Lecture 4
Polynomial-time formula classes
Horn-SAT, 2-SAT, X-SAT, Walk-SAT

Print version of the lecture in *Introduction to Logic for Computer Science*

presented by Prof Hongseok Yang

These lecture notes are very minor variants of the ones made by Prof James Worrell and Prof Christoph Haase for their 'Logic and Proof' course at Oxford.

4.1

1 Polynomial-time fragments of propositional logic

So far, the only method we have to solve the propositional satisfiability problem that can be automated is to use truth tables, which takes exponential time in the formula size in the worst case. In this lecture we show that for Horn formulas, 2-CNF and X-CNF formulas, satisfiability can be decided in polynomial time, whereas for 3-CNF formulas, satisfiability is as hard as the general case.

Horn formulas have numerous computer-science applications due to their good algorithmic properties. In particular, the programming languages Prolog and Datalog are based on Horn clauses in first-order logic.

Definition 1. A CNF formula is a **Horn formula** if each clause contains at most one positive literal.

Example 2. An example of a Horn formula is the following:

$$p_1 \wedge (\neg p_2 \vee \neg p_3) \wedge (\neg p_1 \vee \neg p_2 \vee p_4)$$

Horn formulas can be rewritten in a more intuitive way as conjunctions of implications, called **implication form**. For example,

$$(\mathit{true} \rightarrow p_1) \wedge (p_2 \wedge p_3 \rightarrow \mathit{false}) \wedge (p_1 \wedge p_2 \rightarrow p_4)$$

The satisfiability problem for Horn formulas is called **Horn-SAT**. There is a simple polynomial-time algorithm to determine whether a given Horn formula F is satisfiable using the algorithm below. This algorithm maintains a (partial) assignment $\mathcal{A}$ whose domain is the set $\{p_1, \ldots, p_n\}$ of propositional variables mentioned by F. We consider the set of such assignments ordered pointwise: $\mathcal{A} \leq \mathcal{B}$ if $\mathcal{A}(p_i) \leq \mathcal{B}(p_i)$ for $i = 1, \ldots, n$. Initially $\mathcal{A}$ is assigned the *zero assignment* **0** that sets all variables to the truth value 0, i.e., $\mathcal{A}$ is such that $\mathcal{A}(p_i) = 0$ for $i = 1, \ldots, n$. Thereafter each iteration of the main loop changes $\mathcal{A}(p_i)$ from 0 to 1 for some i until either the input formula is satisfied or a contradiction is reached.

```
INPUT: Horn formula $F$
$T := \emptyset$
while $T$ does not satisfy $F$ do
begin
```

```
    pick an unsatisfied clause $p_1 \wedge \cdots \wedge p_k \to G$
    if $G$ is a variable then $T := T \cup \{G\}$
    if $G = \mathit{false}$ then return UNSAT
end
return $T$
```

It is clear that there can be at most n iterations of the while loop, and so the algorithm terminates in time polynomial in the size of the input formula.

Any assignment $\mathcal{A}$ returned by the algorithm must satisfy F since the termination condition of the while loop is that all clauses are satisfied by $\mathcal{A}$. It thus remains to show that if the algorithm returns "unsat" then the input formula F really is unsatisfiable. To show this, suppose that $\mathcal{B}$ is an assignment that satisfies F. We claim that $\mathcal{A} \le \mathcal{B}$ is a *loop invariant.*[1]

The initialisation $\mathcal{A} := \mathbf{0}$ establishes the invariant. To see that the invariant is maintained by the execution of the loop body, consider an implication $p_1 \wedge \cdots \wedge p_k \to G$ that is not satisfied by $\mathcal{A}$. Then $\mathcal{A}$ satisfies $p_1, \ldots, p_k$ but not G. Since $\mathcal{A} \le \mathcal{B}$, $\mathcal{B}$ also satisfies $p_1, p_2, \cdots, p_k$. It follows that $\mathcal{B}$ satisfies G—so $G \neq \mathit{false}$ and the algorithm does not return "unsat". Moreover, since $\mathcal{B}(G) = 1$ the assignment $\mathcal{A}(G) := 1$ preserves the invariant. This completes the proof of correctness.

