Theory of Computation (VI)

Yijia Chen
Fudan University

Review

Turing Recognizable

The collection of strings that M accepts is the language of M, or the language recognized by M, denoted $L(M)$.

Definition

A language is Turing-recognizable, if some Turing machine recognizes it.

On an input, the machine M may accept, reject, or loop. By loop we mean that the machine simply does not halt.

If M always halts, then it is a decider. A decider that recognizes some language is said to decide that language.

Definition

A language is Turing-decidable or simply decidable if some Turing machine decides it.

Variants of Turing Machines

Multitape Turing Machines

A multitape Turing machine M has several tapes:

1. Each tape has its own head for reading and writing.
2. The input is initially on tape 1, with all the other tapes being blank.
3. The transition function is

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k,$$

where k is the number of tapes.

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$$

means that if M is in state q_i and heads 1 through k are reading symbols a_1 through a_k, the machine goes to state q_j, writes symbols b_1 through b_k, and directs each head to move left or right, or to stay put, as specified.

Theorem

Every multitape Turing machine has an equivalent single-tape Turing machine.

Corollary

A language is Turing-recognizable if and only if some multitape Turing machine recognizes it.

Nondeterministic Turing Machines

1. The transition function for a nondeterministic Turing machine has the form

$$\delta : Q \times \Gamma \to \mathcal{P}\big(Q \times \{L, R\}\big).$$

2. The computation of a nondeterministic Turing machine is a tree whose branches correspond to different possibilities for the machine.
3. If some branch of the computation leads to the accept state, the machine accepts its input.

Theorem

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Corollary

A language is Turing-recognizable if and only if some nondeterministic Turing machine recognizes it.

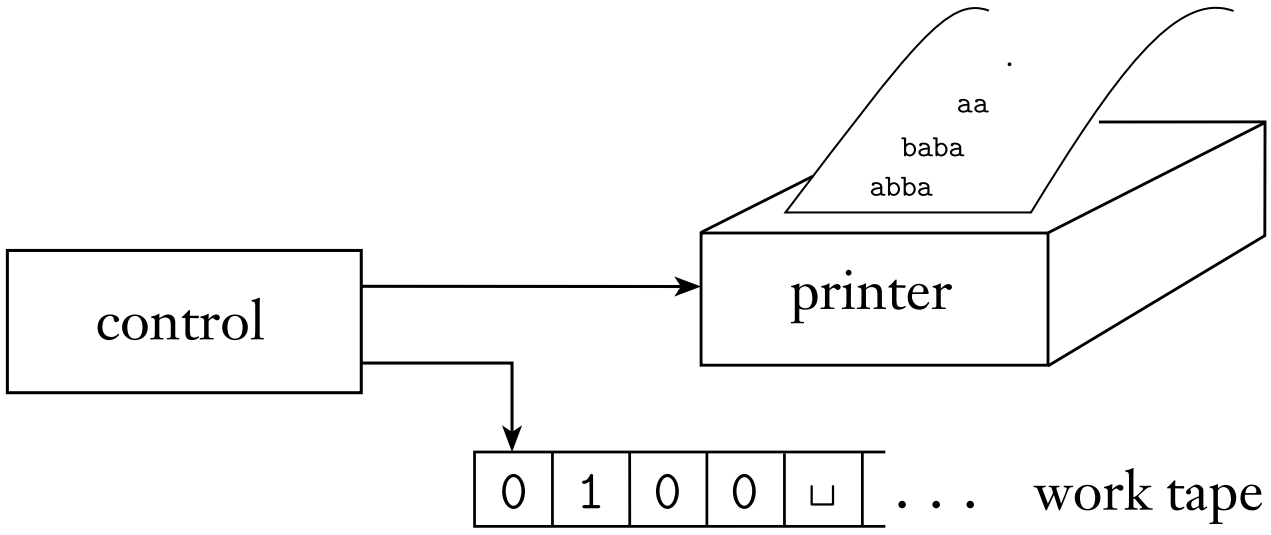
Corollary

A language is decidable if and only if some nondeterministic Turing machine decides it.

