Decision Procedures

Azadeh Farzan

CS410 - Fall 2020

Decision Procedures

- A decision procedure is an algorithm that given a decision problem, terminates with a correct yes/no answer.
- Play an important part in automated verification, theorem proving, compiler optimization, synthesis, and many areas of AI.
- The problems are inherently difficult.
- Procedures need to be a very efficient.

Quick Example

Are these two code fragments equivalent?

```
if(!a && !b) h();
else
    if(!a) g();
    else f();
```

```
if(a) f();
else
    if(b) g();
    else h();
```

It seems to be a YES/NO type of problem. Can we state this generally as a decision problem?

$$(\texttt{if}\ x\ \texttt{then}\ y\ \texttt{else}\ z) \quad \equiv \quad (x \wedge y) \vee (\neg x \wedge z)$$

$$\begin{aligned} & (\neg a \wedge \neg b) \wedge h \ \vee \ \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \ \vee \ a \wedge f) \\ \iff \quad & \boxed{a \wedge f} \vee \ \neg a \wedge \boxed{(b \wedge g \ \vee \ \neg b \wedge h)}\ . \end{aligned}$$

Reminders

A propositional formula is satisfiable if there is an assignment (to propositions) that makes the formal to evaluate to true.

$$A \wedge B$$

A formula is valid if all assignments make it to evaluate to true.

$$A \Rightarrow A$$

Theories

- A theory is (informally):
  - A finite or infinite set of formulas, which are characterized by common grammatical rules, functions and predicates, and a domain of values.

Theory name	Example formula
Propositional logic	$x_1 \wedge (x_2 \vee \neg x_3)$
Equality	$y_1 = y_2 \wedge \neg(y_1 = y_3) \implies \neg(y_1 = y_3)$
Linear arithmetic	$(2z_1 + 3z_2 \leq 5) \vee (z_2 + 5z_2 - 10z_3 \geq 6)$
Bit vectors	$((a >> b)\ \&\ c) < c$
Arrays	$(i = j \wedge a[j] = 1) \implies a[i] = 1$
Pointer logic	$p = q \wedge *p = 5 \implies *q = 5$
Combined theories	$(i \leq j \wedge a[j] = 1) \implies a[i] < 2$

More Definitions

A procedure for a decision problem is sound if when it answers "YES" to the validity/satisfiability question, the formula is valid/satisfiable.

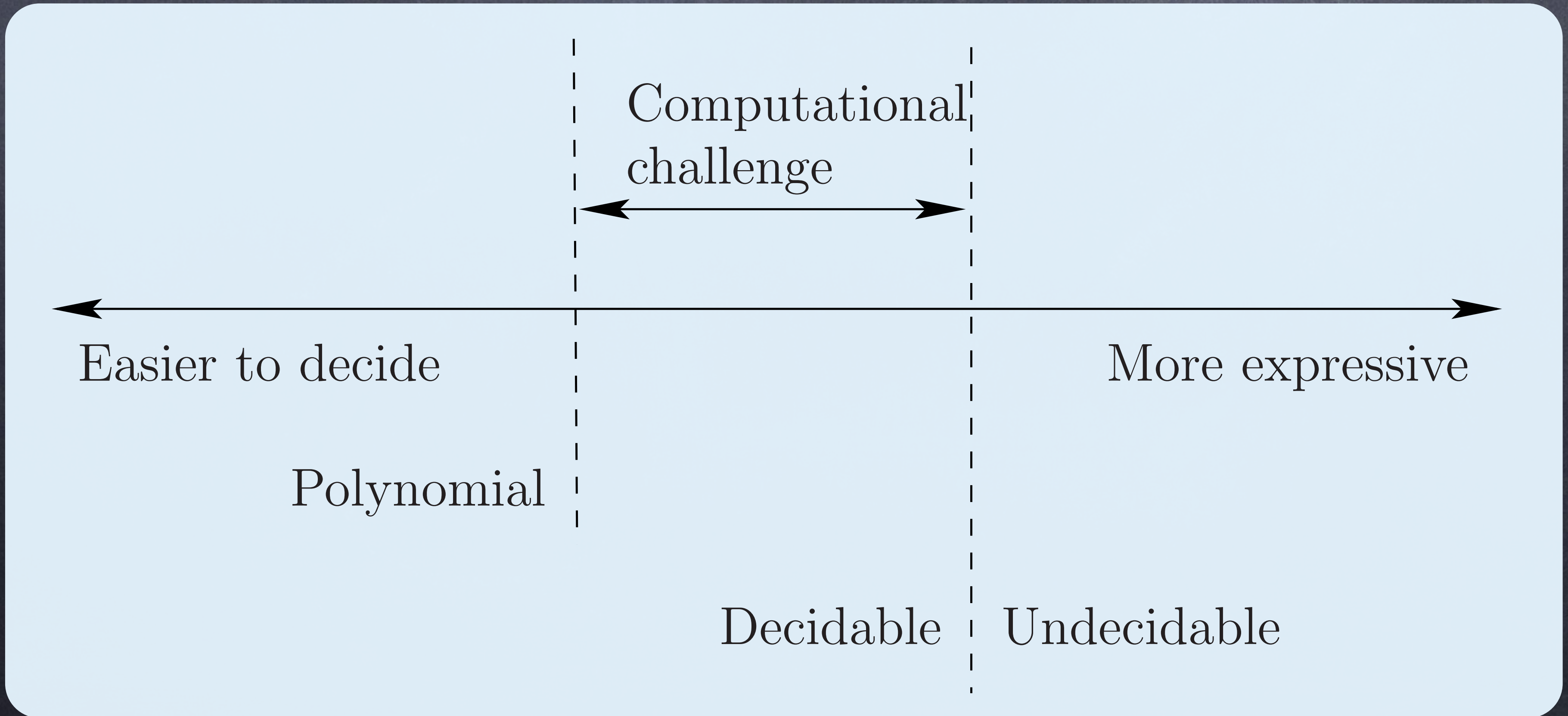
A procedure for a decision problem is complete if it always terminates, and when formula is valid, it it answers "YES" to the validity question.

