Lecture 6
The DPLL Algorithm

Introduction to Logic for Computer Science

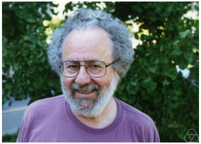
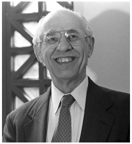
Prof Hongseok Yang
KAIST

These slides are minor variants of those made by Prof Worrell and Dr Haase for their logic course at Oxford.

Davis–Putnam–Logemann–Loveland

DPLL algorithm:

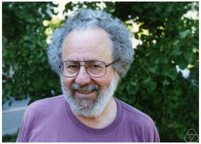
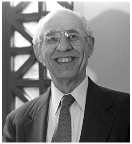
- Combines search and deduction to decide satisfiability.
- Underlies most modern SAT solvers.
- Over 50 years old.

Davis–Putnam–Logemann–Loveland

DPLL algorithm:

- Combines search and deduction to decide satisfiability.
- Underlies most modern SAT solvers.
- Over 50 years old.

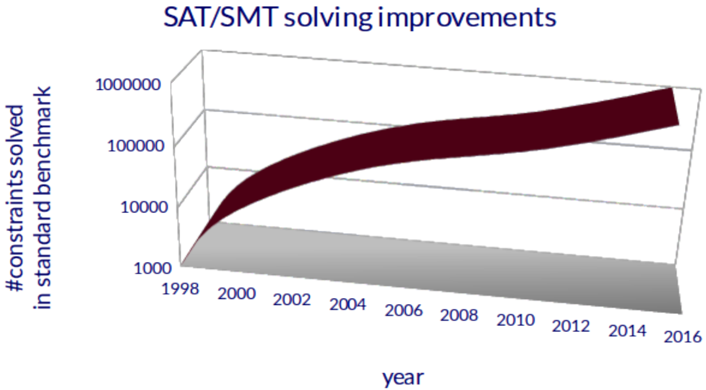



Dramatic progress of DPLL-based SAT solvers since 1990.

Emerged enhancement:

- **Clause learning**.
- **Non-chronological backtracking**.
- Branching heuristics.
- Lazy evaluation.

Performance increase of SAT solvers



DPLL: idea

Depth-first search.

At every unsuccessful leaf of search tree (called **conflict**), use resolution to compute a **conflict clause**.

Add the clause to the formula we're deciding about.

Think of conflict clauses as "caching" previous search results. So we "learn from previous mistakes".

Conflict clauses also determine backtracking.

DPLL: main actors

DPLL: main actors

1. CNF formula F.

   - Set to the input initially.

   - Repeatedly extended with a clause by DPLL.

DPLL: main actors

1. CNF formula F.
   - Set to the input initially.
   - Repeatedly extended with a clause by DPLL.
2. Valuation $\mathcal{A}$.
   - Sequence of assignments $\langle p_1 \mapsto b_1, \ldots, p_r \mapsto b_k \rangle$ where $p_1, \ldots, p_k$ are distinct prop. variables and $b_1, \ldots, b_k \in \{0, 1\}$.
   - $p_i \mapsto b_i$ may be annotated with a clause and other info.

DPLL: main actors

1. CNF formula F.
   - Set to the input initially.
   - Repeatedly extended with a clause by DPLL.
2. Valuation $\mathcal{A}$.
   - Sequence of assignments $\langle p_1 \mapsto b_1, \ldots, p_r \mapsto b_k \rangle$ where $p_1, \ldots, p_k$ are distinct prop. variables and $b_1, \ldots, b_k \in \{0, 1\}$.
   - $p_i \mapsto b_i$ may be annotated with a clause and other info.
3. Specialisation $F|_{\mathcal{A}}$.
   - F simplified under $\mathcal{A}$.
   - Delete any clause containing the true literal under $\mathcal{A}$, and remove from each remaining clause the false literal under $\mathcal{A}$.

The DPLL algorithm

Input: CNF formula F. **Output:** Satisfying valuation $\mathcal{A}$ or UNSAT.

The DPLL algorithm

Input: CNF formula F. **Output:** Satisfying valuation $\mathcal{A}$ or UNSAT.

1. Initialise $\mathcal{A}$ to the empty list of assignments.

The DPLL algorithm

Input: CNF formula F. **Output:** Satisfying valuation $\mathcal{A}$ or UNSAT.

