Theory of Formal Languages and Automata

Lecture 18

Mahdi Dolati

Sharif University of Technology

Fall 2023

December 1, 2023

Definition of Algorithm

- Algorithm: A collection of simple instructions for carrying out some task.
  - Also called procedures or recipes.

- Ancient examples:
  - Algorithm for finding prime numbers,
  - Algorithm for finding greatest common divisors.

- Despite its long history, the notion of algorithm itself was not defined precisely until the twentieth century.
  - Why do we need a formal description?

Hilbert's Problems

Background

- A **polynomial** is a sum of terms, where each term is product of certain variables and a constant, called a coefficient:
  - Example of a term with coefficient 6:

    $$6 \cdot x \cdot x \cdot x \cdot y \cdot z \cdot z = 6x^3yz^2$$

  - Example of a polynomial over the variables x, y, and z:

    $$6x^3yz^2 + 3xy^2 - x^3 - 10$$

- **Root** of a polynomial: An assignment of variables so that the value of the polynomial is zero.
  - For example, x=5, y=3, and z=0 is a root of the previous polynomial.
- A root is **integral** if all the variables are integers,
  - Some polynomials have an integral root and some do not.

Hilbert's Problems

- David Hilbert at International Congress of Mathematicians in Paris, 1900:
  - Presented 23 problems as a challenge for the 20th century.

- The 10th Hilbert problem:
  - Devise *a process according to which it can be determined by a finite number of operations* (=algorithm) that tests whether a polynomial has an integral root.

- Hilbert apparently assumed that such an algorithm must exist—someone need only find it.
  - We now know, no algorithm exists for this task.
  - It is impossible to get this result with an intuitive concept of algorithm.

Hilbert’s Problems

- Definitions of algorithm (they are equivalent):
  - Year 1936,
  - Alonzo Church: With λ-calculus,
  - Alan Turing: With Turing machines.

- Relation between the informal and formal definitions is called the **Church Turing thesis**:

Intuitive notion of algorithms	equals	*Turing machine algorithms*

Church Turing Thesis

- There has never been a proof, but the evidence for its validity comes from the fact that every realistic model of computation, yet discovered, has been shown to be equivalent.
- If there were a device which could answer questions beyond those that a Turing machine can answer, then it would be called an **oracle**.

Hilbert's Problems

- Hilbert's 10th problem in our terminology: Is the set D decidable?

$$D = \{p \mid p \text{ is a polynomial with an integral root }\}$$

- No. D is not decidable but Turing-recognizable.

- **Example**: Show single variable case is Turing-recognizable:

$$D_1 = \{p \mid p \text{ is a polynomial over } x \text{ with an integral root}\}.$$

- Construct a TM M_1 that recognizes D_1:

M_1 = "On input $\langle p \rangle$: where p is a polynomial over the variable x.

1. Evaluate p with x set successively to the values $0, 1, -1, 2, -2, 3, -3, \ldots$. If at any point the polynomial evaluates to 0, *accept*."

Hilbert's Problems

- Hilbert's 10th problem in our terminology: Is the set D decidable?

$$D = \{p \mid p \text{ is a polynomial with an integral root }\}$$

- No. D is not decidable but Turing-recognizable.

- **Example**: Show multivariable case is Turing-recognizable:
  - Similar to single variable case,
  - Build a TM M that goes through all possible settings of the variables.

Hilbert's Problems

- Hilbert's 10th problem in our terminology: Is the set D decidable?

$$D = \{p \mid p \text{ is a polynomial with an integral root }\}$$

- No. D is not decidable but Turing-recognizable.

- **Example**: Can we convert M_1 to be a decider?
  - Yes. We can restrict the search, as root of single variable polynomials lie between the values:

$$\pm k \frac{c_{\max}}{c_1}$$

  - k is the number of terms,
  - $c_{\max}$ is the coefficient with the largest absolute value,
  - c_1 is the coefficient of the highest order term.

Hilbert's Problems

- Hilbert's 10th problem in our terminology: Is the set D decidable?

$$D = \{p \mid p \text{ is a polynomial with an integral root}\}$$

- No. D is not decidable but Turing-recognizable.

- **Example**: Can we convert M to be a decider?
  - No. Matijasevic's theorem shows that it is not possible to find a bound similar to the single variable case here.

TM Description Levels

- Formal description,
- Implementation description,
  - The way that the head moves,
  - The way that content is stored on the tape.
- High-level description,
  - Describe an algorithm.

