The Satisfiability Problem

Cook's Theorem: An NP-Complete Problem

Restricted SAT: CSAT, 3SAT

Boolean Expressions

- Boolean, or propositional-logic expressions are built from variables and constants using the operators AND, OR, and NOT.
  - Constants are true and false, represented by 1 and 0, respectively.
  - We'll use concatenation (juxtaposition) for AND, + for OR, - for NOT, unlike the text.

Example: Boolean expression

- (x+y)(-x + -y) is true only when variables x and y have opposite truth values.
- Note: parentheses can be used at will, and are needed to modify the precedence order NOT (highest), AND, OR.

The Satisfiability Problem (*SAT*)

- Study of boolean functions generally is concerned with the set of *truth assignments* (assignments of 0 or 1 to each of the variables) that make the function true.
- NP-completeness needs only a simpler question (SAT): does there exist a truth assignment making the function true?

Example: SAT

- (x+y)(-x + -y) is satisfiable.
- There are, in fact, two satisfying truth assignments:
  1. x=0; y=1.
  2. x=1; y=0.
- x(-x) is not satisfiable.

SAT as a Language/Problem

- An instance of SAT is a boolean function.
- Must be coded in a finite alphabet.
- Use special symbols (,), +, - as themselves.
- Represent the i-th variable by symbol x followed by integer i in binary.

Example: Encoding for SAT

- (x+y)(-x + -y) would be encoded by the string `(x1+x10)(-x1+-x10)`

SAT is in **NP**

- There is a multitape NTM that can decide if a Boolean formula of length n is satisfiable.
- The NTM takes $O(n^2)$ time along any path.
- Use nondeterminism to guess a truth assignment on a second tape.
- Replace all variables by guessed truth values.
- Evaluate the formula for this assignment.
- Accept if true.

Cook's Theorem

- SAT is NP-complete.
  - Really a stronger result: formulas may be in conjunctive normal form (CSAT) – later.
- To prove, we must show how to construct a polytime reduction from each language L in **NP** to SAT.
- Start by assuming the most resticted possible form of NTM for L (next slide).

Assumptions About NTM for L

1. One tape only.
2. Head never moves left of the initial position.
3. States and tape symbols are disjoint.

- Key Points: States can be named arbitrarily, and the constructions many-tapes-to-one and two-way-infinite-tape-to-one at most square the time.

