

Day28

NP-Complete Problems

3SAT: A literal is a Boolean variable (e.g. x) or a negated Boolean variable (e.g. \bar{x}). A Boolean formula is a **3cnf-formula** if it is a Boolean formula in conjunctive normal form (a conjunction of disjunctive clauses of literals) and each clause has three literals.

$$3SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$$

Example string in $3SAT$

$$\langle (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (x \vee y \vee z) \rangle$$

Example string not in $3SAT$

$$\langle (x \vee y \vee z) \wedge (x \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z}) \rangle$$

Cook-Levin Theorem: $3SAT$ is NP -complete.

Are there other NP-complete problems? To prove that X is NP -complete

- *From scratch:* prove X is in NP and that all NP problems are polynomial-time reducible to X .
- *Using reduction:* prove X is in NP and that a known-to-be NP -complete problem is polynomial-time reducible to X .

CLIQUE: A k -**clique** in an undirected graph is a maximally connected subgraph with k nodes.

$$CLIQUE = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

Example string in *CLIQUE*

Example string not in *CLIQUE*

Theorem (Sipser 7.32):

$$3SAT \leq_P CLIQUE$$

Given a Boolean formula in conjunctive normal form with k clauses and three literals per clause, we will map it to a graph so that the graph has a clique if the original formula is satisfiable and the graph does not have a clique if the original formula is not satisfiable.

The graph has $3k$ vertices (one for each literal in each clause) and an edge between all vertices except

- vertices for two literals in the same clause
- vertices for literals that are negations of one another

Example: $(x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (x \vee y \vee z)$

| Model of Computation | Class of Languages |
|--|---|
| <p>Deterministic finite automata: formal definition, how to design for a given language, how to describe language of a machine? Nondeterministic finite automata: formal definition, how to design for a given language, how to describe language of a machine? Regular expressions: formal definition, how to design for a given language, how to describe language of expression? <i>Also:</i> converting between different models.</p> | <p>Class of regular languages: what are the closure properties of this class? which languages are not in the class? using pumping lemma to prove nonregularity.</p> |
| <p>Push-down automata: formal definition, how to design for a given language, how to describe language of a machine? Context-free grammars: formal definition, how to design for a given language, how to describe language of a grammar?</p> | <p>Class of context-free languages: what are the closure properties of this class? which languages are not in the class?</p> |
| <p>Turing machines that always halt in polynomial time</p> <p>Nondeterministic Turing machines that always halt in polynomial time</p> | <p>P</p> <p>NP</p> |
| <p>Deciders (Turing machines that always halt): formal definition, how to design for a given language, how to describe language of a machine?</p> | <p>Class of decidable languages: what are the closure properties of this class? which languages are not in the class? using diagonalization and mapping reduction to show undecidability</p> |
| <p>Turing machines formal definition, how to design for a given language, how to describe language of a machine?</p> | <p>Class of recognizable languages: what are the closure properties of this class? which languages are not in the class? using closure and mapping reduction to show unrecognizability</p> |

Given a language, prove it is regular

Strategy 1: construct DFA recognizing the language and prove it works.

Strategy 2: construct NFA recognizing the language and prove it works.

Strategy 3: construct regular expression recognizing the language and prove it works.

“Prove it works” means ...

Example: $L = \{w \in \{0,1\}^* \mid w \text{ has odd number of 1s or starts with } 0\}$

Using NFA

Using regular expressions

Example: Select all and only the options that result in a true statement: “To show a language A is not regular, we can...”

- a. Show A is finite
- b. Show there is a CFG generating A
- c. Show A has no pumping length
- d. Show A is undecidable

Example: What is the language generated by the CFG with rules

$$S \rightarrow aSb \mid bY \mid Ya$$

$$Y \rightarrow bY \mid Ya \mid \varepsilon$$

Example: Prove that the language $T = \{\langle M \rangle \mid M \text{ is a Turing machine and } L(M) \text{ is infinite}\}$ is undecidable.

Example: Prove that the class of decidable languages is closed under concatenation.