Decidable Problems

<table>
<thead>
<tr>
<th></th>
<th>DFA</th>
<th>CFG</th>
<th>LBA</th>
<th>TM</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>E</td>
<td>yes</td>
<td>yes</td>
<td>??</td>
<td>no</td>
</tr>
<tr>
<td>All</td>
<td>??</td>
<td>??</td>
<td>??</td>
<td>??</td>
</tr>
<tr>
<td>Eq</td>
<td>yes</td>
<td>??</td>
<td>no</td>
<td>??</td>
</tr>
</tbody>
</table>
Does an LBA accept no strings?

\[ E_{LBA} = \{ \langle M, w \rangle | M \text{ is an LBA and } L(M) = \emptyset \} \]

**Theorem:** \( E_{LBA} \) is undecidable.

- **Proof idea:**
  - Assume \( E_{LBA} \) is decidable and do a reduction from \( A_{TM} \)
  - Assume \( R \) decides \( E_{LBA} \). Use \( R \) in deciding \( A_{TM} \)
  - For a TM \( M \) and input \( w \)
    - Is there an LBA \( B \) that accepts some string when TM accepts \( w \)
      - and recognizes \( \emptyset \) when TM does not accept \( w \)
    - Make \( L(B) \) be all the accepting computation histories for \( M \) on \( w \)
    - If \( M \) does not accept \( w \), \( L(B) \) is empty
  - But
    - Does \( B \) exist, an LBA that accepts accepting computations of a TM?
    - Can a TM construct \( B \) from \( M, w \)? (can we make an algorithm?)

---

**How to construct the LBA \( B \)**

- **Accepting computation** \( C_1, C_2, ..., C_l \)
  - Written on tape as single string, with configurations separated from each other by the \( \# \) symbol
- \( B \) will determine whether
  - \( C_1 \) is the start configuration for \( M \) on \( w \)
    - Check \( C_1 \) by having \( w \) hardcoded into \( B \)
  - \( C_l \) is an accepting configuration for \( M \)
    - Make sure that the state in \( C_l \) is an accepting state
  - \( C_{i+1} \) legally follows from \( C_i \)
    - Verify that \( C_i \) and \( C_{i+1} \) are identical except for the position under and adjacent to the head in \( C_i \), which must be updated according to \( M \)'s \( \delta \)
    - \( B \) does this by zigzagging between corresponding positions of \( C_i \) and \( C_{i+1} \)
    - Marks its position by leaving dots on the tape (increasing the tape alphabet)
- \( B \) is an LBA (with long inputs) and can be constructed by a TM
Can we determine if a CFG accepts everything?

- Can use reduction via computation histories to establish undecidability of certain problems related to CFG and PDA

\[ \text{ALL}_{\text{CFG}} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^* \} \]

**Theorem:** \( \text{ALL}_{\text{CFG}} \) is undecidable

- **Proof Idea:** Proof by contradiction
  - Assume that \( \text{ALL}_{\text{CFG}} \) is decidable
  - Reduce \( A_{TM} \) (whether \( M \) accepts \( w \)) to one about whether a CFG \( G \) contains all strings or not
  - Have \( G \) generate everything but accepting computations of \( w \)
    + If \( M \) rejects \( w \), \( G \) should generate everything
    + If \( M \) accepts \( w \), \( G \) will not generate everything (e.g., accepting computations)
  - Deciding if \( L(G) = \Sigma^* \) allows us to determine if \( M \) accepts \( w \)
Continued: Can a CFG accept such a language?

- Let’s think in terms of a PDA $D$ (with non-determinism)
  - Can just process the input tape from left to right
- Non-deterministically guess where the error is
  - $C_1$ is not a start configuration for $M$ on $w$
    + Check $C_1$ by having $w$ hardcoded into $D$
  - $C_I$ is not an accepting configuration for $M$
    + Make sure that the state in $C_I$ is an accepting state
  - some $C_{i+1}$ does not legally follow from $C_i$
    + Verify that $C_{i+2}$ does not follow $C_i$. Either there is a difference not by the
tape head, or the change by the tape head is not legal according to $M$’s $δ$
    + Cannot do this by zigzagging between corresponding positions of $C_i$ and
      $C_{i+1}$ (since cannot change the tape)
    + Reads the $C_i$ onto its stack. Pops it off its stack as it is comparing it to $C_{i+1}$
    + Assume a configuration is written with every other configuration backwards
      (so it can be compared as it is being popped off stack)

