What are SAT Solvers and Why Should We Care?

Mikoláš Janota

Lero
University College Dublin
Ireland

Contents

What is Satisfiability?

Find out whether a formula has a satisfying assignment

- $x \vee y$ — **SAT**, e.g. $x = T$, $y = F$

What is Satisfiability?

Find out whether a formula has a satisfying assignment

- $x \vee y$ — **SAT**, e.g. $x = T$, $y = F$
- $x \wedge \neg x$ — **UNSAT**, evaluates to F for both $x = T$, $x = F$

What is Satisfiability?

Find out whether a formula has a satisfying assignment

- $x \vee y$ — **SAT**, e.g. $x = T$, $y = F$
- $x \wedge \neg x$ — **UNSAT**, evaluates to F for both $x = T$, $x = F$
- $x \Rightarrow y \wedge y \Rightarrow z \wedge \neg(z \wedge y)$ — **SAT**, e.g. all F

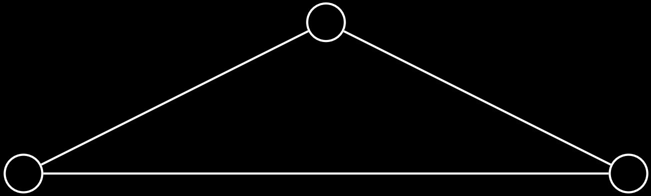
What is Satisfiability?

Find out whether a formula has a satisfying assignment

- $x \vee y$ — **SAT**, e.g. $x = T$, $y = F$
- $x \wedge \neg x$ — **UNSAT**, evaluates to F for both $x = T$, $x = F$
- $x \Rightarrow y \wedge y \Rightarrow z \wedge \neg(z \wedge y)$ — **SAT**, e.g. all F
- $x \Rightarrow y \wedge y \Rightarrow z \wedge \neg(z \wedge y) \wedge x$ — **UNSAT**

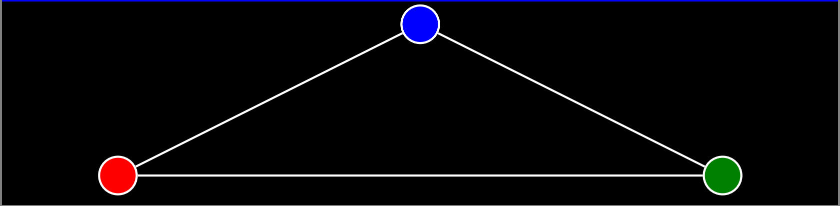
Many Problems Can Be Expressed as Satisfiability

Can this graph be colored with 3 colors?



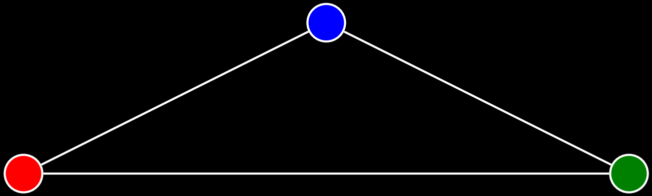
Many Problems Can Be Expressed as Satisfiability

Can this graph be colored with 3 colors?

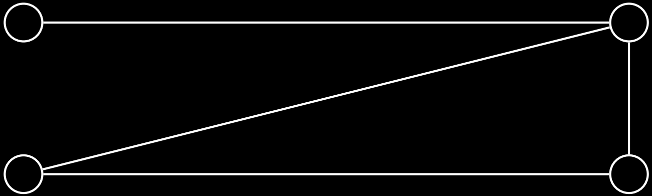


Many Problems Can Be Expressed as Satisfiability

Can this graph be colored with 3 colors?

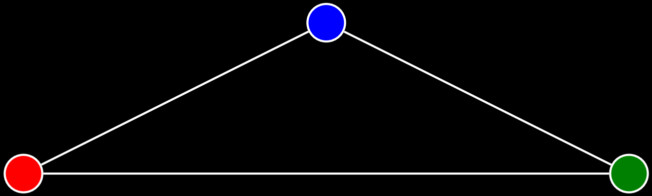


And this one?