A procedure is called a decision procedure for T if it is sound and complete with respect to every formula in T.

A theory is decidable if and only if there is a decision procedure for it.

Theories and Algorithms

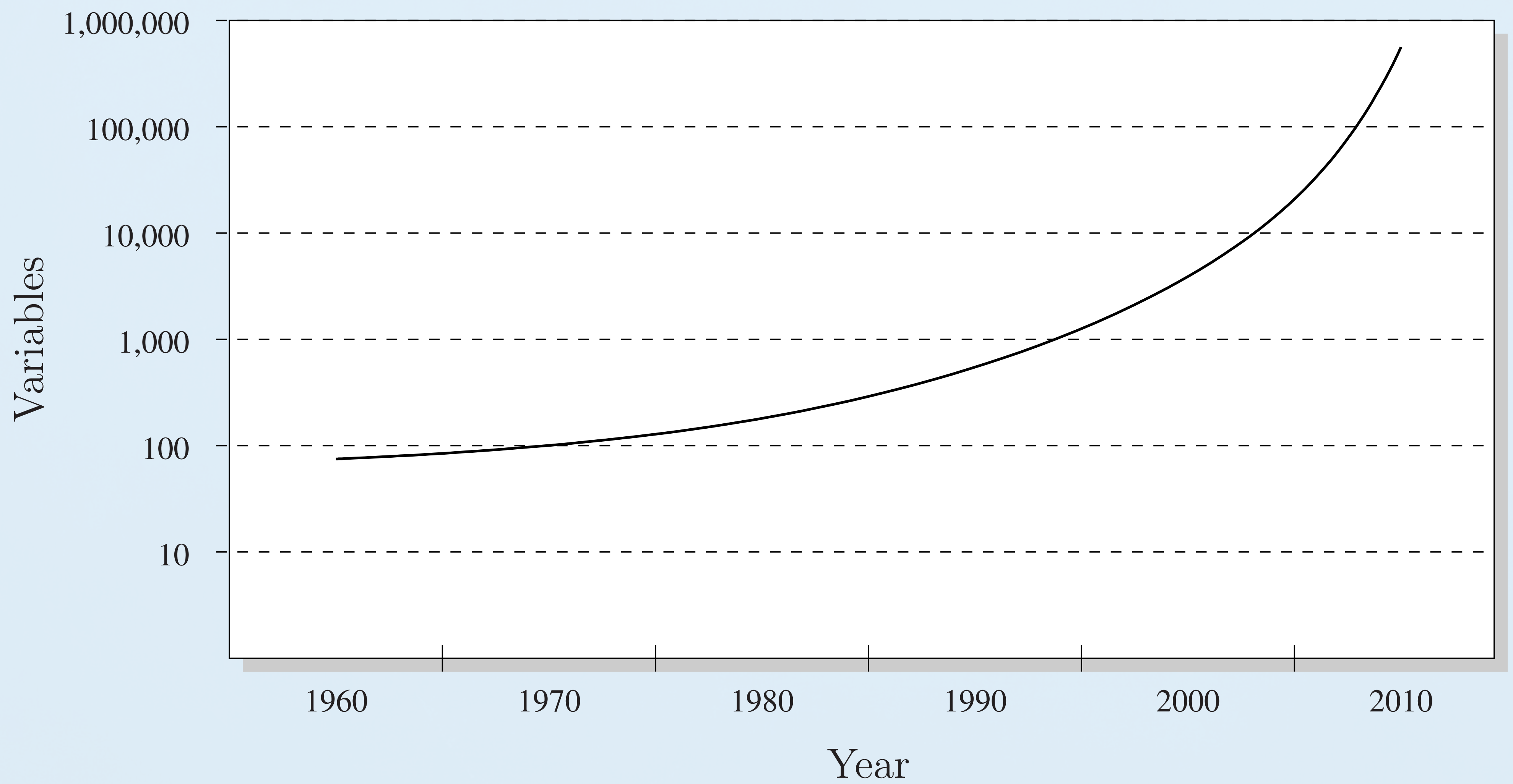
- A theory is interesting if:
  - It is expressive enough to model a real decision problem.
  - It is either decidable or semi-decidable, and more efficiently solvable than more expressive theories.



Propositional Logic

SAT Solvers

- Given a propositional formula F, a SAT Solver decides if F is satisfiable.



Propose a trivial SAT solving Algorithm.

The satisfiability problem for propositional logic is NP-Complete.

SAT Solving Techniques

- DPLL-based
  - Davis-Putnam-Loveland-Logemann
  - Traversing and backtracking on a binary tree, in which internal nodes represent partial assignments, and leaves present full assignments.
- Stochastic Search
  - The solver guesses a full assignment, and if it doesn't work, it starts flipping values based on some greedy heuristic.

Let's see a reasonable SAT-solving algorithm …

CNF

A formula in *Conjunctive Normal Form* (CNF) is a conjunction of clauses

$$C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

each clause C is a disjunction of literals

$$C = L_1 \vee \ldots \vee L_m$$

and each literal is either a plain variable x or a negated variable $\overline{x}$.

Example $(a \vee b \vee c) \wedge (\overline{a} \vee \overline{b}) \wedge (\overline{a} \vee \overline{c})$

For every Propositional formula there exists an equisatisfiable propositional formula in CNF which is at most polynomially larger.