1. Initialise $\mathcal{A}$ to the empty list of assignments.
2. While there is a *unit* clause $\{L\}$ in $F|_{\mathcal{A}}$, add $L \mapsto 1$ to $\mathcal{A}$.

The DPLL algorithm

Input: CNF formula F. **Output:** Satisfying valuation $\mathcal{A}$ or UNSAT.

1. Initialise $\mathcal{A}$ to the empty list of assignments.
2. While there is a *unit* clause $\{L\}$ in $F|_{\mathcal{A}}$, add $L \mapsto 1$ to $\mathcal{A}$.
3. If $F|_{\mathcal{A}}$ contains no clauses, stop & output $\mathcal{A}$.

The DPLL algorithm

Input: CNF formula F. **Output:** Satisfying valuation $\mathcal{A}$ or UNSAT.

1. Initialise $\mathcal{A}$ to the empty list of assignments.
2. While there is a *unit* clause $\{L\}$ in $F|_{\mathcal{A}}$, add $L \mapsto 1$ to $\mathcal{A}$.
3. If $F|_{\mathcal{A}}$ contains no clauses, stop & output $\mathcal{A}$.
4. If $F|_{\mathcal{A}} \ni \square$, do the following:

The DPLL algorithm

Input: CNF formula F. **Output:** Satisfying valuation $\mathcal{A}$ or UNSAT.

1. Initialise $\mathcal{A}$ to the empty list of assignments.
2. While there is a *unit* clause $\{L\}$ in $F|_{\mathcal{A}}$, add $L \mapsto 1$ to $\mathcal{A}$.
3. If $F|_{\mathcal{A}}$ contains no clauses, stop & output $\mathcal{A}$.
4. If $F|_{\mathcal{A}} \ni \Box$, do the following:
    1. Find a clause C by **learning procedure** & add it to F.

The DPLL algorithm

Input: CNF formula F. **Output:** Satisfying valuation $\mathcal{A}$ or UNSAT.

1. Initialise $\mathcal{A}$ to the empty list of assignments.
2. While there is a *unit* clause $\{L\}$ in $F|_{\mathcal{A}}$, add $L \mapsto 1$ to $\mathcal{A}$.
3. If $F|_{\mathcal{A}}$ contains no clauses, stop & output $\mathcal{A}$.
4. If $F|_{\mathcal{A}} \ni \Box$, do the following:
    1. Find a clause C by **learning procedure** & add it to F.
    2. If C is the empty clause, stop & output UNSAT.

The DPLL algorithm

Input: CNF formula F. **Output:** Satisfying valuation $\mathcal{A}$ or UNSAT.

1. Initialise $\mathcal{A}$ to the empty list of assignments.
2. While there is a *unit* clause $\{L\}$ in $F|_{\mathcal{A}}$, add $L \mapsto 1$ to $\mathcal{A}$.
3. If $F|_{\mathcal{A}}$ contains no clauses, stop & output $\mathcal{A}$.
4. If $F|_{\mathcal{A}} \ni \Box$, do the following:
    1. Find a clause C by **learning procedure** & add it to F.
    2. If C is the empty clause, stop & output UNSAT.
    3. Otherwise, backtrack to the highest level where C is a unit clause. Go to step 2.

The DPLL algorithm

Input: CNF formula F. **Output:** Satisfying valuation $\mathcal{A}$ or UNSAT.

1. Initialise $\mathcal{A}$ to the empty list of assignments.
2. While there is a *unit* clause $\{L\}$ in $F|_{\mathcal{A}}$, add $L \mapsto 1$ to $\mathcal{A}$.
3. If $F|_{\mathcal{A}}$ contains no clauses, stop & output $\mathcal{A}$.
4. If $F|_{\mathcal{A}} \ni \Box$, do the following:
    1. Find a clause C by **learning procedure** & add it to F.
    2. If C is the empty clause, stop & output UNSAT.
    3. Otherwise, backtrack to the highest level where C is a unit clause. Go to step 2.
5. Use a **decision strategy** to determine a new assignment $p \mapsto b$. Add it to $\mathcal{A}$. Go to step 2.

Example run with $\{\{\neg p_1\}, \{p_1, p_2, \neg p_3\}, \{p_3, p_4, p_5\}, \{p_4, \neg p_5\}\}$.