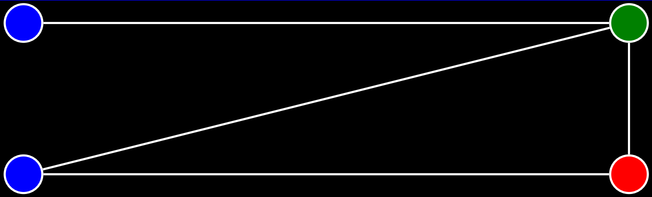


Many Problems Can Be Expressed as Satisfiability

Can this graph be colored with 3 colors?

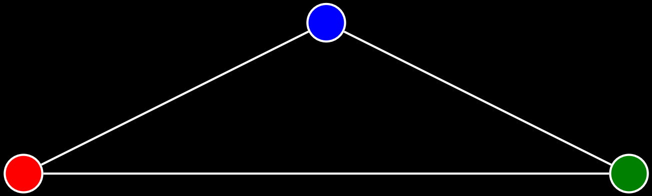


And this one? (Claudia's Coloring)

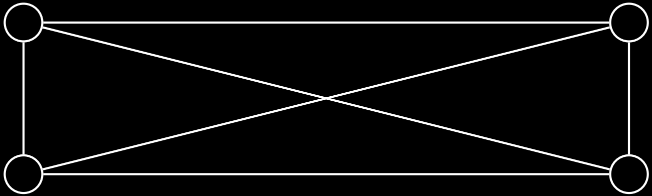


Many Problems Can Be Expressed as Satisfiability

Can this graph be colored with 3 colors?



And this one?



Graph Coloring Can Be Written as Satisfiability Problem

Solution I

- For each node N introduce the variable r_N, g_N, and b_N.
- Each node must have a color: $r_N \vee g_N \vee b_N$.
- Not more than one color though:
  $\neg(r_N \wedge g_N) \wedge \neg(r_N \wedge b_N) \wedge \neg(g_N \wedge b_N)$
- Any two adjacent nodes N and M must not have the same color: $\neg(r_N \wedge r_M) \wedge \neg(g_N \wedge g_M) \wedge \neg(b_N \wedge b_M)$

Graph Coloring Can Be Written as Satisfiability Problem

Solution II

- Encode colors in binary. For each node N introduce variables b_N^0 and b_N^1.
- Assign a color to each combination.

$\neg b_N^1 \wedge \neg b_N^0$
$\neg b_N^1 \wedge b_N^0$
$b_N^1 \wedge \neg b_N^0$
$b_N^1 \wedge b_N^0$

- Red is $\neg b_N^0$, Blue is $\neg b_N^1 \wedge b_N^0$, and Green is $b_N^1 \wedge b_N^0$
- Adjacent node as in Solution I.
- We have less variables *and* less clauses.

Conjunctive Normal Form (CNF)

- *literal:* variable or a negated variable (e.g. x, $\neg y$)

Conjunctive Normal Form (CNF)

- *literal:* variable or a negated variable (e.g. x, $\neg y$)
- *clause:* disjunction of literals (e.g. $x \vee y$, $\neg x$)

Conjunctive Normal Form (CNF)

- *literal:* variable or a negated variable (e.g. x, $\neg y$)
- *clause:* disjunction of literals (e.g. $x \vee y$, $\neg x$)
- *CNF:* conjunction of clauses

$$
\begin{aligned}
&x \vee \neg y \wedge \\
&y \vee x \wedge \\
&x
\end{aligned}
$$

Conjunctive Normal Form (CNF)

- *literal:* variable or a negated variable (e.g. x, $\neg y$)
- *clause:* disjunction of literals (e.g. $x \vee y$, $\neg x$)
- *CNF:* conjunction of clauses

$$
\begin{aligned}
&x \vee \neg y \wedge \\
&y \vee x \wedge \\
&x
\end{aligned}
$$

CNF Satisfiability

- A CNF formula is typically written as a set of clauses.
- A clause is satisfied *if and only if* at least one literal is T
- It is satisfied *if and only if* all the clauses are satisfied.