States of a Clause

- A clause is satisfied if one or more of its literals are satisfied.
- A clause is conflicting if all of its literals are assigned, but not satisfied.
- A clause is unit if it is not satisfied and all but one of its literals are assigned.
- A clause is unresolved if it is none of the above.

$$\{x_1 \mapsto 1,\ x_2 \mapsto 0,\ x_4 \mapsto 1\}$$

$$(x_1 \vee x_3 \vee \neg x_4)$$
$$(\neg x_1 \vee x_2)$$
$$(\neg x_1 \vee \neg x_4 \vee x_3)$$
$$(\neg x_1 \vee x_3 \vee x_5)$$

Simple Algorithm Overview

- Partial Evaluations:
  - We start with the empty evaluation (no variable has been assigned).
  - We step by step extend it to all variables.
- Davis-Putnam-Logemann-Loveland basic structure:

Start with empty

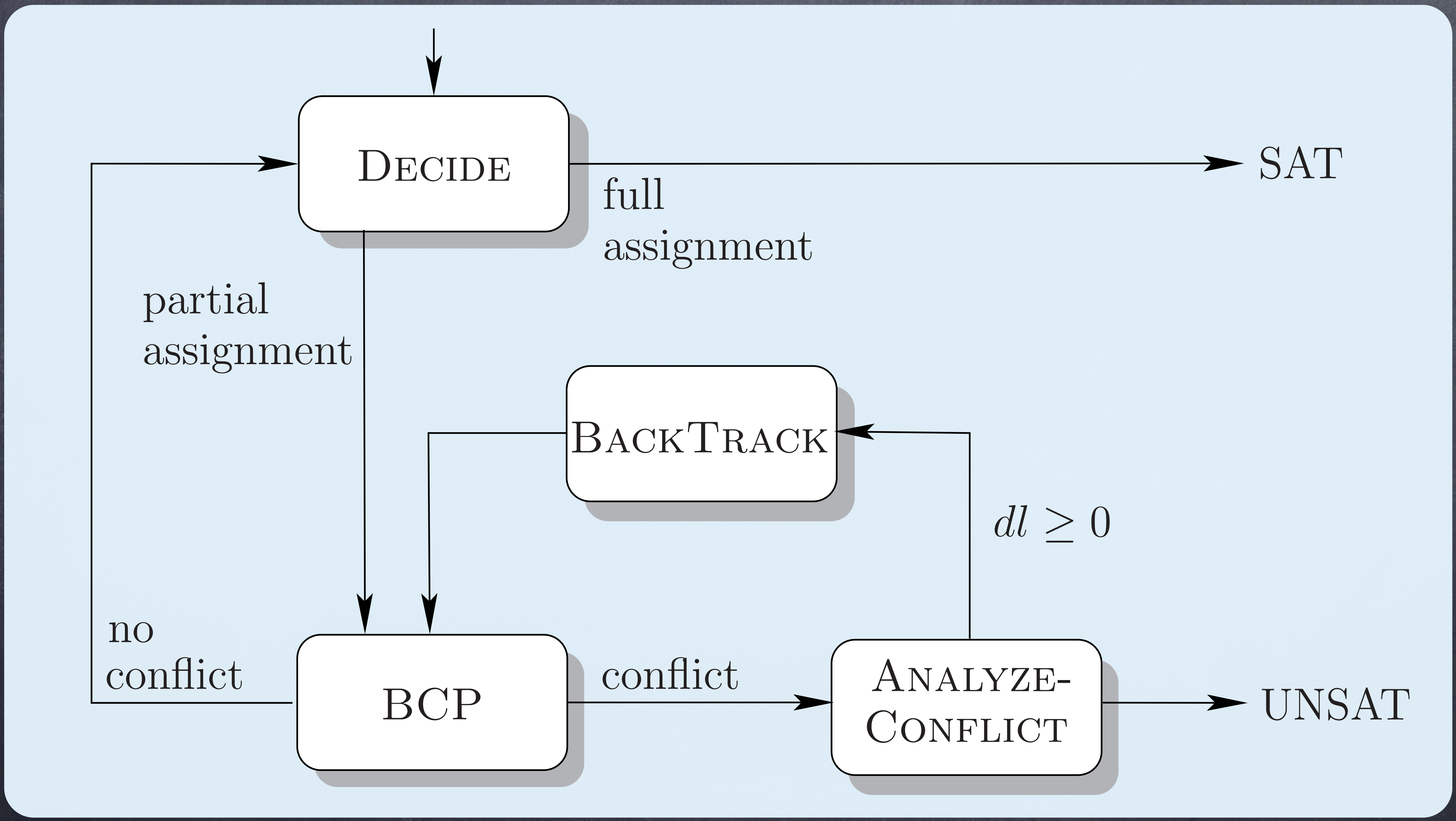
```
boolean DPLL(clause set N, partial valuation 𝒜) {
    if (all clauses in N are true under 𝒜) return true;
    elsif (some clause in N is false under 𝒜) return false;
    elsif (N contains unit clause P) return DPLL(N, 𝒜 ∪ {P ↦ 1});
    elsif (N contains unit clause ¬P) return DPLL(N, 𝒜 ∪ {P ↦ 0});
    else {
        let P be some undefined variable in N;
        if (DPLL(N, 𝒜 ∪ {P ↦ 0})) return true;
        else return DPLL(N, 𝒜 ∪ {P ↦ 1});
    }
}
```

