Artificial Intelligence

9. Propositional Reasoning, Part II: SAT Solvers

How to *Efficiently* Think About What is True or False

Jörg Hoffmann, Daniel Fiser, Daniel Höller, Sophia Saller

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Summer Term 2022

Agenda

1 Introduction

2 The Davis-Putnam (Logemann-Loveland) Procedure

3 DPLL = (A Restricted Form of) Resolution

4 *Why* Did Unit Propagation Yield a Conflict?

5 Clause Learning

6 Phase Transitions: Where the *Really* Hard Problems Are

7 Conclusion

Reminder: Our Agenda for This Topic

→ Our treatment of the topic "Propositional Reasoning" consists of Chapters 8 and 9.

- **Chapter 8:** Basic definitions and concepts; resolution.

  → Sets up the framework. Resolution is the quintessential reasoning procedure underlying most successful solvers.

- **This Chapter:** The Davis-Putnam procedure and clause learning; practical problem structure.

  → State-of-the-art algorithms for reasoning about propositional logic, and an important observation about how they behave.

SAT

The SAT Problem: Given a propositional formula φ, decide whether or not φ is satisfiable.

- The first problem proved to be **NP**-complete!
- φ is commonly assumed to be in CNF. This is without loss of generality, because any φ can in polynomial time be transformed into a satisfiability-equivalent CNF formula (cf. **Chapter 8**).
- Active research area, annual SAT conference, lots of tools etc. available: `http://www.satlive.org/`
- Tools addressing SAT are commonly referred to as SAT solvers.

Reminder: To decide whether KB $\models \varphi$, decide satisfiability of $\theta :=$ KB $\cup \{\neg\varphi\}$: θ is unsatisfiable iff KB $\models \varphi$.

$\rightarrow$ Deduction can be performed using SAT solvers.

Reminder: General Problem Solving using Logic

(some new problem)

model problem in logic $\mapsto$ use off-the-shelf reasoning tool

(its solution)

- "Any problem that can be formulated as reasoning about logic."
- *Very* successful using propositional logic and modern solvers for SAT! (Propositional satisfiability testing, $\rightarrow$ **This Chapter**.)

SAT vs. CSP

Reminder: Constraint network $\gamma = (V, D, C)$ has variables $v \in V$ with finite domains $D_v \in D$, and binary constraints $C_{uv} \in C$ which are relations over u, v specifying the permissible combined assignments to u and v. One extension is to allow constraints of higher arity.

SAT = A kind of CSP:

$\rightarrow$ SAT can be viewed as a CSP problem in which all variable domains are Boolean, and the constraints have unbounded arity.

Encoding CSP as SAT:

$\rightarrow$ Given any constraint network γ, we can in low-order polynomial time construct a CNF formula $\varphi(\gamma)$ that is satisfiable iff γ is solvable.

$\rightarrow$ Anything we can do with CSP, we can (in principle) do with SAT.

Reminder: Conventions

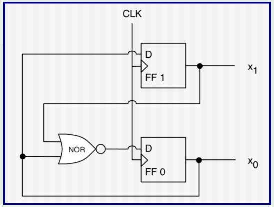
Terminology and Notation

- A literal l is an atom or the negation thereof (e.g., $P, \neg Q$); the negation of a literal is denoted $\bar{l}$ (e.g., $\overline{\neg Q} = Q$).
- A clause C is a disjunction of literals. We identify C with the set of its literals (e.g., $P \vee \neg Q$ becomes $\{P, \neg Q\}$).
- We identify a CNF formula ψ with the set Δ of its clauses (e.g., $(P \vee \neg Q) \wedge R$ becomes $\{\{P, \neg Q\}, \{R\}\}$).
- The empty clause is denoted $\Box$.

$\rightarrow$ For the remainder of this chapter, we assume that the input is a set Δ of clauses.

Example Application: Hardware Verification

Example



- Counter, repeatedly from $c = 0$ to $c = 2$.
- 2 bits x_1 and x_0; $c = 2 * x_1 + x_0$.
- ("FF" Flip-Flop, "D" Data IN, "CLK" Clock)
- **To Verify:** If $c < 3$ in current clock cycle, then $c < 3$ in next clock cycle.

Step 1: Encode into propositional logic.

- Propositions: x_1, x_0; and x_1', x_0' (value in next cycle).
- Transition relation: $x_1' \leftrightarrow x_0$; $x_0' \leftrightarrow \neg(x_1 \vee x_0)$.
- Initial state: $\neg(x_1 \wedge x_0)$. Error property: $x_1' \wedge x_0'$.

Step 2: Transform to CNF, encode as set Δ of clauses.
$\rightarrow \{\{\neg x_1', x_0\}, \{x_1', \neg x_0\}, \{x_0', x_1, x_0\}, \{\neg x_o', \neg x_1\}, \{\neg x_0', \neg x_0\}, \{\neg x_1, \neg x_0\}, \{x_1'\}, \{x_0'\}\}$

Step 3: Call a SAT solver (up next).

Our Agenda for This Chapter

- **The Davis-Putnam (Logemann-Loveland) Procedure:** How to systematically test satisfiability?
  → The quintessential SAT solving procedure, DPLL.
- **DPLL = (A Restricted Form of) Resolution:** How does this relate to what we did in the last chapter?
  → Mathematical understanding of DPLL.
- **Why Did Unit Propagation Yield a Conflict?** How can we analyze which mistakes were made in "dead" search branches?
  → Knowledge is power, see next.
- **Clause Learning:** How can we learn from our mistakes?
  → One of the key concepts, perhaps *the* key concept, underlying the success of SAT.
- **Phase Transitions: Where the Really Hard Problems Are:** Are *all* formulas "hard" to solve?
  → The answer is "no". And in some cases we can figure out exactly when they are/aren't hard to solve.

But – What About Local Search for SAT?

There's a wealth of research on local search for SAT, e.g.:

GSAT Algorithm

INPUT: a set of clauses Δ, Max-Flips, and Max-Tries
OUTPUT: a satisfying truth assignment of Δ, if found

```
for $i := 1$ to Max-Tries
  $I :=$ a randomly-generated truth assignment
  for $j := 1$ to Max-Flips
    if $I$ satisfies $\Delta$ then return $I$
    $X :=$ a proposition reversing whose truth assignment gives
          the largest increase in the number of satisfied clauses
    $I := I$ with the truth assignment of $X$ reversed
  end for
end for
return "no satisfying assignment found"
```

$\rightarrow$ Local search is not as successful in SAT applications, and the underlying ideas are very similar to those presented in **Chapter 4**. Not covered here.

SAT Solvers

SAT solver: Δ, returns interpretation I so that $I \models \Delta$.

Complete SAT solver: If such I does not exist, returns "unsatisfiable".

$\rightarrow$ The DPLL procedure is a complete SAT solver.

Uses of SAT solvers:

- Like a calculus:
  - Wanted: Does Δ entail a formula φ?
  - Reduction to unsatisfiability of $\Delta \cup \{\neg\varphi\}$.
  - Complexity: **co-NP**.
- Like a search for solutions:
  - Wanted: Model of Δ, i.e., I such that $I \models \Delta$.
  - Complexity: **NP**; can be "easier" in practice because we can stop once we found the first I that works.