The DPLL algorithm

Input: CNF formula F. **Output:** Satisfying valuation $\mathcal{A}$ or UNSAT.

1. Initialise $\mathcal{A}$ to the empty list of assignments.
2. While there is a *unit* clause $\{L\}$ in $F|_{\mathcal{A}}$, add $L \mapsto 1$ to $\mathcal{A}$.
3. If $F|_{\mathcal{A}}$ contains no clauses, stop & output $\mathcal{A}$.
4. If $F|_{\mathcal{A}} \ni \Box$, do the following:
    1. Find a clause C by **learning procedure** & add it to F.
    2. If C is the empty clause, stop & output UNSAT.
    3. Otherwise, backtrack to the highest level where C is a unit clause. Go to step 2.
5. Use a **decision strategy** to determine a new assignment $p \mapsto b$. Add it to $\mathcal{A}$. Go to step 2.

Unit propagation – deduction step.

The DPLL algorithm

Input: CNF formula F. **Output:** Satisfying valuation $\mathcal{A}$ or UNSAT.

1. Initialise $\mathcal{A}$ to the empty list of assignments.
2. While there is a *unit* clause $\{L\}$ in $F|_{\mathcal{A}}$, add $L \mapsto 1$ to $\mathcal{A}$.
3. If $F|_{\mathcal{A}}$ contains no clauses, stop & output $\mathcal{A}$.
4. If $F|_{\mathcal{A}} \ni \Box$, do the following:
   1. Find a clause C by **learning procedure** & add it to F.
   2. If C is the empty clause, stop & output UNSAT.
   3. Otherwise, backtrack to the highest level where C is a unit clause. Go to step 2.
5. Use a **decision strategy** to determine a new assignment $p \mapsto b$. Add it to $\mathcal{A}$. Go to step 2.

Decision – search step.

The DPLL algorithm

Input: CNF formula F. **Output:** Satisfying valuation $\mathcal{A}$ or UNSAT.

1. Initialise $\mathcal{A}$ to the empty list of assignments.
2. While there is a *unit* clause $\{L\}$ in $F|_{\mathcal{A}}$, add $L \mapsto 1$ to $\mathcal{A}$.
3. If $F|_{\mathcal{A}}$ contains no clauses, stop & output $\mathcal{A}$.
4. If $F|_{\mathcal{A}} \ni \square$, do the following:
    1. Find a clause C by **learning procedure** & add it to F.
    2. If C is the empty clause, stop & output UNSAT.
    3. Otherwise, backtrack to the highest level where C is a unit clause. Go to step 2.
5. Use a **decision strategy** to determine a new assignment $p \mapsto b$. Add it to $\mathcal{A}$. Go to step 2.

Clause learning – deduction step.

Terminology

Terminology

A **state** of algorithm is a pair of CNF formula F and valuation $\mathcal{A}$.

Successful state when $\mathcal{A} \models F$.

Conflict state when $\mathcal{A} \models \neg F$.

Note: Conflict state if $F|_{\mathcal{A}} \ni \square$. Successful state if $F|_{\mathcal{A}} = \emptyset$.

Terminology

A **state** of algorithm is a pair of CNF formula F and valuation $\mathcal{A}$.

Successful state when $\mathcal{A} \models F$.

Conflict state when $\mathcal{A} \models \neg F$.

Note: Conflict state if $F|_{\mathcal{A}} \ni \square$. Successful state if $F|_{\mathcal{A}} = \emptyset$.

Each assignment $p_i \mapsto b_i$ in $\mathcal{A}$ classified as **decision assignment** or **implied assignment**.

$p_i \mapsto b_i$ by a decision strategy (step 5) is a **decision assignment**. p_i called **decision variable**.

$p_i \mapsto b_i$ by a unit propagation on a clause C (step 2) is an **implied assignment**. Denoted by $p_i \overset{C}{\mapsto} b_i$.

The DPLL algorithm

Input: CNF formula F. **Output:** Satisfying valuation $\mathcal{A}$ or UNSAT.