Not chosen randomly

Iterative Algorithm with smart backtracking

The better algorithm takes shortcuts …

More Sophisticated Algorithm

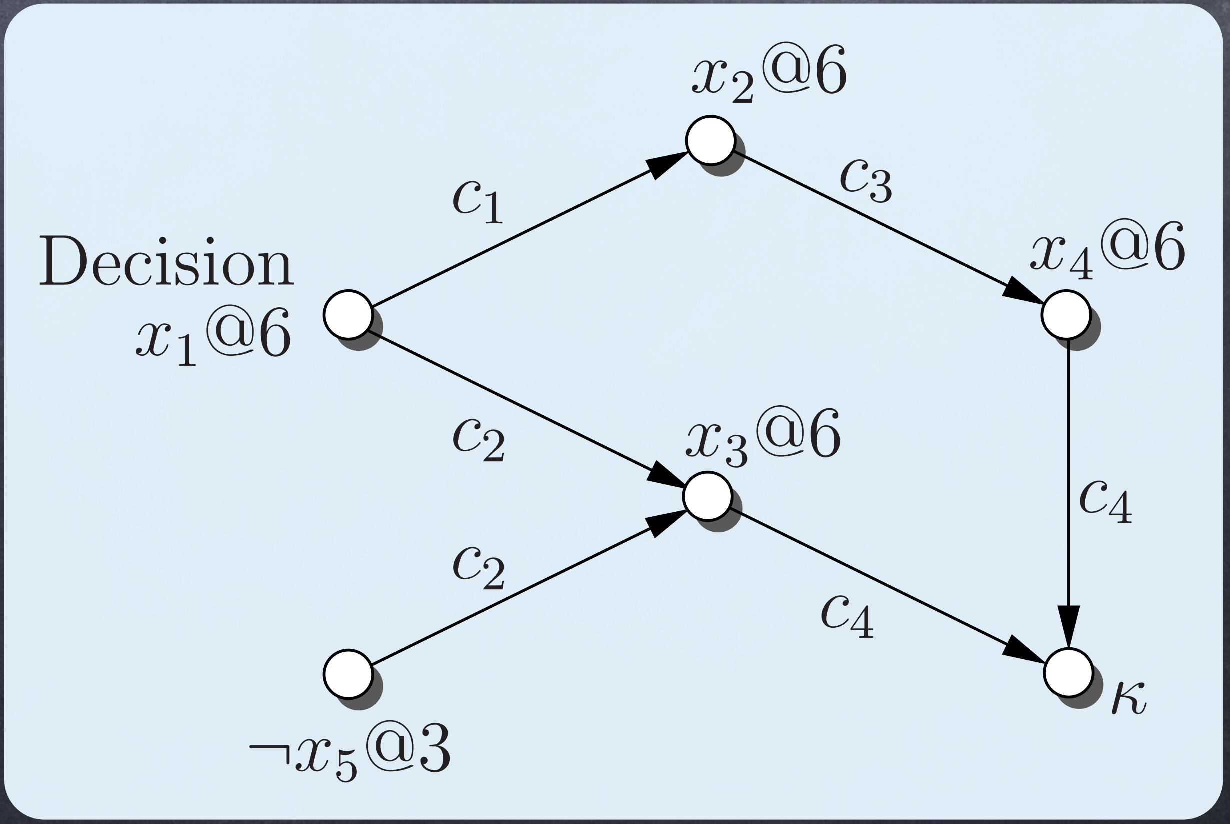


BCP-Conflict Analysis-Backtracking

$x_i@dl$

- Each assignment is associated with a decision level.
  - the level at which a decision or implication was made.
- BCP is performed over an implication graph.
  - the graph represents a current partial assignment and the reason for each implication.

$$
\begin{aligned}
c_1 &= (\neg x_1 \lor x_2)\ , \\
c_2 &= (\neg x_1 \lor x_3 \lor x_5)\ , \\
c_3 &= (\neg x_2 \lor x_4)\ , \\
c_4 &= (\neg x_3 \lor \neg x_4)\ , \\
c_5 &= (x_1 \lor x_5 \lor \neg x_2)\ , \\
c_6 &= (x_2 \lor x_3)\ , \\
c_7 &= (x_2 \lor \neg x_3)\ , \\
c_8 &= (x_6 \lor \neg x_5)\ .
\end{aligned}
$$



