Outline

- Post Correspondence Problem (skip)
- Beyond RE and co-RE
- Recursion Theorem
- On to complexity...

Post Correspondence Problem

- many undecidable problems unrelated to TMs and automata
- classic example: Post Correspondence Problem
  \[ \text{PCP} = \{<(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)> : \]
  \[x_i, y_i \in \Sigma^* \text{ and there exists } (a_1, a_2, \ldots, a_n) \text{ for which } x_1x_2\ldots x_n = y_1y_2\ldots y_n\}\]

**Theorem:** PCP is undecidable.

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

Proof of MPCP \leq_{m} PCP:
- notation: for a string \( u = u_1u_2u_3\ldots u_m \)
  - \( *u \) means the string \( *u_1*u_2*u_3\ldots*u_m \)
  - \( |u| \) means the string \( u_1u_2u_3\ldots u_m \)
  - \( \hat{u} \) means the string \( u_1u_2u_3\ldots u_m^* \)
Proof of $\text{MPCP} \leq_m \text{PCP}$:
- given an instance $(x_1, y_1), \ldots, (x_n, y_n)$ of PCP
- produce an instance of PCP:
  \((x_1, x_i) \cdot (y_1, y_i) \cdot \ldots \cdot (x_n, y_n)\)
- YES maps to YES?
  - given a match in original MPCP instance, can produce a match in the new PCP instance
- NO maps to NO?
  - given a match in the new PCP instance, can produce a match in the original MPCP instance

Proof of $A_{TM} \leq_m \text{MPCP}$:
- given instance of $A_{TM}$: $<M, w>$
- idea: a match will record an accepting computation history for $M$ on input $w$
- start tile records starting configuration:
  - add tile \(#, #q_1, w_1, w_2, \ldots, w_n, \#\)
  - symbols must align

Theorem: PCP is undecidable.
Proof:
- show $A_{TM} \leq_m \text{MPCP}$
  - $\text{MPCP} = \{<x_1, y_1>, \ldots, (x_n, y_n)\} : x_i, y_i \in \Sigma^*$ and there exists \((a_1, a_2, \ldots, a_n)\) for which $x_1 x_2 \ldots x_n = y_1 y_2 \ldots y_n$
- show $\text{PCP} \leq_m \text{MPCP}$
Post Correspondence Problem

- 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 \((#, _)\)

\[
\begin{array}{ccc}
# & \#_\text{q}_{0} & \#_\text{w}_1 \\
# & \#_\text{w}_2 & \#_\text{w}_3 \\
\vdots & \vdots & \vdots \\
# & \#_\text{w}_n & #
\end{array} =
\begin{array}{ccc}
#C_1# & \#C_2# & \ldots & \#C_m#
\end{array}
\]

Post Correspondence Problem

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

\[
\begin{array}{ccc}
# & \#_\text{u} & \#_\text{q}_{\text{accept}} \\
# & \#_\text{v} & \#_\text{q}_{\text{accept}} \\
\vdots & \vdots & \vdots \\
# & \#_\text{q}_{\text{accept}} & \#_\text{v}
\end{array} =
\begin{array}{ccc}
#_\text{C} & \#C_1 & \ldots & \#C_m
\end{array}
\]

Post Correspondence Problem

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

\[
\begin{array}{ccc}
# & \#_\text{q}_{\text{accept}} & \#_\text{v} \\
# & \#_\text{q}_{\text{accept}} & \#_\text{v} \\
\vdots & \vdots & \vdots \\
# & \#_\text{q}_{\text{accept}} & \#_\text{v}
\end{array} =
\begin{array}{ccc}
#_\text{C} & \# & \ldots & \#C_m
\end{array}
\]

Post Correspondence Problem

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

\[
\begin{array}{ccc}
# & \#_\text{q}_{\text{accept}} & \#_\text{v} \\
# & \#_\text{q}_{\text{accept}} & \#_\text{v} \\
\vdots & \vdots & \vdots \\
# & \#_\text{q}_{\text{accept}} & \#_\text{v}
\end{array} =
\begin{array}{ccc}
#_\text{C} & \# & \ldots & \#C_m
\end{array}
\]

We have proved:

**Theorem**: PCP is undecidable.

by showing:
- \(A_{\text{TM}} \leq_m \text{MPCP}\)
- \(\text{MPCP} \leq_m \text{PCP}\)
- conclude \(A_{\text{TM}} \leq_m \text{PCP}\)
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: \( \text{EQ}_{\text{TM}} = \{<M_1, M_2> : L(M_1) = L(M_2)\} \)

• Recall:
  – \( A_{\text{TM}} \) is undecidable, but RE
  – \( \text{co-A}_{\text{TM}} \) is undecidable, but coRE

Therefore, not in \( \text{co-RE} \)
Therefore, not in \( \text{RE} \)

Theorem: \( \text{EQ}_{\text{TM}} \) is neither RE nor coRE.