The DPLL Procedure

Call on input Δ and the empty partial interpretation I:

```
function DPLL($\Delta, I$) returns a partial interpretation $I$, or "unsatisfiable"
/* Unit Propagation (UP) Rule: */
  $\Delta'$ := a copy of $\Delta$; $I' := I$
  while $\Delta'$ contains a unit clause $C = \{l\}$ do
      extend $I'$ with the respective truth value for the proposition underlying $l$
      simplify $\Delta'$      /* remove false literals and true clauses */
/* Termination Test: */
  if $\Box \in \Delta'$ then return "unsatisfiable"
  if $\Delta' = \{\}$ then return $I'$
/* Splitting Rule: */
  select some proposition $P$ for which $I'$ is not defined
  $I'' := I'$ extended with one truth value for $P$; $\Delta''$ := a copy of $\Delta'$; simplify $\Delta''$
  if $I''' :=$ DPLL($\Delta'', I''$) $\neq$ "unsatisfiable" then return $I'''$
  $I'' := I'$ extended with the other truth value for $P$; $\Delta'' := \Delta'$; simplify $\Delta''$
  return DPLL($\Delta'', I''$)
```

$\rightarrow$ In practice, of course one uses flags etc. instead of "copy".

DPLL: Example (Vanilla1)

$$\Delta = \{\{P, Q, \neg R\}, \{\neg P, \neg Q\}, \{R\}, \{P, \neg Q\}\}$$

1. UP Rule: $R \mapsto 1$
   $\{\{P, Q\}, \{\neg P, \neg Q\}, \{P, \neg Q\}\}$
2. Splitting Rule:

2a. $P \mapsto 0$
$\{\{Q\}, \{\neg Q\}\}$

3a. UP Rule: $Q \mapsto 1$
$\{\Box\}$

2b. $P \mapsto 1$
$\{\{\neg Q\}\}$

3b. UP Rule: $Q \mapsto 0$
$\{\}$

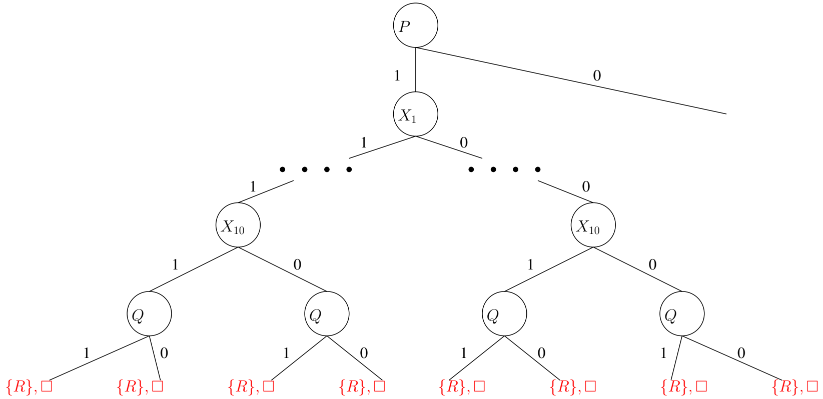
DPLL: Example (Vanilla2)

$$\Delta = \{\{\neg Q, \neg P\}, \{P, \neg Q, \neg R, \neg S\}, \{Q, \neg S\}, \{R, \neg S\}, \{S\}\}$$

1. UP Rule: $S \mapsto 1$
   $\{\{\neg Q, \neg P\}, \{P, \neg Q, \neg R\}, \{Q\}, \{R\}\}$
2. UP Rule: $Q \mapsto 1$
   $\{\{\neg P\}, \{P, \neg R\}, \{R\}\}$
3. UP Rule: $R \mapsto 1$
   $\{\{\neg P\}, \{P\}\}$
4. UP Rule: $P \mapsto 1$
   $\{\Box\}$

DPLL: Example (Redundance1)

$$\Delta = \{\{\neg P, \neg Q, R\}, \{\neg P, \neg Q, \neg R\}, \{\neg P, Q, R\}, \{\neg P, Q, \neg R\},$$
$$\{X_1, \dots, X_{100}\}, \{\neg X_1, \dots, \neg X_{100}\}\}$$



Properties of DPLL

Unsatisfiable case:

- What can we say if “unsatisfiable” is returned?

  $\rightarrow$ In this case, we know that Δ is unsatisfiable: Unit propagation is *sound*, in the sense that it does not reduce the set of solutions. (= Soundness of calculus, cf. next two slides.)

Satisfiable case:

- What can we say when a partial interpretation I is returned?

  $\rightarrow$ Any extension of I to a complete interpretation satisfies Δ. (By construction, I suffices to satisfy all clauses.)

Déjà Vu, Anybody?

- DPLL $\approx$ BacktrackingWithInference, with Inference() = unit propagation.
- Unit propagation is sound: It does not reduce the set of solutions. (Also: = Soundness of calculus, cf. next slide.)

UP = Unit Resolution

The Unit Propagation (UP) Rule . . .

```
while $\Delta'$ contains a unit clause $\{l\}$ do
    extend $I'$ with the respective truth value for the proposition underlying $l$
    simplify $\Delta'$      /* remove false literals */
```

. . . corresponds to a calculus:

Definition (Unit Resolution). *Unit Resolution is the calculus consisting of the following inference rule:*

$$\frac{C\dot{\cup}\{\bar{l}\}, \{l\}}{C}$$

That is, if Δ contains parent clauses of the form $C\dot{\cup}\{\bar{l}\}$ and $\{l\}$, the rule allows to add the resolvent clause C.

$\rightarrow$ Unit propagation = Resolution restricted to the case where one of the parent clauses is unit.

UP/Unit Resolution: Soundness/Completeness

Soundness:

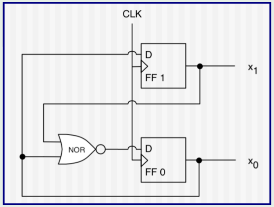
- Need to show: *If Δ' can be derived from Δ by UP, then $\Delta \models \Delta'$.*
- Yes, because any derivation made by unit resolution can also be made by (full) resolution, which we already know has this property (cf. **Chapter 8**).
- (Intuitively: if Δ' contains the unit clause $\{l\}$, then l must be made true so $C \dot\cup \{\bar{l}\}$ implies C.)

Completeness:

- Need to show: *If $\Delta \models \Delta'$, then Δ' can be derived from Δ by UP.*
- No. UP makes only limited inferences, as long as there are unit clauses. It does not guarantee to infer everything that can be inferred.
- Example: $\{\{P,Q\}, \{P,\neg Q\}, \{\neg P, Q\}, \{\neg P, \neg Q\}\}$ is unsatisfiable but UP cannot derive the empty clause $\Box$.

Questionnaire

Example



- Counter, repeatedly from $c = 0$ to $c = 2$.
- **To Verify:** If $c < 3$ in current clock cycle, then $c < 3$ in next clock cycle.
- $\Delta = \{\{\neg x'_1, x_0\}, \{x'_1, \neg x_0\}, \{x'_0, x_1, x_0\}, \{\neg x'_o, \neg x_1\}, \{\neg x'_0, \neg x_0\}, \{\neg x_1, \neg x_0\}, \{x'_1\}, \{x'_0\}\}$

Question!

How many recursive calls to DPLL are made on Δ?

(A): 0	(B): 1
(C): 4	(D): 11

$\rightarrow$ The correct answer is (B): UP derives the empty clause (via $\{x'_1\}$, $\{\neg x'_1, x_0\}$, $\{\neg x'_0, \neg x_0\}$, $\{x'_0\}$) in the first recursive call, so exactly 1 search node is generated.