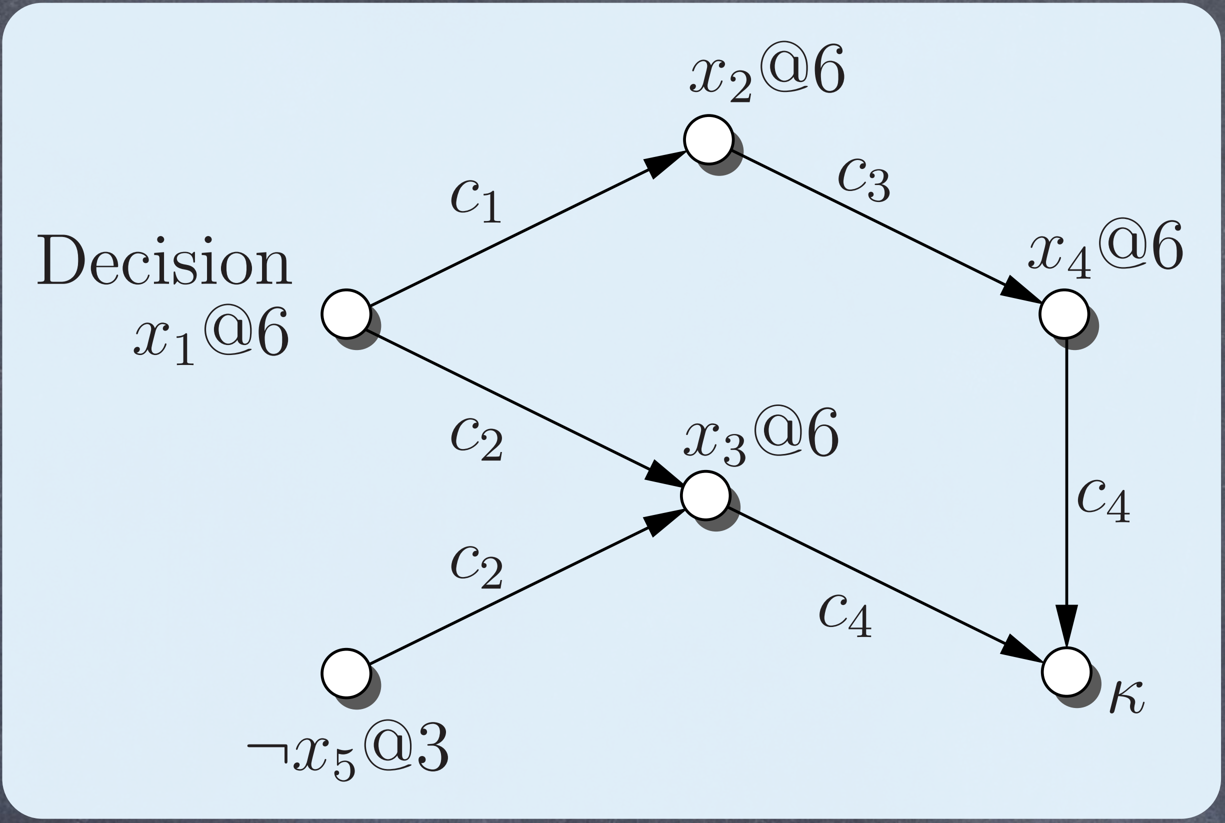
Implication Graph

- A labeled directed acyclic graph G(V,E) where:
  - V represents the literals of the current assignment. Each node is labeled with a literal and its decision level.
  - E represents the connection between literals. (u,v) belongs to the graph if the assignment to u plays a role in what value should be assigned to v (they both belong to a unit clause).
  - G can also contain a single conflict node labeled with k (now the incoming edges are all from the literals that are part of a conflict clause with k).
  - The root nodes correspond to decisions and the internal nodes correspond to implications made by propagation.

The graph may be partial; only referring to a certain decision level.

Analyze Conflict: Example

$$
\begin{aligned}
c_1 &= (\neg x_1 \vee x_2) \;, \\
c_2 &= (\neg x_1 \vee x_3 \vee x_5) \;, \\
c_3 &= (\neg x_2 \vee x_4) \;, \\
c_4 &= (\neg x_3 \vee \neg x_4) \;, \\
c_5 &= (x_1 \vee x_5 \vee \neg x_2) \;, \\
c_6 &= (x_2 \vee x_3) \;, \\
c_7 &= (x_2 \vee \neg x_3) \;, \\
c_8 &= (x_6 \vee \neg x_5) \;.
\end{aligned}
$$



$\neg x_5@3$

$x_1@6$

Therefore, we can safely add a conflict clause:

$$c_9 = (x_5 \vee \neg x_1)$$

to our formula.

generally called learning.

backtrack to the highest decision level before the current one.

We backtrack to level 3 after learning the conflict clause.

We erase all the decisions and implications made after that level, including assignments to x_1, x_2, x_3, x_4.

$$c_9 = (x_5 \vee \neg x_1)$$

$$\begin{aligned}
c_1 &= (\neg x_1 \vee x_2) \ , \\
c_2 &= (\neg x_1 \vee x_3 \vee x_5) \ , \\
c_3 &= (\neg x_2 \vee x_4) \ , \\
c_4 &= (\neg x_3 \vee \neg x_4) \ , \\
c_5 &= (x_1 \vee x_5 \vee \neg x_2) \ , \\
c_6 &= (x_2 \vee x_3) \ , \\
c_7 &= (x_2 \vee \neg x_3) \ , \\
c_8 &= (x_6 \vee \neg x_5) \ .
\end{aligned}$$

$$c_9 = (x_5 \vee \neg x_1)$$

Clause c_9 was a special kind of conflict clause, called asserting clause which forced an immediate implication after backtracking.

Conflict Resolution

How are conflict clauses generated?
(specifically, asserting clauses)

Conflict Resolution

Binary resolution inference rule:

$$\frac{(a_1 \vee \ldots \vee a_n \vee \beta) \qquad (b_1 \vee \ldots \vee b_m \vee \neg\beta)}{(a_1 \vee \ldots \vee a_n \vee b_1 \vee \ldots \vee b_m)}$$

An inference system based on the above rule for propositional logic is sound and complete.

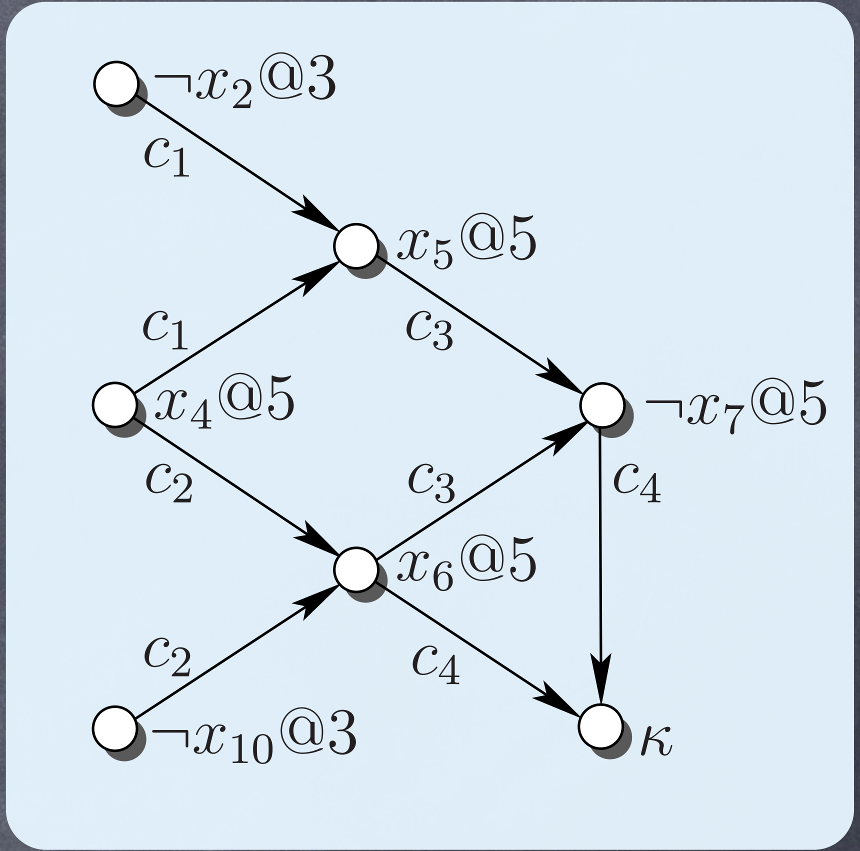
Unsatisfiability of a CNF formula is decided through finitely many applications of the resolution rule.

