6.045: Automata, Computability, and Complexity
Or, Great Ideas in Theoretical Computer Science
Spring, 2010

Class 8
Nancy Lynch
Today

• More undecidable problems:
  – About Turing machines: Emptiness, etc.
  – About other things: Post Correspondence Problem.

• Topics:
  – Undecidable problems about Turing machines.
  – The Post Correspondence Problem: Definition
  – Computation histories
  – First proof attempt
  – Second attempt: Undecidability of modified PCP (MPCP)
  – Finish undecidability of PCP

• Reading: Sipser Sections 4.2, 5.1.
Undecidable Problems about Turing Machines
Undecidable Problems about Turing Machines

• We already showed that $\text{Acc}_{\text{TM}}$ and $\text{Halt}_{\text{TM}}$ are not Turing-decidable (and their complements are not even Turing-recognizable).

• Now consider some other problems:
  – $\text{Acc}_{\text{TM}}^{01} = \{ <M> \mid M \text{ is a TM that accepts the string } 01 \}$$
  – $\text{Empty}_{\text{TM}} = \{ <M> \mid M \text{ is a TM that accepts no strings}\}$
  – $\text{Reg}_{\text{TM}} = \{ <M> \mid M \text{ is a TM and } L(M) \text{ is regular}\}$
  – $\text{EQ}_{\text{TM}}$, equivalence for TMs, = $\{ <M_1, M_2> \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2) \}$
Acc01_{TM}

- Acc01_{TM} = \{ <M> | M accepts the string 01 \}
- Theorem 1: Acc01_{TM} is not Turing-decidable.
- This might seem surprising---it seems simpler than the general acceptance problem, since it involves just one particular string.

Proof attempt:
- Try a reduction---show if you could decide Acc01_{TM} then you could decide general acceptance problem Acc_{TM}.
- Let R be a TM that decides Acc01_{TM}; design S to decide Acc_{TM}.
- S: On input <M,w>:
  - Run R on <M>.
  - If R accepts… ??? Gives useful information only if w = 01.
  - Doesn’t work.
Acc01_{TM}

• **Theorem 1**: Acc01_{TM} is not Turing-decidable.

• **Proof attempt**:
  - Let R be a TM that decides Acc01_{TM}.
  - S: On input <M,w>:
    • Run R on <M>.
    • If R accepts... ???
    • Doesn’t work.

• How can we use information about what a machine does on 01 to help decide what a given machine M will do on an arbitrary w?

• **Idea**: Consider a different machine---modify M.
Theorem 1: \( \text{Acc}^{01}_{\text{TM}} \) is not Turing-decidable.

Proof:

- Let \( R \) be a TM that decides \( \text{Acc}^{01}_{\text{TM}} \); design \( S \) to decide \( \text{Acc}^{\text{TM}} \).

- \( S \): On input \( \langle M, w \rangle \):
  - Instead of running \( M \) on \( w \), \( S \) constructs a new machine \( M'_{M,w} \) that depends on \( M \) and \( w \).
  - \( M'_{M,w} \): On any input \( x \), ignores \( x \) and runs \( M \) on \( w \).
  - Thus, the new machine is the same as \( M \), but hard-wires in the given input \( w \).

- More precisely:
Theorem 1: \text{Acc01}^\text{TM} \text{ is not Turing-decidable.}

Proof:

– R decides \text{Acc01}^\text{TM}; design S to decide \text{Acc}^\text{TM}.

– S: On input \text{<M,w>}: 

  • Step 1: Construct a new machine \text{<M' M,w>}, where 
    – \text{M' M,w}: On input x:
      • Run M on w and accept/reject if M does.
  
  • Step 2: Run R on \text{<M' M,w>}, and accept/reject if R does.

– Note that S can construct \text{<M' M,w>} algorithmically, from inputs M and w.
Theorem 1: Acc01_{TM} is not Turing-decidable.

