Decidability

• With Turing machines, we have a precise model of what an algorithm is
• We can use it to determine what problems are solvable and what problems are not solvable
  - By problem, we mean determining if a string is in a certain language
  - By solvable, we mean whether a Turing machine can decide the language

• We will first prove that a number of problems are solvable
  - Could just build an algorithm, but will use a TM formalism instead
• Real power is in showing that a problem is not solvable
Acceptance Problem for DFAs

• Is there an algorithm that when given a DFA and a word $w$ can decide if the DFA will accept $w$?
• Let’s phrase this as a language that we need to decide:
  - Both the DFA and $w$ are part of the ‘input’ that needs to be ‘decided’
  - $A_{\text{DFA}} = \{B, w \mid B$ is a DFA that accepts input string $w\}$
• Prove that $A_{\text{DFA}}$ is a decidable language
• Intuitively, a DFA either accepts or rejects a string
  - Does so after $|w|$ transitions of the DFA
  - So this should be a decidable language

Overview

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

• Proof Idea: build a TM $M$ that uses the description of $B$ and $w$ on the input tape, and simulates DFA $B$ on input $w$
  - At end of processing $w$, if $B$ is in an accept state then accept, otherwise reject
  - Note that it is not sufficient to build a Turing machine that corresponds to a specific DFA. It needs to simulate any DFA
  - We could do this proof by showing the existence of an 'algorithm', but we want practice with Turing machines

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:
    + Match current position of input and current state
  - Update state and current position
• 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
Acceptance Problem for NFAs

- Is there an algorithm that when given an NFA and a word $w$ can decide if the NFA will accept $w$
  - $A_{\text{NFA}} = \{ \langle B, w \rangle | B \text{ is an NFA that accepts input string } w \}$
- Proof: the following TM decides language $A_{\text{NFA}}$
  \[\text{N = "On input } \langle B, w \rangle \text{ where } B \text{ is an NFA, and } w \text{ is a string}
  \begin{enumerate}
  \item Convert NFA $B$ to an equivalent DFA $C$, using the procedure in Chapter 1.
  \item Run TM $M$ from previous proof on input $\langle C, w \rangle$.
  \item If $M$ accepts, accept; otherwise, reject."
\]
- Running TM $M$ in stage 2 means incorporating $M$ into the design of $N$ as a subprocedure

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 $w$
  - $A_{\text{REX}} = \{ \langle R, w \rangle | R \text{ is a regular expression that accepts input string } w \}$
- Proof: the following TM decides $A_{\text{REX}}$
  \[\text{P = "On input } \langle R, w \rangle \text{ where } R \text{ is a regular expression, and } w \text{ is a string}
  \begin{enumerate}
  \item Convert $R$ to an equivalent NFA $C$, using procedure in Chapter 1.
  \item Run TM $N$ from previous proof on input $\langle C, w \rangle$.
  \item If $M$ accepts, accept; otherwise, reject."\]
- Existence of the 3 proofs shouldn’t be too surprising, as we can convert between the 3 formalisms using an algorithm
Does a DFA accept no strings?

- Formulate as a language
  - \( E_{\text{DFA}} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset \} \)
- Is \( E_{\text{DFA}} \) decidable?
  - **Proof:**
    
    \[ T = \text{"On input } \langle A \rangle \text{ where } A \text{ is a DFA} \]
    
    1. Mark the start state of \( A \)
    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.”

Do two DFAs accept the same language?

- Is \( EQ_{\text{DFA}} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \} \)
  - **Proof Idea:** \( L(A) = L(B) \) iff \( L(A) \text{ xor } L(B) = \emptyset \)
    - Construct DFA \( C \) that accepts xor of two DFAs
      + Similar to how we did union/intersection of two DFAs
      + Take cross product of states and simulate \( A \) and \( B \)
      + Only accept if just one of the two DFAs is in an accept state
    - Test if \( C \) accepts the empty set
  
  \[ T = \text{"On input } \langle A, B \rangle \text{ where } A \text{ and } B \text{ are a DFAs} \]
  
  1. Construct DFA \( C \) as described above.
  2. Run TM \( T \) from previous proof on input \( \langle C \rangle \)
  3. If \( T \) accepts, accept; otherwise reject.”
Overview

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

Acceptance Problem for CFGs

• $A_{CFG} = \{(G, w)|G$ is a CFG that generates string $w\}$
• One idea is to use $G$ to go through all derivations to determine whether any is a derivation of $w$
  - But, $G$ might have infinitely many derivations
  - If $G$ does not generate $w$, this algorithm would never halt
  - This would just be a recognizer, not a decider
Proof

- If grammar is in Chomsky normal form, any derivation of $w$ has $2n-1$ steps, where $n = |w|
- So, just need to check derivations up to $2n-1$ steps
- Only finitely many such derivations exist

$S = \"On input $\langle G, w \rangle$ where $G$ is a CFG an $w$ is a string:\n1. Convert $G$ to Chomsky normal form
2. List all derivations with $2n-1$ steps, where $n = |w|$, except if $n=0$, then instead list all derivations with 1 step (this is because grammar generates $\epsilon$ using one rule $S \rightarrow \epsilon$)
3. If any of these derivations generate $w$, accept; if not, reject.\"$

Does a CFG accept no strings?

- $E_{CFG} = \{ \langle G \rangle | G$ is a CFG and $L(G) = \emptyset \}$
- 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.
  - e.g. $S \Rightarrow aB \Rightarrow abCEd \Rightarrow abcEd \Rightarrow abced$
  - 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 can generate a string (e.g. $C, B, S$)
  - Until we reach the start symbol
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

- Proof: Let $G$ be a CFG for $A$ and design a TM $M_G$ that decides $A$. We build a copy of $G$ into $M_G$. It works as follows:

  $M_G = \text{"On input } \langle w \rangle \text{"}$
  1. Run TM $S$ on input $\langle G, w \rangle$
  2. If this machine accepts, accept; otherwise, reject."
Introduction

• We have shown a number of problems to be decidable (solvable by an algorithm or Turing machine)
  - 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?
The Halting Problem

\[ A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ accepts } w \} \]

- Given a Turing machine \( M \) and an input \( w \), can we determine whether \( M \) will accept \( w \)?
- This is the acceptance problem for TMs
  - Previously showed acceptance problem for DFA and CFG are decidable
- We will eventually show that \( A_{TM} \) is undecidable

\[ A_{TM} \text{ is Turing-Recognizable} \]

- Proof:
  \[ U = \text{"On input } \langle M, w \rangle \text{ where } M \text{ is a TM and } w \text{ is a string} \]
  1. Simulate \( M \) on input \( w \)
  2. If \( M \) ever enters its accept state, accept; if \( M \) ever enters its reject state, reject.”
- Note that \( U \) loops forever on \( w \) if \( M \) loops forever on \( w \),
  - So this construction just shows \( A_{TM} \) is Turing recognizable, and not necessarily decidable
  - If \( U \) had a way to tell if \( M \) was looping forever, it could reject, which is why \( A_{TM} \) is called the halting problem
Universal Turing Machine

- $U$ 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