Today’s Topics

Computer Science
Noncomputability

Upcoming
Special Topic: Enabled by Computer --
Decoding the Human Genome

Reading
Great Ideas, Chapter 15
On the Limits of Computing

- **Noncomputability**
  - Certain Problems *Not* Amenable to Computer Solution
  - Examples given here may seem strained and artificial.
- **However, computers have very real limitations**

- **Will Use Two Approaches to Prove Noncomputability**
  1. Show *Existence* of Noncomputable Functions
  2. Prove That Certain Programs *Can Not Exist*
Existence of Noncomputable Functions

- **Approach**
  - Matching up Programs and Functions
  - E.g., assume 3 functions, only 2 programs
  - Without details, conclude one function has no program

- **Have:** *Uncountable Infinity of Functions Mapping int to int*
  - How can we show that is true?
  - Functions can be seen as columns in tables
  - Put all functions into a huge *(infinite!)* table
  - Show that even that cannot hold them all
  - *Can you identify the functions in the following table?*
# Table of All Integer to Integer Functions

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>2</th>
<th>6</th>
<th>0</th>
<th>0</th>
<th>8</th>
<th>2</th>
<th>1</th>
<th>4</th>
<th>.</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>16</td>
<td>3</td>
<td>13</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>10</td>
<td>5</td>
<td>16</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>12</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>36</td>
<td>8</td>
<td>19</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>14</td>
<td>12</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>14</td>
<td>13</td>
<td>22</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>16</td>
<td>13</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>64</td>
<td>21</td>
<td>25</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>18</td>
<td>14</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>18</td>
<td>34</td>
<td>28</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

...
A Function *NOT* in this (inclusive!) Table

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1+1</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>. .</td>
</tr>
<tr>
<td>2</td>
<td>4+1</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>. .</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>6+1</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>. .</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>8</td>
<td>9+1</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>16</td>
<td>3</td>
<td>13</td>
<td>. .</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>10</td>
<td>10</td>
<td>1+1</td>
<td>0</td>
<td>8</td>
<td>10</td>
<td>5</td>
<td>16</td>
<td>. .</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>12</td>
<td>11</td>
<td>1</td>
<td>1+1</td>
<td>8</td>
<td>36</td>
<td>8</td>
<td>19</td>
<td>. .</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>14</td>
<td>12</td>
<td>1</td>
<td>0</td>
<td>8+114</td>
<td>13</td>
<td>22</td>
<td>. .</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>16</td>
<td>13</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>64+121</td>
<td>25</td>
<td>. .</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>18</td>
<td>14</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>18</td>
<td>34+128</td>
<td>. .</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>20</td>
<td>15</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>100</td>
<td>55</td>
<td>31+1</td>
<td>. .</td>
</tr>
<tr>
<td>. . . . . . . . . . . .</td>
<td>. . . . . . . . . . . .</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CompSci 001
Existence of Noncomputable Functions

- All Programs Can be Ordered (Thus Countable)
  - By size, shortest program first
  - Just use alphabetical order

- Try to Draw Lines Between Functions and Programs
  - Could draw lines from every program to every function
  - But, have proved functions uncountable...
  - Thus, There Must be Functions With NO Programs!

- Hard to come up with function that computer can't produce
  - Possible example: random generator
    (No algorithm can produce truly random number sequence)
  - Use Table
  - Program must be of finite size; Requires infinite table
Noncomputable Programs

* Programs that Read Programs
  - What programs have we used that read in programs?
  - Express programs as a single string (formatting messed up)
  - Therefore, could write program to see if there is an `if` statement in the program: answers YES or NO
  - How about, *Does program halt?*
  - Lack of `while` (and functions) guarantees a halt
  - Not very sophisticated
  - *Not Halting for All Inputs* is usually considered a Bug

* Solving the Halting Problem
  - Write specific code to check out more complicated cases
  - Gets more and more involved...
The Halting Problem: Does it Halt?

Consider Following Program: Does it "halt" for all input?

```c
// input an integer value for k
while (k > 1)
{
    if ((k/2) * 2 == k) // is k even?
        k = k / 2;
    else
        k = 3 * k + 1;
}
```

Try It!
- e.g. input 17: value of k: 52 26 13, 40 20 10 5, 16 8 4 2 1
- For a long time, no one knew whether this quit for all inputs.
Proving Noncomputability

- Mathematicians have proven that no one, finite program can check this property for all possible programs

- Examples of non-computable problems
  - Equivalence: Define by same input > same output
  - Use variation of above program; not sure it ends
  - Cannot generally prove equivalence

- Use **Proof by Contradiction** (Indirect Proof)

- Proving non-computability
  - Sketch of proof
  - Find more details in book
Noncomputability Proof

- **Assume Existence of Function `halt`:**
  
  
  \[
  \text{String } \text{halt}(\text{String } p, \text{ String } x);
  \]
  
  - Inputs: \(p = \text{program}, x = \text{input data}\)
  - Returns: "Halts"
    
    or "Does not halt"

- **Can now write:**

  
  \[
  \text{String } \text{selfhalt}(\text{String } p);
  \]
  
  - Inputs: \(p = \text{program}\)
  - Returns: "Halts on self"
    
    or "Does not halt on self"

  - Uses: `halt(p, p)`;
  - i.e.: asking if halts when program \(p\) uses *itself* as data
Noncomputability Proof.2

- Now write function `contrary`:
  ```java
  void contrary()
  {
      TextField program = new TextField(1000);
      String p, answer;
      p = program.getText();
      answer = selfhalt(p);
      if (answer.equals("Halts on self"))
      {
          while (true) // infinite loop
              answer = "x";
      }
      else
          return; // i.e., halts
  }
  
- "Feed it" this program as data.
Noncomputability Proof.3

- **Paradox!**
  - If the \texttt{halt} program decides it halts, it goes into infinite loop and goes on forever
  - If the \texttt{halt} program decides it doesn't halt, it quits immediately
- **Therefore** \texttt{halt} cannot exist!

- Whole classes of programs on program behavior are non-computable
  - Equivalence
  - Many other programs that deal with the behavior of a program
Living with Noncomputability

- What Does It All Mean?
  - Not necessarily a very tough constraint unless you get too greedy.
  - Programs can't do everything.
    - Beware of people who say they can!
  - Programs probably can't do things we don't know how to do...