DPLL Satisfiability Algorithm

Deepak D'Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

24, 29, 31 Mar 2021

Outline

Boolean Satisfiability Problem (SAT)

Input: A Boolean formula F.
Output: SAT and a valuation v which satisfies F if F is satisfiable; and UNSAT if F is not satisfiable.

Example formula F

$$(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_1)$$

Output: SAT, $x_1 \mapsto \mathit{false}$, $x_2 \mapsto \mathit{false}$.

Example formula F

$$x_1 \wedge \neg x_1$$

Output: UNSAT.

Importance of Sat Solving

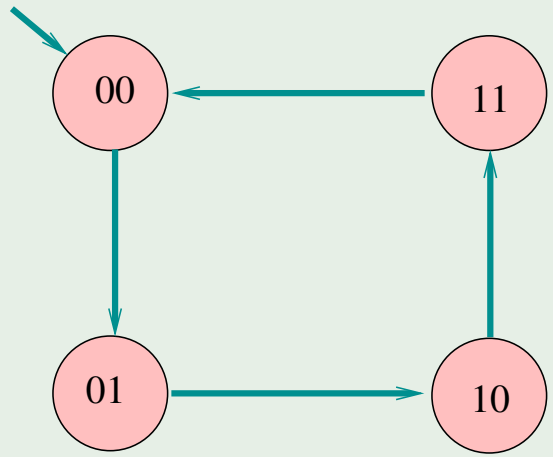
Many practical applications of sat solving

- Bounded Model-Checking,
- (Hardware) Circuit Equivalence checking, Testing, Verification
- Planning, Scheduling, and Optimization

Also the basis of SMT solvers, which have many applications as well.

Boolean SAT solving

Does the system satisfy the temporal logic formula $G(b \implies X(\neg b))$?



In bounded model-checking we could ask for a path of length 2 that violates the specification: Is

$$\neg a_0 \wedge \neg b_0 \wedge T(a_0, b_0, a_1, b_1) \wedge T(a_1, b_1, a_2, b_2) \wedge b_1 \wedge b_2,$$

where $T(a, b, a', b') = (\neg a \wedge a' \wedge b \iff b') \vee (a \wedge \neg a' \wedge b \iff \neg b')$, satisfiable?

A Computationally Difficult Problem

- One of the first problems shown to be NP-Complete (Cook-Levin $\sim$ 1971)
- Easy $2^{O(n)}$-time algorithm (enumerate all valuations and check if F is satisfied).
- Not known to have a $2^{O(\delta n)}$-time algorithm, for $\delta < 1$.
- Yet modern Sat Solvers based on DPLL routinely solve many large size SAT instances.

