Mathematical Foundation of Computer Sciences IV

Decidability and Undecidability

Guoqiang Li

School of Software, Shanghai Jiao Tong University

Decidability on Regular Languages

Decidable problems concerning regular languages (1)

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

That is, for every $w \in \Sigma^*$ and DFA B, $w \in L(B) \iff \langle B, w\rangle \in A_{DFA}$

Decidable problems concerning regular languages (1)

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

That is, for every $w \in \Sigma^*$ and DFA B, $w \in L(B) \iff \langle B, w\rangle \in A_{DFA}$

Theorem

A_{DFA} is a decidable language.

Proof (1)

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

Proof (1)

M on $\langle B, w \rangle$: 此处的M应该为一个Turing

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:

Proof (2)

Some implementation details:

- The representation of B is a list of Q, Σ, δ, q_0, and F.

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.

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.

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.

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.

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.

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.

Decidable problems concerning regular languages (2)

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

That is, for every $w \in \Sigma^*$ and NFA B, $w \in L(B) \iff \langle B, w\rangle \in A_{NFA}$

Decidable problems concerning regular languages (2)

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

That is, for every $w \in \Sigma^*$ and NFA B, $w \in L(B) \iff \langle B, w\rangle \in A_{NFA}$

Theorem

A_{NFA} is a decidable language.

Proof (1)

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 $\langle B, w \rangle$:

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.

Decidable problems concerning regular languages (3)

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

Theorem

A_{REX} *is a decidable language.*

Proof

上述的三个问题是利用了之前讲过的DFA NFA REX之间的等价关系进行问题的等价证明

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

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

Testing the emptiness

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

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.

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 $\langle A \rangle$:

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

Testing equality

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

Testing equality

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

Theorem

EQ_{DFA} *is a decidable language.*

Proof (1)

From A and B we construct a DFA C such that

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

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

$$L(A) = L(B) \iff L(C) = \emptyset$$

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.

Decidability on Context-Free Languages

Decidable problems concerning context-free languages

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

Theorem

A_{CFG} is a decidable language.

Proof

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: Chomsky Normal Form

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

$$\begin{aligned} A &\rightarrow BC \\ A &\rightarrow a \end{aligned}$$

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

Theorem

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

Theorem

If G is a context-free grammar in Chomsky normal form then any $w \in L(G)$ such that $w \neq \varepsilon$ can be derived from the start state in exactly $2|w| - 1$ steps.

Proof

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 $S \to \epsilon$.
3. If any of these derivations generates w, then accept; otherwise reject.

Testing the emptiness

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

Testing the emptiness

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

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 $\langle G \rangle$:

- Mark all terminal symbols in R.

CFG的空集合的证明是使用回溯，即从终止符开始往回走，
(回溯法在确定图论中的最短路径时也常用)，而DFA的
空集合的证明是迭代，即从start开始去看能否到达F states

- Repeat until no new variables get marked:
- Mark any variable A where G contains a rule $A \rightarrow U_1 \ldots U_k$ and all U_i's have already been marked.
- If the start variable is not marked, then accept; otherwise, reject.

Testing equality

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

Testing equality

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

Theorem

EQ_{CFG} *is a* **not** *decidable language*.

CFG对于交集与补集运算不封闭

Inclusion result

Theorem

Every context-free language is decidable.

Recall using Chomsky normal form, we have shown:

Theorem

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

is a decidable language.

Undecidability

Testing equality between context-free languages

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

Theorem

*EQ_{CFG} is a **not** decidable language.*

Testing membership of Turing recognized languages

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

Testing membership of Turing recognized languages

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

Theorem

*A_{TM} is **not** decidable.*

Turing recognizable

Theorem

A_{TM} is Turing-recognizable.

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.

Turing recognizable

Theorem

A_{TM} is Turing-recognizable.

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.

The Diagonalization Method

Functions

Definition

Let $f : A \to B$ be a function.

- f is one-to-one if $f(a) \neq f(a')$ whenever $a \neq a'$.
- f is onto if for every $b \in B$ there is an $a \in A$ with $f(a) = b$.

A and B are the same size if there is a one-to-one, onto function $d : A \to B$.

A function that is both one-to-one and onto is a correspondence.

injective	one-to-one
surjective	onto
bijective	one-to-one and onto

Cantor's Theorem

Definition

A is countable if it is either finite or has the same size as $\mathbb{N}$.

Cantor's Theorem

Definition

A is countable if it is either finite or has the same size as $\mathbb{N}$.

Theorem

$\mathbb{R}$ *is not countable.*

Cantor's Theorem

Corollary

Some languages are not Turing-recognizable.

Proof

We fix an alphabet Σ.

Proof

We fix an alphabet Σ.

- Σ^* is countable.

Proof

We fix an alphabet Σ.

- Σ^* is countable.
- The set of all TMs is countable, as every M can be identified with a string $\langle M \rangle$.

Proof

We fix an alphabet Σ.

- Σ^* is countable.
- The set of all TMs is countable, as every M can be identified with a string $\langle M \rangle$.
- The set of all languages over Σ is uncountable.

