Rice’s Theorem

- We have seen that the following properties of TM’s are undecidable:
  - TM accepts string w
  - TM halts on input w
  - TM accepts the empty language
  - TM accepts a regular language
- Can we describe a single generic reduction for all these proofs?
- Yes. Every property of TMs undecidable!

Rice’s Theorem: Every nontrivial TM property is undecidable.

Proof:
- reduce from $A_{TM}$ (i.e. show $A_{TM} \leq_{m} P$)
- what should $f(<M, w>)$ produce?
- $f(<M, w>) = <M’>$ described below:

<table>
<thead>
<tr>
<th>on input x,</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept if M accepts w</td>
</tr>
<tr>
<td>and $M’$ accepts x</td>
</tr>
<tr>
<td>(intersection of two RE languages)</td>
</tr>
</tbody>
</table>

- $f$ computable?
- $YES$ maps to $YES$?
  - $<M, w> \in A_{TM} \Rightarrow L(M) \neq L(M’)$
  - $f(M, w) \in P$
Post Correspondence Problem

Proof: reduce from \( A_{TM} \) (i.e. show \( A_{TM} \subseteq PCP \))
- what should \( f(M, w) \) produce?
- \( f(M, w) = \langle M' \rangle \) described below:

• NO maps to NO?
  \( \langle M, w \rangle \in A_{TM} \Rightarrow L(M, w) = L(TM) \Rightarrow f(M, w) \notin P \)

经典例子：Post Correspondence Problem

\[ PCP = \{ \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \ldots, \langle x_n, y_n \rangle \} : \]
\[ x_i, y_i \in \Sigma^* \text{ and there exists } (a_1, a_2, \ldots, a_n) \text{ for which } x_1x_2 \ldots x_n = y_1y_2 \ldots y_n \]

Rice's Theorem

Proof:
- reduce from \( A_{TM} \) (i.e. show \( A_{TM} \subseteq \text{P} \))
- what should \( f(M, w) \) produce?
- \( f(M, w) = \langle M' \rangle \) described below:

• NO maps to NO?
  \( \langle M, w \rangle \in A_{TM} \Rightarrow L(M, w) = L(TM) \Rightarrow f(M, w) \notin P \)

MPCP = \{ \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \ldots, \langle x_n, y_n \rangle \} :
\[ x_i, y_i \in \Sigma^* \text{ and there exists } (a_1, a_2, \ldots, a_n) \text{ for which } x_1x_2 \ldots x_n = y_1y_2 \ldots y_n \]

Proof of MPCP \( \subseteq \text{P} \) PCP:
- notation: for a string \( u = u_1u_2u_3 \ldots u_n \)
  - \( x_u \) means the string \( x_{u_1}x_{u_2}x_{u_3} \ldots x_{u_n} \)
  - \( y_u \) means the string \( y_{u_1}y_{u_2}y_{u_3} \ldots y_{u_n} \)
- given a match in original MPCP instance, can produce a match in the new PCP instance
- for \( N \) maps to NO?
  - given a match in the new PCP instance, can produce a match in the original MPCP instance

Post Correspondence Problem

Theorem: PCP is undecidable.

Proof:
- reduce from \( A_{TM} \) (i.e. show \( A_{TM} \subseteq \text{PCP} \))
- two step reduction makes it easier
- first, show \( A_{TM} \subseteq \text{MPCP} \)
  (MPCP = "modified PCP"
- next, show \( \text{MPCP} \subseteq \text{PCP} \)

- classic example: Post Correspondence Problem

\[ PCP = \{ \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \ldots, \langle x_n, y_n \rangle \} : \]
\[ x_i, y_i \in \Sigma^* \text{ and there exists } (a_1, a_2, \ldots, a_n) \text{ for which } x_1x_2 \ldots x_n = y_1y_2 \ldots y_n \]
### CS21 Lecture 15

#### Post Correspondence Problem

**Theorem:** PCP is undecidable.

**Proof:**
- show $A_{TM} \leq_m$ MPCP
  
  $$\text{MPCP} = \langle (x_i, y_i), (x_0, y_0), \ldots, (x_n, y_n) \rangle$$
  
  where $x_i, y_i \in \Sigma^*$ and there exists $(a_0, a_1, \ldots, a_n)$ for which $x_0x_1x_2\ldots x_n = y_0y_1y_2\ldots y_n$

- show MPCP $\leq_m$ PCP

#### Proof of $A_{TM} \leq_m$ MPCP:
- given instance of $A_{TM}$: $\langle M, w \rangle$
- idea: a match will record an accepting computation history for $M$ on input $w$
- start tile records starting configuration:
  - add tile $(\#, \#qaww\ldots\#w\#)$

- tiles for head motions to the right:
  - for all $a, b \in \Gamma$ and all $q, r \in Q$ with $q \neq q_{start}$, if $\delta(q, a) = (r, b, R)$, add tile $(qa, br)$

- tiles for head motions to the left:
  - for all $a, b, c \in \Gamma$ and all $q, r \in Q$ with $q \neq q_{start}$, if $\delta(q, a) = (r, b, L)$, add tile $(ca, rcb)$

- tiles for copying (not near head)
  - for all $a \in \Gamma$, add tile $(a, a)$

- tiles for copying # marker
  - add tile $(\#, \#)$

- tiles for copying # marker and adding _ to end of tape
  - add tile $(\#, _$)
Post Correspondence Problem

- tiles for deleting symbols to left of $q_{\text{accept}}$
  - for all $a \in \Gamma$, add tile $(a q_{\text{accept}}, q_{\text{accept}})$