The above argument shows that the Horn-SAT algorithm returns the *minimum model* of a Horn formula F, i.e., a model $\mathcal{A}$ such that $\mathcal{A} \le \mathcal{B}$ for any other model $\mathcal{B}$ of F.

Another subclass of formulas of propositional logic with a polynomial-time decidable satisfiability problem are formulas in 2-CNF.

Definition 3. A **2-CNF formula**, or **Krom formula** is a CNF formula F such that every clause has at most two literals.

The satisfiability problem for formulas in 2-CNF is called **2-SAT**. A clause of a 2-CNF formula can be written as an implication $L \to M$ for literals L and M. The key idea underlying the algorithm for satisfiability for 2-CNF formulas is that those implications can be represented by a directed graph:

- For a literal L, define
$$\overline{L} := \begin{cases} p & \text{if } L = \neg p \\ \neg p & \text{otherwise} \end{cases}$$
- The **implication graph** of a 2-CNF formula F is a directed graph $\mathcal{G} = (V, E)$, where
$$V := \{p_1, p_2, \ldots, p_n\} \cup \{\neg p_1, \neg p_2, \ldots, \neg p_n\},$$
with $p_1, p_2, \ldots, p_n$ the propositional variables mentioned in F. For each pair of literals L and M, there is an edge (L, M) iff the clause $(\overline{L} \vee M)$ or $(M \vee \overline{L})$ appears in F.

Figure 1 gives an example of an implication graph. Paths in $\mathcal{G}$ correspond to chains of implications. Observe that for an edge (L, M) there is a corresponding edge $(\overline{M}, \overline{L})$. This edge represents the *contrapositive* implication $\overline{M} \to \overline{L}$ corresponding to the implication $L \to M$. We say that $\mathcal{G}$ is **consistent** if there is no propositional variable p with a path from p to $\neg p$ and a path from $\neg p$ to p in $\mathcal{G}$. Notice that this property can be checked in polynomial time using standard graph algorithms for finding strongly connected components.

Theorem 4. *A 2-CNF formula F is satisfiable iff its implication graph $\mathcal{G}$ is consistent.*

[1]Recall that a predicate I is an invariant of a loop **while** C **do** *body* if I holds initially and whenever the conjunction of the invariant and loop guard $I \wedge C$ holds before the execution of *body*, then I holds after the execution.

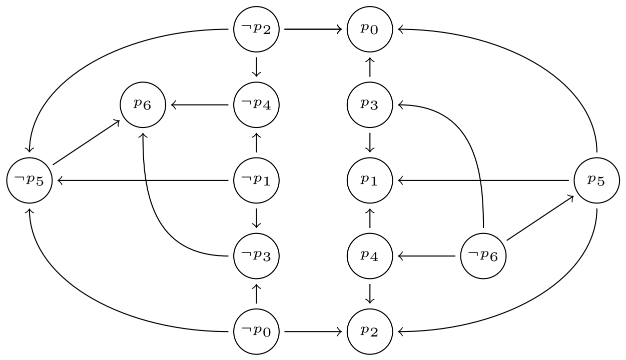

Figure 1: Example of an implication graph for the 2-CNF formula $(p_0 \vee p_2) \wedge (p_0 \vee \neg p_3) \wedge (p_1 \vee \neg p_3) \wedge (p_1 \vee \neg p_4) \wedge (p_2 \vee \neg p_4) \wedge (p_0 \vee \neg p_5) \wedge (p_1 \vee \neg p_5) \wedge (p_2 \vee \neg p_5) \wedge (p_3 \vee p_6) \wedge (p_4 \vee p_6) \wedge (p_5 \vee p_6)$.

