Mapping Reducibility

Monday, April 4, 2022
Announcements

• HW 8 extended
  • Due Wed 4/6 11:59pm EST

• HW 9 out soon
Last Time: TM Accepting Computations

A TM accepting computation is sequence of configurations, where:

1. **Start Config:**
   - State: start state,
   - Head: at leftmost cell
   - Tape: has input string

2. **End Config:**
   - State: accept state

3. **Middle Configs:**
   - State + Head + Tape: each step must be valid according to δ

So: any machine that can recognize TM accepting sequences ...

"w" ... can be used to implement $A_{TM}$ decider!

I.e., ... can be used to prove undecidability!
Last Time: What Makes CFLs “Context-Free”?

- $A_{CFG} = \{\langle G, w \rangle | G$ is a CFG that generates string $w \}$
  - Decidable
- $E_{CFG} = \{\langle G \rangle | G$ is a CFG and $L(G) = \emptyset \}$
  - Decidable
- $ALL_{CFG} = \{\langle G \rangle | G$ is a CFG and $L(G) = \Sigma^* \}$
  - Undecidable

This unintuitive result is explained by ...

... the fact that PDAs can recognize non-accepting TM config sequences

Can be computed in a “context-free” way:
check that pairs of configs are valid nondeterministically, ... and accept if any are not

... but PDAs cannot recognize accepting TM config sequences

Cannot be computed in a “context-free” way:
check that pairs of configs are valid nondeterministically, ... and accept if all are not

This gives insight into what makes context-free languages “context-free”
The Post Correspondence Problem (PCP)
A unique undecidable problem
A Non-Formal Languages Undecidable Problem: \textit{PCP}

- Let \( P \) be a set of "\textit{dominos}"
  - Where each \( t_i \) and \( b_i \) are strings
    
    \[
    \left\{ \frac{t_1}{b_1}, \frac{t_2}{b_2}, \ldots, \frac{t_k}{b_k} \right\}
    \]

- \text{E.g.,} \( P = \left\{ \frac{b}{ca}, \frac{a}{ab}, \frac{ca}{a}, \frac{abc}{c} \right\} \)

- A match is:
  - A sequence of dominos with the same top and bottom strings

- \text{E.g.,} \[
\begin{pmatrix}
\frac{a}{ab} & \frac{b}{ca} & \frac{ca}{a} & \frac{a}{ab} & \frac{abc}{c}
\end{pmatrix}
\] \rightarrow

- \text{Then:} \( PCP = \{ <P> \mid P \text{ is a set of dominos with a match} \} \)
Theorem: \( PCP \) is undecidable

\[ PCP = \{ <P> \mid P \text{ is a set of dominos with a match} \} \]

Proof by contradiction:

Assume \( PCP \) is decidable, has decider \( R \); use it to create decider for \( A_{TM} \):

On input \( <M, w> \):
1. Construct a set of dominos \( P \) that has a **match** only when \( M \) accepts \( w \)
2. Run \( R \) with \( P \) as input
3. Accept if \( R \) accepts, else reject

So a match is a sequence of configs showing \( M \) accepting \( w \)!

Idea: \( P \) has \( M \)'s TM configurations as its domino strings
**PCP Dominos**

- **First domino:** \[
\begin{array}{c}
\# \\
\#q_0 w_1 w_2 \cdots w_n \# \\
\end{array}
\]

- **Key idea:** add dominos representing valid TM steps:
  - if \( \delta(q, a) = (r, b, R) \), put \( \begin{array}{c} qa \\ br \end{array} \) into \( P \)
  - if \( \delta(q, a) = (r, b, L) \), put \( \begin{array}{c} cq a \\ rcb \end{array} \) into \( P \)

- For the tape cells that don’t change: put \( \begin{array}{c} a \\ a \end{array} \) into \( P \)

