resolution



A run of DPLL with result UNSAT corresponds to a tree resolution proof

1. replace all derived $\emptyset$ leaves by the corresponding original input clauses
2. to each unit propagation step, add the original clause of the unit clause that triggered the unit propagation
3. complete the resolution

Corollary (Time Complexity)
DPLL has exponential time complexity (e.g., for PHP formulas).

Theorem (Space Complexity)
DPLL has polynomial space complexity.

DPLL in practice

- DPLL is almost never used in practice
- basis of Conflict-Driven Clause Learning (CDCL) used in most of the modern SAT solvers

Implementing DPLL

Real implementation of DPLL

- the previous theoretical description is not suitable for practical implementation
- each modification of formula Φ is too expensive
- do not modify the formula, modify the partial assignment instead

Clause status

- contains satisfied literal $\rightarrow$ satisfied
- all literals are assigned opposite values $\rightarrow$ falsified / conflict clause
- one literal is unassigned, other literals are assigned opposite values $\rightarrow$ unit clause
- otherwise undetermined

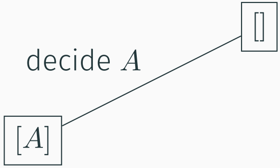
DPLL: Searching in assignments

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

$$\boxed{[]}$$

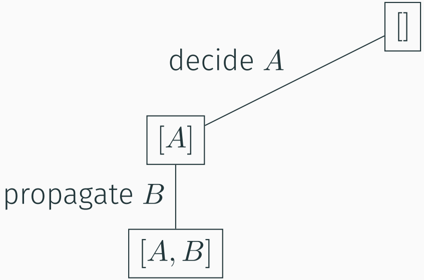
DPLL: Searching in assignments

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



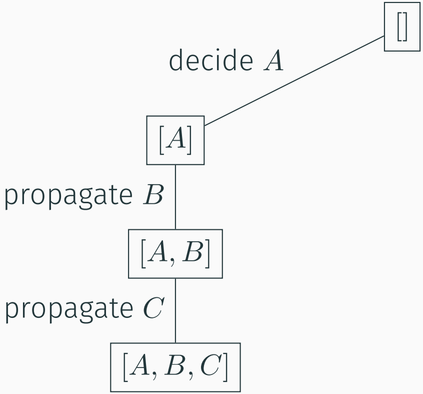
DPLL: Searching in assignments

$$(A \lor B) \land (\neg A \lor B) \land (\neg A \lor C \lor \neg B) \land (\neg A \lor \neg C \lor \neg B)$$



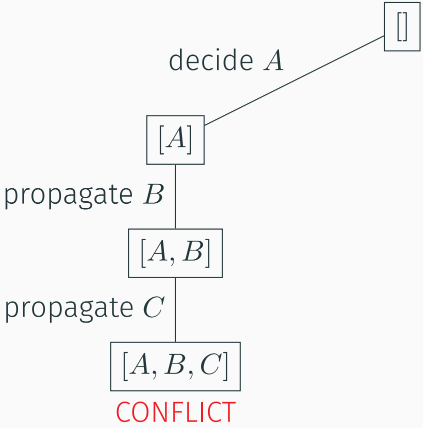
DPLL: Searching in assignments

$$(A \lor B) \land (\neg A \lor B) \land (\neg A \lor C \lor \neg B) \land (\neg A \lor \neg C \lor \neg B)$$



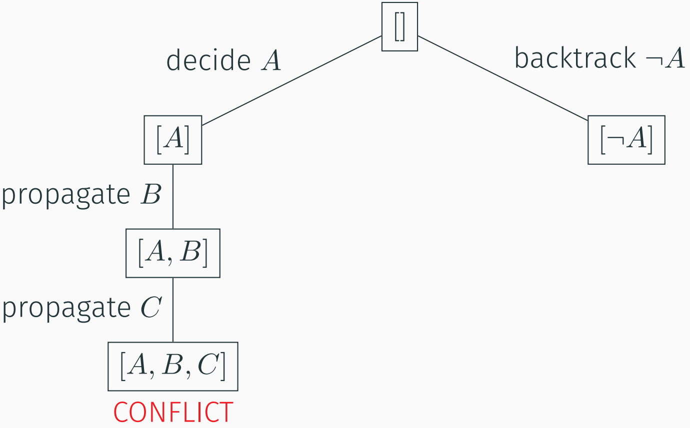
DPLL: Searching in assignments

$$(A \lor B) \land (\neg A \lor B) \land (\neg A \lor C \lor \neg B) \land (\neg A \lor \neg C \lor \neg B)$$