Proof:
- R decides Acc01_{TM}; design S to decide Acc_{TM}.
- S: On input <M,w>:
  - Step 1: Construct a new machine <M'_{M,w}>, where
    - M'_{M,w}: On input x:
      - Run M on w and accept/reject if M does.
  - Step 2: Run R on <M'_{M,w}>, and accept/reject if R does.
- Running R on <M'_{M,w}> tells us whether or not M'_{M,w} accepts 01.
- Claim: M'_{M,w} accepts 01 if and only if M accepts w.
  - M'_{M,w} always behaves the same, ignoring its own input and simulating M on w.
  - If M'_{M,w} accepts 01 (or anything else), then M accepts w.
  - If M accepts w, then M'_{M,w} accepts 01 (and everything else).
- So S gives the right answer for whether M accepts w.
Acc01_{TM}

- **Theorem 1**: Acc01_{TM} is not Turing-decidable.
- **Theorem**: Acc01_{TM} is Turing-recognizable.
- **Corollary**: (Acc01_{TM})^c is not Turing-recognizable.
Empty_{TM}

- Empty_{TM} = \{ <M> | M is a TM and L(M) = \emptyset \}
- **Theorem 2:** Empty_{TM} is not Turing-decidable.
- **Proof:**
  - Reduce Acc_{TM} to Empty_{TM}.
  - Modify the given machine M: Given <M,w>, construct a new machine M’ so that asking whether L(M’) = \emptyset gives the right answer to whether M accepts w:
  - Specifically, M accepts w if and only if L(M’) \neq \emptyset.
  - Use the same machine M’ as for Acc01_{TM}.
  - **S:** On input <M,w>:
    - Step 1: Construct <M’_{M,w}> as before, which acts on every input just like M on w.
    - Step 2: Ask whether L(M’_{M,w}) = \emptyset and output the opposite answer.
Empty$_{\text{TM}}$

• **Theorem 2**: Empty$_{\text{TM}}$ is not Turing-decidable.
• **Proof**:
  – Reduce Acc$_{\text{TM}}$ to Empty$_{\text{TM}}$.
  – S: On input $<M,w>$:
    • Step 1: Construct $<M'_M,w>$ as before, which acts on every input just like $M$ on $w$.
    • Step 2: Ask whether $L(M'_M,w) = \emptyset$ and output the opposite answer.
  – Now $M$ accepts $w$
    if and only if $M'_M,w$ accepts everything
    if and only if $M'_M,w$ accepts something
    if and only if $L(M'_M,w) \neq \emptyset$.
  – So $S$ decides Acc$_{\text{TM}}$, contradiction.
  – So Empty$_{\text{TM}}$ is not Turing-decidable.
• Theorem 2: Empty$_{TM}$ is not Turing-decidable.
• Theorem: (Empty$_{TM}$)$^c$ is Turing-recognizable.
• Proof: On input $<M>$, run M on all inputs, dovetailed, accept if any accept.
• Corollary: Empty$_{TM}$ is not Turing-recognizable.
Reg\textsubscript{TM}  
\begin{itemize}  
\item \(\text{Reg}_{\text{TM}} = \{ <M> \mid M \text{ is a TM and } L(M) \text{ is regular} \}\)  
\item That is, given a TM, we want to know whether its language is also recognized by some DFA.  
\item For some, the answer is yes: TM that recognizes \(0^*1^*\)  
\item For some, no: TM that recognizes \(\{0^n1^n \mid n \geq 0 \}\)  
\item We can prove that there is no algorithm to decide whether the answer is yes or no.  
\item **Theorem 3:** \(\text{Reg}_{\text{TM}}\) is not Turing-decidable.  
\item **Proof:**  
\begin{itemize}  
\item Reduce \(\text{Acc}_{\text{TM}}\) to \(\text{Reg}_{\text{TM}}\).  
\item Assume TM \(R\) that decides \(\text{Reg}_{\text{TM}}\), design \(S\) to decide \(\text{Acc}_{\text{TM}}\).  
\item \(S\): On input \(<M,w>\):  
\begin{itemize}  
\item Step 1: Construct a new machine \(<M'_{M,w}>\) that accepts a regular language if and only if \(M\) accepts \(w\).  
\item Tricky…  
\end{itemize}  
\end{itemize}  
\end{itemize}
\textbf{Reg}_{TM}