DPLL vs. Resolution

Notation: Define the number of decisions of a DPLL run as the total number of times a truth value was set by either unit propagation or the splitting rule.

Theorem. *If DPLL returns "unsatisfiable" on Δ, then $\Delta \vdash \Box$ with a resolution derivation whose length is at most the number of decisions.*

Proof Sketch. Consider first DPLL without the unit propagation rule.

Consider any leaf node N, for proposition X, both of whose truth values directly result in a clause C that has become empty.

Then for $X = 0$ the respective clause C must contain X; and for $X = 1$ the respective clause C must contain $\neg X$. Thus we can resolve these two clauses to a clause $C(N)$ that does not contain X.

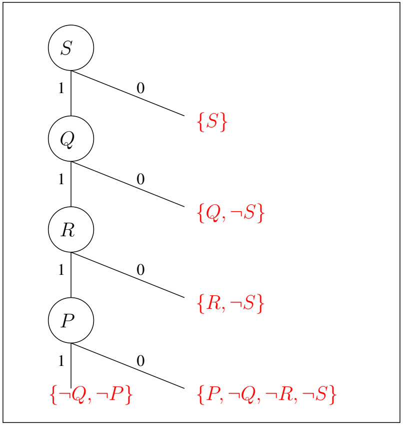
$C(N)$ can contain only the negations of the decision literals $l_1, \dots, l_k$ above N. Remove N from the tree, then iterate the argument. Once the tree is empty, we have derived the empty clause.

Unit propagation can be simulated via applications of the splitting rule, choosing a proposition that is constrained by a unit clause: One of the two truth values then immediately yields an empty clause.

DPLL vs. Resolution: Example (Vanilla2)

Example: $\Delta = \{\{\neg Q, \neg P\}, \{P, \neg Q, \neg R, \neg S\}, \{Q, \neg S\}, \{R, \neg S\}, \{S\}\}$

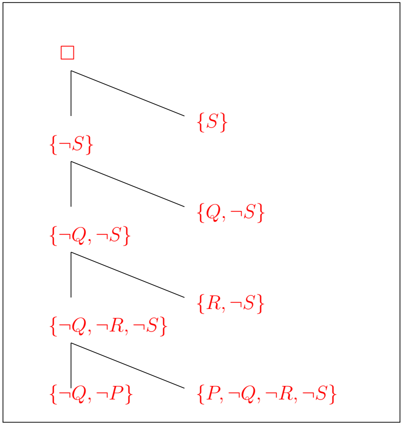
DPLL: (Without UP; leaves annotated with clauses that became empty)



DPLL vs. Resolution: Example (Vanilla2)

Example: $\Delta = \{\{\neg Q, \neg P\}, \{P, \neg Q, \neg R, \neg S\}, \{Q, \neg S\}, \{R, \neg S\}, \{S\}\}$

Resolution Proof from that DPLL Tree:



DPLL vs. Resolution

For reference only:

Theorem. *If DPLL returns "unsatisfiable" on Δ, then $\Delta \vdash \Box$ with a resolution derivation whose length is at most the number of decisions.*

Proof. Consider first DPLL with no unit propagation. If the search tree is not empty, then there exists a leaf node N, i.e., a node associated to proposition X so that, for each value of X, the partial assignment directly results in an empty clause. Denote the parent decisions of N by $l_1, \dots, l_k$, where l_i is a literal for proposition X_i and the search node containing X_i is N_i. Denote the empty clause for X by $C(N, X)$, and denote the empty clause for $\neg X$ by $C(N, \neg X)$.

For each $x \in \{X, \neg X\}$ we have the following properties: (a) $\neg x \in C(N, x)$; and (b) $C(N, x) \subseteq \{\neg x, \overline{l_1}, \dots, \overline{l_k}\}$. Due to (a), we can resolve $C(N, X)$ with $C(N, \neg X)$; denote the outcome clause by $C(N)$.

We obviously have that (1) $C(N) \subseteq \{\overline{l_1}, \dots, \overline{l_k}\}$. The proof now proceeds by removing N from the search tree and attaching $C(N)$ at the l_k branch of N_k, in the role of $C(N_k, l_k)$ as above. Then we select the next leaf node N' and iterate the argument; once the tree is empty, by (1) we have derived the empty clause. What we need to show is that, in each step of this iteration, we preserve the properties (a) and (b) for all leaf nodes. Since we did not change anything in other parts of the tree, the only node we need to show this for is $N' := N_k$.

Due to (1), we have (b) for N_k. But we do not necessarily have (a): $C(N) \subseteq \{\overline{l_1}, \dots, \overline{l_k}\}$, but there are cases where $\overline{l_k} \not\in C(N)$ (e.g., if X_k is not contained in any clause and thus branching over it was completely unnecessary). If so, however, we can simply remove N_k and all its descendants from the tree as well. We attach $C(N)$ at the l_{k-1} branch of N_{k-1}, in the role of $C(N_{k-1}, l_{k-1})$. If $\overline{l_{k-1}} \in C(N)$ then we have (a) for $N' := N_{k-1}$ and can stop. If $\neg l_{k-1} \not\in C(N)$, then we remove N_{k-1} and so forth, until either we stop with (a), or have removed N_1 and thus must already have derived the empty clause (because $C(N) \subseteq (\{\overline{l_1}, \dots, \overline{l_k}\} \setminus \{\overline{l_1}, \dots, \overline{l_k}\})$).

Unit propagation can be simulated via applications of the splitting rule, choosing a proposition that is constrained by a unit clause: One of the two truth values then immediately yields an empty clause.

DPLL vs. Resolution: Discussion

So What? The theorem we just proved helps to *understand* DPLL:

$\rightarrow$ DPLL is an effective practical method for conducting resolution proofs.

In Fact:

- DPLL = tree resolution.
- This is a fundamental weakness! There are inputs Δ whose shortest tree-resolution proof is exponentially longer than their shortest (general) resolution proof.

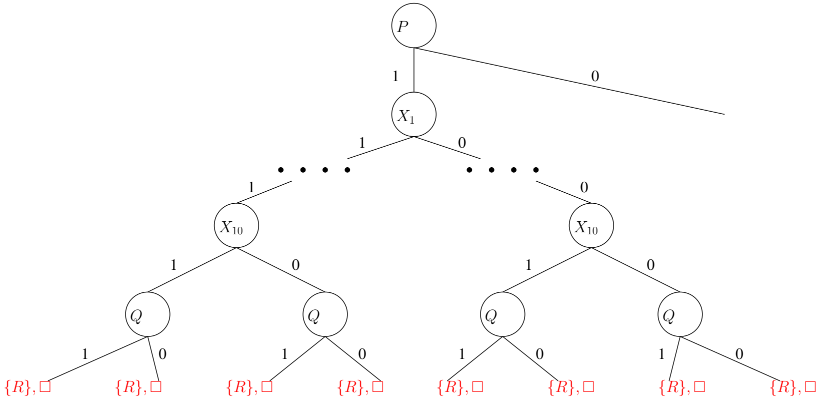
  $\rightarrow$ In a tree resolution, each derived clause C is used only once (at its parent). The same C is derived anew every time it is used!

$\rightarrow$ DPLL "makes the same mistakes over and over again".

$\rightarrow$ To the rescue: clause learning.

