Theory of Computation
Chapter 7

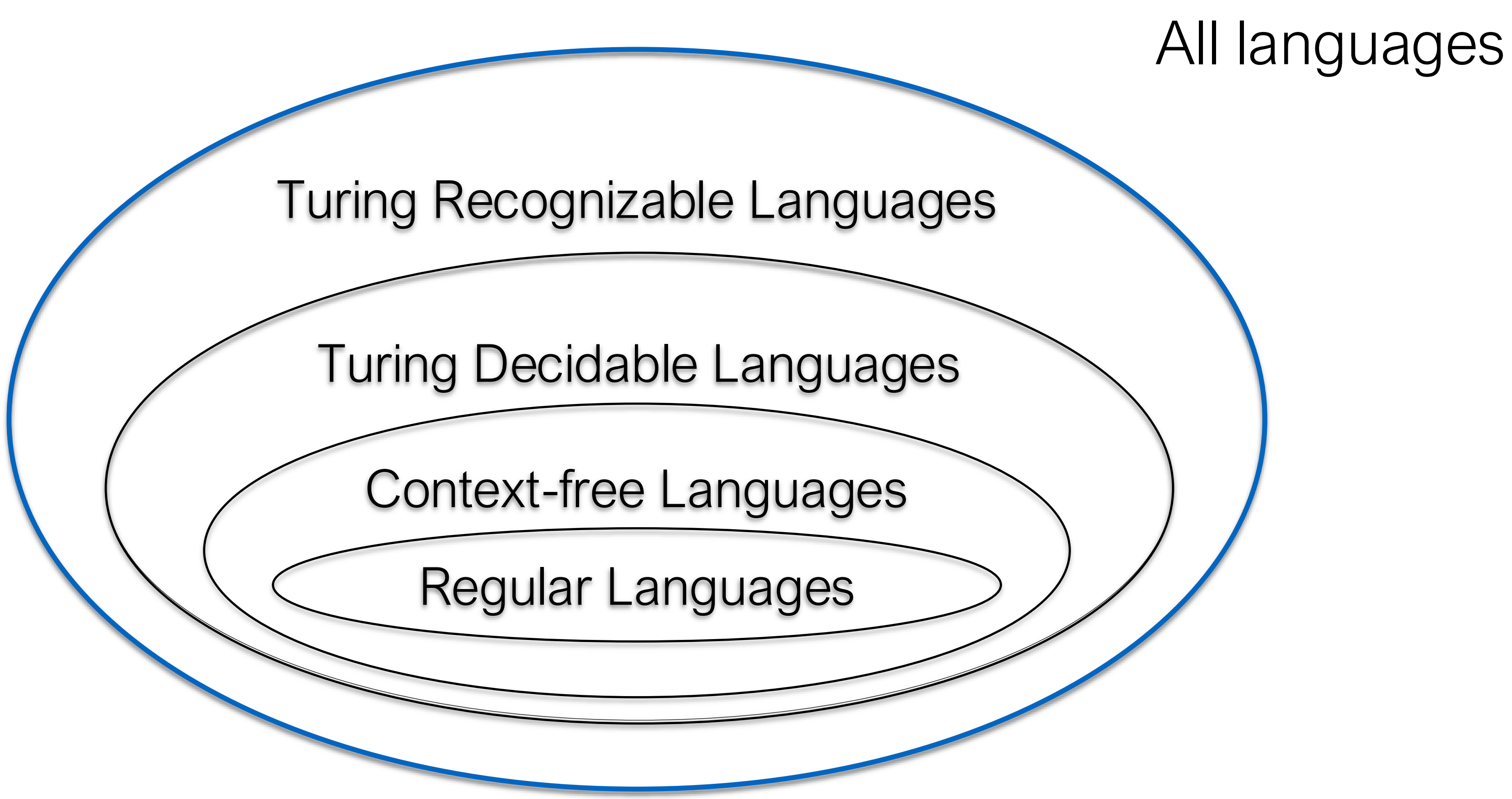
NP Problems & Verifiers of Them

School of Engineering | Computer Science

Stephen Cook & Leonid Levin

- Levin, Russian and Cook, American, independently discovered the existence of NP-complete problems.
- This NP-completeness theorem, often called the Cook–Levin theorem, was a basis for one of the seven Millennium Prize Problems declared by the Clay Mathematics Institute with a $1,000,000 prize offered.
- The Cook–Levin theorem was a breakthrough in computer science and an important step in the development of the theory of computational complexity.

NP - Undecidable



Review of P and NP

- Recall that:
  - P is the class of languages that can be solved efficiently on a deterministic TM
  - NP is the class of languages that can be solved efficiently on a non-deterministic TM
- Defined formally as:
  - P $= \bigcup_k TIME(n^k)$
  - NP $= \bigcup_k NTIME(n^k)$
- Let's look at some NP problems.

NP Problems

- Cook and Levin independently discovered that there are certain problems in NP whose individual complexity is related to that of the entire class.
- If a polynomial-time algorithm exists for any of these problems, all problems in NP would be polynomial-time solvable.
- The NP problem Cook related all NP problems to was 3-SAT
- These problems are NP-Complete problems
- We will now look at this special NP problem and others like it.