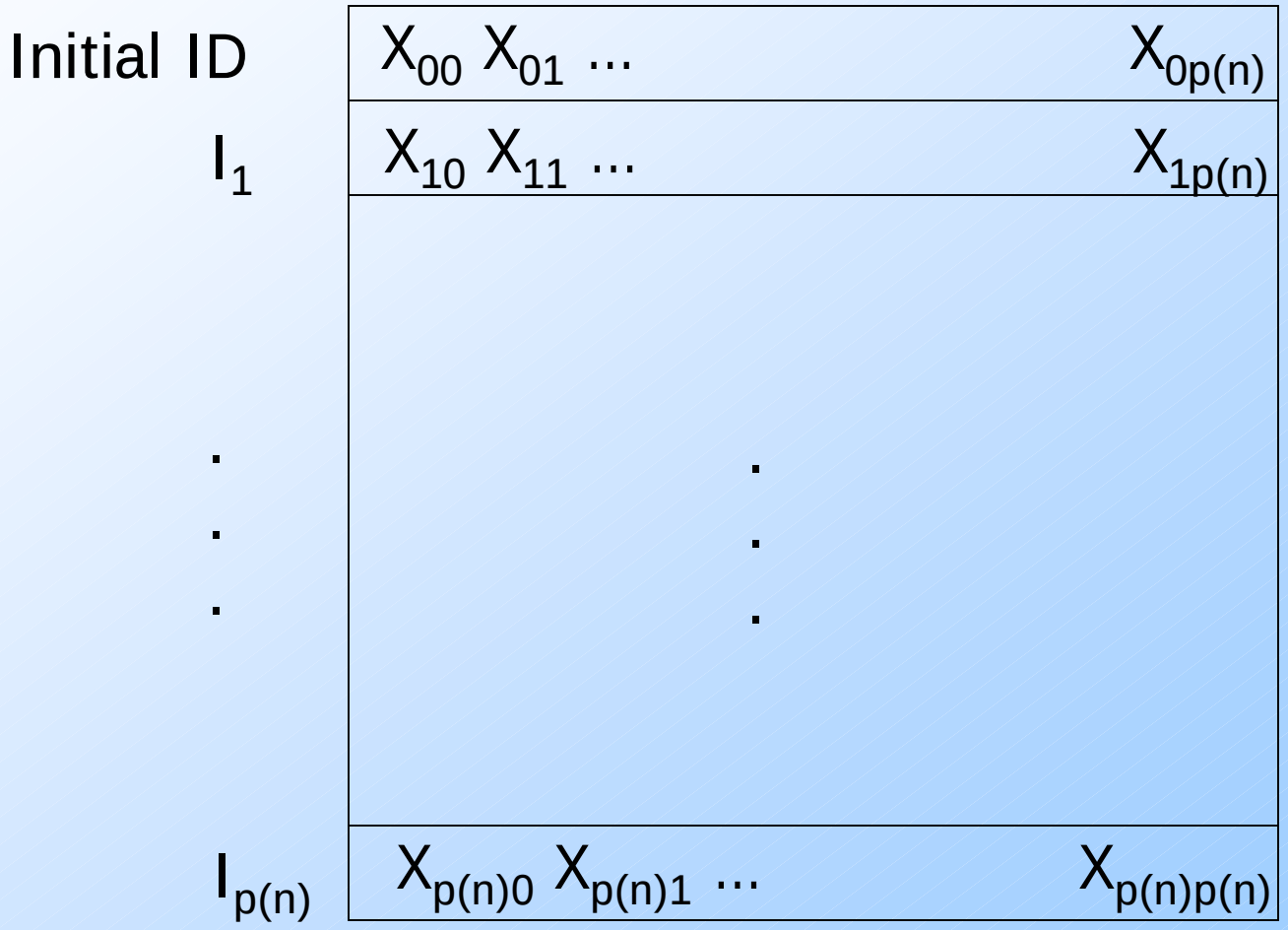
More About the NTM M for L

- Let p(n) be a polynomial time bound for M.
- Let w be an input of length n to M.
- If M accepts w, it does so through a sequence $I_0 \vdash I_1 \vdash \ldots \vdash I_{p(n)}$ of p(n)+1 ID's.
  - Assume trivial move from a final state.
- Each ID is of length at most p(n)+1, counting the state.

From ID Sequences to Boolean Functions

- The Boolean function that the transducer for L will construct from w will have $(p(n)+1)^2$ "*variables*."
- Let variable X_{ij} represent the j-th position of the i-th ID in the accepting sequence for w, if there is one.
  - i and j each range from 0 to p(n).

Picture of Computation as an Array



Intuition

- From M and w we construct a boolean formula that forces the X's to represent one of the possible ID sequences of NTM M with input w, if it is to be satisfiable.
- It *is* satisfiable iff some sequence leads to acceptance.

From ID's to Boolean Variables

- The X_{ij}'s are not boolean variables; they are states and tape symbols of M.
- However, we can represent the value of each X_{ij} by a family of Boolean variables y_{ijA}, for each possible state or tape symbol A.
- y_{ijA} is true if and only if $X_{ij} = A$.

Points to Remember

1. The boolean function has components that depend on n.
   - These must be of size polynomial in n.
2. Other pieces depend only on M.
   - No matter how many states/symbols m has, these are of constant size.
3. Any logical formula about a set of variables whose size is independent of n can be written somehow.

Designing the Function

- We want the Boolean function that describes the X_{ij}'s to be satisfiable if and only if the NTM M accepts w.
- Four conditions:
  1. Unique: only one symbol per position.
  2. Starts right: initial ID is q_0w.
  3. Moves right: each ID follows from the next by a move of M.
  4. Finishes right: M accepts.

Unique

- Take the AND over all i, j, Y, and Z of $(-y_{ijY} + -y_{ijZ})$.
- That is, it is not possible for X_{ij} to be both symbols Y and Z.

Starts Right

- The Boolean Function needs to assert that the first ID is the correct one with $w = a_1 \ldots a_n$ as input.
  1. $X_{00} = q_0$.
  2. $X_{0i} = a_i$ for i = 1,..., n.
  3. X_{0i} = B (blank) for i = n+1,..., p(n).
- Formula is the AND of y_{0iZ} for all i, where Z is the symbol in position i.

Finishes Right

- Somewhere, there must be an accepting state.
- Form the OR of Boolean variables y_{ijq}, where i and j are arbitrary and q is an accepting state.
- Note: differs from text.

Running Time So Far

- Unique requires $O(p^2(n))$ symbols be written.
  - Parentheses, signs, propositional variables.
- Algorithm is easy, so it takes no more time than $O(p^2(n))$.
- Starts Right takes $O(p(n))$ time.
- Finishes Right takes $O(p^2(n))$ time.