- Top can only “catch up” if there is an accepting config sequence

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}) \]
**PCP Example**

- Let $w = 0100$ and $\delta(q_0, 0) = (q_7, 2, R)$ so $\left[ \frac{q_00}{2q_7} \right]$ in $P$.
**PCP Dominos (accepting)**

- When accept state reached, let top “catch” up:

  For every \( a \in \Gamma \),

  
  
  
  put \[ \frac{a}{q_{accept}} \] and \[ \frac{q_{accept}}{a} \] into \( P \)

  Bottom “eats” one char

---

Only possible match: accepting sequence of TM configs

```
# | 2 1 q_{accept} 0 2 | # |
... |
# 2 1 q_{accept} 0 2 | # |
```

“eat” one char
Mapping Reducibility
Flashback: “Reduced”

\[
A_{TM} = \{(M, w) \mid M \text{ is a TM and } M \text{ accepts } w\}
\]

\[
HALT_{TM} = \{(M, w) \mid M \text{ is a TM and } M \text{ halts on input } w\}
\]

Thm: \(HALT_{TM}\) is undecidable

Proof, by contradiction:

- **Assume** \(HALT_{TM}\) has decider \(R\); use to create \(A_{TM}\) decider:

  - **Contradiction:** \(A_{TM}\) is undecidable and has no decider!

Let’s *formalize* this conversion, i.e., *mapping reducibility*
Flashback: \( A_{\text{NFA}} \) is a decidable language

\[
A_{\text{NFA}} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts input string } w \}\]

Decider for \( A_{\text{NFA}} \):

\[N = \text{“On input } \langle B, w \rangle, \text{ where } B \text{ is an NFA and } w \text{ is a string:} \]

1. Convert NFA \( B \) to an equivalent DFA \( C \), using the procedure \( \text{NFA}\rightarrow\text{DFA} \).
2. Run TM \( M \) on input \( \langle C, w \rangle \).
3. If \( M \) accepts, accept; otherwise, reject.”

We said this \( \text{NFA}\rightarrow\text{DFA} \) algorithm is a TM, but it doesn’t accept/reject?

More generally, we’ve been saying “programs = TMs”, but programs do more than accept/reject?
Definition: Computable Functions

• Has TM that, instead of accept/reject, “outputs” final tape contents

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.

• Example 1: All arithmetic operations

• Example 2: Converting between machines, like DFA $\rightarrow$ NFA
  • E.g., adding states, changing transitions, wrapping TM in TM, etc.
**Definition: Mapping Reducibility**

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 $w$,

$$w \in A \iff f(w) \in B.$$  

“if and only if”

The function $f$ is called the **reduction** from $A$ to $B$.

- **“forward” direction ($\Rightarrow$):** if $w \in A$ then $f(w) \in B$
- **“reverse” direction ($\Leftarrow$):** if $f(w) \in B$ then $w \in A$

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.
Flashback: Equivalence of Contrapositive

“If $X$ then $Y$” is equivalent to ... ?

• “If $Y$ then $X$” (converse)
  • No!

• “If not $X$ then not $Y$” (inverse)
  • No!

✓ “If not $Y$ then not $X$” (contrapositive)
  • Yes!
**Definition: Mapping Reducibility**

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** from $A$ to $B$.

"forward" direction ($\Rightarrow$): if $w \in A$ then $f(w) \in B$

"reverse" direction ($\Leftarrow$): if $f(w) \in B$ then $w \in A$

Equivalent (contrapositive): if $w \notin A$ then $f(w) \notin B
Proving Mapping Reducibility: 2 Steps

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** from $A$ to $B$.

