The Legacy of Alan Turing

Based on slides by team of CMU’s 15-251
Sanjeev Arora and Bernard Chazelle

The HELLO assignment

Write a JAVA program to output the words “Hello World!” on the screen and halt.

Space and time are not an issue.

PASS for any working HELLO program, no partial credit.

Grading Program?

A grading program G must be able to take any Java program P and grade it.

\[
G(P) = \begin{cases} 
\text{Pass, if P prints only the words “Hello World!” and halts.} \\
\text{Fail, otherwise.}
\end{cases}
\]

How exactly might such a program work?

Nasty Program

```java
n := 0;
while (n is not a counter-example to the Riemann Hypothesis) {
    n++;
}
print “Hello World!“;
```

What kind of program could a student who hated his/her TA hand in?

The Riemann Hypothesis

• Considered by many mathematicians to be the most important unresolved problem in pure mathematics
• Conjecture about the distribution of zeros of the Riemann zeta - function
• 1 Million dollar prize offered by Clay Institute
Nasty Program

\[
\begin{align*}
n &:= 0; \\
\text{while } (n \text{ is not a counter-example to the Riemann Hypothesis}) \\n\quad &\{ \\
\quad &\quad n++; \\
\quad &\} \\
\text{print } "\text{Hello World!}"; \\
\end{align*}
\]

The nasty program is a PASS if and only if the Riemann Hypothesis is false.

A TA nightmare: Despite the simplicity of the HELLO assignment, there is no program to correctly grade it!

We will develop the ideas needed to prove this.

The theory of what can and can’t be computed by an ideal computer is called Computability Theory.

Turing develops a model of computation

- Wanted a model of human calculation.
- Wanted to strip away inessential details.
- What are the important features?
  - Paper (size? shape?)
  - The ability to read or write what’s on the paper.
  - The ability to shift attention to a different part of the paper
  - The ability to have what you do next depend on what part of the paper you are looking at and on what your state of mind is
  - Limited number of possible states of mind.

A variant on Turing’s model: Turing-Post programs

- 1 dimensional unlimited scratchpad ("infinite")
- Only symbols are 0/1 (tape has a finite number of 1s)
- Can only scan/write one symbol per step
- Legal instructions
  
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PRINT 0</td>
<td>PRINT 1</td>
<td>GO RIGHT</td>
</tr>
<tr>
<td>GO LEFT</td>
<td>GO TO STEP i if 1 SCANNED</td>
<td>GO TO STEP i if 0 SCANNED</td>
</tr>
<tr>
<td>STOP</td>
<td>STOP</td>
<td>STOP</td>
</tr>
</tbody>
</table>

What does this program do?

1. PRINT 0
2. GO LEFT
3. GO TO STEP 2 if 1 SCANNED
4. PRINT 1
5. GO RIGHT
6. GO TO STEP 5 if 1 SCANNED
7. PRINT 1
8. GO RIGHT
9. GO TO STEP 1 if 1 SCANNED
10. STOP
Example: What does this program do?

1. PRINT 0
2. GO RIGHT
3. GO TO STEP 1 if 1 SCANNED
4. GO TO STEP 2 if 0 SCANNED

T-P “programming language” has these instructions

1. PRINT 0
2. PRINT 1
3. GO RIGHT
4. GO LEFT
5. GO TO STEP i if 1 SCANNED
6. GO TO STEP i if 0 SCANNED
7. STOP

What kind of computations can be performed in this model?

Amazing fact about this mickey-mouse model:

It is equivalent to Java!

In fact, all of the following are equivalent computational models:
- Turing-Post programs
- Turing machines (which we haven’t defined precisely)
- Pseudocode (which we haven’t defined precisely
- Python
- C++

• Equivalent = If something can be computed in one of these models, it can also be computed in the others.
• = what can be computed on a digital computer (with no bound on memory)

CHURCH-TURING THESIS

This model captures the notion of computation.

Anything “computable” is computable by Turing machine.

Any “reasonable, physically realizable” model of computation can be simulated on a Turing machine “efficiently”.

Any well-defined procedure that can be grasped and performed by the human mind and pencil/paper, can be performed on a conventional digital computer with no bound on memory.

Turing’s next great insight: duality between programs and data

• A program can be viewed either as
  - a program whose execution does whatever that program is designed to do -- P
  - or as plain data -- the code of the program -- code(P).