NP Problem – 3-SAT

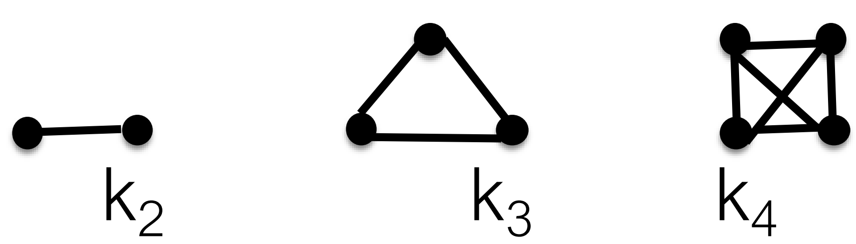
- 3-Satisfiability (3-SAT) Problem:
  - Some Definitions
    - Let $\{x_i\}$, where i runs from 1 to n, be a set of Boolean variables (i.e.: $x_i \in \{0, 1\}$ for each i)
    - Literal: a variable x_i or its negation, $\neg x_i$
    - Clause: $x_1 \lor x_2 \lor x_3 \lor \ldots$ is a set of literals OR-ed together ($\lor$ = OR)
    - Conjunctive Normal Form (CNF): set of clauses c_i connected by ANDs ($C_1 \land C_2$ ...) ($\land$ = AND)
    - 3-CNF Formula: a CNF formula with 3 variables in each clause $((x_1 \lor \neg x_2 \lor x_3) \land (\neg x_3 \lor \neg x_4 \lor x_2))$

NP Problem – 3-SAT

- 3-SAT Problem defined as:
  - Input: Boolean formula ϕ in CNF form (where ϕ: $\{0, 1\}^n \rightarrow \{0,1\}$
  - Output: 1, if there exists an assignment $x \in \{0, 1\}^n$ such that $\phi(x) = 1$ (meaning: there exists an assignment that "satisfies" ϕ), else 0.
  - Ex:
    - Input ϕ: $(x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor x_5) \land (\neg x_3 \lor \neg x_5 \lor x_4)$ if $x_2 = x_4 = 1$ and $x_1 = x_3 = x_5 = 0$
    - Output: $(0 \lor 1 \lor 0) \land (\neg 0 \lor 1 \lor 0) \land (\neg 0 \lor \neg 0 \lor 1) =$ $1 \land (1 \lor 1 \lor 0) \land (1 \lor 1 \lor 1) = 1 \land 1 \land 1 = \underline{1}$ (which satisfies ϕ)

NP Problem – CLIQUE

- CLIQUE defined:
  - Input: Undirected graph G = (V, E) and an integer $k \geq 1$
    - V is the vertex set and E is the edge set
  - Output: Answer the question: Does G contain a clique of size $\geq k$?
  - A clique is a set of k vertices, all of which are connected by an edge



  - Can define CLIQUE as a language: L = {<G, k> | G is an undirected graph with a clique of size $\geq$ k}

Verifier for NP Problems

- 3-SAT and CLIQUE are NP problems and very hard problems to solve. However, verifying a candidate solution is much easier.
- Definition 7.19 (Alternate Characterization of NP):
  - NP is the class of languages that have polynomial time verifiers (set of decision problems whose answer can be efficiently verified using a Turing Machine)

Verifier for NP Problems

- Definition 7.19 Formally: A language $L \in NP$ if there exists polynomials p (proof size) and q (run-time) and a deterministic TM V (verifier) such that:
  - For any input $\in \Sigma^*$:
    1. If $x \in L$, there exists a polynomial-sized "proof" $y \in \Sigma^{p(|x|)}$ such that V accepts $\langle x, y \rangle$
    2. If $x \notin L$, for all polynomial-sized "proofs" $y \in \Sigma^{p(|x|)}$ such that V rejects $\langle x, y \rangle$.
    3. Also, V runs in time $q(|x|)$

Verifier for NP Problems

- Example:
  - An accepting proof for 3-SAT would be a proof satisfying the assignment $x \in \{0,1\}^n$ to ϕ or $\phi(x) = 1$
  - A rejecting proof means that ϕ is unsatisfiable and any candidate assignment evaluates to zero.
- 3-SAT $\in$ NP – this is true because if ϕ is satisfiable, then the proof is a satisfying assignment
- CLIQUE $\in$ NP – this is true because if G has a k-clique, then the proof is the set of vertices in the clique

Verifier for NP Problems

- Definition 7.20: There exists a non-deterministic TM deciding language L if and only if there exists a polynomial-time verifier for L. (Prove the two definitions of NP are equivalent.)
- Proof Sketch:
  - First Part: Build the TM
    - Assume L has a polynomial-time verifier V
    - The following non-deterministic TM decides L:
      1. Non-deterministically "guess" a proof y
      2. Run the verifier V on y, accept if and only if V does

School of Engineering | Computer Science

Verifier for NP Problems

- Definition 7.20: There exists a non-deterministic TM deciding language L if and only if there exists a polynomial-time verifier for L. (Prove the two definitions of NP are equivalent)
- Proof Sketch:
  - Second Part: Build the Verifier
    - Assume there exists a non-deterministic TM deciding L
    - Build a verifier V for L (use it to verify a solution)
      - Because a computational tree of the TM runs in polynomial-time for each branch of the TM, we describe the accepting branch: V simulates the TM on that branch and accepts if and only if the leaf accepts.

