Decidable Languages

... 

For those who can’t make up their mind!
Chapter Objectives

Investigate the power of algorithms.

Understand the unsolvability of some problems.
Concerning Regular Languages

Yes, you have to remember what they are!

Acceptance problem: Testing whether a particular DFA accepts a string.

\[ A_{DFA} = \{ <B, w> \mid B \text{ is the DFA that accepts input string } w \} \]

Testing if \( w \) is in \( B \) is the same as testing whether \( <B, w> \) is in \( A_{DFA} \).

Showing that the language is \textit{decidable} is the same as showing that the computational problem is \textit{decidable}.
Theorem: $A_{DFA}$ is a decidable language

Idea: Present a TM $M$ that decides $A_{DFA}$.

$M$ = “On input $<B, w>$, 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:
To represent $B$, simply list its components $Q$, $\Sigma$, $\delta$, $q_0$, and $F$ on the tape.

Simulation is straightforward.

$M$ tracks $B$’s current state and position within $w$ using markings on the tape.

$B$’s initial state is $q_0$, and initial position is the leftmost symbol of $w$.

States and positions are updated according to $\delta$.

After $M$ finishes 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.

Hence, $A_{DFA}$ is decidable.
Theorem(s): $A_{\text{NFA}}$ and $A_{\text{REX}}$ are decidable languages

Idea: Use previous machine $M$ as a subroutine.

Proof:

$N =$ “On input $<B, w>$, where $B$ is a NFA and $w$ is a string:

1. Convert $B$ into an equivalent DFA $C$ using the procedure for this conversion given in Theorem 1.39.
2. Run $M$ on input $<C, w>$.
3. If $M$ accepts, accept; otherwise, reject.”

Hence, $A_{\text{NFA}}$ is decidable.

Idea: Use previous machine $N$ as a subroutine.

Proof:

$P =$ “On input $<R, w>$, where $R$ is a regular expression and $w$ is a string:

1. Convert $R$ into an equivalent NFA $A$ using the procedure for this conversion given in Theorem 1.54.
2. Run $N$ on input $<A, w>$.
3. If $N$ accepts, accept; otherwise, reject.”

Hence, $A_{\text{REX}}$ is decidable.
For **decidability purposes**, it is equivalent to present a TM with a DFA, NFA, or Regular Expression.

Remember: Showing that the language is **decidable** is the same as showing that the computational problem is **decidable**.
Emptiness Testing

Previous examples tested whether or not a string was in a language.

Now we test to see whether or not the finite automaton accepts any strings at all.

\[ E_{\text{DFA}} = \{ <A> \mid A \text{ is a DFA and } L(A) = \emptyset \} \]

Can you come up with an algorithm to perform this test?

**Proof:** Determine whether or not a final state can be reached from the starting state.

\[ T = \text{“On input } <A>, \text{ where } A \text{ is a DFA:} \]

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.”
Equivalence Testing

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

Proof: Construct new DFA \( C \) from \( A \) and \( B \), where \( C \) accepts string accepted by either \( A \) or \( B \), but not both.

If \( A \) and \( B \) recognize the same language, \( C \) will accept nothing.

Expression is called the symmetric difference.

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

Remember that Regular Languages are closed under complement, union, and intersection operations.

\[ F = \text{“On input } \langle A, B \rangle \text{, where } A \text{ and } B \text{ are DFAs: “} \]
\[ 1. \text{ Construct DFA } C \text{ as described.} \]
\[ 2. \text{ Run TM } T \text{ from Theorem 4.4 on input } \langle C \rangle . \]
\[ 3. \text{ If } T \text{ accepts, accept. If } T \text{ rejects, reject.”} \]
Theorem: $A_{\text{CFG}}$ is a decidable language

Could build a TM that goes through all derivations of $w$ from CFG $G$.

Why should we not do this?

Chapter 2 says that if a CFG is in Chomsky Normal Form, any derivation of $w$ has $(2|w| - 1)$ steps, and is therefore finite.

TM can convert into Chomsky Normal Form!

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 derivation with $(2|w| - 1)$ steps. If $|w|$ is 0, then list all derivations with one step.
3. If any derivation generates $w$, accept; If not, reject.”

Note: This problem is related to that of compiling programming languages!

Note # 2: Everything we say about decidability of CFGs applies to PDAs, too!
Theorem: $E_{CFG}$ (emptiness test) is a decidable language

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

Problem:
We can’t use $S$ from the previous ($A_{CFG}$ is decidable) theorem. Why not?

Different idea:
Try to determine whether or not each variable is capable of producing 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 $A \rightarrow U_1 U_2 \cdots U_k$ and each symbol $U_1, \ldots, U_k$ has already been marked.
4. If the start variable is not marked, accept; otherwise, reject.”
Theorem: $\text{EQ}_{\text{CFG}}$ (equivalence test) is a decidable language

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

Similar tactic as $\text{EQ}_{\text{DFA}}$.

Use symmetric difference to prove equivalence.

CFGs are not closed under complement or intersection (from Exercise 2.2).

$\text{EQ}_{\text{CFG}}$ is not decidable.

Techniques to prove are shown in Chapter 5.
Theorem: Every CFL is decidable

Question: It is simple to simulate a stack on a TM. Also, we know that nondeterministic TMs can be simulated on a deterministic TM. Do we just need to simulate a PDA on a TM? Why or why not?

What other option do we have?

Proof:

\( M_G = \text{“Where } G \text{ is a CFG, on input } w: \)

1. Run TM \( S \) on input \( \langle G, w \rangle \).
2. If this machine accepts, \( \text{accept} \); if it rejects, \( \text{reject} \)."

How does this solution bypass the PDA issue?
Relationships Between Languages

All proofs have been given over the course of the semester lectures.

Don’t forget that these are not only classes of language, but also classes of computational ability.
Relationships Between Languages

Don't forget that these are not only classes of language, but also classes of computational ability. All proofs have been given over the course of the semester lectures.