Same Mistakes over Again: Example (Redundance1)

$$\Delta = \{\{\neg P, \neg Q, R\}, \{\neg P, \neg Q, \neg R\}, \{\neg P, Q, R\}, \{\neg P, Q, \neg R\}, \{X_1, \dots, X_{100}\}, \{\neg X_1, \dots, \neg X_{100}\}\}$$



Note: Here, the problem could be avoided by splitting over different variables. This is not so in general! (See slide 47.)

How To *Not* Make the Same Mistakes Over Again?

. . . it's not that difficult, really:

A. Figure out what went wrong.
B. Learn to not do that again in the future.

And now for DPLL:

A. *Why* Did Unit Propagation Yield a Conflict?

   $\rightarrow$ This section. We will capture the "what went wrong" in terms of graphs over literals set during the search, and their dependencies.

B. What can we learn from that information?

   $\rightarrow$ A new clause! Next section.

Implication Graphs

Notation/Terminology: Literals set along a branch of DPLL

- Value of P set by the splitting rule: choice literal, P for $I(P) = 1$, respectively $\neg P$ for $I(P) = 0$.
- Value of P set by the UP rule: implied literal P respectively $\neg P$.
- Empty clause derived by UP: conflict literal $\Box$.

Definition (Implication Graph). *Let Δ be a set of clauses, and consider any search branch β of DPLL on Δ. The implication graph G^{impl} is a directed graph. Its vertices are the choice and implied literals along β, as well as a separate conflict vertex $\Box_C$ for every clause C that became empty.*

Where $\{l_1, \dots, l_k, l'\} \in \Delta$ became unit with implied literal l', G^{impl} includes the arcs $\overline{l_1} \to l'$, $\dots$, $\overline{l_k} \to l'$. Where $C = \{l_1, \dots, l_k\} \in \Delta$ became empty, G^{impl} includes the arcs $\overline{l_1} \to \Box_C$, $\dots$, $\overline{l_k} \to \Box_C$.

- How do we know that $\overline{l_1}$, $\dots$, $\overline{l_k}$ are vertices in G^{impl}: Because $\{l_1, \dots, l_k, l'\}$ became unit respectively empty.
- Vertices with indegree 0: Choice literals, and unit clauses of Δ.

Implication Graphs: Example (Vanilla1) in Detail

$$\Delta = \{\{P, Q, \neg R\}, \{\neg P, \neg Q\}, \{R\}, \{P, \neg Q\}\}$$

1. UP Rule: $R \mapsto 1$
   Implied literal R.
   $\{\{P, Q\}, \{\neg P, \neg Q\}, \{P, \neg Q\}\}$
2. Splitting Rule:

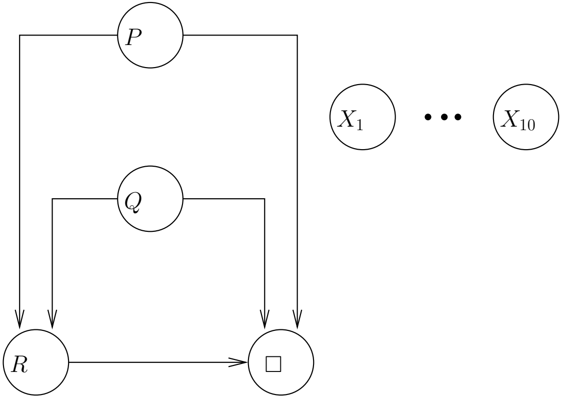
2a. $P \mapsto 0$
Choice literal $\neg P$.
$\{\{Q\}, \{\neg Q\}\}$

3a. UP Rule: $Q \mapsto 1$
Implied literal Q, arcs $R \to Q$ and $\neg P \to Q$.
$\{\Box\}$
Conflict literal $\Box$, arcs $\neg P \to \Box_{\{P,\neg Q\}}$ and $Q \to \Box_{\{P,\neg Q\}}$.

Implication Graphs: Example (Redundance1)

$$\Delta = \{\{\neg P, \neg Q, R\}, \{\neg P, \neg Q, \neg R\}, \{\neg P, Q, R\}, \{\neg P, Q, \neg R\}, \{X_1, \dots, X_{100}\}, \{\neg X_1, \dots, \neg X_{100}\}\}$$

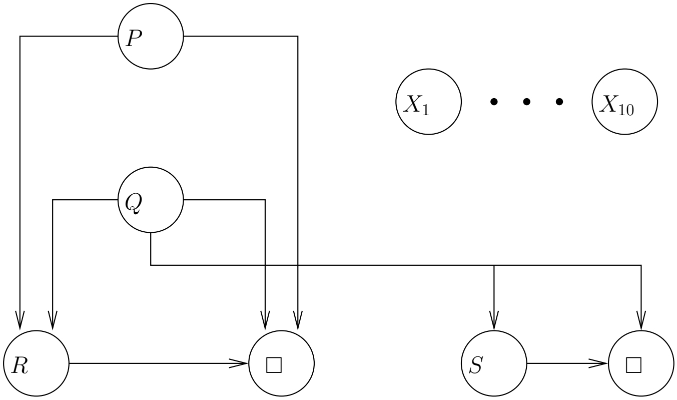
Choice: $P, X_1, \dots, X_{10}, Q$. Implied: R.



Implication Graphs: Example (Redundance2)

$$\Delta = \{\{\neg P, \neg Q, R\}, \{\neg P, \neg Q, \neg R\}, \{\neg P, Q, R\}, \{\neg P, Q, \neg R\}, \{\neg Q, S\}, \{\neg Q, \neg S\} \{X_1, \dots, X_{100}\}, \{\neg X_1, \dots, \neg X_{100}\}\}$$

Choice: $P, X_1, \dots, X_{10}, Q$. Implied: R, S.



Questionnaire

Recall: Where $\{l_1, \dots, l_k, l'\} \in \Delta$ became unit on search branch β, with implied literal l', G^{impl} includes the arcs $\overline{l_1} \rightarrow l'$, …, $\overline{l_k} \rightarrow l'$.

Question!

Can implication graphs have cycles?

(A): Yes (B): No

$\rightarrow$ No, because the implication graph keeps track of chronological behavior along the current DPLL search branch β. Unit propagation cannot derive l' whose value was already set beforehand.

In detail: Assume there is a cycle. Consider the first time point along the search branch where the cycle occurs, because an arc $(\overline{l_i}, l')$ is added to the implication graph. Then (a) l' is implied because $\{l_1, \dots, l_k, l'\} \in \Delta$ just became unit $\{l'\}$; and (b) as $(\overline{l_i}, l')$ closes a cycle, l' must already have been in the implication graph beforehand. (a) and (b) are in contradiction.

Implication Graphs: A Remark

$\rightarrow$ The implication graph is *not* uniquely determined by the choice literals.

Because:

- The implication graph also depends on “ordering decisions” made during UP: Which unit clause is picked first.
- Example: $\Delta = \{\{\neg P, \neg Q\}, \{Q\}, \{P\}\}$

Option 1:

$$\begin{tikzcd}
& Q \arrow[dr] & \\
\Box & & \neg P \arrow[ll]
\end{tikzcd}$$

Option 2:

$$\begin{tikzcd}
& P \arrow[dr] & \\
\Box & & \neg Q \arrow[ll]
\end{tikzcd}$$

Conflict Graphs

→ A conflict graph captures "what went wrong" in a failed node.