Basic idea

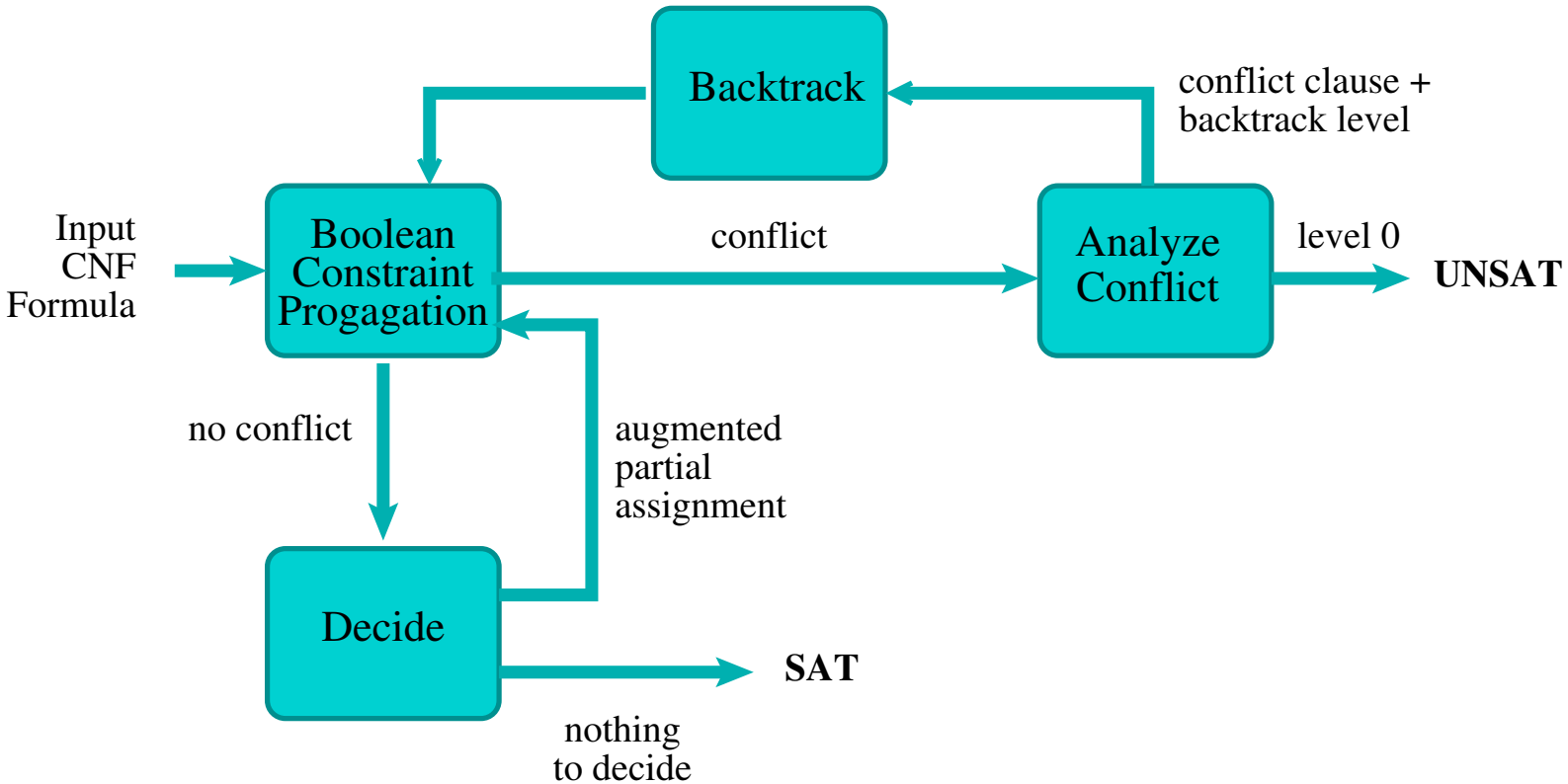
- Given F in Conjunctive Normal Form (CNF).

Example CNF formula

$$(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_1)$$

- Build up partial assignment ρ:
  - if a clause is conflicting, add new clause capturing reason for conflict, and backtrack to ρ' (sub-assignment of ρ)
  - if no confict, extend ρ to larger ρ' by deciding on a new variable to set.
- If Conflict at 0-level, return UNSAT
- If no further variables to set, return SAT.

DPLL Algorithm (Schematic)



Example F_1

Example F_1

$$
\begin{array}{ll}
c_1 & (\neg x_1 \vee x_2) \wedge \\
c_2 & (\neg x_2 \vee x_1) \wedge \\
c_3 & (\neg x_1) \wedge \\
c_4 & (x_2).
\end{array}
$$

Example F_2

Example F_2

$$\begin{aligned} c_1 \quad & (\neg x_1 \vee x_2) \wedge \\ c_2 \quad & (\neg x_2 \vee \neg x_1) \wedge \\ c_3 \quad & (\neg x_3 \vee x_1) \wedge \\ c_4 \quad & (x_1 \vee x_3). \end{aligned}$$

Example F_3

Example F_3

$$\begin{aligned} c_1 \quad & (\neg x_1 \vee x_2) \wedge \\ c_2 \quad & (\neg x_2 \vee x_1). \end{aligned}$$

Example F_4

Example F_4

$$
\begin{aligned}
c_1 \quad & (\neg x_1 \vee x_2) \wedge \\
c_2 \quad & (\neg x_1 \vee x_3 \vee x_2) \wedge \\
c_3 \quad & (\neg x_2 \vee x_4) \wedge \\
c_4 \quad & (\neg x_3 \vee \neg x_4) \wedge \\
c_5 \quad & (x_2 \vee x_3) \wedge \\
c_6 \quad & (x_2 \vee \neg x_3).
\end{aligned}
$$