Variation over symbols Y and Z is independent of n, so covered by constant.

Variation over i and j

Running Time – (2)

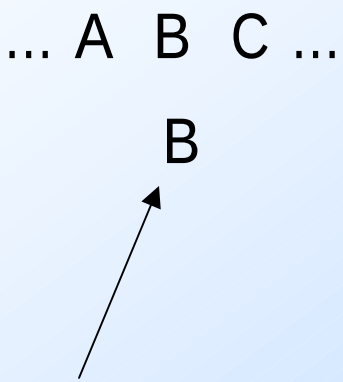
- Caveat: Technically, the propositions that are output of the transducer must be coded in a fixed alphabet, e.g., x10011 rather than y_{ijA}.
- Thus, the time and output length have an additional factor O(log n) because there are $O(p^2(n))$ variables.
- But log factors do not affect polynomials

Moves Right

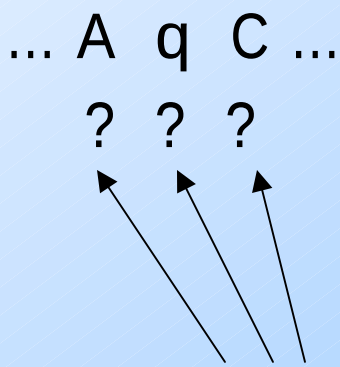
Works because Unique assures only one y_{ijX} true.

- $X_{ij} = X_{i-1,j}$ whenever the state is none of $X_{i-1,j-1}$, $X_{i-1,j}$, or $X_{i-1,j+1}$.
- For each i and j, construct a formula that says (in propositional variables) the OR of "$X_{ij} = X_{i-1,j}$" and all $y_{i-1,k,A}$ where A is a state symbol (k = i-1, i, or i+1).
  - Note: $X_{ij} = X_{i-1,j}$ is the OR of $y_{ijA}.y_{i-1,jA}$ for all symbols A.

Constraining the Next Symbol



Easy case;
must be B



Hard case; all
three may depend
on the move of M

Moves Right – (2)

- In the case where the state is nearby, we need to write an expression that:
  1. Picks one of the possible moves of the NTM M.
  2. Enforces the condition that when $X_{i-1,j}$ is the state, the values of $X_{i,j-1}$, $X_{i,j}$, and $X_{i,j+1}$. are related to $X_{i-1,j-1}$, $X_{i-1,j}$, and $X_{i-1,j+1}$ in a way that reflects the move.

Example: Moves Right

Suppose δ(q, A) contains (p, B, L).

Then one option for any i, j, and C is:

C q A

p C B

If δ(q, A) contains (p, B, R), then an option for any i, j, and C is:

C q A

C B p

Moves Right – (3)

- For each possible move, express the constraints on the six X's by a Boolean formula.
- For each i and j, take the OR over all possible moves.
- Take the AND over all i and j.
- Small point: for edges (e.g., state at 0), assume invisible symbols are blank.

Running Time

- We have to generate $O(p^2(n))$ Boolean formulas, but each is constructed from the moves of the NTM M, which is fixed in size, independent of the input w.
- Takes time $O(p^2(n))$ and generates an output of that length.
  - Times log n, because variables must be coded in a fixed alphabet.

Cook's Theorem – Finale

- In time $O(p^2(n) \log n)$ the transducer produces a boolean formula, the AND of the four components: Unique, Starts, Finishes, and Moves Right.
- If M accepts w, the ID sequence gives us a satisfying truth assignment.
- If satisfiable, the truth values tell us an accepting computation of M.

Picture So Far

- We have one NP-complete problem: SAT.
- In the future, we shall do polytime reductions of SAT to other problems, thereby showing them NP-complete.
- Why? If we polytime reduce SAT to X, and X is in **P**, then so is SAT, and therefore so is all of **NP**.

Conjunctive Normal Form

- A Boolean formula is in *Conjunctive Normal Form* (CNF) if it is the AND of *clauses*.
- Each clause is the OR of *literals*.
- A literal is either a variable or the negation of a variable.
- Problem *CSAT* : is a Boolean formula in CNF satisfiable?

Example: CNF

(x + -y + z)(-x)(-w + -x + y + z) (...

NP-Completeness of CSAT

- The proof of Cook's theorem can be modified to produce a formula in CNF.
- Unique is already the AND of clauses.
- Starts Right is the AND of clauses, each with one variable.
- Finishes Right is the OR of variables, i.e., a single clause.

