Computation Histories and PCP
(not Probabilistically Checkable Proofs)
(also not Angel Dust)

CS 81: Computability and Logic
April 20, 2011
Computation History

- Step-by-step recording of a TM computation.
- Used to show more problems not decidable.
States as Strings

✓ We can describe the configuration of any TM using a string \( C = xqy \in \Gamma^* \).

✓ \( x \in \Gamma^* \) = symbols to the left of head

✓ \( q \in Q \) = current control state

✓ \( y \in \Gamma^* \) = symbols under and to the right of head

✓ Over the course of a computation, we have

✓ \( \cdots \Rightarrow x_1q_1y_1 \Rightarrow x_2q_2y_2 \Rightarrow x_3q_3y_3 \Rightarrow \cdots \)

✓ If the TM halts, we can represent the whole history of the computation as a single (finite) string!

✓ Traditionally written \( \#C_1\#C_2\#C_3\#\cdots\#C_n\# \)
Checking a History

✓ Checker: a Turing Machine C that, given <M, h>, checks whether h is a history of TM M.

✓ Consecutive states should be equal, except around the head (where the change corresponds to the transition table of M).

✓ Can check whether h is a halting (or an accepting) history by looking at the last control state.
Digression: LBAs

✓ In fact, the CH can be checked for validity by a less-than-general TM called an LBA.

✓ LBA = “Linear Bounded Automaton,” a TM that can only use the part of the tape containing input.

✓ An LBA can have a large tape alphabet and can “mark” tape cells. It just can’t grow its tape.

✓ More powerful than a DFA

✓ Number of potential states grows with input size

✓ DFA wouldn’t be able to check a computation history.
Accepting for LBAs

$A_{LBA} = \{ <M, w> \mid M \text{ a LBA accepting } w \}$ is decidable.

Proof: When running $M$ on $w$, there are at most

$$n := |Q| \times |\Gamma|^{|w|} \times |w|$$

distinct “states” during the computation.

So,

Run $M$ on $w$.

If computation takes longer than $n$ steps, it’s in an infinite loop; $M$ doesn’t accept $w$. 
Emptiness for LBAs

\[ E_{LBA} = \{ \langle M \rangle \mid M \text{ a LBA, } L(M) = \emptyset \} \text{ isn’t decidable.} \]

Proof: \( A_{TM} \leq E_{LBA} \).
All\textsubscript{CFG} = \{ <G> \mid G \text{ a CFG}, \text{L}(G) = \Sigma^* \} is undecidable.

**Proof:** \text{A}_{\text{TM}} \leq \text{All}_{\text{CFG}} .

✓ Key: given <M, w> create a PDA/CFG for strings that aren’t accepting computation histories!

✓ PDA accepts strings that

✓ Don’t start with \text{q}_0w

✓ Or, don’t end with \text{xq}_{\text{accept}}y

✓ Or, two successive configurations don’t match properly

✓ Hack: need to reverse every other configuration.

✓ The grammar for this PDA is \Sigma^* iff M, w has no finite, accepting history.
\textbf{Proof:} \text{All}_{\text{CFG}} \leq \text{Eq}_{\text{CFG}}.
Post Correspondence Problem
Emil Post

☑ Named after logician Emil Post (1897-1954)
☑ studied fundamental models of computation
☑ “scooped” by Gödel, Turing, and Church
Why PCP?

✓ Trivial problem to state
  ✓ Looks nothing like Turing Machines
  ✓ A child can understand it
  ✓ Superficially, doesn’t look that hard

✓ Can reduce PCP to other problems, showing them undecidable
**PCP**

Given a set of “dominos” (pairs of strings), find a finite sequence of dominos, repeats allowed, where the top line and bottom line spell out the same string.

<table>
<thead>
<tr>
<th>b</th>
<th>a</th>
<th>ca</th>
<th>abc</th>
<th>10</th>
<th>011</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca</td>
<td>ab</td>
<td>a</td>
<td>c</td>
<td>101</td>
<td>11</td>
<td>011</td>
</tr>
</tbody>
</table>

**Theorem: PCP is undecidable.**

**Proof: Halting ≤ MPCP ≤ PCP**
**Modified PCP (MPCP)**

- Like PCP, but solution starts with **first** domino.
- The following MPCP instance has no solution.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>011</td>
<td>101</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>11</td>
<td>011</td>
<td>0</td>
</tr>
</tbody>
</table>
MPCP ≤ PCP

Given an instance of MPCP, solve by translating dominos and doing PCP.
Halts $\leq$ MPCP

✓ Idea: if a TM halts, it has a computation history
✓ Given a $<M,w>$, construct dominos whose solution would yield such a history (on top and bottom); use MPCP-solver to check for a solution.
The Dominos

First domino: set up the initial state

\[
\begin{array}{c}
# \\
#q_0w# \\
\end{array}
\]

Helper dominos: copy unchanged parts of the tape from $C_i$ to $C_{i+1}$. (optional: expand the tape)
More Dominos

For each transition $q, x \rightarrow q', y, R$

- For each transition $q, x \rightarrow q', y, L$

\[ \begin{align*}
q, x & \rightarrow q', y, R \\
q & \rightarrow q'
\end{align*} \]

\[ \begin{align*}
q, x & \rightarrow q', y, L \\
a & \rightarrow b
\end{align*} \]
Completion Dominoes

*Technical “Trick”*

- When we reach a halting state \( h \), we delete the configuration (one symbol at a time) until it disappears.
- For each halting state \( h \), and each symbol \( x \), we need

\[
\begin{array}{c}
\text{hx} \\
\text{h}
\end{array}
\quad
\begin{array}{c}
\text{xh} \\
\text{h}
\end{array}
\]

- Finally,

\[
\begin{array}{c}
\text{h##} \\
\text{#}
\end{array}
\]
Example: \( w=11 \)
The Recursion Theorem
Recursion Theorem
(Kleene 1938)

Informally: A program can have access to its own description (code).