Exercise

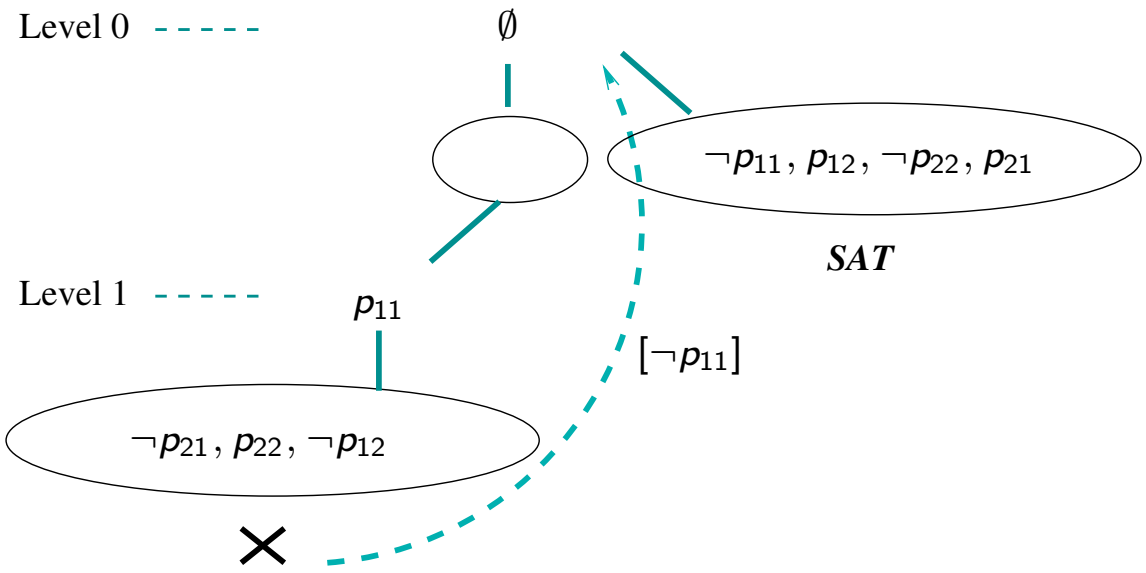
∼Pigeon-Hole(2,2)

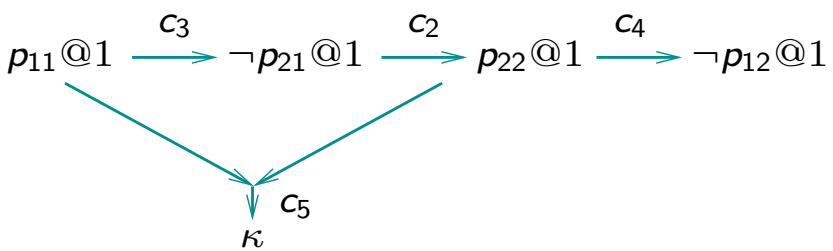
$$
\begin{array}{ll}
c_1 & (p_{11} \vee p_{12}) \wedge \\
c_2 & (p_{21} \vee p_{22}) \wedge \\
c_3 & (\neg p_{11} \vee \neg p_{21}) \wedge \\
c_4 & (\neg p_{12} \vee \neg p_{22}) \wedge \\
c_5 & (\neg p_{11} \vee \neg p_{22})
\end{array}
$$

Solution

∼Pigeon-Hole(2,2)

c_1 $(p_{11} \vee p_{12}) \wedge$
c_2 $(p_{21} \vee p_{22}) \wedge$
c_3 $(\neg p_{11} \vee \neg p_{21}) \wedge$
c_4 $(\neg p_{12} \vee \neg p_{22}) \wedge$
c_5 $(\neg p_{11} \vee \neg p_{22})$





Analyze Conflict

- Pick conflict clause
- Fix backtrack level

Algorithm 2.2.2: ANALYZE-CONFLICT

Input:
Output: Backtracking decision level + a new conflict clause

1. **if** *current-decision-level* $= 0$ **then return** -1;
2. $cl :=$ *current-conflicting-clause*;
3. **while** ($\neg$STOP-CRITERION-MET(cl)) **do**
4. $\quad lit :=$ LAST-ASSIGNED-LITERAL(cl);
5. $\quad var :=$ VARIABLE-OF-LITERAL(lit);
6. $\quad ante :=$ ANTECEDENT(lit);
7. $\quad cl :=$ RESOLVE(cl, $ante$, var);
8. add-clause-to-database(cl);
9. **return** clause-asserting-level(cl); ▷ 2nd highest decision level in cl