Definition (Conflict Graph). *Let Δ be a set of clauses, and let G^{impl} be the implication graph for some search branch of DPLL on Δ. A conflict graph G^{confl} is a sub-graph of G^{impl} induced by a subset of vertices such that:*

(i) *G^{confl} contains exactly one conflict vertex $\Box_C$.*

(ii) *If l' is a vertex in G^{confl}, then all parents of l', i.e. vertices $\overline{l_i}$ with a G^{impl} arc $(\overline{l_i}, l')$, are vertices in G^{confl} as well.*

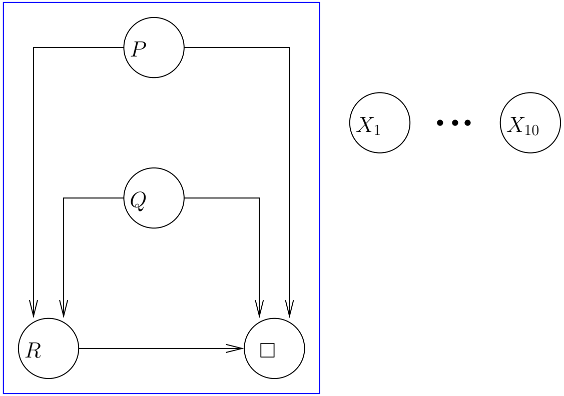
(iii) *All vertices in G^{confl} have a path to $\Box_C$.*

→ Conflict graph = Starting at a conflict vertex, backchain through the implication graph until reaching choice literals.

Conflict Graphs: Example (Redundance1)

$$\Delta = \{\{\neg P, \neg Q, R\}, \{\neg P, \neg Q, \neg R\}, \{\neg P, Q, R\}, \{\neg P, Q, \neg R\}, \{X_1, \dots, X_{100}\}, \{\neg X_1, \dots, \neg X_{100}\}\}$$

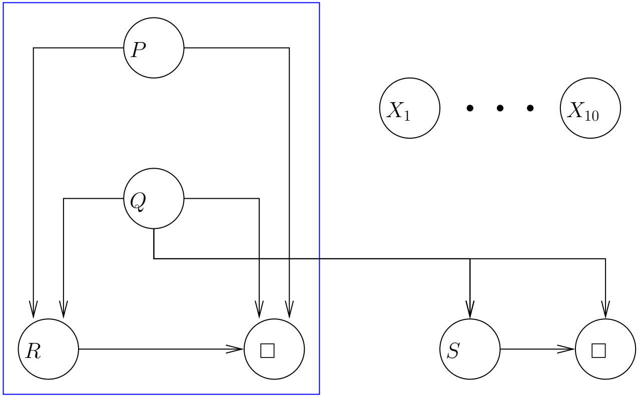
Choice: $P, X_1, \dots, X_{10}, Q$. Implied: R.



Conflict Graphs: Example (Redundance2)

$$\Delta = \{\{\neg P, \neg Q, R\}, \{\neg P, \neg Q, \neg R\}, \{\neg P, Q, R\}, \{\neg P, Q, \neg R\}, \{\neg Q, S\}, \{\neg Q, \neg S\} \{X_1, \dots, X_{100}\}, \{\neg X_1, \dots, \neg X_{100}\}\}$$

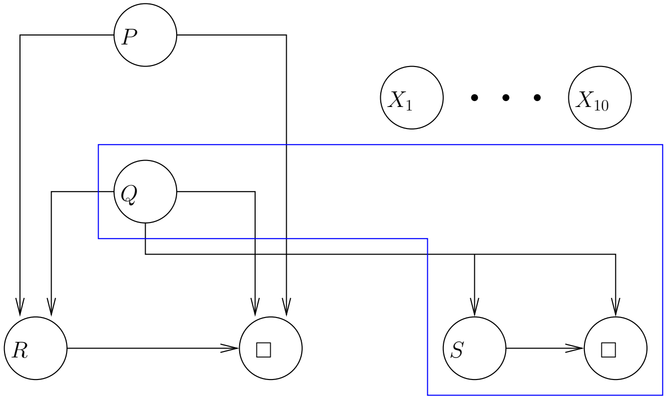
Choice: $P, X_1, \dots, X_{10}, Q$. Implied: R, S.



Conflict Graphs: Example (Redundance2)

$$\Delta = \{\{\neg P, \neg Q, R\}, \{\neg P, \neg Q, \neg R\}, \{\neg P, Q, R\}, \{\neg P, Q, \neg R\}, \{\neg Q, S\}, \{\neg Q, \neg S\} \{X_1, \dots, X_{100}\}, \{\neg X_1, \dots, \neg X_{100}\}\}$$

Choice: $P, X_1, \dots, X_{10}, Q$. Implied: R, S.



Questionnaire

Recall: The implication graph depends on "ordering decisions" during UP: Which unit clause is picked first. E.g. $\Delta = \{\{\neg P, \neg Q\}, \{Q\}, \{P\}\}$.

$$\begin{tikzcd} & Q \arrow[d] & & & P \arrow[d] \\ \Box & \neg P \arrow[l] & & \Box & \neg Q \arrow[l] \end{tikzcd}$$

Question!

Does the existence of a conflict graph depend on these decisions?

(A): Yes (B): No

$\rightarrow$ Observe: A conflict graph exists iff $\Box$ is UP-derivable in the current simplified formula Δ'. So the question is whether it can happen that, when propagating a unit clause $\{l\}$ in Δ', on the resulting simplified Δ'' the UP calculus cannot derive the empty clause anymore.

The answer is no. Δ'' can be obtained in two steps: 1. Remove $\bar{l}$ from every $C \in \Delta'$ where $\bar{l} \in C$ to obtain Δ'_l. 2. Remove $C \in \Delta'_l$ where $l \in C$ to obtain Δ''. 1. cannot hurt $\Box$-derivability because every clause of Δ'_l is a sub-clause of Δ', and smaller clauses can only be better. 2. cannot hurt $\Box$-derivability because $\bar{l}$ does not occur anywhere in Δ'_l (so if $l \in C$ then no derivative of C can ever become empty).

Questionnaire, ctd.

Question!

How many conflict graphs do we get for the choice literal $\neg R$, when running UP on $\{\{P, Q, R\}, \{\neg P, Q, R\}, \{S, R\}, \{\neg S, R\}\}$?

(A): 0 (B): 1

(C): 2 (D): 3

$\rightarrow$ (B) is correct: The only conflict we get is via $\{S, R\}$, $\{\neg S, R\}$, and the choice literal $\neg R$.

Question!

And for the choice literals $\neg Q, \neg R$?

$\rightarrow$ (C) is correct: We get the above conflict, and another one via $\{P, Q, R\}$, $\{\neg P, Q, R\}$, and the choice literals $\neg Q$ and $\neg R$.

(Note: These choices can happen in DPLL on Δ, if we choose $\neg Q$ first.)

Clause Learning

Observe: Conflict graphs encode *logical entailments*

$$\Delta \models (\bigwedge_{l \in choiceLits(G^{\mathsf{confl}})} l) \rightarrow \bot$$

→ Given Δ, setting all choice literals in a conflict graph results in failure.

