Decidability

Topics

- Discuss the power of algorithms to solve problems
- Demonstrate that some problems can be solved by algorithms while other cannot
- Explore the limits of algorithmic solvability
- Demonstrate the unsolvability of certain problems

Rationale for decidability

- Knowing when a problem is algorithmically unsolvable is useful because this shows us that the problem must be simplified
- Like any tool, computers have capabilities and limitations that must be appreciated if they are to be used well
- A glimpse of the unsolvable problem stimulates imagination and help to gain important perspective on computation

Problems versus languages

- Problem solving methodology

  Steps: formalize problem, develop a solution algorithms, execute the algorithm, check the solution

- Any problem that can be formalized can be expressed as a language

  Mechanism: If P be a problem then $L_P = \{E | E$ is an expression of $P\}$ is the language of P.

  Solution: if P is solvable then there is an algorithm that solves it. This is equivalent to saying that a TM M_P decides L_P.

- We use languages to represent various computational problems because we have a terminology for dealing with languages

Decidable languages

Recall: a language L is decidable if there exits a TM M which halts on ever element of L, accepting or rejecting it.

- We develop examples of languages that are decidable by algorithms
- We present algorithms that tests whether a string is a member of a regular language or whether it is a member of a CF language
- These kind of problems have practical applications on compiler construction.

  Example: if the lexicon of a PL is specifi ed by a regular language and regular languages are decidable than we can develop correct lexical analyzers for PL.

Problem solving

- **Direct method:** formulate the problem as a statement asking to show that the language of the problem is decidable and construct a TM that decides it.

Example:

1. **Problem:** design an algorithm that perform lexical analysis in a programming language.
2. **Language:** specify the lexicon of a programming language by regular expressions and design an algorithm that accept an expression if it is specifi ed by a regular expression and reject it if it is not specifi ed by a regular expression.

Pragmatic questions: how do we make the algorithm effi cient and

Problem solving

- **Indirect method:**
  1. Formulate the problem as a statement asking to show that the language L of the problem is decidable.
  2. Express the language $L = E(L_1, \ldots, L_k)$ in terms of the languages L_1, ..., L_k that are decided by the TM-s M_1, ..., M_k.

     Note: the expression $L = E(L_1, \ldots, L_k)$ must be constructed using only closure operators.
  3. Construct a TM that decides the language L using the Turing machines M_1,...,M_k that decide the languages L_1, ..., L_k as subprocedures of M.

Example: specify a PL as an expression $E(RL, CFL)$ where RL is a regular language, CFL is a context-free language. Knowing the TMs M that decide RL and M that decide CFL construct the

Methodology

- Start with well-known problems and languages, such as Finite Automata and Regular Expressions and their closure operators.
- Advance on Chomsky's hierarchy to Pushdown Automata and Context-Free Languages using their closure operators.
- Develop closure operators for TMs and use them in the framework developed so far.

Note: the closure operators for TMs are subject of the problems given in the assignment 5.

Example problems

- Consider the acceptance problem for DFAs:
  test whether a particular finite automaton accepts a given string.
  This can be expressed as a language A_{DFA}
- A_{DFA} contains the encodings of all DFAs together with strings the DFAs accept, i.e.. $A_{DFA} = \{\langle B, w\rangle | B \text{ is a DFA that accepts } w\}$
- Hence, testing whether DFA B accepts w is the same as testing whether $\langle B, w\rangle \in A_{ADF}$

Methodology review

- Computational problems are formulated in terms of testing membership in a language.
- Showing that a language is decidable is the same as showing that a computational problem is solvable.
- We will show first that A_{DFA} is decidable, i.e., testing whether a given finite automaton accepts a string is solvable.

Theorem 4.1

A_{DFA} is a decidable language

Proof idea: construct 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 w
2. If the simulation ends in an accept state then *accept*;
   if it ends in a nonaccepting state then *reject*."

Note: w is finite and simulation always ends

Performing the simulation

- $\langle B, w \rangle$ is a representation of a DFA B together with a string w. One can represent B by a list of its five components: $Q, \Sigma, \delta, q_0, F$
- When M receives an input it checks first whether this input represents a DFA B and a string w; if not reject
- If input is right, M keeps track of B's current state and B's current position in w by writing this info on its tape
- Initially the state of B is q_0 and B's current position is the leftmost symbol of w; the states and position are updated as shown by δ
- When M finishes processing the last symbol of w, M accepts if B is in a final state and reject if B is not in a final state

A DFA simulator

```
State := q0;
C     := First(Input);
while (C≠ EOI)
      {
       State := δ(State,C);
       C := Next(Input);
      }
if (State∈ F)
  accept;
else
  reject;
```

Note: EOI stands for end of input.

Note

We can prove a similar theorem for nondeterministic finite automata

For that we consider the language

$A_{NFA} = \{\langle B, w\rangle | B$ is an NFA and w is a string$\}$

Theorem 4.2

A_{NFA} is a decidable language

Proof: Construct a TM N that decides A_{NFA}.