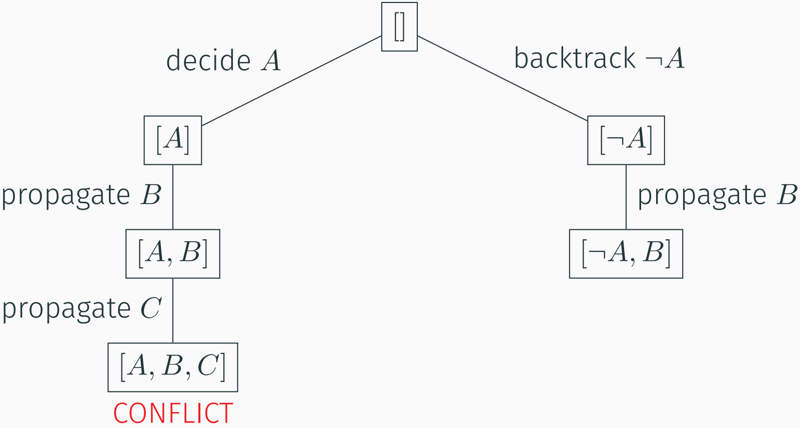
DPLL: Searching in assignments

$$(A \lor B) \land (\neg A \lor B) \land (\neg A \lor C \lor \neg B) \land (\neg A \lor \neg C \lor \neg B)$$



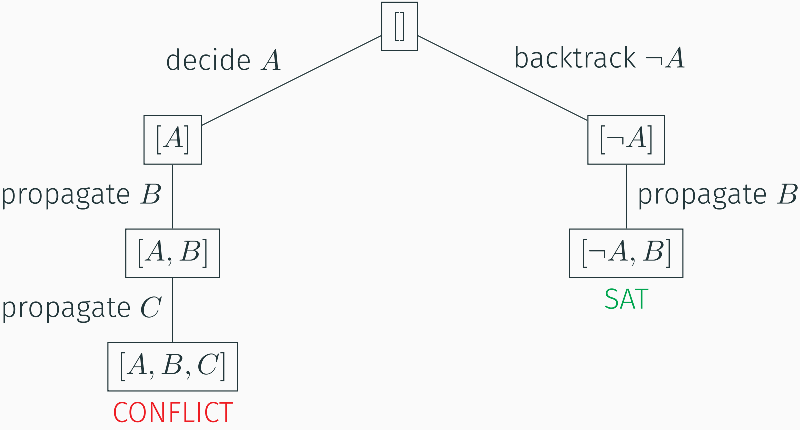
DPLL: Searching in assignments

$$(A \vee B) \;\wedge\; (\neg A \vee B) \;\wedge\; (\neg A \vee C \vee \neg B) \;\wedge\; (\neg A \vee \neg C \vee \neg B)$$



DPLL: Searching in assignments

$$(A \lor B) \land (\neg A \lor B) \land (\neg A \lor C \lor \neg B) \land (\neg A \lor \neg C \lor \neg B)$$



Partial assignment representation

Trail

- stack of currently assigned literals
- `trail = [A, ¬C]`
- used during backtracking

Map of values

- maps each variable to `true`/`false`/`unknown`
- `value[A] = true, value[B] = unknown, value[C] = false`
- used to evaluate clauses

Decision and Backtracking

- do not use recursion for backtracking, manage the stack explicitly (faster and will be useful later)
- keep list of positions of decision literals that can be reverted if needed
- e.g. for `trail = [A, ¬B, C, D, ¬E]`, `decisions = [0, 2]`:
  - literals `trail[0] = A` and `trail[2] = C` were decisions
  - other literals were unit propagated or set during backtracking

Desired functionalities

- `Decide(x,v)`: sets x to v; can be flipped using backtracking
- `Assign(x,v)`: sets x to v; cannot be flipped using backtracking
- `Backtrack()`: undo all assignments up to the last decision, `Assign` the decided variable to the opposite value
- How to implement?

