**Logic and Proof** Hilary 2024

# Polynomial-Time Formula Classes

*James Worrell*

So far the only method we have to solve the propositional satisfiability problem 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 and 2-CNF formulas satisfiability can be decided in polynomial time, whereas for 3-CNF formulas satisfiability is as hard as the general case. We also show that if we replace disjunction in CNF formulas with exclusive-or then satisfiability can again be determined in polynomial time.

## 1 Horn Formulas

We say that a disjunctive clause is a *Horn clause* if it has most one positive literal, called the *head* of the clause, and any number of negative literals, called the *body* of the clause. A CNF formula all of whose clauses are Horn clauses is called a *Horn formula.* For example

$$p_1 \wedge (\neg p_2 \vee \neg p_3) \wedge (\neg p_1 \vee \neg p_2 \vee p_4) \tag{1}$$

is a Horn formula.

Horn clauses can be rewritten in a more intuitive way as implications in which the body of the clause implies the head. For example, the Horn formula (1) can be rewritten

$$(\mathbf{true} \to p_1) \wedge (p_2 \wedge p_3 \to \mathbf{false}) \wedge (p_1 \wedge p_2 \to p_4)\,.$$

Horn clauses have numerous computer-science applications. In particular, the programming languages Prolog and Datalog are based on Horn clauses in first-order logic.

There is a simple polynomial-time algorithm to determine whether a given Horn formula $F$ is satisfiable, see Figure 1. This algorithm maintains a valuation $\mathcal{A}$ whose domain is the set $\{p_1, \ldots, p_n\}$ of propositional variables mentioned by $F$. We consider the set of such valuations 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 valuation $\mathbf{0}$, where $\mathbf{0}[\![p_i]\!] = 0$ 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.

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 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 $\mathcal{A} \leq \mathcal{B}$ is a *loop invariant.*[1]

The initialisation $\mathcal{A} := \mathbf{0}$ establishes the invariant. To see that the invariant is maintained by an 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 $\mathcal{A} \leq \mathcal{B}$, $\mathcal{B}$ also satisfies $p_1, p_2, \cdots, p_k$. It follows that $\mathcal{B}$

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

**Input:** Horn formula $F$
$\mathcal{A} := \mathbf{0}$
**while** $\mathcal{A}$ 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** $\mathcal{A}[\![G]\!] := 1$ **else return** "UNSAT"
**end**
**return** $\mathcal{A}$

Figure 1: Horn-SAT algorithm

satisfies $G$—so $G \neq$ **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 $\mathcal{A} \leq \mathcal{B}$ for any other model $\mathcal{B}$ of $F$.

## 2 2-CNF Formulas

A *2-CNF formula* is a CNF formula $F$ in which every clause has at most two literals. Such clauses can be written in the form $L \to M$ for literals $L$ and $M$. 2-CNF formulas are also known as *Krom* formulas. In this section we show that the satisfiability problem for 2-CNF formulas can be solved in polynomial time. In fact we show that this problem can be reduced to the reachability problem for directed graphs, which can be solved in linear time.

Let $F$ be a 2-CNF formula. We define a directed graph $\mathcal{G} = (V, E)$, called the *implication graph* of $F$, as follows. The set of vertices is

$$V \stackrel{\text{def}}{=} \{p_1, p_2, \ldots, p_n\} \cup \{\neg p_1, \neg p_2, \ldots, \neg p_n\},$$

where $p_1, p_2, \ldots, p_n$ are the propositional variables mentioned in $F$. For each pair of literals $L, M$ such that $L \to M$ is a clause of $F$ we include directed edges $(L, M)$ and $(\overline{M}, \overline{L})$ in $E$, where

$$\overline{L} = \begin{cases} \neg p & \text{if } L = p \\ p & \text{if } L = \neg p \end{cases}$$

denotes the complementary literal of $L$. Note that the edge $(\overline{M}, \overline{L})$ corresponds to the *contrapositive* implication $\overline{M} \to \overline{L}$.

Paths in $\mathcal{G}$ correspond to chains of implications. We say that $\mathcal{G}$ is *consistent* if there is no literal $L$ such that both $L$ and $\overline{L}$ lies in the same SCC of $\mathcal{G}$. Note that if $\mathcal{G}$ is consistent then the SCC's can be arranged in *distinct* dual pairs: if $C$ is an SCC then $\overline{C} := \{\overline{L} : L \in C\}$ is a different SCC.

**Theorem 1.** A 2-CNF formula $F$ is satisfiable iff its implication graph $\mathcal{G}$ is consistent.

*Proof.* 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.