Why Do We Like CNF?

- Any formula can be rewritten in the CNF.
- Many useful constructs can be rewritten *easily*:

$$\begin{aligned} x \Rightarrow y &\quad\leadsto\quad \{\neg x \vee y\} \\ \neg(x \wedge y) &\quad\leadsto\quad \{\neg x \vee \neg y\} \\ x \Leftrightarrow y &\quad\leadsto\quad \{\neg x \vee y, \neg y \vee x\} \end{aligned}$$

- Simple to think about and program on (tree vs. list).

Why Do We Like CNF?

- Any formula can be rewritten in the CNF.
- Many useful constructs can be rewritten *easily*:

$$\begin{aligned} x \Rightarrow y &\quad\rightsquigarrow\quad \{\neg x \vee y\} \\ \neg(x \wedge y) &\quad\rightsquigarrow\quad \{\neg x \vee \neg y\} \\ x \Leftrightarrow y &\quad\rightsquigarrow\quad \{\neg x \vee y, \neg y \vee x\} \end{aligned}$$

- Simple to think about and program on (tree vs. list).

WARNING

- The conversion may blow-up in size. Try $(x \wedge y) \vee (w \wedge z)$.
- **BUT** can be converted to CNF in linear size by introducing new variables.

The Bad and The Good News

A *SAT solver* is a program that decides whether a given CNF formula is satisfiable or not.

- ☹ Deciding satisfiability of a CNF formula is NP-complete.
- ☺ SAT solvers in the past decade have become really, really good (see `http://www.satcompetition.org/`).

Basic Idea

Trying All Variable Assignments

- Assign values to variables, until either. . .
  1. If all variables have a value, respond **SAT**.
  2. If all literals in some clause are F, then backtrack and try something else. If there is nothing else to try, respond *UNSAT*.

Basic Idea

Trying All Variable Assignments

- Assign values to variables, until either...
  1. If all variables have a value, respond **SAT**.
  2. If all literals in some clause are F, then backtrack and try something else. If there is nothing else to try, respond *UNSAT*.

Terminology

- Assigning a value to a variable is called a *decision*.
- A clause with all literals F, is called a *conflict*.

Example

Clauses

$\neg x \lor \neg y$

$x \lor \neg y$

Legend:

false

true

Example

Clauses

$\boxed{\neg x} \vee \neg y$

$\boxed{x} \vee \neg y$

Legend:

false

true

Decisions

$x = T$

Example

Clauses

$\neg x \lor \neg y$

$x \lor \neg y$

Legend:

false

true

Decisions

$x = T$

$y = T$

Example

Clauses

$\neg x \lor \neg y$

$x \lor \neg y$

Legend:

false

true

Decisions

$x = T$

Example

Clauses

$\neg x \vee \neg y$

$x \vee \neg y$

Legend:

false

true

Decisions

$x = T$

$y = F$

Programming Basic Idea

- We will keep a *stack* of decisions as a global variable. We will always first assign the value T to a variable.
- Try to backtrack in case of a conflict.
- Make a new decisions if there is no conflict.

Stack Operations

$\textsc{Backtrack}$ ▷ Flip a T-variable, return FALSE if there is none.

while $\neg\, stack.isEmpty()$
    do $(Var, Value) \leftarrow stock.pop()$
        if $Value = T$ ▷ We found a variable to flip.
            then $stack.push((Var, F))$
                return TRUE
return FALSE ▷ All variables tried with both values.

Stack Operations

BACKTRACK ▷ Flip a T-variable, return FALSE if there is none.

while $\neg$ *stack.isEmpty*()
  do (*Var*, *Value*) $\leftarrow$ *stock.pop*()
    if *Value* $= T$ ▷ We found a variable to flip.
      then *stack.push*((*Var*, F))
        return TRUE
return FALSE ▷ All variables tried with both values.

NEWDECISION

let *Var* be a variable that is not on the stack,
then *stack.push*((*Var*, T))

Basic SAT Solver

```
IsSAT
while true
    do if all clauses satisfied
          then return SAT
       if conflict
          then if ¬Backtrack
                  then return UNSAT
          else NewDecision
```

How Can We Speed-Up?