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

– not co-RE:
  • reduce from \( A_{\text{TM}} \) (i.e. show \( A_{\text{TM}} \leq_m \text{EQ}_{\text{TM}} \))
  • what should \( f(<M, w>) \) produce?

Proof (\( A_{\text{TM}} \leq_m \text{EQ}_{\text{TM}} \))
– \( f(<M, w>) = <M_1, M_2> \) 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?
    \(<M, w> \in A_{\text{TM}} \Rightarrow L(M_1) = \Sigma^* \)
    \( \Rightarrow f(<M, w>) \in \text{EQ}_{\text{TM}} \)

  • NO maps to NO?
    \(<M, w> \not\in A_{\text{TM}} \Rightarrow L(M_1) = \emptyset \)
    \( \Rightarrow f(<M, w>) \not\in \text{EQ}_{\text{TM}} \)

Proof (\( \text{co-A}_{\text{TM}} \leq_m \text{EQ}_{\text{TM}} \))
– \( f(<M, w>) = <M_1, M_2> \) 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?
    \(<M, w> \in \text{co-A}_{\text{TM}} \Rightarrow L(M_1) = \emptyset \)
    \( \Rightarrow f(<M, w>) \in \text{EQ}_{\text{TM}} \)

  • NO maps to NO?
    \(<M, w> \not\in \text{co-A}_{\text{TM}} \Rightarrow L(M_1) = \Sigma^* \)
    \( \Rightarrow f(<M, w>) \not\in \text{EQ}_{\text{TM}} \)

Summary

- A very useful, and non-obvious, capability of Turing Machines:
  – in the course of computation, can print out a description of itself!

- how is this possible?
  – an example of a program that prints out self:
    
    Print two copies of the following, the 2\textsuperscript{nd} one in quotes:
    "Print two copies of the following, the 2\textsuperscript{nd} one in quotes:"
The Recursion Theorem

- 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 $M$ rejects, accept; if $M$ accepts, reject.

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 $P_w$ 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

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

- Output:
  - $<B>$
  - read contents of tape
  - apply $q$ to it
  - prepend result to tape

Combine with description on tape to produce a complete TM $AB$.

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 $P_w$ 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

Note: <A> = q(<B>)

Recall: q(w) is a description of a TM P_w that prints out w and then halts.

Theorem: Let T be a TM that computes fn:
\[ t: \Sigma^* \times \Sigma^* \rightarrow \Sigma^* \]
There is a TM R that computes the fn:
\[ r: \Sigma^* \rightarrow \Sigma^* \]
defined as \( r(w) = t(w, <R>). \)

This allows “obtain own description” as valid step in TM program
- first modify TM so that it takes an additional input (that is own description); use at will

Proof outline: TM R has 3 parts
Part A: output description of BT
Part B: prepend description of A
Part “T”: run TM T

Proof details: TM R has 3 parts
Part A: output description of BT
  • <A> = q(<BT>)
Part B: prepend description of A
  • read contents of tape <BT>
  • apply q to it q(<BT>) = <A>
  • prepend to tape <ABT>
Part “T”: run TM T
  • 2nd argument on tape is description of R

*combine with description on tape to produce a complete TM
Summary

• full-fledged model of computation: TM
• many equivalent models
• Church-Turing Thesis
• encoding of inputs
• Universal TM

Summary

• classes of problems:
  – decidable ("solvable by algorithms")
  – recursively enumerable (RE)
  – co-RE
• counting:
  – not all problems are decidable
  – not all problems are RE

Summary

• diagonalization: HALT is undecidable
• reductions: other problems undecidable
  – many examples
  – Rice's Theorem
• natural problems that are not RE
• Recursion Theorem: non-obvious capability of TMs: printing out own description
• Incompleteness Theorem

Complexity

• So far we have classified problems by whether they have an algorithm at all.
• In real world, we have limited resources with which to run an algorithm:
  – one resource: time
  – another: storage space
• need to further classify decidable problems according to resources they require

Complexity

• Complexity Theory = study of what is computationally feasible (or tractable) with limited resources:
  – running time
  – storage space
  – number of random bits
  – degree of parallelism
  – rounds of interaction
  – others…

Worst-case analysis

• Always measure resource (e.g. running time) in the following way:
  – as a function of the input length
  – value of the fn. is the maximum quantity of resource used over all inputs of given length
  – called "worst-case analysis"
• "input length" is the length of input string, which might encode another object with a separate notion of size