An undecidable language

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

An undecidable language

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

Theorem

A_{TM} *is undecidable.*

Proof (1)

Assume H is a decider for A_{TM}. That is

$$H(\langle M, w \rangle) = \begin{cases} \text{accept} & \text{if } M \text{ accepts } w \\ \text{reject} & \text{if } M \text{ does not accept} \end{cases}$$

下面即证明H不存在

Proof (2)

D on $\langle M \rangle$, where M is a TM:

Proof (2)

D on $\langle M \rangle$, where M is a TM:

1. Run H on input $\langle M, \langle M \rangle \rangle$.

Proof (2)

D on $\langle M \rangle$, where M is a TM:

1. Run H on input $\langle M, \langle M \rangle \rangle$.
2. Output the opposite of what H outputs. That is, if H accepts, then reject; and if H rejects, then accept.

Proof (2)

D on $\langle M \rangle$, where M is a TM:

1. Run H on input $\langle M, \langle M \rangle \rangle$.
2. Output the opposite of what H outputs. That is, if H accepts, then reject; and if H rejects, then accept.

$$D(\langle M \rangle) = \begin{cases} \text{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \text{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

Proof (2)

D on $\langle M \rangle$, where M is a TM:

1. Run H on input $\langle M, \langle M \rangle \rangle$.
2. Output the opposite of what H outputs. That is, if H accepts, then reject; and if H rejects, then accept.

$$D(\langle M \rangle) = \begin{cases} \text{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \text{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

$$D(\langle D \rangle) = \begin{cases} \text{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\ \text{reject} & \text{if } D \text{ accepts } \langle D \rangle \end{cases}$$

罗素悖论

Proof (3)

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\dots$
M_1	accept		accept		
M_2	accept	accept	accept	accept	
M_3					$\dots$
M_4	accept	accept			
$\vdots$			$\vdots$		

Entry i, j is accept if M_i accepts $\langle M_j \rangle$.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\dots$
M_1	accept	reject	accept	reject	
M_2	accept	accept	accept	accept	
M_3	reject	reject	reject	reject	$\dots$
M_4	accept	accept	reject	reject	
$\vdots$			$\vdots$		

Entry i, j is the value of H on input $M_i, \langle M_j \rangle$

Proof (4)

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\ldots$	$\langle D \rangle$	$\ldots$
M_1	accept	reject	accept	reject		accept	
M_2	accept	accept	accept	accept	$\ldots$	accept	
M_3	reject	reject	reject	reject		reject	$\ldots$
M_4	accept	accept	reject	reject		accept	
$\vdots$			$\vdots$			$\vdots$	
D	reject	reject	accept	accept		**?**	
$\vdots$			$\vdots$				

If D is in the figure, then a contradiction occurs at **?**

co-Turing-recognizable

Definition

A language is co-Turing-recognizable if it is the complement of a Turing-recognizable language.

co-Turing-recognizable

Definition

A language is co-Turing-recognizable if it is the complement of a Turing-recognizable language.

Theorem

A language is decidable if and only if it is Turing recognizable and co-Turing-cognizable.

Proof

If A is decidable, then both A and $\overline{A}$ are Turing-recognizable: Any decidable language is Turing-recognizable, and the complement of a decidable language also is decidable.

Proof

If A is decidable, then both A and $\overline{A}$ are Turing-recognizable: Any decidable language is Turing-recognizable, and the complement of a decidable language also is decidable.

Assume both A and $\overline{A}$ are Turing recognizable by M_1 and M_2 respectively.

Proof

If A is decidable, then both A and $\overline{A}$ are Turing-recognizable: Any decidable language is Turing-recognizable, and the complement of a decidable language also is decidable.

Assume both A and $\overline{A}$ are Turing recognizable by M_1 and M_2 respectively.

The TM M on input w:

Proof

If A is decidable, then both A and $\overline{A}$ are Turing-recognizable: Any decidable language is Turing-recognizable, and the complement of a decidable language also is decidable.

Assume both A and $\overline{A}$ are Turing recognizable by M_1 and M_2 respectively.

The TM M on input w:

1. Run M_1 and M_2 on input w in parallel.

Proof

If A is decidable, then both A and $\overline{A}$ are Turing-recognizable: Any decidable language is Turing-recognizable, and the complement of a decidable language also is decidable.

Assume both A and $\overline{A}$ are Turing recognizable by M_1 and M_2 respectively.

The TM M on input w:

1. Run M_1 and M_2 on input w in parallel.
2. If M_1 accepts, then accept; and if M_2 accepts, then reject.

Proof

If A is decidable, then both A and $\overline{A}$ are Turing-recognizable: Any decidable language is Turing-recognizable, and the complement of a decidable language also is decidable.

Assume both A and $\overline{A}$ are Turing recognizable by M_1 and M_2 respectively.

The TM M on input w:

1. Run M_1 and M_2 on input w in parallel.
2. If M_1 accepts, then accept; and if M_2 accepts, then reject.

Clearly, M decides A.

Corollary

$\overline{A_{TM}}$ is not Turing-recognizable.

Corollary

Corollary

$\overline{A_{TM}}$ is not Turing-recognizable.

Proof.

A_{TM} is Turing-recognizable but not decidable.