- $\text{Reg}_{TM} = \{ \langle M \rangle \mid L(M) \text{ is regular} \}$
- **Theorem 3:** $\text{Reg}_{TM}$ is not Turing-decidable.
- **Proof:**
  - Assume $R$ decides $\text{Reg}_{TM}$, design $S$ to decide $\text{Acc}_{TM}$.
  - **$S$:** On input $\langle M, w \rangle$:
    - **Step 1:** Construct a new machine $\langle M'_{M,w} \rangle$ that accepts a regular language if and only if $M$ accepts $w$.
      - $M'_{M,w}$: On input $x$:
        - If $x$ is of the form $0^n1^n$, then accept.
        - If $x$ is not of this form, then run $M$ on $w$ and accept if $M$ accepts.
    - **Step 2:** Run $R$ on input $\langle M'_{M,w} \rangle$, and accept/reject if $R$ does.
Theorem 3: $\text{Reg}_{TM}$ is not Turing-decidable.

Proof:
- $S$: On input $<M, w>$:
  - Step 1: Construct a new machine $< M'_{M,w} >$ that accepts a regular language if and only if $M$ accepts $w$.
    - $M'_{M,w}$: On input $x$:
      - If $x$ is of the form $0^n1^n$, then accept.
      - If $x$ is not of this form, then run $M$ on $w$ and accept if $M$ accepts.
  - Step 2: Run $R$ on input $< M'_{M,w} >$, and accept/reject if $R$ does.
    - If $M$ accepts $w$, then $M'_{M,w}$ accepts everything, hence recognizes the regular language $\{0,1\}^*$.
    - If $M$ does not accept $w$, then $M'_{M,w}$ accepts exactly the strings of the form $0^n1^n$, which constitute a non-regular language.
    - Thus, $M$ accepts $w$ iff $M'_{M,w}$ recognizes a regular language.
And more questions

- Many more questions about what TMs compute can be proved undecidable using the same method.

- One more example: \( \text{EQ}_{\text{TM}} = \{<M_1, M_2> | M_1 \text{ and } M_2 \text{ are basic TMs that recognize the same language} \} \)

- **Theorem 4**: \( \text{EQ}_{\text{TM}} \) is not Turing-decidable.

- **Proof**:
  - Reduce \( \text{Empty}_{\text{TM}} \) to \( \text{EQ}_{\text{TM}} \).
  - Assume R is a TM that decides \( \text{EQ}_{\text{TM}} \); design S to decide \( \text{Empty}_{\text{TM}} \).
  - Define any particular TM \( M_\emptyset \) with \( L(M) = \emptyset \) (M accepts nothing).
  - S: On input \(<M>\):
    - Run R on input \(<M, M_\emptyset>\); accept/reject if R does.
  - R tells whether \(<M, M_\emptyset> \in \text{EQ}_{\text{TM}} \), that is, whether \( L(M) = L(M_\emptyset) = \emptyset \).
An Undecidable Problem not involving Turing Machines
Post Correspondence Problem

• A simple string-matching problem.
• Given a finite set of “tile types”, e.g.:

\[
\begin{align*}
T & = \{ a \}, & c & a b, & b & c, & b & d \\
& \{ a & b \}, & c & a b, & b & c, & b & d \\
\end{align*}
\]

• Is there a nonempty finite sequence of tiles (allowing repetitions, and not necessarily using all the tile types) for which the concatenation of top strings = concatenation of bottom strings?
• Example: 

\[
\begin{align*}
& a & b \quad b & d \\
& a & b \quad b & d
\end{align*}
\] 

or

\[
\begin{align*}
& a & b \quad b & d \\
& a & b \quad b & d
\end{align*}
\] 

• No limit on length, but must be finite.
• Call such a sequence a match, or correspondence.
• Post Correspondence Problem (PCP) = 

\{ < T > | T is a finite set of tile types that has a match \}
Post Correspondence Problem

- Given a finite set of tile types, is there a nonempty finite sequence of tiles for which the concatenation of top strings = concatenation of bottom strings?
- Call sequence a match, or correspondence.
- Post Correspondence Problem (PCP) = \{ < T > | T is a finite set of tile types that has a match \}.
- Theorem: PCP is undecidable.
- Proof:
  - Reduce Acc_{TM} to PCP.
  - Previous reductions involved reducing one question about TMs (usually Acc_{TM}) to another question about TMs.
  - Now we reduce TM acceptance to a question about matching strings.
  - Do this by encoding TM computations using strings…
Computation Histories
Computation Histories

• **Computation History (CH):** A formal, stylized way of representing the computation of a TM on a particular input.