“Code” for a program

= Binary Representation

Many conventions possible (e.g., ASCII)

One possible convention:

<table>
<thead>
<tr>
<th>Code</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>PRINT 0</td>
</tr>
<tr>
<td>001</td>
<td>PRINT 1</td>
</tr>
<tr>
<td>010</td>
<td>GO LEFT</td>
</tr>
<tr>
<td>011</td>
<td>GO RIGHT</td>
</tr>
<tr>
<td>100</td>
<td>GO TO STEP i if 0 SCANNED</td>
</tr>
<tr>
<td>101</td>
<td>GO TO STEP i if 1 SCANNED</td>
</tr>
<tr>
<td>111</td>
<td>STOP</td>
</tr>
</tbody>
</table>

P ← ← Code (P) Start with 1, end with III
### P vs code (P)

P:
1. PRINT 0
2. GO RIGHT
3. GO TO STEP 1 if 1 SCANNED
4. GO TO STEP 2 if 0 SCANNED

<table>
<thead>
<tr>
<th>Code</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>PRINT 0</td>
</tr>
<tr>
<td>001</td>
<td>PRINT 1</td>
</tr>
<tr>
<td>010</td>
<td>GO LEFT</td>
</tr>
<tr>
<td>011</td>
<td>GO RIGHT</td>
</tr>
<tr>
<td>100</td>
<td>GO TO STEP 1 if 0 IS SCANNED</td>
</tr>
<tr>
<td>101</td>
<td>GO TO STEP 1 if 1 IS SCANNED</td>
</tr>
<tr>
<td>110</td>
<td>STOP</td>
</tr>
</tbody>
</table>

Start with 1, and with III

### Turing’s next great insight: duality between programs and data

- A program can be viewed either as
  - a program whose execution does whatever that program is designed to do -- P
  - or as plain data -- the code of the program -- code(P).

- If code(P) can be viewed as data, it can be the input to another program!

### Duality leads to Universality!

- There is a universal Turing machine \( U \)
  - On input code(P) and an input \( x \), \( U \) outputs the same thing as \( P \) does on input \( x \)
  - At each step it decodes which operation \( P \) would have performed and simulates it
- One Turing machine, the Universal TM, is enough!
  - Basis for modern stored-program computer
  - Von Neumann studied Turing’s UTM design

### You be the universal computer

101111000110111111 code(P) followed by input

1 011 1010 001 100 111 111

<table>
<thead>
<tr>
<th>Code</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>PRINT 0</td>
</tr>
<tr>
<td>001</td>
<td>PRINT 1</td>
</tr>
<tr>
<td>010</td>
<td>GO LEFT</td>
</tr>
<tr>
<td>011</td>
<td>GO RIGHT</td>
</tr>
<tr>
<td>100</td>
<td>GO TO STEP 1 if 0 IS SCANNED</td>
</tr>
<tr>
<td>101</td>
<td>GO TO STEP 1 if 1 IS SCANNED</td>
</tr>
<tr>
<td>110</td>
<td>STOP</td>
</tr>
</tbody>
</table>

Start with 1, and with III

### Before Turing...

- Brain
- Control
- Data
- Program
- Data

### After Turing...

- Brain
- Control
- Data
- Program
- Data
control

brain

program data

Print this Let ‘em eat cake

Let ‘em eat cake

Fishing ...

Fishing ...

Fishing manual

program data

control = hardware knows nothing

program data

programs = software
Duality leads to Universality!

- There is a universal Turing machine $U$
  - On input $\text{code}(P)$ and an input $x$, $U$ outputs the same thing as $P$ does on input $x$
  - At each step it decodes which operation $P$ would have performed and simulates it.
- One Turing machine, the Universal TM, is enough!
  - Basis for modern stored-program computer
  - Von Neumann studied Turing’s UTM design

• “existence of software industry lemma” -- Scott Aaronson

The Halting Problem
(or Universal Termination Detector)

Is there a program $\text{HALT}$ such that:

$\text{HALT}(\text{code}(P), x) = \text{True}$, if $P(x)$ halts
$\text{HALT}(\text{code}(P), x) = \text{False}$, if $P(x)$ does not halt

1. GO RIGHT
2. GO TO STEP 1 IF 0 IS SCANNED
3. GO TO STEP 1 IF 1 IS SCANNED
4. STOP

THEOREM: There is no program to solve the halting problem
(Alan Turing 1937)

We’ll use a “proof by contradiction”

“When something’s not right, it’s wrong.”
Bob Dylan

THEOREM: There is no program to solve the halting problem
(Alan Turing 1937)

Suppose a program $\text{HALT}$ existed that solved the halting problem.

$\text{HALT}(\text{code}(P), x) = \text{True}$, if $P(x)$ halts
$\text{HALT}(\text{code}(P), x) = \text{False}$, if $P(x)$ does not halt

We will call $\text{HALT}$ as a subroutine in a new program called $\text{CONFUSE}$.

CONFUSE

CONFUSE($\text{code}(P)$)
{  if ($\text{HALT}(\text{code}(P),\text{code}(P))=\text{True}$)
    then loop forever; // i.e., we don’t halt
  else exit; // i.e., we halt
}

Does CONFUSE(CONFUSE) halt?
Alan Turing (1912-1954)

**Theorem: [1937]**
There is no program to solve the halting problem

Similar to the fact that there are different kinds of infinity

“...impossibility spreads like a shock wave through the space of problems...” Kleinberg/Papadimitriou

• No Hello World Tester

There is no program to grade the HELLO assignment

- Does P halt?
- Let P’ be P with all print statements removed.
- Is [P’; print HELLO] a hello program?

“...impossibility spreads like a shock wave through the space of problems...” Kleinberg/Papadimitriou

• No Hello World Tester
• No automated checking of pretty much any property of software!