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
- Problems are not solvable
- We can use it to determine if problems are solvable and what
- With Turing machines, we have a precise model of what an

Overview

- Preliminaries to Halting Problem
- Decidable Problems of Context-Free Languages
- Decidable Problems of Regular Languages
- Introduction: Decidability
Acceptance Problem for DFAs

- So this should be a decidable language.
- Does so after |w| transitions of the DFA.
- Intuitively, an DFA either accepts or rejects a string.

Prove that $\text{ADFA}$ is a decidable language.

$\{\langle m, w \rangle \mid \text{DFA} m \text{ accepts input string } w \}$

- Both the DFA and $w$ are part of the input, hence needs to be decided.
- Let's phrase this as a language that we need to decide:

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

Preliminaries to Halting Problem

Decidable Problems of Context-Free Languages

Decidable Problems of Regular Languages

Introduction: Decidability

Overview
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 \( B \) when applicable transition:
  - Find the applicable transition.
  - Update state and current position.
  - Mark current position of input and current state.
  - Update input position and current state.

• 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 non-accepting state.

Proof Idea

• Proof Idea: build a TM \( M \) that uses the description of \( B \) and \( w \) to simulate \( B \) on input \( w \).

- \( M \) writes start state on end of the tape, and marks start of input.

- \( M \) then simulates \( B \) when applicable transition:
  - Find the applicable transition.
  - Update state and current position.
  - Mark current position of input and current state.
  - Update input position and current state.

- 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 non-accepting state.

Notice that it is not sufficient to build a Turing machine that corresponds to a specific DFA. It needs to simulate any DFA. Nonetheless, we could do this proof by showing the existence of an algorithm that converts an input DFA into a TM that accepts the same language.

• Proof Idea: build a TM \( M \) that uses the description of \( B \) and \( w \) to simulate \( B \) on input \( w \).
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 \( w \)

- \( \text{REX} = \{ \langle R, w \rangle | R \text{ is a regular expression that accepts input string } w \} \)

Proof: the following TM decides \( \text{REX} \)

1. Convert \( R \) to an equivalent NFA \( C \), using procedure in Chapter 1.
2. Run TM \( M \) from previous proof on input \( \langle C, w \rangle \).
3. If \( M \) 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 and a word \( w \) can decide if the NFA will accept \( w \)

- \( \text{NFA} = \{ \langle B, w \rangle | B \text{ is a NFA that accepts input string } w \} \)

Proof: the following TM decides \( \text{NFA} \)

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.

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

Acceptance Problem for NFAs

• Is there an algorithm that when given an NFA and a word \( w \) can decide if the NFA will accept \( w \)

- \( \text{NFA} = \{ \langle B, w \rangle | B \text{ is a NFA that accepts input string } w \} \)

Proof: the following TM decides \( \text{NFA} \)

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.

Is there an algorithm that when given an NFA and a word \( w \) can...
Do two DFAs accept the same language?

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

Proof Idea:

- $L(A) = L(B)$ if and only if $L(A) \text{ xor } L(B) = \emptyset$.

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

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

- Construct DFA $C$ that accepts $\emptyset$.

Test if $C$ accepts the empty set.

$\{ \emptyset \}$

Does a DFA accept no strings?

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

Proof:

- $\text{E}$ is decidable.

- $\{ \emptyset = \langle \emptyset \rangle \}$ is a DFA and $\emptyset$ is 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 \} = \text{ACFG}
Does a CFG accept no strings?

- Until we reach the start symbol, variables that can generate a string (e.g., C, B, S)
- As we move backwards through the derivation, we get more and more
- of generating a string (e.g., δ).
- If we take the last derivation, we see there is a variable that is capable
  of generating a string, we know there is a derivation.
- If a CFG generates a string, we know there is a derivation.
- We would need to check this with all possible w, but infinitely many.
- One idea is to use S, which can test whether a CFG generates
  \{\emptyset \mid G \text{ is a CFG and } L(G) = \emptyset\}

Proof

- If any of these derivations generate \( w \), accept; if not, reject.

  - this is because grammar generates \( \epsilon \) using one rule \( S \rightarrow \epsilon \)
  - except if \( n = 0 \), then instead list all derivations with 1 step
    \(|n| = 1\), list all derivations with 2n − 1 steps, where \( n \) is a string:
    1. Convert to Chomsky normal form
    2. Lists all derivations with 2n − 1 steps, where \( n \) is a string:
      - Only finitely many such derivations exist
      - So, just need to check derivations up to 2n-1 steps
        \(|n| = 2n-1\) steps, where \( n \) is a string.

  If grammar is in Chomsky normal form, any derivation of \( w \) has
Every Context-Free Language is Decidable

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.

We prove that for a given CFL we can build a TM that decides it.

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

Construction

1. Run TM $S$ on input $(G, w)$.
2. If this machine accepts $G$, accept; otherwise, reject.

Construction

1. On input $(G)$
2. Mark all terminal symbols in $G$.
3. Mark any variable $A$ whose $G$ has a rule $A \rightarrow U_1 \ldots U_k$.
4. Repeat until no new variables are marked:
   - Mark all terminal symbols in $G$.
   - Mark any variable $A$ where $G$ has a rule $A \rightarrow V_1 \ldots V_l$.

R = "On input $(G)$ where $G$ is a CFG in $w$:
   1. Mark all terminal symbols in $G$.
   2. Repeat until no new variables are marked:
      - Mark any variable $A$ where $G$ has a rule $A \rightarrow U_1 \ldots U_k$.
   3. Mark any variable $A$ whose $G$ has a rule $A \rightarrow V_1 \ldots V_l$.
   4. If the start symbol is not marked, accept; otherwise, reject."
Introduction

- Are there any useful ones?

What kind of problems are not decidable?

There are problems that are not decidable.

But, part of the reason for introducing this machinery was to show that

by an algorithm or Turing machine

we have shown a number of problems to be decidable (solvable).

Overview

- Preliminaries to Halting Problem

- Decidable Problems of Context-Free Languages

- Decidable Problems of Regular Languages

- Undecidability: Decidability
A_TM is Turing-recognizable

**Proof:**

1. Simulate M on input w
2. If M ever enters its accept state, accept.
3. If M ever enters its reject state, reject.

Note that A_TM loops forever on w if M loops forever on w.

So this construction just shows A_TM is Turing recognizable, and not necessarily decidable.

We will eventually show that A_TM is undecidable.

Previous 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 whether M will accept w?

\[ \{ m \mid \langle m, w \rangle \in A_{TM} \} = \mathbb{N} \]

The Halting Problem
Universal Turing Machine

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