1. Initialise $\mathcal{A}$ to the empty list of assignments.
2. While there is a *unit* clause $\{L\}$ in $F|_{\mathcal{A}}$, add $L \mapsto 1$ to $\mathcal{A}$.
3. If $F|_{\mathcal{A}}$ contains no clauses, stop & output $\mathcal{A}$.
4. If $F|_{\mathcal{A}} \ni \square$, do the following:
   1. Find a clause C by **learning procedure** & add it to F.
   2. If C is the empty clause, stop & output UNSAT.
   3. Otherwise, backtrack to the highest level where C is a unit clause. Go to step 2.
5. Use a **decision strategy** to determine a new assignment $p \mapsto b$. Add it to $\mathcal{A}$. Go to step 2.

Unit propagation.

Unit propagation

Unit propagation refers to the loop at step 2, which repeatedly adds an assignment $L \mapsto 1$ to $\mathcal{A}$ whenever there is a unit clause $\{L\} \in F|_{\mathcal{A}}$.

Unit propagation

Unit propagation refers to the loop at step 2, which repeatedly adds an assignment $L \mapsto 1$ to $\mathcal{A}$ whenever there is a unit clause $\{L\} \in F|_{\mathcal{A}}$.

Example: start with set of clauses $F = \{C_1, \ldots, C_5\}$, where

$$
\begin{aligned}
C_1 &= \{\neg p_1, \neg p_4, p_5\}, \\
C_2 &= \{\neg p_1, p_6, \neg p_5\}, \\
C_3 &= \{\neg p_1, \neg p_6, p_7\}, \\
C_4 &= \{\neg p_1, \neg p_7, \neg p_5\}, \\
C_5 &= \{p_1, p_4, p_6\}.
\end{aligned}
$$

Let the current valuation is $\mathcal{A} = \langle p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 1\rangle$.

Notice that $F|_{\mathcal{A}}$ contains the unit clause $\{p_5\}$.

Unit propagation

Unit propagation refers to the loop at step 2, which repeatedly adds an assignment $L \mapsto 1$ to $\mathcal{A}$ whenever there is a unit clause $\{L\} \in F|_{\mathcal{A}}$.

Example: start with set of clauses $F = \{C_1, \ldots, C_5\}$, where

$$\begin{aligned} C_1 &= \{\neg p_1, \neg p_4, p_5\}, \\ C_2 &= \{\neg p_1, p_6, \neg p_5\}, \\ C_3 &= \{\neg p_1, \neg p_6, p_7\}, \\ C_4 &= \{\neg p_1, \neg p_7, \neg p_5\}, \\ C_5 &= \{p_1, p_4, p_6\}. \end{aligned}$$

Let the current valuation is $\mathcal{A} = \langle p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 1 \rangle$.

Notice that $F|_{\mathcal{A}}$ contains the unit clause $\{p_5\}$.

Ex: Run unit propagation on F and $\mathcal{A}$ completely.

Unit propagation

Unit propagation refers to the loop at step 2, which repeatedly adds an assignment $L \mapsto 1$ to $\mathcal{A}$ whenever there is a unit clause $\{L\} \in F|_{\mathcal{A}}$.

Example: start with set of clauses $F = \{C_1, \ldots, C_5\}$, where

$$\begin{aligned} C_1 =& \{\neg p_1, \neg p_4, p_5\}, \\ C_2 =& \{\neg p_1, p_6, \neg p_5\}, \\ C_3 =& \{\neg p_1, \neg p_6, p_7\}, \\ C_4 =& \{\neg p_1, \neg p_7, \neg p_5\}, \\ C_5 =& \{p_1, p_4, p_6\}. \end{aligned}$$

Let the current valuation is $\mathcal{A} = \langle p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 1\rangle$.

Notice that $F|_{\mathcal{A}}$ contains the unit clause $\{p_5\}$.

Ex: Run unit propagation on F and $\mathcal{A}$ completely.

Answer: Unit propagation generates $p_5 \overset{C_1}{\mapsto} 1, p_6 \overset{C_2}{\mapsto} 1, p_7 \overset{C_3}{\mapsto} 1$. This leads to conflict, with C_4 being made false.

The DPLL algorithm

Input: CNF formula F. **Output:** Satisfying valuation $\mathcal{A}$ or UNSAT.