- tiles for deleting symbols to right of $q_{\text{accept}}$
  - for all $a \in \Gamma$, add tile $(q_{\text{accept}} a, q_{\text{accept}})$

- tiles for completing the match
  - for all $a \in \Gamma$, add tile $(q_{\text{accept}} a, q_{\text{accept}})$

-- YES maps to YES?
  - by construction, if $M$ accepts $w$, there is a way to assemble the tiles to achieve this match:

- NO maps to NO?
  - sketch: at any step if the "intended" next tile is not used, then it is impossible to recover and produce a match in the end (case analysis)

Beyond RE and co-RE

- We saw (by a counting argument) that there is some language that is neither RE nor co-RE.
- We will prove this for a natural language: $\text{EQ}_{TM} = \{<M_1, M_2> : L(M_1) = L(M_2)\}$
- Recall:
  - $A_{TM}$ is undecidable, but RE
  - $\text{co-A}_{TM}$ is undecidable, but coRE

We have proved:

**Theorem:** PCP is undecidable.

by showing:
- $A_{TM} \leq_{m} \text{MPCP}$
- $\text{MPCP} \leq_{m} \text{PCP}$
- conclude $A_{TM} \leq_{m} \text{PCP}$
Beyond RE and co-RE

**Theorem:** $EQ_{TM}$ is neither RE nor coRE.

**Proof:**
- not RE:
  - reduce from co-A$_{TM}$ (i.e. show co-A$_{TM}$ $\leq_m$ $EQ_{TM}$)
  - what should $f(<M, w>)$ produce?
- not co-RE:
  - reduce from A$_{TM}$ (i.e. show A$_{TM}$ $\leq_m$ $EQ_{TM}$)
  - what should $f(<M, w>)$ produce?

---

Beyond RE and co-RE

**Proof** ($A_{TM}$ $\leq_m$ $EQ_{TM}$)

- $f(<M, w>) = <M_1, M_2>$ described below:

  **TM $M_1$:**
  - on input $x$,
    - accept

  **TM $M_2$:**
  - on input $x$,
    - simulate $M$ on input $w$
    - accept if $M$ accepts $w$

  - YES maps to YES?
    - $<M, w> \in A_{TM}$ $\Rightarrow$ $L(M_1) = \Sigma^*$ and $L(M_2) = \Sigma^*$ $\Rightarrow$ $f(<M, w>) \in EQ_{TM}$
  - NO maps to NO?
    - $<M, w> \notin A_{TM}$ $\Rightarrow$ $L(M_1) = \Sigma^*$ and $L(M_2) = \emptyset$ $\Rightarrow$ $f(<M, w>) \notin EQ_{TM}$

---

Beyond RE and co-RE

**Proof** (co-A$_{TM}$ $\leq_m$ $EQ_{TM}$)

- $f(<M, w>) = <M_1, M_2>$ described below:

  **TM $M_1$:**
  - on input $x$,
    - reject

  **TM $M_2$:**
  - on input $x$,
    - simulate $M$ on input $w$
    - accept if $M$ accepts $w$

  - YES maps to YES?
    - $<M, w> \in$ co-A$_{TM}$ $\Rightarrow$ $L(M_1) = \emptyset$ and $L(M_2) = \emptyset$ $\Rightarrow$ $f(<M, w>) \in EQ_{TM}$
  - NO maps to NO?
    - $<M, w> \notin$ co-A$_{TM}$ $\Rightarrow$ $L(M_1) = \Sigma^*$ and $L(M_2) = \Sigma^*$ $\Rightarrow$ $f(<M, w>) \notin EQ_{TM}$

---

Summary

- regular languages
- context free languages
- all languages
- RE
- PCP
- some language
- co-RE
- co-HALT
- decidable
- $\{a^nb^n: n \geq 0\}$
- HALT
- $\{a^nb^nc^n: n \geq 0\}$

---

The Recursion Theorem

- A very useful, and non-obvious, capability of Turing Machines:
  - in the course of computation, can print out a description of itself!
  - why is this useful?
  - example: slick proof that $A_{TM}$ undecidable
  - assume TM $M$ decides $A_{TM}$
  - construct machine $M'$ as follows:

  **on input $x$:**
  - obtain own description $<M>$
  - run $M$ on input $<M, x>$
  - if $M$ accepts, reject. if $M$ rejects, accept.

  - if $M'$ on input $x$:
    - accepts, then $M$ rejects $<M', x>$, but then $M'$ does not accept!
    - rejects, then $M$ accepts $<M', x>$, but then $M'$ accepts!
The Recursion Theorem

• Lemma: there is a computable function
  \( q: \Sigma^* \rightarrow \Sigma^* \)
such that \( q(w) \) is a description of a TM \( P_w \) that prints out \( w \) and then halts.

• Proof:
  – on input \( w \), construct TM \( P_w \) that has \( w \) hard-coded into it; output \( <P_w> \).

The Recursion Theorem

• Warm-up: produce a TM SELF that prints out its own description.

• Two parts:
  – Part A:
    • output a description of B
    • pass control to B.
  – Part B:
    • prepend a description of A
    • done

The Recursion Theorem

– Part A:
  • output a description of B
  • pass control to B.
– Part B:
  • prepend a description of A
  • done

Note: \( <A> = q(<B>) \)
Recall: \( q(w) \) is a description of a TM \( P_w \) that prints out \( w \) and then halts.

The Recursion Theorem

– watch closely as TM AB runs:
  – A runs. Tape contents: \( <B> \)
  – B runs. Tape contents: \( q(<B>)<B> = <AB> \)
  – AB is our desired machine SELF.