Unit propagation

`UnitPropagate()`

- detects unit clauses
- keeps a queue of unit assignments that have to be performed
- assigns value to all unit literals until fixed point
- can detect conflicts

DPLL: Realistic

```
DPLL(formula Φ):
  InitializeDatastructures()

  if UnitPropagation() == CONFLICT:
      return UNSAT

  while not all variables are assigned:
      v ← PickUnassignedVariable()

      Decide(v, false)
      while UnitPropagation() == CONFLICT:
          if decisions == []:
              return UNSAT
          Backtrack()

  return SAT
```

Unit propagation: naive

Naive unit propagation

- go through the list of clauses
- for each unit clause `Assign` the unassigned literal and repeat
- found clause that has all literals assigned to `false` $\rightarrow$ return CONFLICT

Naive unit propagation

- go through the list of clauses
- for each unit clause `Assign` the unassigned literal and repeat
- found clause that has all literals assigned to `false` $\rightarrow$ return CONFLICT

Less naive unit propagation

- all unit propagations (except the first one) occur after variable decision/assignment
- precompute for each literal `occurs[l]`, the list of clauses that contain l
- after decision/assignment of l, only check the clauses in `occurs[¬l]`

Unit propagation: need something better

Still not good enough, a variable can occur in a large number of clauses.

Most of the runtime is spent in unit propagation $\rightarrow$ must be as cheap as possible!

Idea

Do not check clauses for which we are sure that contain at least two unassigned literals.

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$\underset{\uparrow}{x} \vee \neg y \vee z \vee \neg v \vee \neg w \vee \overset{\downarrow}{u}$$

Variable assignment:

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$\underset{\uparrow}{x} \vee \neg y \vee z \vee \neg v \vee \neg w \vee \overset{\downarrow}{u}$$

Variable assignment:

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$\underset{\uparrow}{x} \vee \neg y \vee z \vee \neg v \vee \neg w \vee \overset{\downarrow}{u}$$

Variable assignment: y

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$\underset{\uparrow}{x} \vee \neg y \vee z \vee \neg v \vee \neg w \vee \overset{\downarrow}{u}$$

Variable assignment: y

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$\underset{\uparrow}{x} \vee \neg y \vee z \vee \neg v \vee \neg w \vee \overset{\downarrow}{u}$$

Variable assignment: y , v

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$\underset{\uparrow}{x} \vee \neg y \vee z \vee \neg v \vee \neg w \vee \overset{\downarrow}{u}$$

Variable assignment: y , v

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$\underset{\uparrow}{x} \vee \neg y \vee z \vee \neg v \vee \neg w \vee \overset{\downarrow}{u}$$

Variable assignment: y , v , $\neg x$

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$x \vee \neg y \vee \underset{\uparrow}{z} \vee \neg v \vee \neg w \vee \overset{\downarrow}{u}$$

Variable assignment: y , v , $\neg x$

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$x \lor \neg y \lor \underset{\uparrow}{z} \lor \neg v \lor \neg w \lor \overset{\downarrow}{u}$$

Variable assignment: y , v , $\neg x$, $\neg u$

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$x \vee \neg y \vee \underset{\uparrow}{z} \vee \neg v \vee \neg \overset{\downarrow}{w} \vee u$$

Variable assignment: y , v , $\neg x$, $\neg u$

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$x \lor \neg y \lor \underset{\uparrow}{z} \lor \neg v \lor \overset{\downarrow}{\neg w} \lor u$$

Variable assignment: y , v , $\neg x$, $\neg u$, w

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$x \lor \neg y \lor \underset{\uparrow}{z} \lor \neg v \lor \overset{\downarrow}{\neg w} \lor u$$

Variable assignment: y , v , $\neg x$, $\neg u$, w

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$x \vee \neg y \vee \underset{\uparrow}{z} \vee \neg v \vee \overset{\downarrow}{\neg w} \vee u$$