1. Initialise $\mathcal{A}$ to the empty list of assignments.
2. While there is a *unit* clause $\{L\}$ in $F|_{\mathcal{A}}$, add $L \mapsto 1$ to $\mathcal{A}$.
3. If $F|_{\mathcal{A}}$ contains no clauses, stop & output $\mathcal{A}$.
4. If $F|_{\mathcal{A}} \ni \square$, do the following:
    1. Find a clause C by **learning procedure** & add it to F.
    2. If C is the empty clause, stop & output UNSAT.
    3. Otherwise, backtrack to the highest level where C is a unit clause. Go to step 2.
5. Use a **decision strategy** to determine a new assignment $p \mapsto b$. Add it to $\mathcal{A}$. Go to step 2.

Clause learning via conflict analysis.

Conflict analysis

After unit propagation:

- if not in conflict nor successful, make a decision (line 5);
- if in conflict, a **learned clause** is added (line 4).

Conflict analysis

After unit propagation:

- if not in conflict nor successful, make a decision (line 5);
- if in conflict, a **learned clause** is added (line 4).

Learned clause desiderata: If unit propagation from state $(F, \mathcal{A})$ leads to conflict, a clause C is learned such that:

Conflict analysis

After unit propagation:

- if not in conflict nor successful, make a decision (line 5);
- if in conflict, a **learned clause** is added (line 4).

Learned clause desiderata: If unit propagation from state $(F, \mathcal{A})$ leads to conflict, a clause C is learned such that:

- $F \equiv F \cup \{C\}$;

Conflict analysis

After unit propagation:

- if not in conflict nor successful, make a decision (line 5);
- if in conflict, a **learned clause** is added (line 4).

Learned clause desiderata: If unit propagation from state $(F, \mathcal{A})$ leads to conflict, a clause C is learned such that:

- $F \equiv F \cup \{C\}$;
- C is a **conflict clause**, i.e., every literal in C is made false by $\mathcal{A}$;

Conflict analysis

After unit propagation:

- if not in conflict nor successful, make a decision (line 5);
- if in conflict, a **learned clause** is added (line 4).

Learned clause desiderata: If unit propagation from state $(F, \mathcal{A})$ leads to conflict, a clause C is learned such that:

- $F \equiv F \cup \{C\}$;
- C is a **conflict clause**, i.e., every literal in C is made false by $\mathcal{A}$;
- C mentions only decision variables in $\mathcal{A}$.

Conflict analysis

After unit propagation:

- if not in conflict nor successful, make a decision (line 5);
- if in conflict, a **learned clause** is added (line 4).

Learned clause desiderata: If unit propagation from state $(F, \mathcal{A})$ leads to conflict, a clause C is learned such that:

- $F \equiv F \cup \{C\}$;
- C is a **conflict clause**, i.e., every literal in C is made false by $\mathcal{A}$;
- C mentions only decision variables in $\mathcal{A}$.

Ex: How can we find such C from a conflicted state $(F, \mathcal{A})$?

Clause learning algorithm

Suppose $\mathcal{A} = \langle p_1 \mapsto b_1, \ldots, p_k \mapsto b_k \rangle$ leads to conflict.

Find associated clauses $D_1, \ldots, D_{k+1}$ by backward induction:

Clause learning algorithm

Suppose $\mathcal{A} = \langle p_1 \mapsto b_1, \ldots, p_k \mapsto b_k \rangle$ leads to conflict.

Find associated clauses $D_1, \ldots, D_{k+1}$ by backward induction:

1. Pick any conflict clause $D_{k+1} \in F$ under $\mathcal{A}$.

Clause learning algorithm

Suppose $\mathcal{A} = \langle p_1 \mapsto b_1, \ldots, p_k \mapsto b_k \rangle$ leads to conflict.

Find associated clauses $D_1, \ldots, D_{k+1}$ by backward induction:

1. Pick any conflict clause $D_{k+1} \in F$ under $\mathcal{A}$.
2. If $p_i \mapsto b_i$ is a decision assignment or p_i is not mentioned in D_{i+1}, set $D_i = D_{i+1}$.

Clause learning algorithm

Suppose $\mathcal{A} = \langle p_1 \mapsto b_1, \ldots, p_k \mapsto b_k \rangle$ leads to conflict.