Enumerators

1. An enumerator is a Turing machine with an attached printer.
2. The Turing machine can use that printer as an output device to print strings.
3. Every time the Turing machine wants to add a string to the list, it sends the string to the printer.

Schematic of an enumerator



Theorem

A language is Turing-recognizable if and only if some enumerator enumerates it.

Proof (1)

Let E be an enumerator E that enumerates a language A. The desired M on input w:

1. Run E. Every time that E outputs a string, compare it with w.
2. If w ever appears in the output of E, then accept.

Proof (2)

If M recognizes a language A, we can construct the following enumerator E for A. Let $s_1, s_2, s_3, \ldots$, be a list of all possible strings in Σ^*.

1. Repeat the following for $i = 1, 2, 3, \ldots$
2. Run M for i steps on each input, s_1, s_2, $\ldots$, s_i.
3. If any computations accept, print out the corresponding s_j.

The Definition of Algorithm

Polynomials and their roots

A polynomial is a sum of terms, where each term is a product of certain variables and a constant, i.e., coefficient. For example,

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

is a term with coefficient 6, and

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

is a polynomial with four terms, over the variables x, y, and z.

A root of a polynomial is an assignment of values to its variables so that the value of the polynomial is 0. This root is an integral root because all the variables are assigned integer values. Some polynomials have an integral root and some do not.

Hilbert's Problems

Hilbert's tenth problem was to devise an algorithm that tests whether a polynomial has an integral root. He did not use the term algorithm but rather

> *a process according to which it can be determined by a finite number of operations*.

Church-Turing Thesis

In 1936 to formalize the definition of an algorithm:

1. Alonzo Church proposed λ-calculus;
2. Alan Turing proposed Turing machines,

which were shown to be equivalent.

So we have the Church-Turing Thesis:

$$\text{Intuitive notion of algorithms} \quad = \quad \text{Turing machine algorithms.}$$

Hilbert's Tenth Problem

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

Theorem (Yuri Matijasevič, Martin Davis, Hilary Putnam, and Julia Robinson, 1970)

D is not decidable.

A simple variant

$$D_1 = \{p \mid p \text{ is a polynomial on a single variable } x \text{ with integer coefficients and with an integral root}\}.$$

Lemma
Both D and D_1 are Turing-recognizable.

Proof.
On input $p(x)$

> *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, then accept.*

□

A simple variant (con'd)

Lemma

Let

$$p(x) = c_1 x^n + c_2 x^{n-1} + \cdots + c_n x + c_{n+1}$$

with $c_1 \neq 0$ *and* $p(x_0) = 0$. *Define*

$$c_{\max} = \max \left\{ |c_i| \right\}_{i \in [n+1]}.$$

Then

$$|x_0| < \frac{c_{\max} \cdot (n+1)}{|c_1|}.$$

Corollary

D_1 *is decidable.*

Decidability

Decidable Languages

Decidable problems concerning regular languages

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

That is, for every $w \in \Sigma^*$ and DFA B,

$$w \in L(B) \iff \langle B, x\rangle \in A_{\text{DFA}}.$$

Theorem

A_{DFA} is a decidable language.

Proof (1)

M on $\langle B, w \rangle$:

1. Simulate B on input w.
2. If the simulation ends in an accepting state, then accept. If it ends in a nonaccepting state, then reject.

Proof (2)

Some implementation details:

- The representation of B is a list of Q, Σ, δ, q_0, and F.
- When M receives its input, M first determines whether it properly represents a DFA B and a string w. If not, M rejects.
- Then M carries out the simulation directly.
  1. It keeps track of B's current state and position in w by writing this information down on its tape.
  2. Initially, B's current state is q_0 and current input position is the leftmost symbol of w.
  3. The states and position are updated according to the specified transition function δ.
  4. When M finishes processing the last symbol of w, M accepts the input if B is in an accepting state; M rejects the input if B is in a nonaccepting state.