**Input:** 2-CNF formula $F$ with consistent implication graph $\mathcal{G}$
compute topological ordering $C_1 < \cdots < C_n$ of SCC's of $\mathcal{G}$
$\mathcal{A} :=$ empty valuation
**while** there is some unassigned variable **do**
**begin**
  pick the least SCC $C$ whose literals are unassigned
  assign true to all literals in $C$ and assign false to all literals in $\overline{C}$
**end**
**return** $\mathcal{A}$

Figure 2: Algorithm for 2-SAT

Conversely, suppose that $\mathcal{G}$ is consistent. We construct a satisfying assignment $\mathcal{A}$ using the procedure in Figure 2. We start by computing a topological ordering $C_1 < C_2 < \ldots < C_n$ of the SCC's of $\mathcal{G}$, i.e., such that if $i < j$ then there is no path from $C_i$ to $C_j$. The loop invariant is that if a literal $L$ is assigned true by $\mathcal{A}$, then all literals reachable from $L$ are also assigned true by $\mathcal{A}$. This clearly guarantees that the assignment $\mathcal{A}$ satisfies $F$ on termination of the algorithm.

It remains to verify that the invariant is preserved by the loop body. First note that the assignments therein are well-defined. More precisely, for each literal $L \in C$ the dual literal $\overline{L}$ lies in $\overline{C}$, which is a different SCC by the consistency of $\mathcal{G}$; thus we can consistently assign true to all literals in $C$ and false to all literals in $\overline{C}$ (indeed, these two actions are one and the same thing). For preservation of the invariant we argue as follows. Observe that (i) by leastness of $C$ there is no path from $C$ to an unassigned literal (and hence no path from $C$ to $\overline{C}$); (ii) by the loop invariant, there is no path from $C$ to an SCC that is already assigned false (equivalently, there is no path to $C$ from an SCC already assigned true). But the combination of (i) and (ii) entails that assigning true to all literals in $C$ and false to all literals in $\overline{C}$ preserves the invariant.

□

A common feature of the algorithms for deciding satisfiability for Horn formulas and 2-CNF formulas is that they build satisfying assignments incrementally *without backtracking*. This last feature is the main difference with procedures for deciding satisfiability of general CNF formulas.

# 3 Walk-SAT

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".

If $F$ is not satisfiable then clearly the procedure will certainly return "UNSAT". However it is possible for $F$ to be satisfiable and the algorithm to halt before finding a satisfying assignment. We say that Walk-SAT has *one-sided errors*. Below we will show that for a 2-CNF formulas $F$ with $n$

**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 return the current assignment
return "UNSAT"

Figure 3: Walk-SAT algorithm

variables, choosing $r = 2mn^2$ the error probability of Walk-SAT is 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 2** (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

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

□

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 $T_n = 1 + T_{n-1}$. 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

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

To obtain an upper bound on the $T_i$ we consider the situation in which (2) 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 3.** 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 valuation from any given starting point is at most $n^2$, by Markov's inequality the probability that a satisfying valuation is not found in any given phase is at most $n^2/2n^2 = 1/2$. Thus the probability that an unsatisfying valuation 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 left and $1/3$ of going right. However in this case we require the parameter $r$ to be exponential in $n$ to get a decent error bound.

## 4 3-CNF Formulas

A 3-CNF formula is a CNF formula with at most 3 literals per clause. 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.

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 non-atomic 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 $F_i = F_j \vee F_k$ 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.

## 5 XOR-Clauses

In this final section 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.

$$\begin{array}{lclclcl}
p_1 & & & + & p_3 & = & 1 \\
1+p_1 & + & p_2 & & & = & 1 \\
p_1 & & p_2 & + & 1+p_3 & = & 1
\end{array}$$

Simplifying yields:

$$\begin{array}{lclclcl}
p_1 & & & + & p_3 & = & 1 \\
p_1 & + & p_2 & & & = & 0 \\
p_1 & + & p_2 & + & p_3 & = & 0
\end{array}$$

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).

**Exercise 4.** Consider the following combinatorial puzzle. You have an $N \times N$ grid, each cell of which is coloured black or white. A move involves selecting a cell and inverting the colours of that cell and its north, south, east, and west neighbours on the grid. (So a cell has between 2 and 4 neighbours, depending on which boundaries of the grid it lies on.) Given an initial configuration, the goal of the puzzle is to end up with all cells black.

Give a translation of this puzzle to the satisfiability problem for conjunctions of XOR-clauses. Your translation should be such that from a satisfying assignment one can read off a sequence of moves that solves the puzzle.