Find associated clauses $D_1, \ldots, D_{k+1}$ by backward induction:

1. Pick any conflict clause $D_{k+1} \in F$ under $\mathcal{A}$.
2. If $p_i \mapsto b_i$ is a decision assignment or p_i is not mentioned in D_{i+1}, set $D_i = D_{i+1}$.
3. If $p_i \overset{C_i}{\mapsto} b_i$ is an implied assignment and p_i is mentioned in D_{i+1}, define D_i to be a resolvent of C_i and D_{i+1} with respect to p_i.

Clause learning algorithm

Suppose $\mathcal{A} = \langle p_1 \mapsto b_1, \ldots, p_k \mapsto b_k \rangle$ leads to conflict.

Find associated clauses $D_1, \ldots, D_{k+1}$ by backward induction:

1. Pick any conflict clause $D_{k+1} \in F$ under $\mathcal{A}$.
2. If $p_i \mapsto b_i$ is a decision assignment or p_i is not mentioned in D_{i+1}, set $D_i = D_{i+1}$.
3. If $p_i \overset{C_i}{\mapsto} b_i$ is an implied assignment and p_i is mentioned in D_{i+1}, define D_i to be a resolvent of C_i and D_{i+1} with respect to p_i.

The final clause D_1 is the **learned clause**.

Proposition

The learned clause D_1 fulfills the desiderata.

Clause learning algorithm

Suppose $\mathcal{A} = \langle p_1 \mapsto b_1, \ldots, p_k \mapsto b_k \rangle$ leads to conflict.

Find associated clauses $D_1, \ldots, D_{k+1}$ by backward induction:

1. Pick any conflict clause $D_{k+1} \in F$ under $\mathcal{A}$.
2. If $p_i \mapsto b_i$ is a decision assignment or p_i is not mentioned in D_{i+1}, set $D_i = D_{i+1}$.
3. If $p_i \overset{C_i}{\mapsto} b_i$ is an implied assignment and p_i is mentioned in D_{i+1}, define D_i to be a resolvent of C_i and D_{i+1} with respect to p_i.

The final clause D_1 is the **learned clause**.

Proposition

The learned clause D_1 fulfills the desiderata.

Ex1: Run this algorithm on the example in the unit prop. slide.

Ex2: Prove the proposition.

Clause learning: example

In conflict of above example, learning generates clauses

$$
\begin{aligned}
D_8 &:= \{\neg p_1, \neg p_7, \neg p_5\} && \text{(clause } C_4\text{)} \\
D_7 &:= \{\neg p_1, \neg p_5, \neg p_6\} && \text{(resolve } D_8,\ C_3\text{)} \\
D_6 &:= \{\neg p_1, \neg p_5\} && \text{(resolve } D_7,\ C_2\text{)} \\
D_5 &:= \{\neg p_1, \neg p_4\} && \text{(resolve } D_6,\ C_1\text{)} \\
&\vdots \\
D_1 &:= \{\neg p_1, \neg p_4\}
\end{aligned}
$$

Clause learning: example

In conflict of above example, learning generates clauses

$$
\begin{aligned}
D_8 &:= \{\neg p_1, \neg p_7, \neg p_5\} && \text{(clause } C_4\text{)} \\
D_7 &:= \{\neg p_1, \neg p_5, \neg p_6\} && \text{(resolve } D_8, C_3\text{)} \\
D_6 &:= \{\neg p_1, \neg p_5\} && \text{(resolve } D_7, C_2\text{)} \\
D_5 &:= \{\neg p_1, \neg p_4\} && \text{(resolve } D_6, C_1\text{)} \\
&\vdots \\
D_1 &:= \{\neg p_1, \neg p_4\}
\end{aligned}
$$

The learned clause D_1 is a conflict clause with only decision variables, including the top-level one p_4.

D_1 records that conflict arose from decision to make p_1, p_4 true.

Adding D_1 makes assignments validating p_1, p_4 unreachable.

Backtracking to the highest level where D_1 is a unit clause ($p_1 \mapsto 1$) and doing unit propagation lead to $p_4 \mapsto 0$.

Example: 4 queens

Problem: place 4 non-attacking queens on a 4x4 chess board.

Example: 4 queens

Problem: place 4 non-attacking queens on a 4x4 chess board.

Variable p_{ij} models that there is a queen in the square (i, j).