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

That is, for every $w \in \Sigma^*$ and NFA B,

$$w \in L(B) \iff \langle B, x\rangle \in A_{\text{NFA}}.$$

Theorem

A_{NFA} is a decidable language.

Proof (1)

The simplest proof is to simulate an NFA using nondeterministic Turing machine, as we used the (deterministic) Turing machine M to simulate a DFA.

Instead we design a (deterministic) Turing machine N which uses M as a subroutine.

Proof (2)

N on $\langle B, w \rangle$:

1. Convert NFA B to an equivalent DFA C using the subset construction.
2. Run TM M from the previous Theorem on input $\langle C, w \rangle$.
3. If M accepts, then accept; otherwise reject.

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

Theorem

A_{REX} is a decidable language.

Proof

P on $\langle R, w \rangle$:

1. Convert R to an equivalent NFA A.
2. Run TM N from the previous Theorem on input $\langle A, w \rangle$.
3. If N accepts, then accept; otherwise reject.

Testing the emptiness

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

Theorem

E_{DFA} is a decidable language.

Proof

A DFA accepts some string if and only if reaching an accept state from the start state by traveling along the arrows of the DFA is possible.

T on $\langle A \rangle$:

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, then accept; otherwise, reject.

Testing equality

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

Theorem

EQ_{DFA} is a decidable language.

Proof (1)

From A and B we construct a DFA C such that

$$L(C) = \Big(L(A) \cap \overline{L(B)}\Big) \cup \Big(\overline{L(A)} \cap L(B)\Big),$$

i.e., the symmetric difference between $L(A)$ and $L(B)$.

Then

$$L(A) = L(B) \quad \Longleftrightarrow \quad L(C) = \emptyset.$$

Proof (2)

F on $\langle A, B \rangle$:

1. Construct DFA C from A and B.
2. Run TM T from the previous Theorem on input $\langle C \rangle$.
3. If T accepts, then accept; otherwise reject.

Decidable problems concerning context-free languages

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

Theorem

A_{CFG} is a decidable language.

Proof (1)

For CFG G and string w, we want to determine whether G generates w.

One idea is to use G to go through all derivations to determine whether any is a derivation of w. Then if G does not generate w, this algorithm would never halt. It gives a Turing machine that is a recognizer, but not a decider.

Recall:

Definition

A context-free grammar is in Chomsky normal form if every rule is of the form

$$A \to BC \quad \text{and} \quad A \to a$$

where a is a terminal, and A, B, and C are variables – except that B and C may be not the start variable. In addition, we permit the rule $S \to \varepsilon$, where S is the start variable.

Theorem

Any context-free language is generated by a context-free grammar in Chomsky normal form.

Theorem

Let G be CFG in Chomsky normal form, and G generates w with $w \neq \varepsilon$. Then any derivation of w has $2|w| - 1$ steps.

Proof (2)

S on $\langle G, w \rangle$:

1. Convert G to an equivalent grammar in Chomsky normal form.
2. List all derivations with $2|w| - 1$ steps; except if $|w| = 0$, then instead check whether there is a rule $S \to \varepsilon$.
3. If any of these derivations generates w, then accept; otherwise reject.

Testing the emptiness

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

Theorem

E_{CFG} is a decidable language.

Proof (1)

To determine whether $L(G) = \emptyset$, the algorithm might try going through all possible w's, one by one. But there are infinitely many w's to try, so this method could end up running forever.

Instead, the algorithm solves a more general problem: determine for each variable whether that variable is capable of generating a string of terminals.

- First, the algorithm marks all the terminal symbols in the grammar.
- It scans all the rules of the grammar. If it finds a rule that permits some variable to be replaced by some string of symbols, all of which are already marked, then it marks this variable.

Proof (2)

R on $\langle G \rangle$:

1. Mark all terminal symbols in G.
2. Repeat until no new variables get marked:
3. Mark any variable A where G contains a rule $A \to U_1 \cdots U_k$ and all U_i's have already been marked.
4. If the start variable is not marked, then accept; otherwise, reject.