The proof of this theorem consists of two steps. First, suppose that $\mathcal{G}$ is not consistent, i.e., that there are paths from p to $\neg p$ and from $\neg p$ to p. Then for any assignment $\mathcal{A}$ that satisfies F we must have $\mathcal{A}(p) \leq \mathcal{A}(\neg p)$ and $\mathcal{A}(\neg p) \leq \mathcal{A}(p)$. But then $\mathcal{A}(p) = \mathcal{A}(\neg p)$, which is impossible. Thus F must be unsatisfiable.

For the converse direction, suppose that $\mathcal{G}$ is consistent. We construct a satisfying assignment incrementally, starting with the empty assignment, using the procedure below.

```
INPUT: 2-CNF formula $F$
$\mathcal{A}$ := empty partial assignment
while there is some unassigned variable do
  begin
  pick a literal $L$ for which there is no path from $L$ to $\overline{L}$, and
  set $\mathcal{A}(L) := 1$
  while there is an edge $(M, N)$ with $\mathcal{A}(M) = 1$ and $\mathcal{A}(N)$ is undefined
     do $\mathcal{A}(N) := 1$
  end
return $\mathcal{A}$
```

The invariant of the outer loop is that any node reachable from a true node is also true. If the invariant holds and all variables are assigned, then we have a satisfying assignment. The invariant of the inner loop is that there is no path from a true node to a false node. If the invariant of the outer loop holds but not all variables have been assigned, then pick a unassigned literal L with no path from L to $\overline{L}$ (such a literal must exist by consistency of $\mathcal{G}$) and set $\mathcal{A}(L) := 1$. After this assignment the invariant of the inner loop (no path from a true node to a false node) is true. Indeed, by assumption there is no path from L to $\overline{L}$, moreover there can be no path from a true node to $\overline{L}$ (equivalently, from L to a false node). Because otherwise L would already have been assigned.

Clearly the body of the inner loop maintains the invariant that there is no path from a true node to a false node. Thus on termination of the inner loop every node reachable from a true node is true.

A **3-CNF** formula is a CNF formula with at most 3 literals per clause, and the corresponding satisfiability problem is called **3-SAT**. While the satisfiability problem for 2-CNF formulas is “easy”, i.e., polynomial-time solvable, we show that the satisfiability problem for 3-CNF formulas is as hard as the general case. More precisely, given an arbitrary propositional formula F, we build an **equisatisfiable** 3-CNF formula G. By this we mean that G is satisfiable if and only if F is satisfiable. Since the transformation from F to G is straightforward to implement, it follows that if we had an polynomial-time algorithm to decide satisfiability for 3-CNF formulas, then we could also decide satisfiability of arbitrary formulas in polynomial time. Note that two logically equivalent formulas are equisatisfiable, but two equisatisfiable formulas need not be logically equivalent.

Theorem 5. *Given an arbitrary formula F, we can compute an equisatisfiable 3-CNF formula G in polynomial time.*

Proof. Let F be an arbitrary formula. We construct an equisatisfiable 3-CNF formula G as follows. Let $F_1, F_2, \ldots, F_n$ be a list of the subformulas of F, with $F_n = F$. Furthermore let the propositional variables appearing in F be $p_1, \ldots, p_m$ and suppose that $F_1 = p_1, \ldots, F_m = p_m$. Corresponding to the remaining subformulas $F_{m+1}, \ldots, F_n$ of F, we introduce new propositional variables $p_{m+1}, \ldots, p_n$. With each new variable p_i, we associate a formula G_i which intuitively asserts that p_i has the same truth value as the subformula F_i.