- ≥ 1 in each row: $\bigwedge_{i=1}^{4} \bigvee_{j=1}^{4} p_{ij}$.
- ≤ 1 in each row: $\bigwedge_{i=1}^{4} \bigwedge_{j \neq j'=1}^{4} (\neg p_{ij} \vee \neg p_{ij'})$.
- ≤ 1 in each column: $\bigwedge_{j=1}^{4} \bigwedge_{i \neq i'=1}^{4} (\neg p_{ij} \vee \neg p_{i'j})$.
- ≤ 1 on each diagonal:
  $\bigwedge_{i,j=1}^{4} \bigwedge_{k} (\neg p_{i,j} \vee \neg p_{i+k,j+k}) \wedge \bigwedge_{i,j=1}^{4} \bigwedge_{l} (\neg p_{i,j} \vee \neg p_{i-l,j+l})$

Total number of clauses: 4 + 24 + 24 + 28 = 80.

DPLL: 4 queens

Running the DPLL algorithm:

- Start with $p_{11} \mapsto 1$. Then,
  1) delete $\{p_{11}, p_{12}, p_{13}, p_{14}\}$;
  2) delete $\neg p_{11}$ and generate 9 new unit clauses;
  3) delete 65 clauses subsequently!

DPLL: 4 queens

Running the DPLL algorithm:

- Start with $p_{11} \mapsto 1$. Then,
  1) delete $\{p_{11}, p_{12}, p_{13}, p_{14}\}$;
  2) delete $\neg p_{11}$ and generate 9 new unit clauses;
  3) delete 65 clauses subsequently!

- Next, set $p_{23} \mapsto 1$. Then,
  1) generate 4 new unit clauses: $\{\neg p_{24}\}, \{\neg p_{43}\}, \{\neg p_{32}\}, \{\neg p_{34}\}$;
  2) through the unit propagation of $\{\neg p_{34}\}$, reach UNSAT.

DPLL: 4 queens

Running the DPLL algorithm:

- Start with $p_{11} \mapsto 1$. Then,
  1) delete $\{p_{11}, p_{12}, p_{13}, p_{14}\}$;
  2) delete $\neg p_{11}$ and generate 9 new unit clauses;
  3) delete 65 clauses subsequently!

- Next, set $p_{23} \mapsto 1$. Then,
  1) generate 4 new unit clauses: $\{\neg p_{24}\}, \{\neg p_{43}\}, \{\neg p_{32}\}, \{\neg p_{34}\}$;
  2) through the unit propagation of $\{\neg p_{34}\}$, reach UNSAT.

- Fixing only two literals collapsed 80 clauses to 1 and ruled out 2^{14} of 2^{16} possible assignments!

- Backtrack and set $\langle p_{11} \mapsto 0, p_{12} \mapsto 1\rangle$. Then,
  1) Delete $\{\neg p_{12}\}$ and generate 9 new unit clauses;
  2) Do unit propagation, which leaves only 1 clause $\{p_{43}\}$!

DPLL: 4 queens

Running the DPLL algorithm:

- Start with $p_{11} \mapsto 1$. Then,
  1) delete $\{p_{11}, p_{12}, p_{13}, p_{14}\}$;
  2) delete $\neg p_{11}$ and generate 9 new unit clauses;
  3) delete 65 clauses subsequently!

- Next, set $p_{23} \mapsto 1$. Then,
  1) generate 4 new unit clauses: $\{\neg p_{24}\}, \{\neg p_{43}\}, \{\neg p_{32}\}, \{\neg p_{34}\}$;
  2) through the unit propagation of $\{\neg p_{34}\}$, reach UNSAT.

- Fixing only two literals collapsed 80 clauses to 1 and ruled out 2^{14} of 2^{16} possible assignments!

- Backtrack and set $\langle p_{11} \mapsto 0, p_{12} \mapsto 1\rangle$. Then,
  1) Delete $\{\neg p_{12}\}$ and generate 9 new unit clauses;
  2) Do unit propagation, which leaves only 1 clause $\{p_{43}\}$!

- Answer: $p_{12}, p_{24}, p_{31}, p_{43} \mapsto 1$.

Summary

Resolution:

- Very simple sound and complete proof calculus.

Davis–Putnam algorithm:

- Uses resolution to decide SAT via variable elimination.
- But may generate huge intermediate formulas.

DPLL algorithm:

- Improves resolution with clause learning and backtracking.
- Efficient.
- Basis for modern SAT solvers.