Observation

If all literals in a clause are false, except for one, then this one must be true. $\boxed{l_1 \vee l_2 \vee \dots l_{k-1}} \quad \vee \quad l_k \quad \vee \quad \boxed{l_{k+1} \vee \dots \vee l_n}$

How Can We Speed-Up?

Observation

If all literals in a clause are false, except for one, then this one must be true. $l_1 \vee l_2 \vee \dots l_{k-1} \quad \vee \quad l_k \quad \vee \quad l_{k+1} \vee \cdots \vee l_n$

Example: y must be F

$\neg x \quad \vee \quad \neg y$

$x \quad \vee \quad \neg y$

How Can We Speed-Up?

Observation

If all literals in a clause are false, except for one, then this one must be true. $l_1 \vee l_2 \vee \dots l_{k-1} \quad \vee \quad l_k \quad \vee \quad l_{k+1} \vee \dots \vee l_n$

Example: y must be F

$\neg x \quad \vee \quad \neg y$

$x \quad \vee \quad \neg y$

How Can We Speed-Up?

Observation

If all literals in a clause are false, except for one, then this one must be true. $l_1 \vee l_2 \vee \ldots l_{k-1} \quad \vee \quad l_k \quad \vee \quad l_{k+1} \vee \cdots \vee l_n$

Example: y **must** be F

$\neg x \vee \neg y$

$x \vee \neg y$

Boolean **C**onstraint **P**ropagation

- A clause with only one non-false literal is called *unit clause*.
- Transitively deriving variable values based on unit clauses is called *Boolean Constraint Propagation* (BCP).

SAT + BCP, Intuitively

- Perform BCP in each iteration of the solver.
- BCP detects conflicts.
- Does not make sense to backtrack a BCP decision unless its reason is backtracked as well. Otherwise BCP derives the same decision again.
- Hence, we can do as before but whenever we backtrack, we retract all the propagated assignments.

Conflict Detection Example

For decisions r, p

1. $\neg r \;\vee\; \neg q$
2. $\neg p \;\vee\; q \;\vee\; z$
3. $\neg p \;\vee\; q \;\vee\; \neg z$

Conflict Detection Example

For decisions r, p

1. $\boxed{\neg r} \vee \neg q$
2. $\neg p \vee q \vee z$
3. $\neg p \vee q \vee \neg z$

Conflict Detection Example

For decisions r, p

1. $\neg r \lor \neg q$
2. $\neg p \lor q \lor z$
3. $\neg p \lor q \lor \neg z$

Conflict Detection Example

For decisions r, p

1. $\neg r \vee \neg q$
2. $\neg p \vee q \vee z$
3. $\neg p \vee q \vee \neg z$

Conflict Detection Example

For decisions r, p

1. $\neg r \lor \neg q$
2. $\neg p \lor q \lor z$
3. $\neg p \lor q \lor \neg z$

Conflict Detection Example

For decisions r, p

1. $\neg r \vee \neg q$
2. $\neg p \vee q \vee z$
3. $\neg p \vee q \vee \neg z$

Conflict Analysis

- One of $\neg p \vee q \vee \neg z$, must be true (but it isn't).
- But, $(p \wedge \neg q) \Rightarrow z$ hence $\neg p \vee q$ must be true.
- But, $r \Rightarrow \neg q$ hence $\neg p \vee \neg r$ must be true.
- $\neg p \vee \neg r$ is called the *conflict clause*.
- Adding conflict clauses to the set is called *learning*.

Further Reading

- Davis Putnam procedure (DPLL), a more general algorithm for first-order logic, *Davis and Putnam, A Computing Procedure for Quantification Theory*
- Clause learning and backtracking, *Silva and Sakallah*, *GRASP — A New Search Algorithm for Satisfiability*,
- A fast way to detect unit clauses, *Moskewicz et al., Chaff: Engineering an efficient SAT solver*
- A nicely implemented and described solver MiniSAT, *Een and Sorensson, An extensible SAT-solver*