Example

$$c_1 = (\neg x_4 \lor x_2 \lor x_5)$$
$$c_2 = (\neg x_4 \lor x_{10} \lor x_6)$$
$$c_3 = (\neg x_5 \lor \neg x_6 \lor \neg x_7)$$
$$c_4 = (\neg x_6 \lor x_7)$$
$$\vdots$$

Implication Order in BCP:

$$x_4, x_5, x_6, x_7$$

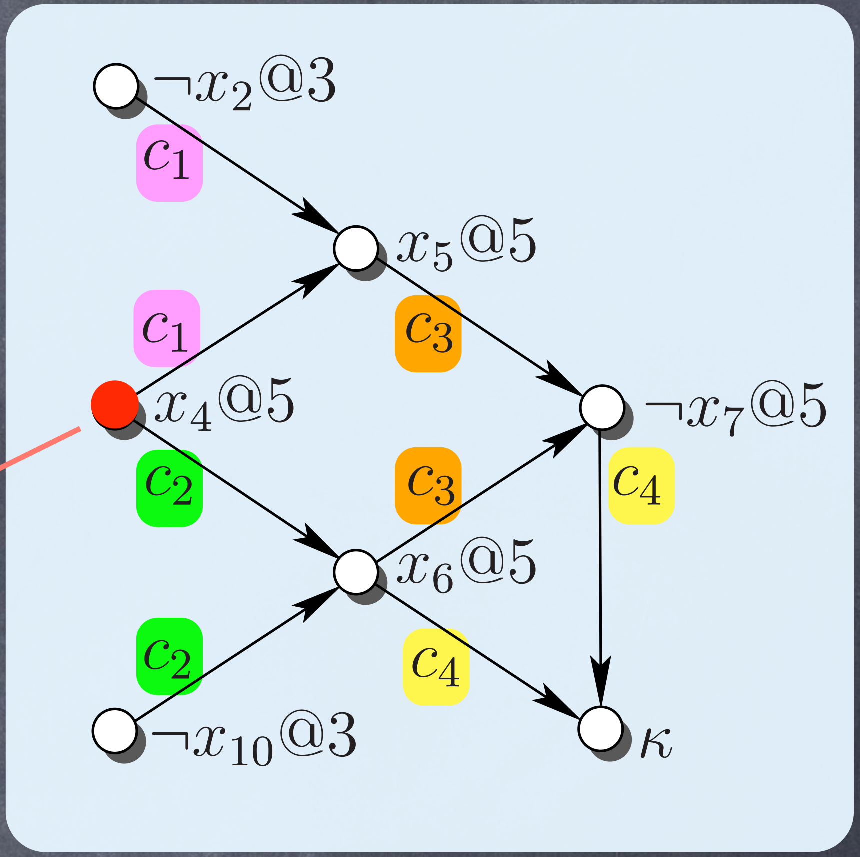


Example

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

Implication Order in BCP:

$$x_4, x_5, x_6, x_7$$



The first UIP

Stopping criteria: negation of first UIP is the only literal from the current decision level

$$(\neg x_6 \lor x_7)$$

$$(\neg x_5 \lor \neg x_6)$$

$$(\neg x_4 \lor x_{10} \lor \neg x_5)$$

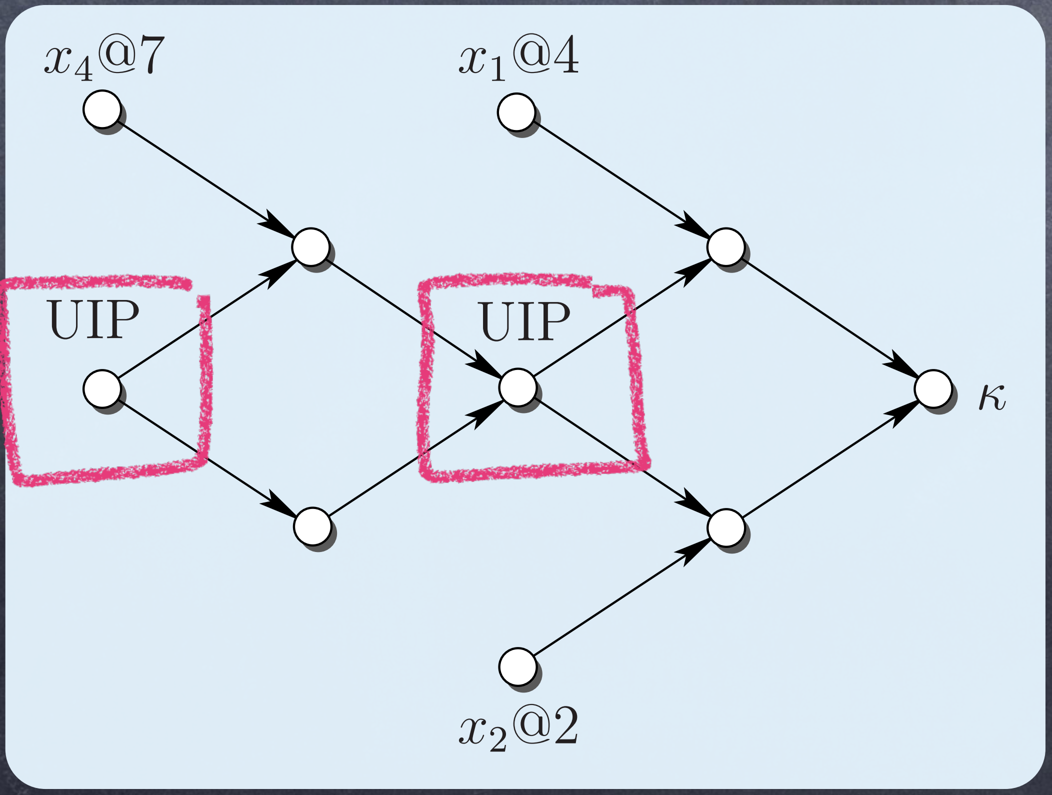
$$(\neg x_4 \lor x_2 \lor x_{10})$$

An asserting clause which contains the negation of x_4.