• **Configuration:**
  – Instantaneous snapshot of the TM’s computation.
  – Includes current state, current tape contents, current head position.
  – Write in standard form: \( w_1 q w_2 \), where \( w_1 \) and \( w_2 \) are strings of tape symbols and \( q \) is a state.
  – **Meaning:**
    • \( w_1 w_2 \) is the string on the non-blank portion of the tape, perhaps part of the blank portion (rest assumed blank).
    • \( w_1 \) is the portion of the string strictly to the left of the head.
    • \( w_2 \) is the portion directly under the head and to the right.
    • \( q \) is the current state.
Configurations

• **Configuration:**
  - $w_1 \ q \ w_2$, where $w_1$ and $w_2$ are strings of tape symbols and $q$ is a state.

• **Meaning:**
  - $w_1 \ w_2$ is the string on the non-blank portion of the tape, perhaps part of the blank portion (rest assumed blank).
  - $w_1$ is the portion of the string strictly to the left of the head.
  - $w_2$ is the portion directly under the head and to the right.
  - $q$ is the current state.

• **Example:** 0011q01 represents TM configuration:

```
0 0 1 1 0 1
```

FSC in state $q$
Computation Histories

• TM begins in a starting configuration, of the form \(q_0 \, w\), where \(w\) is the input string, and moves through a series of configurations, following the transition function.

• Computation History of TM \(M\) on input \(w\):
  – A (finite or infinite) sequence of configs \(C_1, C_2, C_3, \ldots, C_k, \ldots\), where
    • \(C_1, C_2, \ldots\) are configurations of \(M\).
    • \(C_1\) is the starting configuration with input \(w\).
    • Each \(C_{i+1}\) follows from \(C_i\) using \(M\)’s transition function.

• Accepting CH: Finite CH ending in accepting configuration.
• Rejecting CH: Finite CH ending in rejecting configuration.
• Represent CH as a string \(# \, C_1 \, # \, C_2 \, # \, \ldots \, # \, C_k \, #\), where \(#\) is a special separator symbol.
• Claim: \(M\) accepts \(w\) iff there is an accepting CH of \(M\) on \(w\).
Undecidability of PCP: First Attempt
First attempt

• **Theorem:** PCP is undecidable.

• **Proof attempt:**
  – Reduce $\text{Acc}_M$ to PCP, that is, show that, if we can decide PCP, then we can decide $\text{Acc}_M$.
  – Given $<M,w>$, construct a finite set $T_{M,w}$ of tile types such that $T_{M,w}$ has a match iff $M$ accepts $w$.
  – That is, $T_{M,w}$ has a match iff there is an accepting CH of $M$ on $w$.
  – Write the accepting CH twice:
    
    # $C_1$ # $C_2$ # $C_3$ # … # $C_k$ #
    # $C_1$ # $C_2$ # $C_3$ # … # $C_k$ #
  – Split along boundaries of successive configurations:
    
    # \[ \begin{array}{|c|c|c|c|c|c|} \hline
    C_1 & C_2 & C_3 & … & C_k \\
    \hline
    C_1 & C_2 & C_3 & … & C_k \\
    \hline
    \end{array} \]
First attempt

• Given \(<M,w>\), construct a finite set \(T_{M,w}\) of tile types s.t. \(T_{M,w}\) has a match iff there is an accepting CH of \(M\) on \(w\).