Observe: We can re-write this!

$$\Delta \models \bigvee_{l \in choiceLits(G^{\mathsf{confl}})} \bar{l}$$

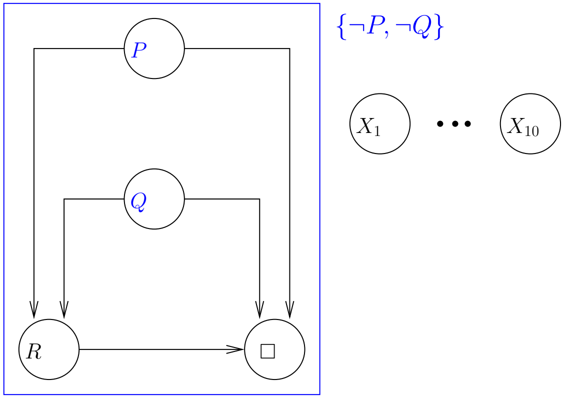
Proposition (Clause Learning). *Let Δ be a set of clauses, and let G^{confl} be a conflict graph at some time point during a run of DPLL on Δ. Let $choiceLits(G^{\mathsf{confl}})$ be the choice literals in G^{confl}. Then $\Delta \models \{\bar{l} \mid l \in choiceLits(G^{\mathsf{confl}})\}$.*

→ The negation of the choice literals in a conflict graph is a valid clause.

Clause Learning: Example (Redundance1)

$$\Delta = \{\{\neg P, \neg Q, R\}, \{\neg P, \neg Q, \neg R\}, \{\neg P, Q, R\}, \{\neg P, Q, \neg R\}, \{X_1, \dots, X_{100}\}, \{\neg X_1, \dots, \neg X_{100}\}\}$$

Choice: $P, X_1, \dots, X_{10}, Q$. Implied: R.



The Effect of Learned Clauses (in Redundance1)

$\rightarrow$ What happens after we learned a new clause C?

1. **We add C into Δ.** $\rightarrow$ Example: $C = \{\neg P, \neg Q\}$.
2. **We retract the last choice l'.**

   $\rightarrow$ Example: Retract the choice $l' = Q$.

Observation: $C = \{\bar{l} \mid l \in \mathit{choiceLits}(G^{\text{confl}})\}$. Before we learn the clause, G^{confl} must contain the most recent choice l': otherwise, the conflict would have occured earlier on. So $C = \{\overline{l_1}, \dots, \overline{l_k}, \overline{l'}\}$ where $l_1, \dots, l_k$ are earlier choices.
$\rightarrow$ Example: $l_1 = P$, $C = \{\neg P, \neg Q\}$, $l' = Q$.

Observation: Given the earlier choices $l_1, \dots, l_k$, after we learned the new clause $C = \{\overline{l_1}, \dots, \overline{l_k}, \overline{l'}\}$, $\overline{l'}$ is now set by UP!

3. **We set the opposite choice $\overline{l'}$ as an implied literal.**

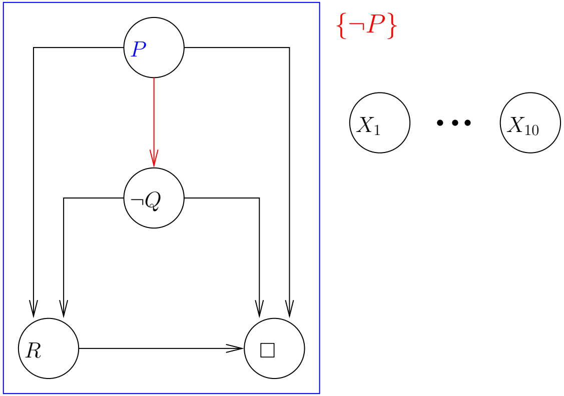
   $\rightarrow$ Example: Set $\neg Q$ as an implied literal.
4. **We run UP and analyze conflicts.** Learned clause: earlier choices only!

   $\rightarrow$ Example: $C = \{\neg P\}$, see next slide.

The Effect of Learned Clauses: Example (Redundance1)

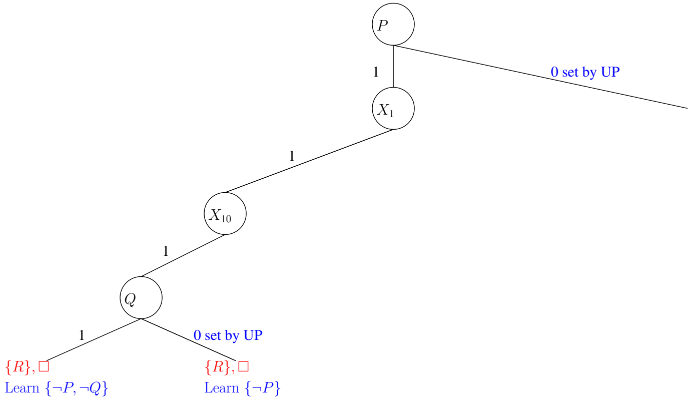
$$\Delta = \{\{\neg P, \neg Q, R\}, \{\neg P, \neg Q, \neg R\}, \{\neg P, Q, R\}, \{\neg P, Q, \neg R\}, \{X_1, \dots, X_{100}\}, \{\neg X_1, \dots, \neg X_{100}\}, \{\neg P, \neg Q\}\}$$

Choice: $P, X_1, \dots, X_{10}$. Implied: $\neg Q$, R.



NOT Same Mistakes over Again: Example (Redundance1)

$$\Delta = \{\{\neg P, \neg Q, R\}, \{\neg P, \neg Q, \neg R\}, \{\neg P, Q, R\}, \{\neg P, Q, \neg R\}, \{X_1, \dots, X_{100}\}, \{\neg X_1, \dots, \neg X_{100}\}\}$$



Note: Here, the problem could be avoided by splitting over different variables. This is not so in general! (see next slide)

Clause Learning vs. Resolution

Remember (slide 26):

1. DPLL $=$ tree resolution: Each derived clause C (not in Δ) is derived anew every time it is used.
2. There exist Δ whose shortest tree-resolution proof is exponentially longer than their shortest (general) resolution proof.

This is no longer the case with clause learning!

1. We add each learned clause C to Δ, can use it as often as we like.
2. Clause learning renders DPLL equivalent to full resolution [Beame *et al.* (2004); Pipatsrisawat and Darwiche (2009)]. (Inhowfar exactly this is the case was an open question for ca. 10 years, so it's not as easy as I made it look here . . .)

$\rightarrow$ In particular: Selecting different variables/values to split on can *provably* not bring DPLL up to the power of DPLL+Clause Learning. (cf. slide 28, and previous slide)

"DPLL + Clause Learning"?

Disclaimer: We have only seen *how to learn a clause from a conflict*. We will *not* cover how the overall DPLL algorithm changes, given this learning. Slides 44 – 46 are merely meant to give a *rough intuition* on "backjumping".

Just for the record: (not exam or exercises relevant)

- One *could* run "DPLL + Clause Learning" by always backtracking to the maximal-level choice variable contained in the learned clause.
- But the actual algorithm is called Conflict-Directed Clause Learning (CDCL), and differs from DPLL more radically:

```
$L := 0$; $I := \emptyset$
repeat
    execute UP
    if a conflict was reached then // $C = \{\overline{l_1}, \dots, \overline{l_k}, \overline{l'}\}$
        if $L = 0$ then return UNSAT
        $L := \max_{i=1}^{k} \mathit{level}(l_i)$; erase $I$ below $L$
        add $C$ into $\Delta$; add $\overline{l'}$ to $I$ at level $L$
    else
        if $I$ is a total interpretation then return $I$
        choose a new decision literal $l$; add $l$ to $I$ at level $L$
        $L := L + 1$
```

Remarks

WHICH clause(s) to learn?