Formally, the formulas $G_{m+1}, \ldots, G_n$ are defined from $F_{m+1}, \ldots, F_n$ as follows:

- If $F_i = \mathit{true}$, then we define G_i so that it is logically equivalent to true:
$$G_i := p_i \,.$$
- If $F_i = \mathit{false}$, then we define G_i so that it is logically equivalent to false:
$$G_i := \neg p_i \,.$$
- If $F_i = F_j \vee F_k$, then we define G_i so that it is logically equivalent to $p_i \leftrightarrow p_j \vee p_k$:
$$G_i := (\neg p_i \vee p_j \vee p_k) \wedge (\neg p_j \vee p_i) \wedge (\neg p_k \vee p_i) \,.$$
- If $F_i = F_j \wedge F_k$, then we define G_i so that it is logically equivalent to $p_i \leftrightarrow p_j \wedge p_k$:
$$G_i := (\neg p_i \vee p_j) \wedge (\neg p_i \vee p_k) \wedge (\neg p_j \vee \neg p_k \vee p_i)$$
- If $F_i = \neg F_j$, then we define G_i so that it is logically equivalent to $p_i \leftrightarrow \neg p_j$:
$$G_i := (\neg p_i \vee \neg p_j) \wedge (p_j \vee p_i) \,.$$

We now define
$$G := G_{m+1} \wedge G_{m+2} \wedge \cdots \wedge G_n \wedge p_n \,.$$
Then any assignment $\mathcal{A}$ with domain $\{p_1, \ldots, p_m\}$ that satisfies F can be uniquely extended to an assignment $\mathcal{A}'$ with domain $\{p_1, \ldots, p_n\}$ that satisfies G by writing $\mathcal{A}'(p_i) = \mathcal{A}(F_i)$ for $i = m+1, \ldots, n$. Conversely any assignment $\mathcal{A}'$ that satisfies G restricts to an assignment that satisfies F. Thus F and G are equisatisfiable. □

Finally, we consider formulas that can be written as conjunctions of XOR-clauses, where each XOR-clause is an exclusive-or of literals. Such formulas look like CNF-formulas, but with exclusive-or instead of disjunction. For example, consider the formula
$$F = (p_1 \oplus p_3) \wedge (\neg p_1 \oplus p_2) \wedge (p_1 \oplus p_2 \oplus \neg p_3) \,.$$

The satisfiability of F can be formulated as a system of linear equations over $\mathbb{Z}_2$ (the integers modulo 2), with one equation for each clause.
$$\begin{array}{ccccccc}
p_1 & & & + & p_3 & = & 1 \\
1 + p_1 & + & p_2 & & & = & 1 \\
p_1 & & p_2 & + & 1 + p_3 & = & 1
\end{array}$$

Simplifying yields:
$$\begin{array}{ccccccc}
p_1 & & & + & p_3 & = & 1 \\
p_1 & + & p_2 & & & = & 0 \\
p_1 & + & p_2 & + & p_3 & = & 0
\end{array}$$

Reducing the system to echelon form using Gaussian elimination and solving yields $p_1 = 1, p_2 = 1, p_3 = 0$.

In general we can reduce the SAT problem for conjunctions of XOR-clauses to solving linear equations over $\mathbb{Z}_2$. Such equations can be solved by Gaussian elimination (which requires a cubic number of arithmetic operations).

2 Walk-SAT: A randomised algorithm for satisfiability

The algorithms that we looked at so far are all exact in the sense that once they stop, they will tell us for sure whether the input formula is satisfiable. In this section, we describe a very simple randomised algorithm **Walk-SAT** for deciding satisfiability of CNF formulas. We show that Walk-SAT yields a polynomial-time algorithm when run on 2-CNF formulas.

Given a CNF formula F, Walk-SAT starts by guessing an assignment uniformly at random. While there is some unsatisfied clause in F, the algorithm picks a literal in that clause (again at random) and flips its truth value. If a satisfying assignment has not been found after r steps, where r is a parameter, then algorithm returns "unsat":

```
INPUT: CNF formula $F$ with $n$ variables, repetition parameter $r$
pick a random assignment
repeat $r$ times
    pick an unsatisfied clause
    pick a literal in the clause uniformly at random, and flip its value
    if $F$ is satisfied, then return the current assignment
return unsat
```

If F is not satisfiable, then clearly the procedure will certainly return "unsat". However, it is possible that F is satisfiable and the algorithm halts before finding a satisfying assignment. We say that Walk-SAT has *one-sided errors*. Below we will show that for any 2-CNF formula F with n variables and any m, choosing $r = 2mn^2$ makes the error probability of Walk-SAT be at most 2^{-m}. Thus, we obtain a polynomial-time algorithm with exponentially small error probability.