• Write the accepting CH twice, and split along boundaries of successive configurations:

\[
\begin{array}{cccccc}
\# & C_1 & \# & C_2 & \# & C_3 & \# & \ldots & \# & C_k & \#
\end{array}
\]

\[
\begin{array}{cccccc}
\# & C_1 & \# & C_2 & \# & C_3 & \# & \ldots & \# & C_k & \#
\end{array}
\]

– What tiles do we need?
– Try \(T_{M,w} = \) \[
\left\{ \begin{array}{ccc}
\# & C_k & \#
\end{array} \right\}
\]

where

\[
\left\{ \begin{array}{ccc}
\# & C_1 & \#
\end{array} \right\}
\]

• \(C_1 = \) starting configuration for \(M\) on \(w\),
• \(C_k = \) accepting configuration (can assume unique, because we can assume accepting machine cleans up its tape).
• \(C_j\) follows from \(C_i\) by rules of \(M\) (one step).
First attempt

\[ T_{M,w} = \left\{ \left( \begin{array}{c} \# \\ \# C_1 \end{array} \right), \left( \begin{array}{c} C_k \\ \# \end{array} \right), \left( \begin{array}{c} C_i \\ \# C_j \end{array} \right) \right\} \]

- \( C_1 \) = starting configuration for \( M \) on \( w \),
- \( C_k \) = accepting configuration.
- \( C_j \) follows from \( C_i \) by rules of \( M \) (one step).

- \( M \) accepts \( w \) iff \( T_{M,w} \) has a match.

But there is a problem:

- \( T_{M,w} \) has infinitely many tile types \( T_{M,w} \), because \( M \) has infinitely many configurations.
- Configuration has tape contents, state, head position---infinitely many possibilities.
- Of course, in any particular accepting computation, only finitely many configurations appear.
- But we don’t know what these are ahead of time.
- So we can’t pick a single finite set of tiles.
First attempt

• M accepts w iff $T_{M,w}$ has a match.
• But:
  – $T_{M,w}$ has infinitely many tile types $T_{M,w}$, because M has infinitely many configurations.
  – In any particular accepting computation, only finitely many configurations appear.
  – But we can’t pick a single finite set of tiles for all computations.
• New insight:
  – Represent infinitely many configurations with finitely many tiles.
  – Going from one configuration to the next involves changing only a few “local” things:
    • State
    • Contents of one tape cell
    • Position of head, by at most 1
  – So let tiles represent small pieces of configs, not entire configs.
Undecidability of Modified PCP
Undecidability of Modified PCP

• **Modified PCP (MPCP):** Like PCP, but we’re given not just a finite set of tiles, but also a designated tile that must start the match.

• **MPCP = { <T, t > | T is a finite set of tiles, t is a tile in T, and there is a match for T starting with t }**.

• **Theorem:** MPCP is undecidable.

• Later, we remove the requirement to start with t:

• **Theorem:** PCP is undecidable.

• **Proof:**
  – By reducing MPCP to PCP.
  – If PCP were decidable, MPCP would be also, contradiction.
MPCP is undecidable

- Reduce $\text{Acc}_{TM}$ to MPCP.
- Given $<M,w>$, construct $(T_{M,w}, t_{M,w})$, an instance of MPCP.
- 7 kinds of tiles:
  - Type 1 tile:
    \[
    \begin{array}{cc}
    \# & \\
    # \  q_0 \  w_1 \  w_2 \  \ldots \  w_n \  \# & \\
    \end{array}
    \]
    - $w = w_1 \ w_2 \ \ldots \ w_n$
    - $q_0 \ w_1 \ w_2 \ \ldots \ w_n$ is the starting configuration for input $w$.
    - Bottom string is long, but there’s only one tile like this.
    - Tile depends on $w$, which is OK.
    - Make this the initial tile $t_{M,w}$. 
MPCP is undecidable

• Now consider how M goes from one configuration to the next.
• E.g., by moving right: $\delta(q,a) = (q',b,R)$.
• Config changes using this transition look like (e.g.):
  – $w_1 w_2 q a w_3 \rightarrow w_1 w_2 b q' w_3$.
  – Only change is to replace $q a$ by $b q'$.
• Type 2 tiles:
  – For each transition of the form $\delta(q,a) = (q',b,R)$:

\[
\begin{pmatrix}
q a \\
b q'
\end{pmatrix}
\]
MPCP is undecidable

• E.g., moving left: \( \delta(q,a) = (q',b,L) \).