Resolution

Given clauses of the form

$$c_1 = (a_1 \vee \cdots a_m \vee a) \text{ and } c_2 = (b_1 \vee \cdots b_n \vee \neg a)$$

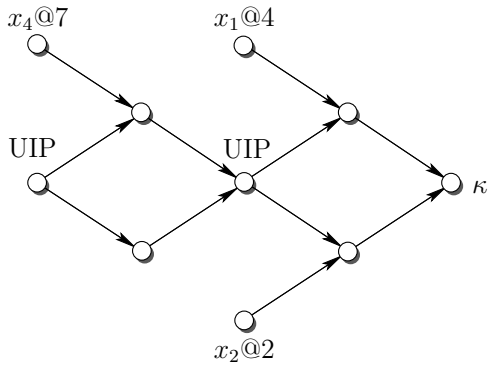
The clause

$$c = (a_1 \vee \cdots a_m \vee b_1 \vee \cdots b_n)$$

(called the resolvant of c_1 and c_2 wrt a), is a logical consequence of $c_1 \wedge c_2$.

Finding a conflict clause to add

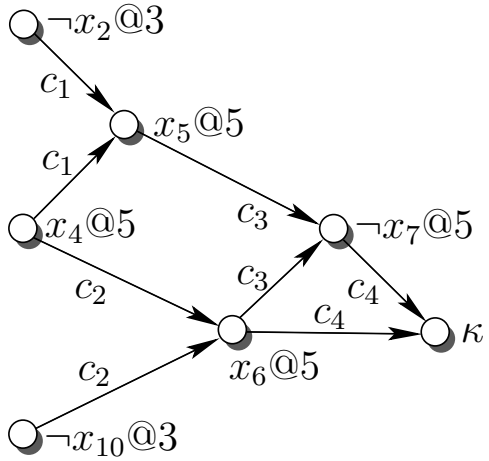
- Unique Implication Point (UIP)



- Stopping criteria: clause should contain the negation of the first UIP as the only literal at current decision level.

Example

$$
\begin{aligned}
c_1 &= (\neg x_4 \vee x_2 \vee x_5) \\
c_2 &= (\neg x_4 \vee x_{10} \vee x_6) \\
c_3 &= (\neg x_5 \vee \neg x_6 \vee \neg x_7) \\
c_4 &= (\neg x_6 \vee x_7) \\
&\vdots
\end{aligned}
$$



[From Decision Procedures by Kroening and Strichman]

Decide Hueristics

Jeroslow-Wang: Pick literal that occurs frequently in small length clauses.

Correctness

- If the algo outputs SAT
- If the algo outputs UNSAT
- Algo always terminates

Some observations

Proposition (1)

Let F be a given formula. Let ρ be a full assignment which does not satisfy F (that is ρ falsifies a clause of F). Then the BCP graph for this level will contain a conflict node.

Proposition (2)

Let F be a given formula. Let ρ be the partial assignment at a decision level d, and let l be an assignment implied by BCP. Then all extensions of ρ that satisfy F must also satisfy l.