TM Description Levels

- Input is always a string:
  - We can represent any object as a string.
  - Examples:
    - Polynomials,
    - Graphs,
    - Grammars,
    - Automata,
    - Any combination of above,
    - …
  - The TM decodes the representation.
    - Tests validity of encoding and rejects if it is not valid.

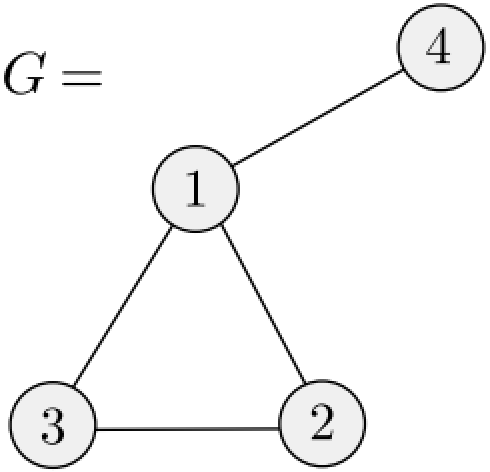
- Use $\langle O \rangle$ to show the encoding of an object O.
- Use $\langle O_1, O_2, \dots, O_k \rangle$ for several objects.

- **Example**: Undirected graphs that are connected:

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}.$$

- A graph and its encoding:

$G =$



$\langle G \rangle =$

`(1,2,3,4)((1,2),(2,3),(3,1),(1,4))`

TM Description Levels

Encoding

- **Example**: Undirected graphs that are connected:

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}.$$

- High-level description of a TM M that decides A:

M = "On input $\langle G \rangle$, the encoding of a graph G:

1. Select the first node of G and mark it.
2. Repeat the following stage until no new nodes are marked:
3. For each node in G, mark it if it is attached by an edge to a node that is already marked.
4. Scan all the nodes of G to determine whether they all are marked. If they are, *accept*; otherwise, *reject*."

Decidability

- Limits of algorithmic solvability: We demonstrate certain problems that can be solved algorithmically and others that cannot.
  - You know some problems must be simplified or altered before you can find an algorithmic solution.

Decidable Languages

- Certain problems of this kind are related to applications.
  - Problem of testing whether a CFG generates a string is related to the problem of recognizing and compiling programs in a programming language.

- Examples of decidability helps you to appreciate the undecidable examples.

Decidable Languages

Regular Languages

- Algorithms for:
  - Whether a finite automaton accepts a string,
  - whether the language of a finite automaton is empty, and
  - whether two finite automata are equivalent.

- Represent **computational problems** by **languages**.
  - We have set up terminology dealing with languages.

- **The acceptance problem for DFAs**: Testing whether a particular deterministic finite automaton accepts a given string expressed as a language:

$$A_{\mathsf{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}.$$

- Test whether $\langle B, w\rangle \in L(A_{DFA})$.

- Language is decidable ↔ Computational problem is decidable.

Theorem
A_{DFA} is a decidable language.

- Proof idea: present a TM M that decides A_{DFA}.

  M = "On input $\langle B, w\rangle$, where B is a DFA and w is a string:
  1. Simulate B on input w.
  2. If the simulation ends in an accept state, *accept*. If it ends in a nonaccepting state, *reject*."
- Proof: A few implementation details to carry out the simulation:
  - Representation of a DFA with its five components.
  - Start from q0, read one symbol from the input, change the current state based on the transition function.
  - When finished the input, check whether the state is final.

- The acceptance problem for NFAs:

$$A_{\mathsf{NFA}} = \{\langle B, w\rangle \mid B \text{ is an NFA that accepts input string } w\}.$$

Theorem
A_{NFA} is a decidable language.

- Proof: We present a TM N that decides A_{NFA}.
  - A new idea: Convert the NFA to a DFA:

$N =$ "On input $\langle B, w\rangle$, where B is an NFA and w is a string:

1. Convert NFA B to an equivalent DFA C, using the procedure for this conversion given in Theorem 1.39.
2. Run TM M from Theorem 4.1 on input $\langle C, w\rangle$.
3. If M accepts, *accept*; otherwise, *reject*."

We know how to convert NFAs to DFAs.

Decidable Languages
Regular Languages

- The acceptance problem for regular expressions:

$$A_{\mathsf{REX}} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}.$$

Theorem
A_{REX} is a decidable language.

- Proof: We present a TM P that decides A_{REX}.

$P =$ "On input $\langle R, w\rangle$, where R is a regular expression and w is a string:

1. Convert regular expression R to an equivalent NFA A by using the procedure for this conversion given in Theorem 1.54.
2. Run TM N on input $\langle A, w\rangle$.
3. If N accepts, *accept*; if N rejects, *reject*."