• **Type 3 tile:**
  – For each transition of the form \( \delta(q,a) = (q',b,L) \), and every symbol \( c \) in the tape alphabet \( \Gamma \):

    \[
    \begin{array}{ccc}
    c & q & a \\
    q' & c & b \\
    \end{array}
    \]

  – Include arbitrary \( c \) because it could be anything.

• Notice, only finitely many tiles (so far).
MPCP is undecidable

• Now, to match unchanged portions of 2 consecutive configurations:

• Type 4 tile:
  – For every symbol a in the tape alphabet \( \Gamma \):

\[
\begin{array}{c}
\text{a} \\
\text{a} \\
\text{a}
\end{array}
\]

• Still only finitely many tiles.
MPCP is undecidable

• What can we do with the tiles we have so far?
• Example: Partial match
  – Suppose the starting configuration is $q_0\ 1\ 1\ 0$ and the first move is $(q_0, 1) \rightarrow (q_4, 0, R)$.
  – Then the next configuration is $0\ q_4\ 1\ 0$.
  – We can start the match with tile 1:
    $$
    \begin{array}{c}
    \text{#} \\
    \text{q}_0 110 \text{#}
    \end{array}
    $$
  
  – Continue with type 2 tile:
    $$
    \begin{pmatrix}
    q_0 & 1 \\
    0 & q_4
    \end{pmatrix}
    $$
  
  – Use type 4 tiles for the 2 unchanged symbols:
  
  – Yields:
    $$
    \begin{array}{c}
    \text{#} \\
    \text{q}_0\ 1\ 1\ 0\ \# \\
    \text{#} \\
    \text{q}_0\ 1\ 1\ 0\ \# \\
    \text{0}\ q_4\ 1\ 0\ \# \\
    \text{#} \\
    \text{#}
    \end{array}
    $$
MPCP is undecidable

• Now we put in the separators.
• Type 5 tiles:

\[
\begin{array}{c}
# \\
# \\
\end{array}
\quad \begin{array}{c}
# \\
--# \\
\end{array}
\]

Allows us to add extra spaces at right end as needed---lets the configuration size grow.

• Example: Extend previous match:

\[
\begin{array}{cccccc}
# & q_0 & 1 & 1 & 0 & # \\
# & q_0 & 1 & 1 & 0 & # & 0 & q_4 & 1 & 0 & # \\
\end{array}
\]
MPCP is undecidable

• How does this end?

• Type 6 tiles:
  – For every \( a \) in \( \Gamma \):
    – A trick…
    – Adds “pseudo-steps” to the end of the computation, where the state “eats” adjacent symbols in the top row.
    – Yields one symbol less in each successive bottom configuration.
  – Do this until the remaining bottom “configuration” is \( q_{\text{acc}} \# \):

\[
\begin{array}{c}
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\ldots \# \quad \ldots \# \\
\end{array}
\]

\[
\begin{array}{c}
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\ldots \# \quad \ldots \# \\
\end{array}
\]

\[
\begin{array}{c}
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\ldots \# \quad \ldots \# \\
\end{array}
\]

\[
\begin{array}{c}
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\ldots \# \quad \ldots \# \\
\end{array}
\]

\[
\begin{array}{c}
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\ldots \# \quad \ldots \# \\
\end{array}
\]

\[
\begin{array}{c}
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\ldots \# \quad \ldots \# \\
\end{array}
\]

\[
\begin{array}{c}
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\ldots \# \quad \ldots \# \\
\end{array}
\]

\[
\begin{array}{c}
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\ldots \# \quad \ldots \# \\
\end{array}
\]

\[
\begin{array}{c}
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\ldots \# \quad \ldots \# \\
\end{array}
\]

\[
\begin{array}{c}
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\ldots \# \quad \ldots \# \\
\end{array}
\]

\[
\begin{array}{c}
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\ldots \# \quad \ldots \# \\
\end{array}
\]

\[
\begin{array}{c}
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\ldots \# \quad \ldots \# \\
\end{array}
\]

\[
\begin{array}{c}
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\ldots \# \quad \ldots \# \\
\end{array}
\]

\[
\begin{array}{c}
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\vdots \# \quad \vdots \# \\
\ldots \# \quad \ldots \# \\
\end{array}
\]
MPCP is undecidable