- While we only select $choiceLits(G^{\text{confl}})$, much more can be done.
- For any cut through G^{confl}, with $choiceLits(G^{\text{confl}})$ “above” the cut and the conflict literals ”below” the cut, the literals directly above the border of the cut yield a learnable clause.
- Must take care to *not learn too many clauses* . . .

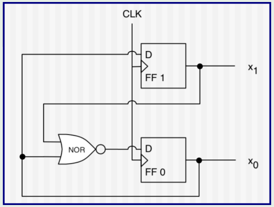
Origins of clause learning:

- Clause learning originates from explanation-based (no-good) learning developed in the CSP community.
- The distinguishing feature here is that the “no-good” is a clause:

  $\rightarrow$ The exact same type of constraint as the rest of Δ.

Questionnaire

Example



- Counter, repeatedly from $c = 0$ to $c = 2$.
- **To Verify:** If $c < 3$ in current clock cycle, then $c < 3$ in next clock cycle.
- $\Delta = \{\{\neg x_1', x_0\}, \{x_1', \neg x_0\}, \{x_0', x_1, x_0\}, \{\neg x_o', \neg x_1\}, \{\neg x_0', \neg x_0\}, \{\neg x_1, \neg x_0\}, \{x_1'\}, \{x_0'\}\}$

Question!

Which clause do we learn after running UP on Δ?

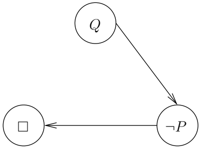
(A): $\Box$ (B): None

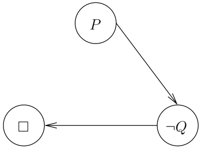
$\rightarrow$ We learn the clause $\Box$. There are no choice literals, so the learned clause is empty.

$\rightarrow$ In case there are no choice literals, the contradiction follows "without assumptions", so we learn immediately that the input formula is unsatisfiable. This special case happens only if the input formula can be proved unsatisfiable using unit propagation (which is rarely the case in practice).

Questionnaire, ctd.

Recall: The implication graph depends on "ordering decisions" during UP: Which unit clause is picked first. E.g. $\Delta = \{\{\neg P, \neg Q\}, \{Q\}, \{P\}\}$.


$$\begin{tikzcd} & Q \arrow[d] \\ \Box & \neg P \arrow[l] \end{tikzcd}$$


$$\begin{tikzcd} & P \arrow[d] \\ \Box & \neg Q \arrow[l] \end{tikzcd}$$

Question!

May the learned clause also change?

(A): Yes (B): No

$\rightarrow$ Yes. Depending on which conflict UP ended up deriving, the conflict graph may differ, and thus the learned clause may differ.

(Note: In the example above, the learned clause in both cases is $\Box$ because there aren't any choice variables.)

Questionnaire, ctd.

Question!

Which clauses can we learn after choosing $\neg R$ and running UP on $\{\{P, Q, R\}, \{\neg P, Q, R\}, \{S, R\}, \{\neg S, R\}\}$?

(A): $\{\neg S, R\}$.

(B): $\{Q, R\}$.

(C): $\{R\}$.

(D): $\Box$.

$\rightarrow$ (A), (B): No: Neither S (in (A)) nor $\neg Q$ (in (B)) is a choice literal.

$\rightarrow$ (C): Yes, via the conflict from $\{S, R\}$, $\{\neg S, R\}$, choice literal $\neg R$.

$\rightarrow$ (D): No: While UP does derive a conflict, that conflict depends on the choice literal $\neg R$.

Questionnaire, ctd.

Question!

Which clauses can we learn after choosing $\neg Q, \neg R$ and running UP on $\{\{P,Q,R\},\{\neg P,Q,R\},\{S,R\},\{\neg S,R\}\}$?

(A): $\{\neg S, R\}$.
(B): $\{Q, R\}$.
(C): $\{R\}$.
(D): $\Box$.

$\rightarrow$ (A): No. S is not a choice literal.

$\rightarrow$ (B): Yes, via the conflict $\{P,Q,R\}$, $\{\neg P,Q,R\}$, with choice literals $\neg Q, \neg R$.

$\rightarrow$ (C): Yes, via the conflict from $\{S,R\}$, $\{\neg S,R\}$, choice literal $\neg R$.

$\rightarrow$ (D): No (same as on previous slide).

(Note: These choices can happen in DPLL on Δ, if we choose $\neg Q$ first.)

Damn, Where Are the Hard Problems?

Err, what?

- SAT is **NP**-hard. Worst case for DPLL is 2^n, with n propositions.
- Imagine I gave you as homework to make a formula family $\{\phi\}$ where DPLL runtime necessarily is in the order of 2^n.

  $\rightarrow$ I promise you're not gonna find this easy . . . (although it is of course possible: e.g., the "Pigeon Hole Problem").
- People noticed by the early 90s that, in practice, the DPLL worst case does not tend to happen.

$\rightarrow$ Modern SAT solvers successfully tackle practical instances where $n > 1000000$.

Damn, Where Are the Hard Problems? Ctd.

So, what's the problem? Science is about *understanding the world*.

→ Are "hard cases" just pathological outliers? Can we say something about the *typical case*?

Difficulty 1: What is the "typical case" in applications? E.g., what is the "average" Hardware Verification instance?

→ Consider precisely defined random distributions instead.

Difficulty 2: Search trees get very complex, and are difficult to analyze mathematically, even in trivial examples. Never mind examples of practical relevance . . .

→ The most successful works are empirical.

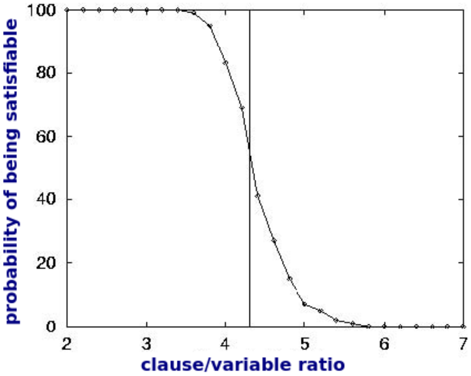
(Interesting theory is mainly concerned with *hand-crafted* formulas, like the Pigeon Hole Problem.)

Phase Transitions in SAT [Mitchell *et al.* (1992)]

Fixed clause length model: Fix clause length k; n variables. Generate m clauses, by uniformly choosing k variables P for each clause C, and for each variable P deciding uniformly whether to add P or $\neg P$ into C.

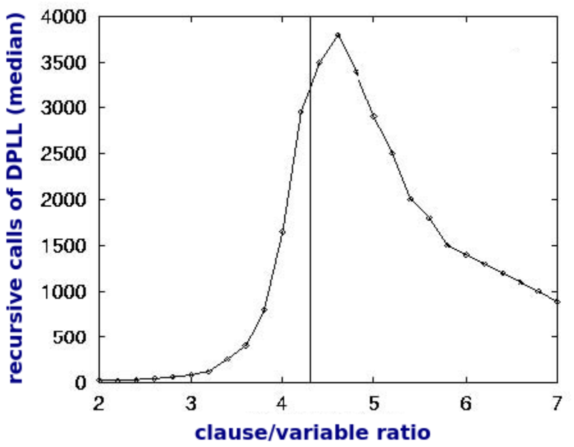
Order parameter: Clause/variable ratio $\frac{m}{n}$.