Continued: Aside

- A CFG cannot determine if a computation history is valid
  - As it would need to make sure that $C_{i+1}$ is valid given $C_i$ and that $C_{i+2}$
    is valid given $C_{i+1}$
  - So, as it is popping $C_i$ off the stack to compare to $C_{i+1}$, it would need to
    be pushing $C_{i+1}$ onto the stack so it can compare it to $C_{i+2}$
  - But, it can make sure make sure there is at least one incorrect transition
Proof

- Assume that $\text{All}_{\text{CFG}}$ is decidable. Let $R$ be a TM that decides it.
- We construct TM $S$ to decide $A_{\text{TM}}$ as follows:
  
  $S = \text{"On input } \langle M, w \rangle \text{ where } M \text{ is a TM and } w \text{ is a string} \text{:
    
    1. Construct PDA } D \text{ from } M \text{ and } w \text{ as described on previous slide}
    \text{. We can do this because we can algorithmically describe how to build } D$
    
    2. Convert $D$ to a CFG $G$ (from Chapter 2)
    
    3. Run $R$ on input $\langle G \rangle$
    
    4. If $R$ rejects (not $\Sigma^*$), accept; if $R$ accepts, reject

Overview

- Reductions via Computational Histories
- $\Rightarrow$ Post Correspondence Problem
- Mapping Reducibility
Post Correspondence Problem

- Are there other problems, not concerned with Automata, that are undecidable?
- Post Correspondence Problem
  - Collection of dominos e.g. \{[A]_{a}, [B]_{ab}, [C]_{a}, [A]_{c}\}
  - Is it possible to make a list of these dominos (repetitions allowed) with the same string on top as on bottom (called a match)?
  - Some collections of dominos do not have a match: e.g. \{[abc]_{a}, [ca]_{a}, [acc]_{a}\}
  - Problem is to determine whether a collection of dominos has a match
- More formally
  - An instance of the PCP is a collection of dominos \(P = ([t_1]_{b_1}, [t_2]_{b_2}, ..., [t_k]_{b_k})\)
  - and a match is a sequence \(i_1, i_2, ..., i_l\), where \(t_{i_1}t_{i_2}...t_{i_l} = b_{i_1}b_{i_2}...b_{i_l}\)
  - \(PCP = \{\{P\}|P\text{ is an instance of PCP with a match}\}\)

Is PCP decidable?

- Show that \(A_{TM}\) can be reduced to PCP
  - Use dominos to describe computation histories
  - A match will be a successful computation of \(M\) on \(w\)
- Simplistic View:
  - A domino with \(C_1\) on bottom, nothing on top
  - A domino for all of the legal ways that any \(C_i\) (on top) can yield some \(C_{i+1}\) (on bottom)
  - A domino for each accepting configuration on top, nothing on bottom
  - Matching makes sure \(C_i\)'s on bottom matches \(C_i\)'s on top
- What's wrong with our simplistic view?
Finite Number of Tiles to End

- Problem: Infinite number of tiles to specify end configurations
- All that is needed is to check that the state is an accepting state
  - For each accepting state $q_f$, create a tile with state on top $[q_f]$
  - For each letter $a \in \Sigma$, create a tile for letter on top $[a]$
  - But what prevents a match from using these special tiles before the last configuration?

```
# | $C_1$ | # | $C_2$ | # | $C_{16}$ | #
- | $C_1$ | - | $C_2$ | - | $C_3$ | -
```

+ $C_{16}$ is the configuration of $C_3$ with an $x$ at the beginning
+ We have just created a way to match that is not a legal computation history

A Better Way to End

- Allow accept state to swallow up a character before it or after it
  - For each $a \in \Sigma$, add $[q_{\text{accept}}a]$ and $[q_{\text{accept}}]$ to $q_{\text{accept}}$
  - For each $a \in \Sigma$, add $[q_{\text{accept}}]$ to $q_{\text{accept}}$
  - Use $[q_{\text{accept}}#]$ to end
  - Will take a number of ‘pseudo-steps to end
    + Think of this as changing TM so that once it gets to accept state, it adds extra transitions to empty out the tape
- Allows us to alter any configuration that has an accept state
  - But still has to legally get to the first accept state
Almost There