Note: we could design N to operate like M, simulating an NFA instead of an DFA. However, we will do it differently, will use M as a procedure called by N.

Constructing N

Because M is designed to work with DFAs, N fi rst converts its input NFA to a DFA by the usual technique

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

1. Convert NFA B to a DFA C (see theorem 1.39)
2. Run TM M from Theorem 4.1 on $\langle C, w\rangle$
3. If M accepts, *accept*; otherwise *reject*"

Note: running M in stage 2 means incorporating M into the design of N as a subprocedure

Regular expressions

Consider the language:

$A_{REX} = \{\langle R, w\rangle |$R is a regular expression and w is a string $\}$.

Theorem 4.3 A_{REX} is a decidable language

Proof

The following TM P decides A_{REX}

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

1. Convert R to an equivalent DFA A (see theorem 1.54)
2. Run TM M on input $\langle A, w \rangle$
3. If M accepts, *accept*; if M rejects, *reject*".

Note

Theorems 4.1, 4.2, 4.3 show that for decidability purpose presenting a TM M with DFA, NFA, or a regular expression, all are equivalent because M is able to convert one form of encoding to another

Emptiness problem

- Another kind of problems concerning FAs is the *emptiness testing*
- **The problem:** test if the language of a DFA is empty
- The language of this problem is:

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

Theorem 4.4

E_{DFA} is a decidable language

Proof idea:

- A DFA accepts some string iff reaching a final state from the start state by traveling along the arrows of the transition diagram of the DFA is possible.
- To test this condition we can construct the TM T that marks states of DFA using the state transition function of the DFA
- Use T to solve emptiness problem

The TM T

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:
   (a) Mark any state that has a transition coming into it from any state that is already marked
3. If no fi nal state is marked, *accept*; otherwise *reject*

Language equality

- **The problem:** for two DFA-s A and B, is $L(A) = L(B)$?
- **The language:**

  $$EQ_{DFA} = \{\langle A, B\rangle | A \wedge B \; are \; DFAs \; \wedge L(A) = L(B)\}$$

Theorem 4.5

EQ_{DFA} is a decidable language

Proof idea: (use the indirect method and theorem 4.4)

- Construct a DFA C from A and B where C accepts only those strings that are accepted either by A or B but not by both.
- If A and B recognize the same language then C accepts nothing
- The language C is defi ned by
  $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$
  which is called *symmetric difference* of $L(A)$ and $L(B)$
- Use machine T to check if C is empty

Symmetric difference

The expression $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$ called the symmetric difference of $L(A)$ and $L(B)$ is illustrated in Figure 1

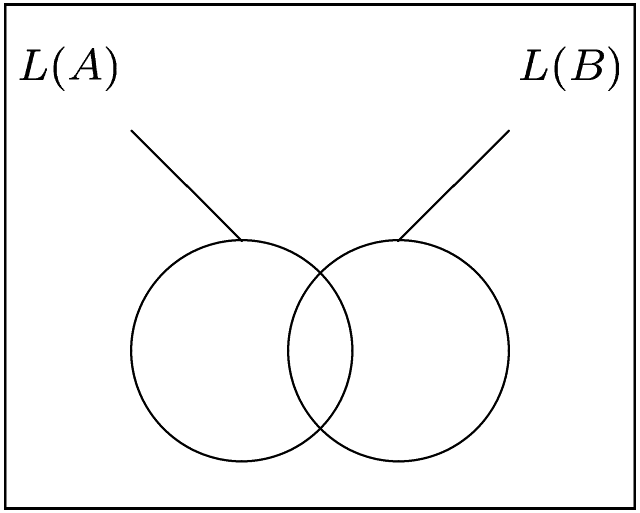

Figure 1: Symmetric difference of L(A) and L(B)

Note: If $L(A) = L(B)$ then $L(A) \cap \overline{L(B)} = L(A) \cap \overline{L(A)} = \emptyset$;

similarly, $\overline{L(A)} \cap L(B) = \emptyset$ and thus $C = \emptyset$.

Construction

$L(C) = \emptyset$ iff $L(A) = L(B)$.
Symmetric difference of $L(A)$ and $L(B)$ is constructed by:

1. Use construction employed by the proof showing that the class of regular languages is closed under complementation (for $\overline{L(A)}$ and $\overline{L(B)}$);
2. Use construction at (1) in conjunction with the construction that proves that class of regular languages is closed under intersection
3. Use the constriction at (2) in conjunction with the construction that proves that class of regular languages is closed under union.

Proving Theorem 4.5

Construct the TM F:

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

1. Construct DFA C that recognizes $L(C)$ as described above
2. Rum TM T from Theorem 4.4 on input $\langle C\rangle$
3. If T accepts, *accept*; if T rejects, *reject*."