Testing equality

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

Theorem

EQ_{CFG} is not decidable.

Theorem

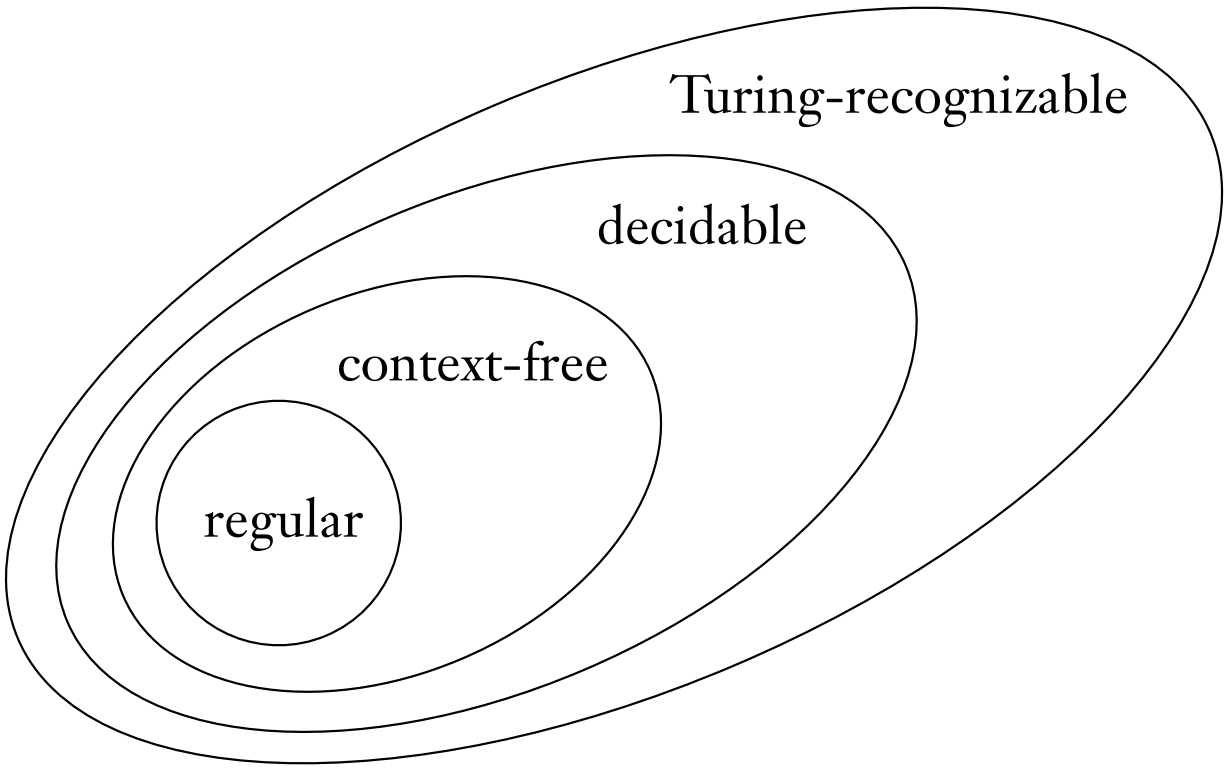
Every context-free language is decidable.

Recall using Chomsky normal form, we have shown:

Theorem

$A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates } w\}$ *is a decidable language.*

Relationship among classes of languages



Undecidability

Testing membership

$$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}.$$

Theorem

A_{TM} is not decidable.

Theorem
A_{TM} is Turing-recognizable.

Proof.
U on $\langle M, w \rangle$:

1. Simulate M on w.
2. If M enters its accept state, then accept; if it enters its reject state, reject.

□

U is a universal Turing machine first proposed by Alan Turing in 1936. This machine is called universal because it is capable of simulating any other Turing machine from the description of that machine.