- **Use Modified Post Correspondence Problem**
  - Must start match with a specified domino
    - We can use it to force the domino with $C_1$ on bottom to be used first
    - Now can’t use domino with same letter on top and bottom to end right away
  - Assume $M$ never goes off the left-end
    - A TM can transform any $M$ to an equivalent one that does this by adding an extra character to denote the beginning of the tape, and extra transition rules that check for the beginning of tape character
  - If $w = \epsilon$, use string $\omega$ (which is character on right-end of tape)
  - So, given $\langle M, w \rangle$, a TM $S$ can build a $MPCP$ that has a match iff $M$ accepts $w$
    - Acceptance can be reduced to $MPCP$!
    - Since acceptance problem is undecidable, so is $MPCP$

Finite number of dominos for Transition

- Next configuration $C_{i+1}$ depends on
  - state of the machine in $C_i$
  - content of tape to the left of head in $C_i$
- Next configuration $C_{i+1}$ might change
  - state
  - position of head (forward/backward)
  - content to the left of former position of head
- Just put this part of configurations on dominos
  - e.g. for $\delta(q_0, 0) = (q_7, 2, R)$, use $\begin{bmatrix} q_0 & 0 \\ 2 & q_7 \end{bmatrix}$
- Rest of transition from $C_i$ to $C_{i+1}$ stays the same
  - Use dominos that we added earlier: $\begin{bmatrix} \alpha \end{bmatrix}$ for each $\alpha \in \Sigma$
- This is close but ...
Modified Post Correspondence Problem

• Can MPCP be reduced to PCP?
  - Can we get PCP to start with a certain tile without forcing it?
• Let $\star \diamond$ be two characters not used in the tiles
  - Convert every domino so that top has $\star$ before every character and
    bottom has $\star$ after each character
  - Add extra domino for start domino that also has $\star$ before bottom string
  - Add a special domino $\begin{bmatrix} \star \diamond \\ \diamond \star \end{bmatrix}$
• Only way to start is to use the special start domino
  - Otherwise first character of the top and bottom strings won’t be the same
  - All intermediate solutions cannot reuse the special start tile
  - Only way to end is with ending tile
  - $\star$s are every other character, and don’t interfere with solution to MPCP

Example of Converting MPCP to PCP

Tiles: \{ \begin{bmatrix} a \\ ab \end{bmatrix}, \begin{bmatrix} b \\ ca \end{bmatrix}, \begin{bmatrix} ca \\ a \end{bmatrix}, \begin{bmatrix} abc \\ c \end{bmatrix}, \begin{bmatrix} a \end{bmatrix} \}

Start tile: \begin{bmatrix} a \\ ab \end{bmatrix}
Overview

- Reductions via Computational Histories
- Post Correspondence Problem
  ⇒ Mapping Reducibility

Mapping Reducibility

- Used reducibility to prove a number of problems are undecidable
- Let’s formalize the notion of **reducibility** so that we can use it in more refined ways
- Idea: Reducing problem \( A \) to problem \( B \) means that there is a computable function that converts instances of problem \( A \) to instances of problem \( B \)
  - If we have such a reduction, we can solve \( A \) with a solver for \( B \)
  - If instance of \( B \) is true (accept), so is instance of \( A \)
  - Preserves acceptance (cannot switch it: reject \( A \) if accept \( B \))
Computable Function

Definition: A function \( f : \Sigma^* \rightarrow \Sigma^* \) is a **computable function** if some Turing machine \( M \), on every input \( w \) halts with just \( f(w) \) on its tape.

- All arithmetic operations on integers are computable functions
  - can make a machine that takes input \( \langle m, n \rangle \) and returns sum of \( m \) and \( n \)
- Computable functions can transform machine descriptions
  - Can make \( f \) that transforms a machine \( M \) into \( M' \) so that it never attempts to move off the left-hand side of its tape
  + \( f \) does this by adding several states to the description of \( M \) so that \( M' \) inserts a special character at the beginning of the tape when it starts (and moving all of the other characters back one), and then has special states if it reads that character later on
  - So \( f \) takes input \( w \). If \( w \) is the description of a TM, say \( M \), \( f \) writes out the description of \( M' \). Otherwise it writes out \( \epsilon \)

Formal Definition

Definition: Language \( A \) is **mapping reducible** to language \( B \), written \( A \leq_m B \), if there is a computable function \( f : \Sigma^* \rightarrow \Sigma^* \), where for every \( w \),

\[ w \in A \iff f(w) \in B. \]

The function \( f \) is called the **reduction** of \( A \) to \( B \).