Variable assignment: y , v , $\neg x$, $\neg u$, w

Unit: z

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$x \vee \neg y \vee \underset{\uparrow}{z} \vee \neg v \vee \overset{\downarrow}{\neg w} \vee u$$

Variable assignment: y , v , $\neg x$, $\neg u$

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$x \lor \neg y \lor \underset{\uparrow}{z} \lor \neg v \lor \neg \overset{\downarrow}{w} \lor u$$

Variable assignment: y , v , $\neg x$, $\neg u$

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$x \lor \neg y \lor \underset{\uparrow}{z} \lor \neg v \lor \overset{\downarrow}{\neg w} \lor u$$

Variable assignment: y , v , $\neg x$

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$x \vee \neg y \vee \underset{\uparrow}{z} \vee \neg v \vee \neg w \vee \overset{\downarrow}{u}$$

Variable assignment: y , v , $\neg x$

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$x \lor \neg y \lor \underset{\uparrow}{z} \lor \neg v \lor \neg w \lor \overset{\downarrow}{u}$$

Variable assignment: y , v

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$\underset{\uparrow}{x} \lor \neg y \lor z \lor \neg v \lor \neg w \lor \overset{\downarrow}{u}$$

Variable assignment: y , v

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$\underset{\uparrow}{x} \vee \neg y \vee z \vee \neg v \vee \neg w \vee \overset{\downarrow}{u}$$

Variable assignment: y

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$\underset{\uparrow}{x} \vee \neg y \vee z \vee \neg v \vee \neg w \vee \overset{\downarrow}{u}$$

Variable assignment: y

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$\underset{\uparrow}{x} \vee \neg y \vee z \vee \neg v \vee \neg w \vee \overset{\downarrow}{u}$$

Variable assignment:

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$\underset{\uparrow}{x} \vee \neg y \vee z \vee \neg v \vee \neg w \vee \overset{\downarrow}{u}$$

Variable assignment:

Unit propagation: head-tail lists

Head-tail lists (SATO solver, 1997)

- for each clause, remember positions of its first and last unassigned literals (head and tail)
- for each literal, remember list of clauses where it is head and where it is tail
- during unit propagation, only check clauses where the negation of literal is head/tail
- needs to maintain the invariant during backtracking

$$\underset{\uparrow}{x} \lor \neg y \lor z \lor \neg v \lor \neg w \lor \overset{\downarrow}{u}$$

Variable assignment:

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$\underset{\uparrow}{x} \vee \overset{\downarrow}{\neg y} \vee z \vee \neg v \vee \neg w \vee u$$

Variable assignment:

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$\underset{\uparrow}{x} \lor \overset{\downarrow}{\neg y} \lor z \lor \neg v \lor \neg w \lor u$$

Variable assignment:

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$\underset{\uparrow}{x} \vee \overset{\downarrow}{\neg y} \vee z \vee \neg v \vee \neg w \vee u$$

Variable assignment: y

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$\underset{\uparrow}{x} \lor \neg y \lor \overset{\downarrow}{z} \lor \neg v \lor \neg w \lor u$$

Variable assignment: y

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$\underset{\uparrow}{x} \lor \neg y \lor \overset{\downarrow}{z} \lor \neg v \lor \neg w \lor u$$

Variable assignment: y , v

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$\underset{\uparrow}{x} \vee \neg y \vee \overset{\downarrow}{z} \vee \neg v \vee \neg w \vee u$$

Variable assignment: y , v

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$\underset{\uparrow}{x} \lor \neg y \lor \overset{\downarrow}{z} \lor \neg v \lor \neg w \lor u$$

Variable assignment: y , v , $\neg x$

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \lor \neg y \lor \overset{\downarrow}{z} \lor \neg v \lor \underset{\uparrow}{\neg w} \lor u$$

Variable assignment: y , v , $\neg x$

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \lor \neg y \lor \overset{\downarrow}{z} \lor \neg v \lor \underset{\uparrow}{\neg w} \lor u$$