Consider a 2-CNF formula F with a satisfying assignment $\mathcal{A}$. We bound the expected number of flips to find this assignment. Of course, the algorithm may terminate successfully by finding another satisfying assignment, but we only seek an upper bound on the expected running time.

We will need the following result from elementary probability theory.

Proposition 6 (Markov's Inequality)**.** Let X be a non-negative random variable. Then for all $a > 0$, $\Pr(X \geq a) \leq \dfrac{\mathbf{E}[X]}{a}$.

Proof. Define a random variable

$$I = \begin{cases} 1 & X \geq a \\ 0 & \text{otherwise.} \end{cases}$$

Then $I \leq X/a$, since $X \geq 0$. Hence

$$\frac{\mathbf{E}[X]}{a} \geq \mathbf{E}[I] = \Pr(I = 1) = \Pr(X \geq a)\,.$$

$\square$

Define the distance between two assignments to be the number of variables on which they differ. Let T_i be the maximum over all assignments $\mathcal{B}$ at distance i from $\mathcal{A}$ of the expected number of variable-flipping steps to reach $\mathcal{A}$ starting from $\mathcal{B}$. By definition, $T_0 = 0$ and clearly $T_n \leq 1 + T_{n-1}$. Otherwise, when we flip, we choose from among two literals in a clause that is not satisfied by the current assignment. Since such a clause is satisfied by $\mathcal{A}$, at least one of those literals must have a different value under $\mathcal{A}$ than $\mathcal{B}$. Thus the probability of moving closer to $\mathcal{A}$ is at least $1/2$ and the probability of moving farther from $\mathcal{A}$ is at most $1/2$. In summary we have

$$\begin{aligned} T_0 &= 0 \\ T_n &\leq 1 + T_{n-1} \\ T_i &\leq 1 + (T_{i+1} + T_{i-1})/2 \qquad 0 < i < n \end{aligned} \tag{1}$$

To obtain an upper bound on the T_i we consider the situation in which (1) holds as an equality. Defining $H_0, \ldots, H_n$ by the equations

$$\begin{aligned} H_0 &= 0 \\ H_n &= 1 + H_{n-1} \\ H_i &= 1 + (H_{i+1} + H_{i-1})/2 \qquad 0 < i < n \end{aligned}$$

we have $T_i \leq H_i$ for $i = 0, \ldots, n$.

The above is a system of $n+1$ linearly independent equations in $n+1$ unknowns, which therefore has a unique solution. Adding all the equations together we get $H_1 = 2n - 1$. Then solving the H_1-equation for H_2, we get $H_2 = 4n - 4$. Continuing in this manner yields $H_i = 2in - i^2$. So the worst expected time to hit $\mathcal{A}$ is $H_n = n^2$.

Theorem 7. *Consider a run of Walk-SAT on a satisfiable 2-CNF formula with n variables. Choosing $r = 2mn^2$, the probability of returning a satisfying assignment is at least $1 - 2^{-m}$.*

Proof. We can divide the $2mn^2$ iterations of the main loop into m phases, each consisting of $2n^2$ iterations. Since the expected number of iterations to find a satisfying assignment from any given starting point is at most n^2, by Markov's inequality the probability that a satisfying assignment is not found in any given phase is at most $n^2/2n^2 = 1/2$. Thus the probability that an unsatisfying assignment is not found over all m phases is at most 2^{-m}. $\square$

We have analysed Walk-SAT in terms of a one-dimensional random walk on line $\{0, \ldots, n\}$ with absorbing barrier 0 and reflecting barrier n. A similar analysis can be carried out for 3-CNF formulas, but with a probability $2/3$ of going right and $1/3$ of going left. However in this case we require the parameter r to be exponential in n to get a decent error bound.