Code for TM S

Take input x.
...
put $\langle S \rangle$ on the tape
...
do stuff with x and $\langle S \rangle$
...
Recursion Theorem
(Kleene 1938)

Formally:
If \( R \) is a Turing machine computing a binary function \( R(x, y) \),
then there is a Turing machine \( S \) computing a unary function
such that:
\[
S(x) = R(x, \langle S \rangle)
\]
where \( \langle S \rangle \) is the description of \( S \) itself.
Application:
Undecidability of $A_{\text{TM}}$ (again)

- Suppose $A_{\text{TM}}$ were decidable using TM $A$.

- Define $M(x)$ as follows:
  1. Take input $x$;
  2. Use $A$ to decide whether $M$ accepts $x$, i.e., whether $\langle M, x \rangle \in A_{\text{TM}}$;
  3. Return the opposite answer.

- $M$ can't exist, so $A$ must not exist. QED
Recursion Theorem

to the rescue

Code for TM R

Take input x and y.
Put y and x on tape.
Run A as subroutine.
Return opposite answer.

Recursion Theorem

Code for TM M

Take input x.
Put \langle M \rangle and x on tape.
Run A as subroutine.
Return opposite answer.
Self-Printing Machines

✓ Even if a machine is not given a handle to its own code on its tape at the outset, there are ways for it to construct it.

✓ Such programs are now called “Quines”
A Java Quine

http://www.knet.ro/lsantha/

class Q{public static void main(String[]v){char
c=34;System.out.print(s+c+s+c+';'+'+);}static String
s="class Q{public static void main(String[]v){char
c=34;System.out.print(s+c+s+c+';'+'+);}static String
s=";};

javac Q.java

djava Q

class Q{public static void main(String[]v){char
c=34;System.out.print(s+c+s+c+';'+'+);}static String
s="class Q{public static void main(String[]v){char
c=34;System.out.print(s+c+s+c+';'+'+);}static String
s=";}

C and C++ Quines
(authors unknown)

```
char f[] =
"char f[] =\%c\%c%s\%c;\%cmain() {printf(f,10,34,f,34,10,10);}\%c"
main() {printf(f,10,34,f,34,10,10);}"
```

```
#include <iostream>
define a(b) std::cout<<"#include <iostream>\n#define a(b) "<<#b<<"\nmain(){a("<<#b<<");}"
main(){a(std::cout<<"#include <iostream>\n#define a(b) "<<#b<<"\nmain(){a("<<#b<<");}}
```
rex Quine
(by a Pomona College Student)

```
a="\"a\"; bb="b";
c="\"c\"; cc="c"];d="print(
  a,a,c, b, a,a,b, f,
  a,a,a, c,b,a,b,f,g,bb,c,b,
  a,b,b,f ,bb,bb,c,b,bb,b,f,g,
  cc,c,b,c, b,f, cc,c,c,b,cc,
  b, f,g .dd c,b,
  d, b,f , g,g,
  dd, dd, c,b,
  dd, b,f .g,ee
  c,b e, b,f.
  ee, ee, c,b,
  ee,b,f, g,ff,c, b,f,
  b,f,ff,ff,c, b,f,ff,f,f, g,gg,
  c,b,a ,nn,b,
  f,gg ,gg,c,b,
  gg,b,f, g,nn,nn,c
b,nn,b, f,g,g,d,g);"

print(
  a,a,c, b, a,a,b, f,
  a,a,a, c,b,a,b,f,g,bb,c,b,
  a,b,b,f ,bb,bb,c,b,bb,b,f,g,
  cc,c,b,c, b,f, cc,c,c,b,cc,
  b, f,g .dd c,b,
  d, b,f , g,g,
  dd, dd, c,b,
  dd, b,f .g,ee
  c,b e, b,f.
  ee, ee, c,b,
  ee,b,f, g,ff,c, b,f,
  b,f,ff,ff,c, b,f,ff,f,f, g,gg,
  c,b,a ,nn,b,
  f,gg ,gg,c,b,
  gg,b,f, g,nn,nn,c
b,nn,b, f,g,g,d,g);```

continued next col.
Applications of Quines

✓ Entertainment
✓ Computer viruses
✓ Artificial life?
Recursion from the Recursion Theorem

- If you have $\langle M \rangle$ on the tape, you can run it.
- A machine $M$ can put $\langle M \rangle$ on the tape.
- Therefore, a TM can compute “recursively” in the modern sense.

**Code for TM R**

- Take input $x$ and $M$.
- If $x = 0$, write 1 and halt.
- Otherwise, run $M$ on $(x-1)$ and $M$.
- Multiply result by $x$.

**Code for TM F**

- Take input $x$.
- If $x = 0$, write 1 and halt.
- Otherwise, run $F$ on $(x-1)$.
- Multiply result by $x$. 

---

**Recursion Theorem**
The language \( \text{MIN} = \{ \langle M \rangle \mid M \text{ is minimal} \} \) is not even recognizable.

- Minimal
  
  =

  no machine with the same behavior has smaller description.
Proof

✓ Suppose MIN were recognizable, hence enumerated by some machine E.

✓ Consider the TM C:

✓ Take input x.

✓ Start running enumerator E.

✓ Stop when we produce a $\langle D \rangle$ that is strictly longer than $\langle C \rangle$ (This must happen. Why?)

✓ Simulate D running on x.

✓ C cannot exist (why?), so E cannot exist.

✓ Thus MIN cannot be recognizable.