- A mapping reduction of \( A \) to \( B \) provides a way to convert questions about membership testing in \( A \) to membership testing in \( B \)
Mapping Reduction and Decidability

**Theorem:** If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.

- **Proof:**
  - Let $M$ be a decider for $B$ and $f$ be the reduction from $A$ to $B$
  - A decider $N$ for $A$ is as follows:
    
    $N =$ “On input $w$:
    1. Compute $f(w)$
    2. Run $M$ on input $f(w)$ and output whatever $M$ outputs

  - Is $N$ a decider of $A$?
    
    + Clearly if $w \in A$, then $f(w) \in B$ because $f$ is a reduction from $A$ to $B$
    + Thus, $M$ accepts $f(w)$ whenever $w \in A$. Therefore $N$ works as desired

**Corollary:** If $A \leq_m B$ and $A$ is undecidable, then $B$ is undecidable.

Let’s Use Mapping Reducibility

Earlier we proved:

- Assume that $HALT_{TM}$ is decidable. Let $R$ be TM that decides it
- We construct TM $S$ to decide $A_{TM}$ as follows:
  
  $S =$ “On input $\langle M, w \rangle$ where $M$ is a TM, and $w$ is a string
  1. Run TM $R$ on input $\langle M, w \rangle$
  2. If $R$ rejects, $reject$
  3. If $R$ accepts, simulate $M$ on $w$ until $M$ halts
  4. If $M$ accepted, $accept$, if $M$ has rejected, $reject$

- Does $S$ decide $A_{TM}$?
  
  - $S$ always halts, and so it is a decider
  - It accepts $\langle M, w \rangle$ exactly when $M$ accepts $w$. So, it decides $A_{TM}$

- So is $A_{TM}$ decidable. Contradiction. So, our assumption is false
Use with Halting Problem

- Make a computable function $f$ that takes input $\langle M, w \rangle$ and returns output $\langle M', w' \rangle$ where $\langle M, w \rangle \in A_{TM}$ if and only if $\langle M', w' \rangle \in HALT_{TM}$
- The following machine $F$ computes a reduction $f$
  $F =$ "On input $\langle M, w \rangle$:
  1. Construct the following machine $M'$
     $M' =$ "On input $x$
     (a) Run $M$ on $x$
     (b) If $M$ accepts, accept
     (c) If $M$ rejects, enter a loop."
  2. Output $\langle M', w \rangle$."
- Minor Issue: If $F$ determines it’s input is not in right format, just has to output some string not in $HALT_{TM}$

More

- PCP is undecidable:
  - Contains two reductions $A_{TM} \leq_{m} MP$ and then from $MP \leq_{m} PCP$
  - Mapping reducibility is transitive (Exercise 5.6)
  - So $A_{TM} \leq_{m} PCP$
  - Since $A_{TM}$ is undecidable, so is $PCP$
- $EQ_{TM}$ is undecidable
  - $E_{TM} \leq_{m} EQ_{TM}$
    + $f$ on input $\langle M \rangle$ outputs $\langle M, M_1 \rangle$ where $M_1$ is a machine that rejects all inputs
  - Since $E_{TM}$ is undecidable, so is $EQ_{TM}$
Complementation

- In previous proof
  - Assume that $E_{TM}$ is decidable, say by $R$
  - On input $\langle M, w \rangle$
    + Construct $M_1$ that on input $x \neq w$ rejects, otherwise run $M$ on $w$
    + If $R$ accepts $M_1$, then $M_1$ accepts no strings, so $M$ rejects $w$, so accept.
    Otherwise, reject
- Mapping reduction cannot change accepts and rejects
  + We do have a mapping reduction from $A_{TM}$ to $E_{TM}$
  + Decidability is not affected by complementation, so $E_{TM}$ is undecidable
  + But, we do not have mapping reduction from $A_{TM}$ to $E_{TM}$
  + In fact no such mapping reduction exists Exercise 5.5

Turing-recognizable

**Theorem:** If $A \leq_m B$ and $B$ is Turing-recognizable, then $A$ is Turing-recognizable.

**Corollary:** If $A \leq_m B$ and $A$ is not Turing-recognizable, then $B$ is not Turing-recognizable.