• To finish off:
• Type 7 tile:
\[
\left\{ \begin{array}{c}
q_{\text{acc}} \\
\#
\end{array} \right. \\
\#
\]

• That completes the definition of \( T_{M,w} \) and \( t_{M,w} \).
• Note that \( T_{M,w} \), for a given \( M \) and \( w \), is a finite set of tiles.
MPCP is undecidable

- That completes the definition of $T_{M,w}$ and $t_{M,w}$.
- Note that $T_{M,w}$, for a given $M$ and $w$, is a finite set of tiles.
- Why does this work?
- Must show:
  - If $M$ accepts $w$, then $T_{M,w}$ has a match beginning with $t_{M,w}$, that is, $<T_{M,w}, t_{M,w}> \in \text{MPCP}$.
  - If $<T_{M,w}, t_{M,w}> \in \text{MPCP}$, then $M$ accepts $w$.

- If $M$ accepts $w$, then there is an accepting computation history, which can be described by a match using the given tiles, starting from the distinguished initial tile:
MPCP is undecidable

• If M accepts w, then there is an accepting computation history, which can be described by a match using the given tiles, starting from the distinguished initial tile:

\[
\begin{array}{c}
\# \ C_1 \ # \ C_2 \ # \ C_3 \ # \ldots \ # \ C_k \ # \ C_{k+1} \ # \ldots \ # \ q_{\text{acc}} \ a \ # \ q_{\text{acc}} \ # \ # \\
\# \ C_1 \ # \ C_2 \ # \ C_3 \ # \ldots \ # \ C_k \ # \ C_{k+1} \ # \ldots \ # \ q_{\text{acc}} \ a \ # \ q_{\text{acc}} \ # \ #
\end{array}
\]

Accepting configuration  Shrink by one symbol  Keep shrinking  Final, special tile

• So $T_{M,w}$ has a match beginning with $t_{M,w}$, that is, $<T_{M,w}, t_{M,w}> \in \text{MPCP}$.
MPCP is undecidable

- If $<T_{M,w}, t_{M,w}> \in \text{MPCP}$, that is, if $T_{M,w}$ has a match beginning with the designated tile $t_{M,w}$, then $M$ accepts $w$.

- The rules are designed so the only way we can get a match beginning with the designated tile:

\[
\begin{array}{c}
\# \\
\# q_0 w_1 w_2 \ldots w_n \# 
\end{array}
\]

is to have an actual accepting computation of $M$ on $w$. Hand-wave, in the book, LTTR.

- Combining the two directions, we get:

\[ M \text{ accepts } w \iff <T_{M,w}, t_{M,w}> \in \text{MPCP}, \text{ that is, } <M, w> \in \text{Acc}_M \iff <T_{M,w}, t_{M,w}> \in \text{MPCP}. \]
MPCP is undecidable

• \(<M, w> \in \text{Acc}_\text{TM}\) iff \(<T_{M,w}, t_{M,w}> \in \text{MPCP}\).

• Theorem: MPCP is undecidable.

