Decidability

Real power is in showing that a problem is not solvable.

- Could just build an algorithm, but we use a TM formalism instead.
- We will first prove that a number of problems are solvable.

By solvable, we mean whether a Turing machine can decide the language.

- By problem, we mean determining if a string is in a certain language.

We can use this to determine which problems are solvable and which are not.

With Turing machines, we have a precise model of what an algorithm is.
Acceptance Problem for DFAs

Prove that $A_{DFA}$ is a decidable language

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

Let's phrase this as a language that we need to decide

Is there an algorithm that when given a DFA and a word $w$ can

Overview

• Introduction: Decidability
  ⇒ Decidable Problems of Regular Languages
  • Decidable Problems of Context-Free Languages
  • Preliminaries to Halting Problem

Decidability: Decidable Problems of Regular Languages
Proof

We can assume that $B$ is written as a list of its five components $Q, \Sigma, \delta, q_0$ and $F$.

- When $M$ receives its input, $M$ first determines whether it properly represents a DFA and a string $w$. If not, $M$ rejects.
- $M$ writes start state on end of the tape, and marks start of input.
- $M$ then simulates $B$.
  - Find the applicable transition.
  - Update current symbol and current position.
  + Which current position of input and current state.
- When $M$ finishes processing the last symbol of $w$.
- $M$ accepts if $B$ is in an accepting state.
- $M$ rejects if $B$ is in a non-accepting state.

Proof Idea

- Proof Idea: build a TM $M$ that uses the description of $B$ and $w$.
  - When $M$ finishes processing the last symbol of $w$.
  - $M$ accepts if $B$ is in an accepting state.
  - $M$ rejects if $B$ is in a non-accepting state.

We want practice with Turing machines.

- We could do this proof by showing the existence of a function $f$.
- A specific DFA $D$ needs to simulate any DFA.
- Note that it is not sufficient to build a Turing machine that corresponds to
  all inputs of $B$.
  $\forall w \in \Sigma^*$, if $B$ is in an accepting state then $w$.

Proof Idea: build a TM $M$ that uses the description of $B$ and $w$.
Acceptance Problem for Regular Expressions

• Is there an algorithm that when given a regular expression and a word w can decide if the regular expression will generate w - $A_{REX} = \{ \langle R, w \rangle | R$ is a regular expression that accepts input string w $\}$

• Proof: the following TM decides $A_{REX}$
P = "On input $\langle R, w \rangle$ where R is a regular expression and w is a string
2. Run TM N from previous proof on input $\langle C, w \rangle$.
3. If N accepts, accept; otherwise, reject."

Existence of the 3 proofs shouldn’t be too surprising, as we can convert between the 3 formalisms using an algorithm.

Acceptance Problem for NFAs

• Is there an algorithm that when given an NFA B and a word w can decide if the NFA will accept w - $A_{NFA} = \{ \langle B, w \rangle | B$ is an NFA that accepts input string w $\}$

• Proof: the following TM decides $A_{NFA}$
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 in Chapter 1.
2. Run TM M from previous proof on input $\langle C, w \rangle$.
3. If M accepts, accept; otherwise, reject."

Running TM M in stage 2 means incorporating M into the design of N as a subprocedure.

Is there an algorithm that when given an NFA and a word w can decide if the NFA will accept w?
Do two DFAs accept the same language?

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

- **Proof Idea:** $L(A) = L(B)$ if and only if $L(A) \cup L(B)$ is an accepting language since $\phi = (\forall \langle \mathcal{A}, \mathcal{B} \rangle) (\forall \langle \mathcal{A}, \mathcal{B} \rangle) (\exists \langle \mathcal{C} \rangle)$ is $\text{DFA}$ accepting the empty set.

- Test if $\mathcal{C}$ accepts the empty set.

- Similar to how we did union/intersection of two DFAs.

- Construct DFA $\mathcal{C}$ that accepts $\mathcal{A} \cup \mathcal{B}$.

- Construct DFA $\mathcal{C}$ such that accepts xor of two DFAs.

- $\{ \langle \mathcal{C} \rangle \}$ is $\text{DFA}$ accepting a language.

---

Does a DFA accept no strings?

- Is $\text{E} \text{DFA} = \{ \langle \mathcal{A} \rangle \mid \mathcal{A} \text{ is a DFA and } L(\mathcal{A}) = \phi \}$

- **Proof:**
  
  1. Mark the start state of $\mathcal{A}$.
  2. Repeat until no new states get marked:
     3. For each state that is not marked:
        4. Mark it if has a transition into it from any state that is already marked.
  5. If no accept state is marked, accept; otherwise reject.