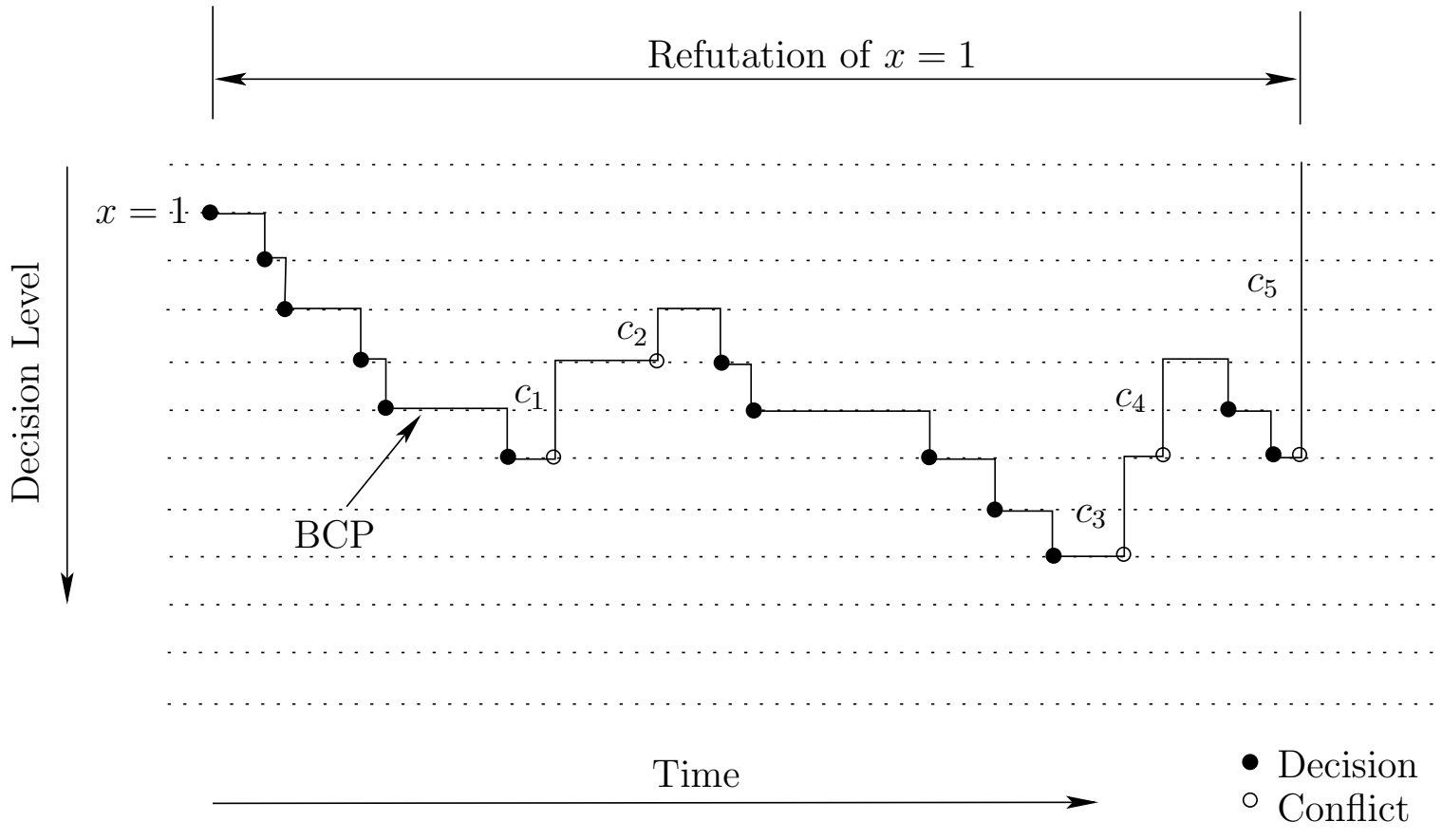
Proposition (3)

If the solver adds a conflict clause c to formula F, then F and $F \wedge c$ have exactly the same set of satisfying assignments.

Correctness

- If the algo outputs SAT
  - Let ρ be the current assignment. Then all variables have been assigned in ρ. Further there are no conflicts detected in BCP at this level.
  - By Prop. 1 ρ must satisfy all clauses in F' and hence F too.
- If the algo outputs UNSAT
  - Then BCP has detected a conflict (say clause c) at level 0. If there was a satisfying assignment ρ for F, it must agree with the assignments made by BCP (Prop 2).
  - But then a clause c is conflicting under ρ.

Vizualizing the progress of the solver



[Figure from Decision Procedures, Kroening and Strichman, p38]

Termination

- Assuming that ANALYZE-CONFLICT always produces an asserting clause (which forces an implied assignment immediately).
- Algo never re-enters a decision level d with the same partial assignment.
- Suppose the algo went on forever. Then at least one decision level must be entered infinitely many times; and since there are only finitely many partial assignments, it must also enter with the same partial assignment infinitely many times. This contradicts previous observation.