Conflict Resolution

What is the stopping condition for binary resolution steps?

- Given a partial graph for a decision level, a unique implication point (UIP) is a node (other than the conflict node) that is on all paths from the decision node to the conflict node.
- First UIP is a UIP that is closest to the conflict node.

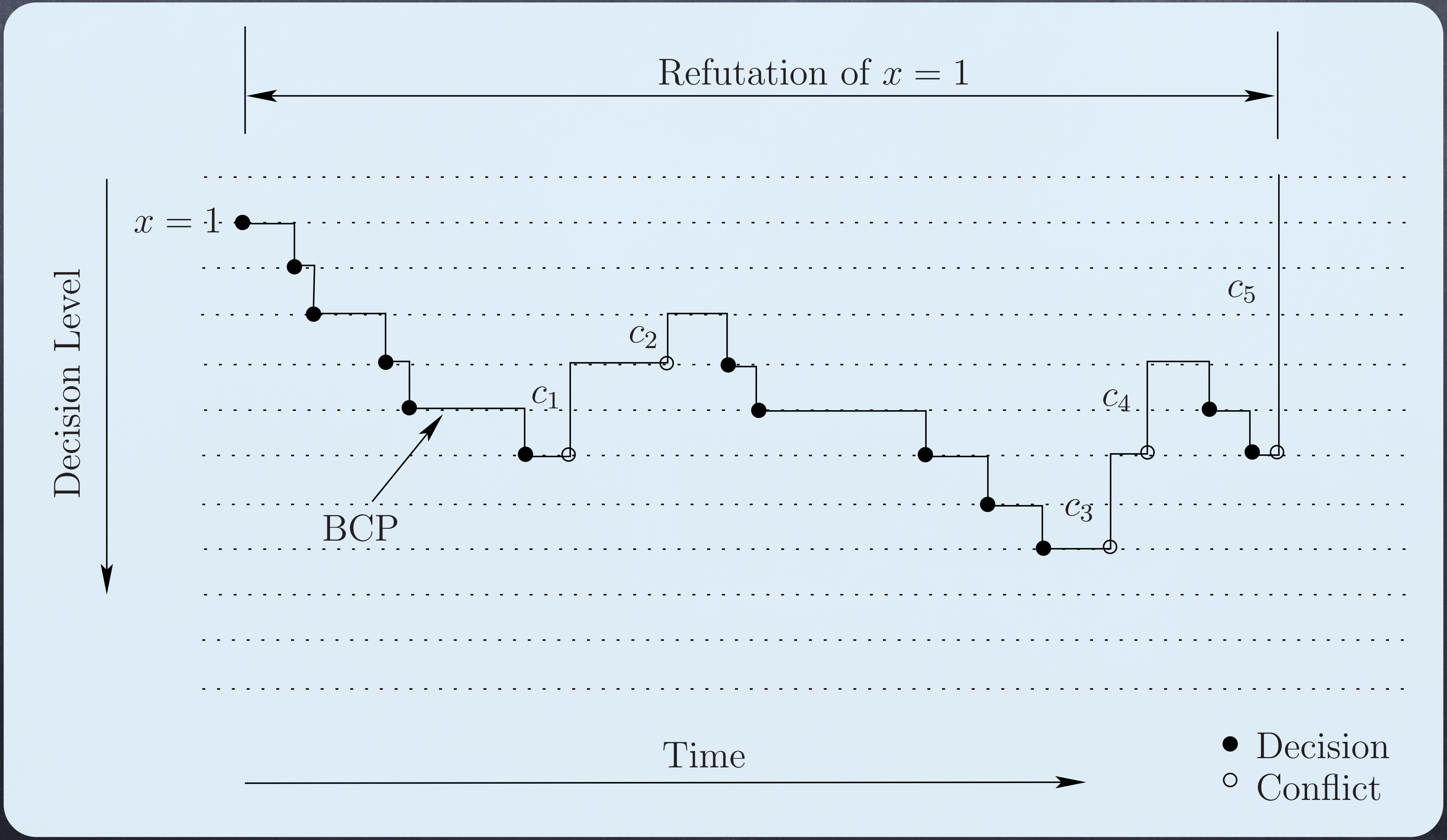


Why first? It is faster and one backtracks to the lowest level.

Termination

Why does this terminate? How do we know that a partial assignment cannot be repeated forever?

Theorem: It is never the case that the solver enters decision level dl again with the same partial assignment.



Decision Procedures for Other Theories

Equality Logic and Uninterpreted Functions

Equality Logic

An equality logic formula is defined by the following grammar:

$$\begin{aligned} formula &: formula \wedge formula \mid \neg formula \mid (formula) \mid atom \\ atom &: term \;=\; term \\ term &: \underline{identifier \mid constant} \end{aligned}$$

infinite domain: integers, reals

The satisfiability problem for equality logic is NP-Complete.

It is more natural and convenient to use equality logic for modeling some problems compared to propositional logic.