Phase transition: (Fixing $k = 3$, $n = 50$)



Does DPLL Care?

Oh yes, it does! Extreme runtime peak at the phase transition!



Why Does DPLL Care?

Intuitive explanation:

Under-Constrained:	Satisfiability likelihood close to 1. Many solutions, first DPLL search path usually successful. ("Deep but narrow")
Over-Constrained:	Satisfiability likelihood close to 0. Most DPLL search paths short, conflict reached after few applications of splitting rule. ("Broad but shallow")
Critically Constrained:	At the phase transition, many *almost-successful* DPLL search paths. ("Close, but no cigar")

The Phase Transition Conjecture

Conjecture: [Cheeseman *et al.* (1991)]

Phase Transition Conjecture

"All **NP**-complete problems have at least one order parameter, and the hard to solve problems are around a critical value of this order parameter. This critical value (a phase transition) separates one region from another, such as over-constrained and under-constrained regions of the problem space."

$\rightarrow$ [Cheeseman *et al.* (1991)] confirmed this for Graph Coloring and Hamiltonian Circuits. Later work confirmed it for SAT (see previous slides), and for numerous other **NP**-complete problems.

Why Should *We* Care?

Enlightenment:

- Phase transitions contribute to the fundamental understanding of the behavior of search, even if it's only in random distributions.
- There are interesting theoretical connections to phase transition phenomena in physics. (See [Gomes and Selman (2005)] for a short summary.)

Ok, but what can we use these results for?

- Benchmark design: Choose instances from phase transition region.

  → Commonly used in competitions etc. (In SAT, random phase transition formulas are the most difficult for DPLL-style searches.)
- Predicting solver performance: Yes, but very limited because:

→ All this works only for the particular considered *distributions of instances*! Not meaningful for any other instances.

Questionnaire, ctd.

Question!

Say I encode a Wumpus problem into Δ that turns out to have clause/variable ratio 10. Which is true?

(A): Δ is very likely to be unsatisfiable.

(B): Δ is very likely to be satisfiable.

$\rightarrow$ Neither is true. The clause/var ratio 4.3 has a meaning only for Mitchell *et al.* (1992)'s "fixed clause length model" for generating random CNF formulas.

$\rightarrow$ For other CNF formulas, the clause/variable ratio is completely meaningless!

Extreme example: Say we generate clauses including only positive literals . . .

Practical example: Many Verification problems have huge numbers of clauses but are still satisfiable.

$\rightarrow$ For example, consider the straightforward encoding for "Exactly one of n variables $x_1, \dots, x_n$ is true." We get the clause $\{x_1, \dots, x_n\}$ ("at least one is true") and, for every $1 \leq i \neq j \leq n$, the clause $\{\neg x_i, \neg x_j\}$ ("at most one is true"). The clause/variable ratio is $\frac{n^2-n+2}{2n}$, but the formula is satisfiable.

Questionnaire, ctd.

Question!

Say I sit down tonight and write a random Δ with clause/variable ratio 1.1. Which are true?

(A): I'm bored.

(B): Δ is satisfiable.

(C): All slides for next week are prepared already.

(D): Δ is very likely to be satisfiable.

$\rightarrow$ (A), (C): Definitely true ...

$\rightarrow$ (B): Definitely not a certainty, for any way of generating random CNFs (unless we include only positive, or only negative, literals).

$\rightarrow$ (D): Depends on *how* I generate the CNF. If I use Mitchell *et al.* (1992)'s methods then yes. If I use a different method, then not necessarily.

Summary

- SAT solvers decide satisfiability of CNF formulas. This can be used for deduction, and is highly successful as a general problem solving technique (e.g., in Verification).
- DPLL = backtracking with inference performed by unit propagation (UP), which iteratively instantiates unit clauses and simplifies the formula.
- DPLL proofs of unsatisfiability correspond to a restricted form of resolution. The restriction forces DPLL to "makes the same mistakes over again".
- Implication graphs capture how UP derives conflicts. Their analysis enables us to do clause learning. DPLL with clause learning is called CDCL. It corresponds to full resolution, not "making the same mistakes over again".
- CDCL is state of the art in applications, routinely solving formulas with millions of propositions.
- In particular random formula distributions, typical problem hardness is characterized by phase transitions.

State of the Art in SAT

SAT competitions:

- Since beginning of the 90s `http://www.satcompetition.org/`
- Distinguish random vs. industrial vs. handcrafted benchmarks.
- Largest industrial instances: > 1000000 propositions.

State of the art is CDCL:

- Vastly superior on handcrafted and industrial benchmarks.
- Key techniques: Clause Learning! Also: Efficient implementation (UP!), good branching heuristics, random restarts, portfolios.

What about local search?

- Better on random instances.
- No “dramatic” progress in last decade.
- Parameters are difficult to adjust.

Topics We Didn't Cover Here

- Variable/value selection heuristics: A whole zoo is out there.
- Implementation techniques: One of the most intensely researched subjects. Famous "watched literals" technique for UP had huge practical impact.
- Local search: In space of all truth value assignments. GSAT (slide 11) had huge impact at the time (1992), caused huge amount of follow-up work. Less intensely researched since clause learning hit the scene in the late 90s.
- Portfolios: How to combine several SAT solvers effectively?
- Random restarts: Tackling heavy-tailed runtime distributions.
- Tractable SAT: Polynomial-time sub-classes (most prominent: 2-SAT, Horn formulas).
- MaxSAT: Assign weight to each clause, maximize weight of satisfied clauses (= optimization version of SAT).
- Resolution special cases: There's a universe in between unit resolution and full resolution: trade-off inference vs. search.
- Proof complexity: Can one resolution special case X simulate another one Y polynomially? Or is there an exponential separation (example families where X is exponentially less effective than Y)?

Reading

- *Chapter 7: Logical Agents*, Section 7.6.1 [Russell and Norvig (2010)].

  Content: Here, RN describe DPLL, i.e., basically what I cover under "The Davis-Putnam (Logemann-Loveland) Procedure".

  → That's the only thing they cover of this Chapter's material. (And they even mark it as "can be skimmed on first reading".)

  → This does not do the state of the art in SAT any justice.

- *Chapter 7: Logical Agents*, Sections 7.6.2, 7.6.3, and 7.7 [Russell and Norvig (2010)].

  Content: Sections 7.6.2 and 7.6.3 say a few words on local search for SAT, which I recommend as additional background reading. Section 7.7 describes in quite some detail how to build an agent using propositional logic to take decisions; nice background reading as well.

References I

Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the *really* hard problems are. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 331–337, Sydney, Australia, August 1991. Morgan Kaufmann.

Carla Gomes and Bart Selman. Can get satisfaction. *Nature*, 435:751–752, 2005.

David Mitchell, Bart Selman, and Hector J. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the 10th National Conference of the American Association for Artificial Intelligence (AAAI'92)*, pages 459–465, San Jose, CA, July 1992. MIT Press.

Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers with restarts. In Ian P. Gent, editor, *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP'09)*, volume 5732 of *Lecture Notes in Computer Science*, pages 654–668. Springer, 2009.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (Third Edition)*. Prentice-Hall, Englewood Cliffs, NJ, 2010.