- The emptiness testing for regular languages:

$$E_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}.$$

Theorem
E_{DFA} is a decidable language.

- Proof: Reaching an accept state from the start state:

T = "On input $\langle A\rangle$, where A is a DFA:

1. Mark the start state of A.
2. Repeat until no new states get marked:
3. Mark any state that has a transition coming into it from any state that is already marked.
4. If no accept state is marked, *accept*; otherwise, *reject*."

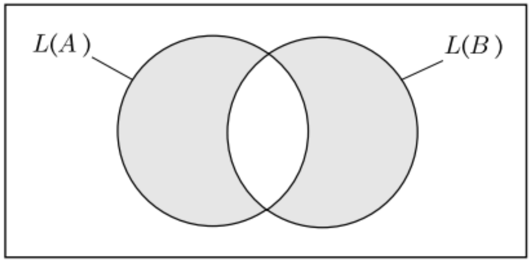
- The equivalency problem for DFAs:

$$EQ_{\text{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}.$$

Theorem

EQ_{DFA} is a decidable language.

- Proof: Construct a new DFA C that accepts the symmetric difference of L(A) and L(B):



$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right)$$

F = "On input $\langle A, B\rangle$, where A and B are DFAs:

1. Construct DFA C as described.
2. Run TM T from Theorem 4.4 on input $\langle C\rangle$.
3. If T accepts, *accept*. If T rejects, *reject*."

Test emptiness.

- The acceptance problem for CFGs:

$$A_{\mathsf{CFG}} = \{\langle G, w\rangle |\ G \text{ is a CFG that generates string } w\}.$$

Theorem
A_{CFG} is a decidable language.

- Proof Idea 1 (does not work):
  - Go through all derivations to determine whether any is a derivation of w,
  - gives a Turing machine that is a recognizer, but not a decider.

- The acceptance problem for CFGs:

$$A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}.$$

Theorem
A_{CFG} is a decidable language.

- Proof Idea 2:
  - If G is in CNF, any derivation of w has 2n − 1 steps, where n is the length of w,
  - Checking only derivations with 2n − 1 steps to determine whether G generates w would be sufficient.

- The acceptance problem for CFGs:

$$A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}.$$

Theorem
A_{CFG} is a decidable language.

- Proof:

$S =$ "On input $\langle G, w\rangle$, where G is a CFG and w is a string:

1. Convert G to an equivalent grammar in Chomsky normal form.

2. List all derivations with $2n-1$ steps, where n is the length of w; except if $n = 0$, then instead list all derivations with one step.

3. If any of these derivations generate w, *accept*; if not, *reject*."

Decidable Languages

Context-Free Languages

- The emptiness problem for CFLs:

$$E_{\text{CFG}} = \{\langle G\rangle |\ G \text{ is a CFG and } L(G) = \emptyset\}.$$

Theorem
E_{CFG} is a decidable language.

- Proof Idea 1 (does not work):
  - Going through all possible w’s, one by one.
  - There are infinitely many w’s.

- The emptiness problem for CFLs:

$$E_{\mathsf{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}.$$

Theorem
E_{CFG} is a decidable language.

- Proof: Keep track whether each variable is capable of generating a string of terminals:

$R =$ "On input $\langle G\rangle$, where G is a CFG:

1. Mark all terminal symbols in G.

2. Repeat until no new variables get marked:

3. Mark any variable A where G has a rule $A \to U_1U_2\cdots U_k$ and each symbol $U_1, \ldots, U_k$ has already been marked.

4. If the start variable is not marked, *accept*; otherwise, *reject*."

- The equivalency problem for CFGs:

$$EQ_{\mathsf{CFG}} = \{\langle G, H\rangle |\ G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}.$$

Theorem
EQ_{CFG} is NOT a decidable language.

- Proof: We prove this in later (Chapter 5).

Theorem
Every context-free language is decidable.

- Let A be a CFL. Our objective is to show that A is decidable.
- Proof idea 1 (does not work): Simulate the PDA of the language with a TM:
  - TM is powerful enough to simulate a stack with its tape,
  - However, some branches of the PDA's computation may go on forever, reading and writing the stack without ever halting.
  - The TM would not be a decider.

Theorem
Every context-free language is decidable.

- Proof: Let G be a CFG for A and design a TM M_G that decides A. We build a copy of G into M_G. It works as follows.

M_G = "On input w:

1. Run TM S on input $\langle G, w \rangle$.
2. If this machine accepts, *accept*; if it rejects, *reject*."

- The relationship among classes of languages: