CS262 Logic and Verification
Lecture 9: Satisfiability

Boolean satisfiability problem

Problem SAT: Given a propositional formula in conjunctive normal form, is there a satisfying assignment for it?

Further questions: If yes, how can we compute it efficiently? If not, how can we prove this?

Example: $\langle [x, y, \neg z], [\neg x, \top], [\neg y, z, \bot] \rangle$
$= (x \vee y \vee \neg z) \wedge (\neg x \vee \top) \wedge (\neg y \vee z \vee \bot)$

W.l.o.g. we may assume that $\top$ or $\bot$ do not appear: Eliminate them by neutral element rules $[x, \bot] = [x]$, $[x, \top] = [\top] = \top$, $\langle X, \top \rangle = \langle X \rangle$.

Example (continued):

$\langle [x, y, \neg z], [\neg x, \top], [\neg y, z, \bot] \rangle$
$= \langle [x, y, \neg z], [\top], [\neg y, z] \rangle$
$= \langle [x, y, \neg z], [\neg y, z] \rangle$

Definitions:

(positive/negative) literal - a variable, a negated variable
clause - disjunction of literals
CNF formula - formula in conjunctive normal form
k-**CNF formula** - every clause has at most k literals
k-**SAT problem** - input of SAT is a k-CNF

Computational complexity

We can solve the problem SAT in time $2^n \cdot L$ by computing the entire truth table, where L is the total number of literals of the input formula, and n is the number of variables.

This problem is the 'mother' of all NP-complete problems.

If we could the problem efficiently, i.e., in polynomial time, then many other problems could be solved efficiently as well.

1.000.000$ Millenium prize problem: Is P=NP?

Most likely, there is no efficient algorithm known to solve this problem.

Next steps

Possible approaches from here:

- solve special cases efficiently: 2-SAT, Horn formulas
- improve the exponential running time from 2^n to c^n for some $c < 2$
- compact reductions: translate problems into SAT problems with small blowup
- use SAT solvers and heuristics

Theorem

There is a polynomial time algorithm that, given an instance F of SAT, produces a 3-CNF $G(F)$ such that F is satisfiable if and only if $G(F)$ is satisfiable.

If we can solve 3-SAT efficiently, then we can solve SAT efficiently (in polynomial time).

Proof:

Consider a clause containing two literals X and Y. Replace $X \vee Y$ in the clause by a new variable Z (so the clause gets shorter by one literal), and express $X \vee Y \equiv Z$ by a 3-CNF $F(X, Y, Z)$: [...]

Take the conjunction of the current CNF with $F(X, Y, Z)$.

Repeat the process on clauses of length > 3. $\square$

Reductions

Problem COLORING: Given a graph $G = (V, E)$ and integer k, can its vertices be colored properly with k colors?

Definition: A coloring of G is **proper**, if the end vertices of every edge receive distinct colors.

Theorem

There is a polynomial time algorithm that, given an instance (G, k) of COLORING, produces a CNF formula $F(G, k)$ such that G has a proper coloring with k colors if and only if $F(G, k)$ is satisfiable.

If we can solve SAT efficiently, then we can solve COLORING efficiently.

If we can count the number of satisfying assignment of a CNF formula efficiently, then we can count proper k-colorings efficiently.

A concrete way to tackle a problem with SAT solvers

Proof:

For each vertex v and each color k, introduce a variable

$$x_{v,k} = \begin{cases} 1 & \text{if vertex } v \text{ receives color } k \\ 0 & \text{else} \end{cases}$$

Add constraints that every vertex receives exactly one color: [...]

Add constraints that the end vertices of every edge receive two distinct colors: [...]

Number of variables: $|V| \cdot k$.

Size of the formula: polynomial in $|V|$ and $|E|$. $\square$

Solving 2-SAT in polynomial time

First method: **Resolution**

Recall the resolution proof method:

- maintains a conjunction of disjunctions
- first step: resolution expansion into CNF
- second step: resolution rule

Key observation: The resolution rule on clauses of size 2 produces another clause of size ≤ 2. (Example: $[x, y], [\neg x, \neg z] \rightsquigarrow [y, \neg z]$)

At most $1 + 2n + 4\binom{n}{2} = 2n^2 + 1$ clauses can ever occur in the process (n is the number of variables).

If formula is unsatisfiable, then the empty clause [ ] will occur; otherwise it is satisfiable.

Hence resolution is a polynomial method for deciding 2-SAT.

A linear-time algorithm for 2-SAT

Recall that

$$u \lor v \;\equiv\; \neg u \to v \;\equiv\; \neg v \to u$$

Idea: Capture this implication information of a 2-CNF F in a directed graph $D = D(F)$:

- vertex set $V(D) := V \cup \overline{V}$, i.e., all variables $V = V(F)$ and their negation

- edge set
  $E(D) := \{(\neg u, v), (\neg v, u) \mid [u \lor v] \in F\} \cup \{(\neg u, u) \mid [u] \in F\}$

2-clauses lead to two directed edges, whereas a unit clause leads to a single directed edge.

The graph D has $2n$ vertices, where $n := |V|$, and at most $2m$ edges, where $m := |F|$.

A linear-time algorithm for 2-SAT

Examples:

$F = \langle [\neg x_1, x_2], [\neg x_2, x_3], [\neg x_3, x_4], [\neg x_4, x_1], [\neg x_5, x_3], [x_5, x_1] \rangle$

$F = \langle [x, y], [\neg x, y], [z, \neg y], [\neg z, \neg y] \rangle$

Definition: Write $x \rightsquigarrow y$ if there is a directed path from x to y in the graph $D(F)$.

A linear-time algorithm for 2-SAT

Lemma

F is not satisfiable if and only if there is a variable $x \in V$ such that $x \rightsquigarrow \neg x \rightsquigarrow x$.

With this result, satisfiability can be reduced to computing the strongly connected components of the graph D.

Definition: Two vertices u and v in a directed graph are **strongly connected**, if $u \rightsquigarrow v$ and $v \rightsquigarrow u$. The strongly connected components are the maximal subsets of vertices with this property.

The strongly connected components of a directed graph can be computed in linear time in the size of the graph, i.e., in time that is linear in the number of vertices and edges, by well-known standard graph algorithms.

A linear-time algorithm for 2-SAT

Lemma

F is not satisfiable if and only if there is a variable $x \in V$ such that $x \rightsquigarrow \neg x \rightsquigarrow x$.

Proof: Let F' denote the CNF obtained from F by exhaustively applying the resolution rule.

Consider the resolvent $[u, v]$ of $[x, u]$ and $[\neg x, v]$. This resolvent would add two new edges to the graph, namely $\neg u \to v$ and $\neg v \to u$. But the edges $\neg u \to x \to v$ and $\neg v \to \neg x \to u$ were already present. So adding these edges does not alter the relation $\rightsquigarrow$.

F is not satisfiable $\iff$
$[\,]$ appears in F' $\iff$
$[x], [\neg x]$ appears in F' for some variable $x \in V$ $\iff$
$x \to \neg x \to x$ in $D(F')$ $\iff$
$x \rightsquigarrow \neg x \rightsquigarrow x$ in $D(F)$. $\square$

Idea: Want to use our polynomial-time algorithms for solving 2-SAT for solving 3-SAT more efficiently, i.e., in less than 2^n steps (n is the number of variables).

Definition: Given a 3-CNF F over n variables. A subset G of clauses of F is called **independent**, if no two clauses share any variables. We consider such a subset G of **maximal** size, i.e., no further clauses can be added without violating independence.

Example: $F = \langle [a, b, \neg c], [c, \neg d, e], [\neg e, f, g], [\neg f, g, \neg h], [h, i, \neg j] \rangle$

$G = \langle [a, b, \neg c], [\neg e, f, g], [h, i, \neg j] \rangle$

$G' = \langle [c, \neg d, e], [\neg f, g, \neg h] \rangle$

Lemma

Consider a maximal set G of independent 3-clauses in F. Then we have

- $|G| \leq n/3$.
- *For any truth assignment α to the variables in G, $F^{[\alpha]}$ is a 2-CNF. Here, $F^{[\alpha]}$ is the formula obtained from F by setting all variables defined by α to true or false (remove clauses with true literals and remove false literals from clauses).*
- *The number of truth assignments satisfying G is $7^{|G|} \leq 7^{n/3}$.*

Proof: [...]

$\square$

Algorithm: Go through all truth assignments α satisfying G, and check satisfiability of the 2-CNF $F^{[\alpha]}$ in polynomial time (quadratic or linear).

Theorem

Satisfiability of a 3-CNF formula can be decided in time $O(7^{n/3}\mathrm{poly}(n)) = O(1.913^n)$.

Horn satisfiabiliy

We saw that 2-SAT is a special case of SAT that can be solved efficiently

We now consider another special case, where there is no restriction on the size of clauses, but on their structure

Definition: A **Horn clause** is a clause in which there is at most one positive literal (non-negated variable)

A **Horn CNF** is a CNF that has only Horn clauses.

Examples:

$[\neg x, \neg y, \neg z, a] = \neg x \vee \neg y \vee \neg z \vee a = (x \wedge y \wedge z) \rightarrow a$
$[\neg x, \neg y, \neg z]$
$[z]$
$[\,]$

Horn clauses in Prolog

Prolog statements are Horn clauses.

The clause $[\neg x, \neg y, \neg z, a] = (x \wedge y \wedge z) \rightarrow a$ is written as:

```
a :- x, y, z.
```

Satisfiability of Horn CNFs

Here are some simple observations:

If $F = \langle\,\rangle$, then F is satisfied by any assignment.

If every clause has size ≥ 2, then the formula can be satisfied by setting all variables to false.

If the formula has a clause of size $= 1$, then we have to set the literal in this clause to true to satisfy the formula.

If the formula contains the empty clause $[\,]$, then the formula is not satisfiable.

A linear-time algorithm

These observations give the following algorithm:

Input: Horn CNF F
Output: 'Yes' if F is satisfiable, 'No' if F is not satisfiable
while $([\,] \notin F)$ {
    If $F = \langle\,\rangle$ or every clause in F has size ≥ 2, return 'Yes'
    Pick a clause $[u] \in F$ of size $= 1$.
    Remove all clauses containing u from F, and
        remove the literal $\neg u$ from all clauses containing it.
}
return 'No'

Example: $F = \langle[\neg y, \neg x, z], [\neg y, z], [\neg z], [\neg z, x]\rangle$

Theorem

This algorithm decides satisfiability of a Horn CNF in linear time.