• Proof:
  – By contradiction.
  – Assume MPCP is decidable, and decide \(\text{Acc}_\text{TM}\), using S:
    – S: On input \(<M, w>:\)
      • Step 1: Construct \(<T_{M,w}, t_{M,w}>\), instance of MPCP, as described.
      • Step 2: Use MPCP to decide if \(T_{M,w}\) has a match beginning with \(t_{M,w}\). If so, accept; if not, reject.
  – Thus, if MPCP is decidable, then also \(\text{Acc}_\text{TM}\) is decidable, contradiction.
Undecidability of (Unmodified) PCP
Undecidability of PCP

• We showed that MPCP, in which the input is a set of tiles + designated input tile, is undecidable, by reducing Acc$_{TM}$ to MPCP.
• Now we want:
• Theorem: PCP is undecidable.
• Why doesn’t our construction reduce Acc$_{TM}$ to PCP?
• $T_{M,v}$ has trivial matches, e.g., just

\[
\begin{array}{c}
\text{a} \\
\text{a}
\end{array}
\]

• Proof of the theorem:
  – To show that PCP is undecidable, reduce MPCP to PCP, that is, show that if PCP is decidable, then so is MPCP.
Undecidability of PCP

• **Theorem:** PCP is undecidable.
• **Proof:**
  – Reduce MPCP to PCP.
  – To decide MPCP using PCP, suppose we are given:
    • **T:** \[
    \begin{pmatrix}
    u_1 \\
    v_1
    \end{pmatrix}
    \quad \begin{pmatrix}
    u_2 \\
    v_2
    \end{pmatrix}
    \quad \ldots 
    \quad \begin{pmatrix}
    u_k \\
    v_k
    \end{pmatrix}
    \]
    • **t:** \[
    \begin{pmatrix}
    u_1 \\
    v_1
    \end{pmatrix}
    \]
  – We want to know if there is a match beginning with **t**.
  – Construct an instance **T’** of ordinary PCP that has a match (starting with any tile) iff **T** has a match starting with **t**.
Undecidability of PCP

- Given T: \[ \begin{pmatrix} u_1 \\ v_1 \end{pmatrix}, \begin{pmatrix} u_2 \\ v_2 \end{pmatrix}, \ldots, \begin{pmatrix} u_k \\ v_k \end{pmatrix} \] t: \[ \begin{pmatrix} u_1 \\ v_1 \end{pmatrix} \]
- Construct an instance T’ of PCP that has a match iff T has a match starting with t.
- Construction (technical):
  - Add 2 new alphabet symbols, ♠ and ♦
  - If \( u = u_1 u_2 \ldots u_n \) then define:
    - \( ♠ u = ♠ u_1 ♠ u_2 \ldots ♠ u_n \)
    - \( u ♠ = u_1 ♠ u_2 \ldots ♠ u_n \)
    - \( ♠ u ♠ = ♠ u_1 ♠ u_2 \ldots ♠ u_n \)
  - Instance T’ of PCP:
    \[ \begin{pmatrix} ♠ u_1 \\ ♠ v_1 ♠ \end{pmatrix}, \begin{pmatrix} ♠ u_1 \\ ♠ v_1 ♠ \end{pmatrix}, \begin{pmatrix} ♠ u_2 \\ ♠ v_2 ♠ \end{pmatrix}, \ldots, \begin{pmatrix} ♠ u_k \\ ♠ v_k ♠ \end{pmatrix}, \begin{pmatrix} ♦ \end{pmatrix} \]
Undecidability of PCP

• **Claim:** $T$ has a match starting with $t$ iff $T'$ has any match.

$\implies$ Suppose $T$ has a match starting with $t$:

Mimic this match with $T'$ tiles, starting with $u_1$
and ending with $v_1$

Yields the same matching strings, with $\heartsuit$s interspersed, and with $\spadesuit$ at the end.

$\Leftarrow$ If $T'$ has any match, it must begin with $u_1$
because that’s the only tile in which top and bottom start with the same symbol.

Other tiles are like $T$ tiles but with extra $\heartsuit$s.
Stripping out $\heartsuit$s yields match for $T$ beginning with $t$. 
Undecidability of PCP

• So, to decide MPCP using a decider for PCP:

• Given instance <T, t> for MPCP,
  – Step 1: Construct instance T’ for PCP, as above.
  – Step 2: Ask decider for PCP whether T’ has any match.
    • If so, answer yes for <T, t>.
    • If not, answer no.

• Since we already know MPCP is undecidable, so is PCP.
Next time…

• Mapping reducibility
• Rice’s Theorem
• **Reading:**
  – Sipser Section 5.3, Problems 5.28-5.30.