Verify CLIQUE is in NP

- Definition 7.20 Example: Show CLIQUE is in NP
- Proof 1 (Show using a non-deterministic TM):
  - N = “On input <G, k> (G is the graph, k is the size of the set of connected vertices (k-clique), C is the string passed in):
    1. Non-deterministically select a subset C of k nodes of G
    2. Test whether G contains all edges connecting nodes in C
    3. If yes, accept; otherwise, reject.”

Verify CLIQUE is in NP

- Definition 7.20 Example: Show CLIQUE is in NP
- Proof 2 (Show using the verifier V for CLIQUE):
  - V = “On input <<G, k>, C> (G is the graph, k is the size of the set of connected vertices (k-clique), and C is the string passed in)
    1. Test if C is a subgraph with k nodes in G
    2. Test if G contains all edges connecting nodes in C
    3. If both pass, accept; otherwise, reject”

Verify SUBSET-SUM is in NP

- Use the definition of a polynomial-time verifier to show SUBSET-SUM is in NP
  - SUBSET-SUM = {<S, t> | S = {x_1, …, x_k} and for some {y_1, ..., y_i} ⊆ {x_1, …, x_k}, we have $\sum y_i$ = t
  - Ex: ({4, 11, 16, 21, 27}, 25) ∈ SUBSET-SUM because 4 + 21 = 25
  - The sets {x_1, …, x_k} and {y_1, ..., y_i} are multisets and allow repetition of elements.

Verify SUBSET-SUM is in NP

- Use the definition of a polynomial-time verifier to show SUBSET-SUM is in NP
  - SUBSET-SUM = {<S, t> | S = {x_1, …, x_k} and for some {y_1, ..., y_i} ⊆ {x_1, …, x_k}, we have $\sum y_i$ = t
  - Ex: ({4, 11, 16, 21, 27}, 25) ∊ SUBSET-SUM because 4 + 21 = 25
  - Solution:
    - Non-deterministic TM N = “On input <S, t>
      1. Non-deterministically select a subset C from the set S.
      2. Test whether the elements of C add to t.
      3. If yes, accept; otherwise reject.”

Verify SUBSET-SUM is in NP

- Use the definition of a polynomial-time verifier to show SUBSET-SUM is in NP
  - SUBSET-SUM = {<S, t> | S = {x_1, …, x_k} and for some {y_1, ..., y_i} ⊆ {x_1, …, x_k}, we have $\sum y_i$ = t
  - Ex: ({4, 11, 16, 21, 27}, 25) ∈ SUBSET-SUM because 4 + 21 = 25
  - Solution:
    - Verifier V = "On input <<S, t>, C> (C is input string)
      1. Test whether C is a collection of numbers that sum to t.
      2. Test whether S contains all the numbers in C.
      3. If both pass, accept; otherwise reject."

Try It

- Use the definition of a polynomial-time verifier to show PACKING is in NP
  - PACKING = {<S, L, H> | S = $\{x_1, \ldots, x_k\}$ and for some $\{y_1, \ldots, y_i\} \subseteq \{x_1, \ldots, x_k\}$, we have $L \leq \sum y_i \leq H$
  - Ex: ({5, 8, 16, 4, 27}, 20, 25) ∈ PACKING because $20 \leq 16 + 5 \leq 25$

VCU School of Engineering | Computer Science

Try It

- Use the definition of a polynomial-time verifier to show PACKING is in NP
  - PACKING = {<S, L, H> | S = {x_1, …, x_k} and for some {y_1, ..., y_i} ⊆ {x_1, …, x_k}, we have L ≤ $\sum y_i$ ≤ H
  - Ex: ({5, 8, 16, 4, 27}, 20, 25) ∊ PACKING because 20 ≤ 16 + 5 ≤ 25
  - Solution:
    - Non-deterministic TM N = "On input <S, t>
      1. Non-deterministically select a subset C from the set S.
      2. Test whether the elements of C sum to greater than or equal to L, but less than or equal to H.
      3. If yes, accept; otherwise reject."

Try It

- Use the definition of a polynomial-time verifier to show PACKING is in NP
  - PACKING = {<S, L, H> | S = {x_1, …, x_k} and for some {y_1, ..., y_i} ⊆ {x_1, …, x_k}, we have L ≤ $\sum y_i$ ≤ H
  - Ex: ({5, 8, 16, 4, 27}, 20, 25) ∈ PACKING because 20 ≤ 16 + 5 ≤ 25
  - Solution:
    - Verifier V = "On input <<S, L, H>, C> (C is input string)
      1. Test whether C is a collection of numbers that sum to greater than or equal to L, but less than or equal to H.
      2. Test whether S contains all the numbers in C.
      3. If both pass, accept; otherwise reject."