Uninterpreted Functions

An equality logic formula with uninterpreted functions is defined by the following grammar:

$$
\begin{aligned}
\mathit{formula} &: \mathit{formula} \wedge \mathit{formula} \mid \neg \mathit{formula} \mid (\mathit{formula}) \mid \mathit{atom} \\
\mathit{atom} &: \mathit{term} \ = \ \mathit{term} \ \mid \mathit{predicate\text{-}symbol}\ (\mathit{list\ of\ terms}) \\
\mathit{term} &: \mathit{identifier} \mid \mathit{function\text{-}symbol}\ (\mathit{list\ of\ terms})
\end{aligned}
$$

Functional Consistency: same function same output on same input.

$$F(x) = F(G(y)) \vee x + 1 = y$$

The satisfiability problem is reduced to that of equality logic.

Example: Translation Validation

Consider the statement:

$$z = (x_1 + y_1) * (x_2 + y_2)$$

A compiler translates this to:

$$u_1 = x_1 + y_1;\; u_2 = x_2 + y_2;\; z = u_1 * u_2$$

Correctness of this translation ties to the validity of the verification condition:

$$u_1 = x_1 + y_1 \wedge u_2 = x_2 + y_2 \wedge z = u_1 * u_2 \implies z = (x_1 + y_1) * (x_2 + y_2)$$

which can be turned into an EUF formula.

Example: Translation Validation

Starting with:

$$u_1 = x_1 + y_1 \wedge u_2 = x_2 + y_2 \wedge z = u_1 * u_2 \implies z = (x_1 + y_1) * (x_2 + y_2)$$

We make addition and multiplication uninterpreted to get:

$$(u_1 = F(x_1, y_1) \wedge u_2 = F(x_2, y_2) \wedge z = G(u_1, u_2))$$
$$\implies z = G(F(x_1, y_1), F(x_2, y_2)) \; .$$

If this formula is satisfiable, then the translation is valid.

How is this solved?

Example: Translation Validation

Starting with:

$$
\left(u_1 = \overset{f_1}{\boxed{F(x_1, y_1)}} \wedge u_2 = \overset{f_2}{\boxed{F(x_2, y_2)}} \wedge z = \overset{g_1}{\boxed{G(u_1, u_2)}}\right)
\implies z = \underset{g_2}{\boxed{G(F(x_1, y_1), F(x_2, y_2))}}.
$$

Then we apply Ackerman's reduction:

Functional Consistency

$$
\left(\begin{array}{l}
\boxed{(x_1 = x_2 \wedge y_1 = y_2 \implies f_1 = f_2)} \wedge \\
(u_1 = f_1 \wedge u_2 = f_2 \implies g_1 = g_2)
\end{array}\right) \implies
$$
$$
\boxed{(u_1 = f_1 \;\wedge\; u_2 = f_2 \;\wedge\; z = g_1) \implies z = g_2)}
$$

to reduce to this to standard propositional validity.

original formula

Linear Arithmetic

Linear Arithmetic

A linear arithmetic formula is defined by the following rules:

$$
\begin{aligned}
\mathit{formula} &: \mathit{formula} \wedge \mathit{formula} \mid (\mathit{formula}) \mid \mathit{atom} \\
\mathit{atom} &: \mathit{sum}\ \mathit{op}\ \mathit{sum} \\
\mathit{op} &: = \ \mid\ \leq\ \mid\ < \\
\mathit{sum} &: \mathit{term} \mid \mathit{sum} + \mathit{term} \\
\mathit{term} &: \underline{\mathit{identifier} \mid \mathit{constant} \mid}\ \mathit{constant}\ \ \mathit{identifier}
\end{aligned}
$$

rational numbers and integers

The satisfiability problem for linear arithmetic theory is polynomial for rational numbers and NP-Complete for integers.

A variant of the simplex method works as a decision procedure.

Difference Logic

A difference logic formula is defined by the following rules:

$$\begin{aligned} formula &: formula \wedge formula \mid atom \\ atom &: identifier - identifier \ \ op \ \ constant \\ op &: \ \leq \ \mid \ < \end{aligned}$$

rational numbers or integers

The satisfiability problem for difference logic is polynomial time solvable in both cases.

Further info

DPLL-SAT Algorithm

Input: A propositional CNF formula $\mathcal{B}$
Output: "Satisfiable" if the formula is satisfiable and "Unsatisfiable" otherwise

1. **function** DPLL
2. **if** BCP() = "conflict" **then return** "Unsatisfiable";
3. **while** (TRUE) **do**
4. **if** $\neg$DECIDE() **then return** "Satisfiable";
5. **else**
6. **while** (BCP() = "conflict") **do**
7. *backtrack-level* := ANALYZE-CONFLICT();
8. **if** *backtrack-level* < 0 **then return** "Unsatisfiable";
9. **else** BackTrack(*backtrack-level*);

DPLL-SAT Components

Name	DECIDE()
Output	FALSE if and only if there are no more variables to assign.
Description	Chooses an unassigned variable and a truth value for it.

Name	BCP()
Output	"conflict" if and only if a conflict is encountered.
Description	Repeated application of the unit clause rule until either a conflict is encountered or there are no more implications.

Name	ANALYZE-CONFLICT()
Output	Minus 1 if a conflict at decision level 0 is detected (which implies that the formula is unsatisfiable). Otherwise, a decision level which the solver should backtrack to.

Name	BACKTRACK(dl)
Description	Sets the current decision level to dl and erases assignments at decision levels larger than dl.

Conflict Resolution Algorithm

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** (¬Stop-criterion-met(*cl*)) **do**
4. *lit* := Last-assigned-literal(*cl*);
5. *var* := Variable-of-literal(*lit*);
6. *ante* := Antecedent(*lit*);
7. *cl* := Resolve(*cl*, *ante*, *var*);
8. add-clause-to-database(*cl*);
9. **return** clause-asserting-level(*cl*); ▷ 2nd highest decision level in *cl*