**Step 1:** Show there is computable fn $f$ ... by creating a TM

**Step 2:** Prove the iff is true

**Step 2a:** “forward” direction ($\Rightarrow$): if $w \in A$ then $f(w) \in B$

**Step 2b:** “reverse” direction ($\Leftarrow$): if $f(w) \in B$ then $w \in A$

**Step 2b:** Equivalent (contrapositive): if $w \notin A$ then $f(w) \notin B

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.
Thm: $A_{TM}$ is mapping reducible to $HALT_{TM}$

- To show: $A_{TM} \leq_m HALT_{TM}$

Step 1: computable fn $f$: $<M, w> \rightarrow <M', w>$ where:

$$\langle M, w \rangle \in A_{TM} \iff \langle M', w' \rangle \in HALT_{TM}$$

The following machine $F$ computes a reduction $f$.

$F = \text{"On input } \langle M, w \rangle:\$

1. Construct the following machine $M'$.
   $M' = \text{"On input } x:\$
   1. Run $M$ on $x$.
   2. If $M$ accepts, accept.
   3. If $M$ rejects, enter a loop.

2. Output $\langle M', w \rangle$.”

Output new $M'$

$M'$ is like $M$, except it always loops when it doesn’t accept

Converting $M$ to $M'$

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 $w$,

$$w \in A \iff f(w) \in B.$$
⇒ If $M$ accepts $w$, then $M'$ halts on $w$

- $M'$ accepts (and thus halts) if $M$ accepts

⇐ If $M'$ halts on $w$, then $M$ accepts $w$

⇐ (Alternatively) If $M$ doesn’t accept $w$, then $M'$ doesn’t halt on $w$ (contrapositive)

- Two possibilities for non-acceptance:
  1. $M$ loops: $M'$ loops and doesn’t halt
  2. $M$ rejects: $M'$ loops and doesn’t halt

The following machine $F$ computes a reduction $f$.

$F =$ “On input $\langle M, w \rangle$:

1. Construct the following machine $M'$.
   $M'$ = “On input $x$:
   1. Run $M$ on $x$.
   2. If $M$ accepts, accept.
   3. If $M$ rejects, enter a loop.”

2. Output $\langle M', w \rangle$.”
Uses of Mapping Reducibility

• To prove Decidability

• To prove Undecidability
**Thm:** If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.

**Proof:** We let $M$ be the decider for $B$ and $f$ be the reduction from $A$ to $B$. We describe a decider $N$ for $A$ as follows.

\[ N = \text{“On input } w:\]

1. Compute $f(w)$.
2. Run $M$ on input $f(w)$ and output whatever $M$ outputs.

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** from $A$ to $B$. 

We know this is true bc of the iff (specifically reverse direction)
Corollary: If \( A \leq_m B \) and \( A \) is undecidable, then \( B \) is undecidable.

- Proof by contradiction.

- Assume \( B \) is decidable.

- Then \( A \) is decidable (by the previous thm).

- Contradiction: we already said \( A \) is undecidable.
Summary: Showing Mapping Reducibility

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** from $A$ to $B$.

**Step 1:** Show there is computable fn $f$ ... by creating a TM

**Step 2:** Prove the iff is true

**Step 2a:** “forward” direction ($\Rightarrow$): if $w \in A$ then $f(w) \in B$

**Step 2b:** “reverse” direction ($\Leftarrow$): if $f(w) \in B$ then $w \in A$

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.

**Step 2b:** Equivalent (contrapositive): if $w \notin A$ then $f(w) \notin B$
**Summary: Using Mapping Reducibility**

To prove decidability ...

- If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.

To prove undecidability ...

- If $A \leq_m B$ and $A$ is undecidable, then $B$ is undecidable.

Be careful with the **direction of the reduction!**
Alternate Proof: The Halting Problem

$\text{HALT}_TM$ is undecidable

- If $A \leq_m B$ and $A$ is undecidable, then $B$ is undecidable.