NP-Completeness of CSAT – (2)

- Only Moves Right is a problem, and not much of a problem.
- It is the product of formulas for each i and j.
- Those formulas are fixed, independent of n.

NP-Completeness of CSAT – (3)

- You can convert any formula to CNF.
- It may exponentiate the size of the formula and therefore take time to write down that is exponential in the size of the original formula, but these numbers are all fixed for a given NTM M and independent of n.

k-SAT

- If a boolean formula is in CNF and every clause consists of exactly k literals, we say the boolean formula is an instance of *k-SAT*.
  - Say the formula is in *k-CNF*.
- Example: 3-SAT formula

(x + y + z)(x + -y + z)(x + y + -z)(x + -y + -z)

k-SAT Facts

- Every boolean formula has an equivalent CNF formula.
  - But the size of the CNF formula may be exponential in the size of the original.
- Not every boolean formula has a k-SAT equivalent.
- 2SAT is in **P**; 3SAT is NP-complete.

Proof: 2SAT is in **P** (Sketch)

- Pick an assignment for some variable, say x = true.
- Any clause with –x forces the other literal to be true.
  - Example: (-x + -y) forces y to be false.
- Keep seeing what other truth values are forced by variables with known truth values.

Proof – (2)

- One of three things can happen:
  1. You reach a contradiction (e.g., z is forced to be both true and false).
  2. You reach a point where no more variables have their truth value forced, but some clauses are not yet made true.
  3. You reach a satisfying truth assignment.

Proof – (3)

- Case 1: (Contradiction) There can only be a satisfying assignment if you use the other truth value for x.
  - Simplify the formula by replacing x by this truth value and repeat the process.
- Case 3: You found a satisfying assignment, so answer “yes.”

Proof – (4)

- Case 2: (You force values for some variables, but other variables and clauses are not affected).
  - Adopt these truth values, eliminate the clauses that they satisfy, and repeat.
- In Cases 1 and 2 you have spent $O(n^2)$ time and have reduced the length of the formula by ≥ 1, so $O(n^3)$ total.

3SAT

- This problem is NP-complete.
- Clearly it is in **NP**, since SAT is.
- It is not true that every Boolean formula can be converted to an equivalent 3-CNF formula, even if we exponentiate the size of the formula.

3SAT – (2)

- But we don’t need equivalence.
- We need to reduce every CNF formula F to some 3-CNF formula that is satisfiable if and only if F is.
- Reduction involves introducing new variables into long clauses, so we can split them apart.

Reduction of CSAT to 3SAT

- Let $(x_1+\ldots+x_n)$ be a clause in some CSAT instance, with $n \geq 4$.
  - Note: the x's are literals, not variables; any of them could be negated variables.
- Introduce new variables $y_1,\ldots,y_{n-3}$ that appear in no other clause.

CSAT to 3SAT – (2)

- Replace $(x_1+\ldots+x_n)$ by
  $(x_1+x_2+y_1)(x_3+y_2+ -y_1) \ldots (x_i+y_{i-1}+ -y_{i-2})$
  $\ldots (x_{n-2}+y_{n-3}+ -y_{n-4})(x_{n-1}+x_n+ -y_{n-3})$
- If there is a satisfying assignment of the x's for the CSAT instance, then one of the literals x_i must be made true.
- Assign y_j = true if $j < i-1$ and y_j = false for larger j.

CSAT to 3SAT – (3)

- We are not done.
- We also need to show that if the resulting 3SAT instance is satisfiable, then the original CSAT instance was satisfiable.

CSAT to 3SAT – (4)

- Suppose $(x_1+x_2+y_1)(x_3+y_2+ -y_1)$ ...
  $(x_{n-2}+y_{n-3}+ -y_{n-4})(x_{n-1}+x_n+ -y_{n-3})$
  is satisfiable, but none of the x's is true.
- The first clause forces y_1 = true.
- Then the second clause forces y_2 = true.
- And so on ... all the y's must be true.
- But then the last clause is false.

CSAT to 3SAT – (5)

- There is a little more to the reduction, for handling clauses of 1 or 2 literals.
- Replace (x) by $(x+y_1+y_2)$ $(x+y_1+ -y_2)$ $(x+ -y_1+y_2)$ $(x+ -y_1+ -y_2)$.
- Replace (w+x) by (w+x+y)(w+x+ -y).
- Remember: the y's are different variables for each CNF clause.

CSAT to 3SAT Running Time

- This reduction is surely polynomial.
- In fact it is linear in the length of the CSAT instance.
- Thus, we have polytime-reduced CSAT to 3SAT.
- Since CSAT is NP-complete, so is 3SAT.