---

Formulate as a language.
Acceptance Problem for CFGs

- This would just be a recognizer, not a decider.
- If $G$ does not generate $w$, this algorithm would never halt.
- But, $G$ might have infinitely many derivations.

One idea is to use $G$ to go through all derivations to determine whether any is a derivation of $w$.

\[ \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \]

Overview

- Introduction: Decidability
- Decidable Problems of Regular Languages \( \subseteq \)
- Decidable Problems of Context-Free Languages
- Preliminaries to Halting Problem
- Overview: Decidability
Does a CFG accept no strings?

- If we reach the start symbol while generating a string (e.g., $S$, $B$, $G$) without generating a string (e.g., $S$, $G$)
  - As we move backwards through the derivation, we see that there is a variable that is capable of generating a string (e.g., $S$
- If we reach the last derivation, we see that there is a variable that is capable of generating a string (e.g., $S$
- One idea is to use $S$, which can test whether a CFG generates some particular string $w$
  - Would need to check this with all possible $w$, but infinitely many
- If a CFG generates a string, we know there is a derivation.
  - If we take the last derivation, we see that there is a variable that is capable of generating a string (e.g., $E$
- As we move backwards through the derivation, we get more and more variables that are capable of generating a string (e.g., $C$, $B$, $S$
- Until we reach the start symbol.

Proof

- If any of these derivations generate $w$, accept. If not, reject.
  - This is because a grammar generates a string using one rule $S$.
  - Except if $n = 0$, then instead list all derivations with 1 step
  - List all derivations with $2n - 1$ steps, where $n = |w|$
  - Convert to Chomsky normal form
  - Only finitely many such derivations exist
  - So, just need to check derivations up to $2n - 1$ steps.
  - If grammar is in Chomsky normal form, any derivation of $w$ has a string $S = \langle n, G, w \rangle$.
Every Context-Free Language is Decidable

• Given a context-free language, can we build a TM machine that will accept it?

- This is different from the acceptance problem, as here we just have to prove that for a given CFL we can build a TM.

• We can build a different TM for every different language.
- This is different from the acceptance problem, as here we just have to prove that for a given CFL, we can build a TM that will accept it.

Proof: Let $G$ be a CFG for $A$ and design a TM $M_G$ that decides $A$.

- We can build a different TM for every different language.

Construction

1. Run TM $S$ on input $\langle G, w \rangle$.
2. If this machine accepts, accept; otherwise reject.
3. Mark any variable $A$ where $G$ has a rule $A \rightarrow U_1 U_2 \ldots U_k$ and each symbol $U_1$, $U_2$, ..., $U_k$ has already been marked.
4. Repeat until no new variables get marked.
5. If the start symbol is not marked, accept; otherwise, reject.

$R = "\text{On input } \langle G, w \rangle \text{ where } G \text{ is a CFG an } w:\"

Construction
Introduction

We have shown a number of problems to be decidable. But, part of the reason for introducing this machinery was to show that there are problems that are not decidable.

What kind of problems are not decidable? Are there any useful ones?
ATM is Turing-Recognizable

Proof: Suppose ATM is Turing-Recognizable, and note

1. Simulate ATM on input \( \langle m, \langle n, \langle n, \langle n, \rangle \rangle \rangle \rangle = \langle m, n, m \rangle \)
2. If \( m \) ever reaches its accept state, accept.
3. If \( m \) ever enters \( \langle n, m \rangle \), reject.

Thus, ATM is Turing-Recognizable.

The Halting Problem

\[ ATM = \{ \langle m, w \rangle \mid m \text{ is a TM and } w \text{ is a string} \} \]

Why \( ATM \) is called the Halting problem: If \( ATM \) was Turing-recognizable, which is necessarily decidable, \( ATM \) would be Turing-recognizable, and not

\[ = \{ \langle m, w \rangle \mid m \text{ is a TM and } w \text{ is a string} \} \]

So the construction just shows \( ATM \) is Turing-recognizable, and not

\[ = \{ \langle m, n \rangle \mid m \text{ is a TM} \} \]

\[ = \{ \langle m, n \rangle \mid m \text{ is a TM and } w \text{ is a string} \} \]

- Previously showed acceptance problem for DFA and CFG are decidable
- This is the acceptance problem for TMs
- Given a Turing machine \( M \) and an input \( w \), can we determine

\[ ATM = \{ \langle m, w \rangle \mid \langle m, w \rangle \text{ is a TM and } w \text{ is a string} \} \]
Universal Turing Machine

- It is interesting in its own right
- It is an example of a universal Turing machine
- Capable of simulating any other Turing machine from a description of that machine