Variable assignment: y , v , $\neg x$, $\neg u$

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \vee \neg y \vee \overset{\downarrow}{z} \vee \neg v \vee \underset{\uparrow}{\neg w} \vee u$$

Variable assignment: y , v , $\neg x$, $\neg u$

Unit propagation: two watched literals

Two watched literals (zCHAFF solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \lor \neg y \lor \overset{\downarrow}{z} \lor \neg v \lor \underset{\uparrow}{\neg w} \lor u$$

Variable assignment: y , v , $\neg x$, $\neg u$, w

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \vee \neg y \vee \overset{\downarrow}{z} \vee \neg v \vee \underset{\uparrow}{\neg w} \vee u$$

Variable assignment: y , v , $\neg x$, $\neg u$, w

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \vee \neg y \vee \overset{\downarrow}{z} \vee \neg v \vee \underset{\uparrow}{\neg w} \vee u$$

Variable assignment: y , v , $\neg x$, $\neg u$, w

Unit: z

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \lor \neg y \lor \overset{\downarrow}{z} \lor \neg v \lor \underset{\uparrow}{\neg w} \lor u$$

Variable assignment: y , v , $\neg x$, $\neg u$

Unit propagation: two watched literals

Two watched literals (zCHAFF solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \vee \neg y \vee \overset{\downarrow}{z} \vee \neg v \vee \underset{\uparrow}{\neg w} \vee u$$

Variable assignment: y , v , $\neg x$, $\neg u$

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \lor \neg y \lor \overset{\downarrow}{z} \lor \neg v \lor \underset{\uparrow}{\neg w} \lor u$$

Variable assignment: y , v , $\neg x$

Unit propagation: two watched literals

Two watched literals (zCHAFF solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \lor \neg y \lor \overset{\downarrow}{z} \lor \neg v \lor \underset{\uparrow}{\neg w} \lor u$$

Variable assignment: y , v , $\neg x$

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \lor \neg y \lor \overset{\downarrow}{z} \lor \neg v \lor \underset{\uparrow}{\neg w} \lor u$$

Variable assignment: y , v

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \vee \neg y \vee \overset{\downarrow}{z} \vee \neg v \vee \underset{\uparrow}{\neg w} \vee u$$

Variable assignment: y , v

Unit propagation: two watched literals

Two watched literals (zCHAFF solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \lor \neg y \lor \overset{\downarrow}{z} \lor \neg v \lor \underset{\uparrow}{\neg w} \lor u$$

Variable assignment: y

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \lor \neg y \lor \overset{\downarrow}{z} \lor \neg v \lor \neg \underset{\uparrow}{w} \lor u$$

Variable assignment: y

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \vee \neg y \vee \overset{\downarrow}{z} \vee \neg v \vee \underset{\uparrow}{\neg w} \vee u$$

Variable assignment:

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \vee \neg y \vee \overset{\downarrow}{z} \vee \neg v \vee \underset{\uparrow}{\neg w} \vee u$$

Variable assignment:

Unit propagation: two watched literals

Two watched literals (zChaff solver, 2001)

- for each clause, remember positions of its two unassigned literals (watched literals)
- for each literal, remember list of clauses where it is watched
- during unit propagation, only check clauses where the negation of literal is watched
- nothing needs to be done during backtracking!

$$x \lor \neg y \lor \overset{\downarrow}{z} \lor \neg v \lor \underset{\uparrow}{\neg w} \lor u$$

Variable assignment:

Next time

Conflict-Driven Clause Learning (CDCL)

- DPLL search (unit propagation, backtracking)
- + using resolution to learn new clauses after conflict
- + non-chronological backtracking

Next time

Conflict-Driven Clause Learning (CDCL)

- DPLL search (unit propagation, backtracking)
- + using resolution to learn new clauses after conflict
- + non-chronological backtracking

Modern SAT Solvers

- CDCL
- + two watched literal scheme
- + variable decision heuristics
- + dynamic restarts
- + preprocessing/inprocessing

Project

You can already start implementing your SAT solver

- input in DIMACS format
- DPLL-like assignment decisions and backtracking
- unit propagation with two watched literal scheme