- $A_{TM} \leq_m \text{HALT}_{TM}$

- Since $A_{TM}$ is undecidable,
  - ... and we showed mapping reducibility from $A_{TM}$ to $\text{HALT}_{TM}$,
  - then $\text{HALT}_{TM}$ is undecidable.

\[ \blacksquare \]
Flashback: $EQ_{TM}$ is undecidable

$EQ_{TM} = \{\langle M_1, M_2 \rangle | M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

Proof by contradiction:

- **Assume** $EQ_{TM}$ has **decider** $R$; use to create $EQ_{TM}$ **decider**:

  $\quad = \{\langle M \rangle | M \text{ is a TM and } L(M) = \emptyset\}$

  $S = \text{“On input } \langle M \rangle, \text{ where } M \text{ is a TM:}

  1. \text{ Run } R \text{ on input } \langle M, M_1 \rangle, \text{ where } M_1 \text{ is a TM that rejects all inputs.}

  2. \text{ If } R \text{ accepts, } \text{accept}; \text{ if } R \text{ rejects, } \text{reject.”}
Alternate Proof: $E_{Q_{TM}}$ is undecidable

$E_{Q_{TM}} = \{\langle M_1, M_2 \rangle | M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

Show mapping reducibility: $E_{TM} \leq_m E_{Q_{TM}}$

**Step 1:** create computable fn $f$: $<M> \rightarrow <M_1, M_2>$, computed by $S$

$$S = \text{“On input } \langle M \rangle, \text{ where } M \text{ is a TM:}$$

1. **Construct:** $\langle M, M_1 \rangle$, where $M_1$ is a TM that rejects all inputs.
2. **Output:** $\langle M, M_1 \rangle$

**Step 2:** show iff requirements of mapping reducibility (exercise)

And use theorem ...

If $A \leq_m B$ and $A$ is undecidable, then $B$ is undecidable.
Flashback: \( E_{TM} \) is undecidable
\[
E_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \}
\]

Proof, by contradiction:
• Assume \( E_{TM} \) has decider \( R \); use to create \( A_{TM} \) decider:

\[
S = \text{"On input } \langle M, w \rangle, \text{ an encoding of a TM } M \text{ and a string } w:\n1. \text{ Use the description of } M \text{ and } w \text{ to construct the TM } M_1
2. \text{ Run } R \text{ on input } \langle M_1 \rangle.
3. \text{ If } R \text{ accepts, reject; if } R \text{ rejects, accept."
}
\]

• So this only reduces \( A_{TM} \) to \( \overline{E_{TM}} \)

If \( M \) accepts \( w \), \( M_1 \) not in \( E_{TM} \)!
Alternate Proof: $E_{TM}$ is undecidable

$E_{TM} = \{ \langle M \rangle | M$ is a TM and $L(M) = \emptyset \}$

Show mapping reducibility??: $A_{TM} \leq_{m} E_{TM}$

**Step 1:** create computable fn $f$: $\langle M, w \rangle \rightarrow \langle M' \rangle$, computed by $S$

$$S = \text{"On input } \langle M, w \rangle, \text{ an encoding of a TM } M \text{ and a string } w:\$$

1. Use the description of $M$ and $w$ to construct the TM $M_1$

$$M_1 = \text{"On input } x:\$$

1. If $x \neq w$, reject.
2. If $x = w$, run $M$ on input $w$ and accept if $M$ does.”

2. **Output:** $\langle M_1 \rangle$.

3. If $R$ accepts, reject; if $R$ rejects, accept.”

• So this only reduces $A_{TM}$ to $\overline{E_{TM}}$

• It’s good enough! Still proves $E_{TM}$ is undecidable
  • Because undecidable langs are closed under complement

Step 2: show iff requirements of mapping reducibility (exercise)
Undecidable Langs Closed under Complement

Proof by contradiction

• **Assume** some lang \( L \) is undecidable and \( \overline{L} \) is decidable ...
  • Then \( \overline{L} \) has a decider
  
  \[ \text{Contradiction!} \]

• ... **then** we can create decider for \( L \) from decider for \( \overline{L} \) ...
  • Because decidable languages are closed under complement (hw8)!
Check-in Quiz 4/4

On gradescope