CS 360
Programming Languages

Introduction
Computability
Turing Machines
Background

- Some programming language
- Some assembly programming (CS 281)
- Regular Expressions (CS 270)
- Finite State Automata (CS 270, ECE 200)
Topics

• Computability
• What is a language?
• Interpreter vs. Compiler
• Semantics
  – Operationally
What is a language?

A **programming language** is a notational system for describing computation in a machine-readable and human-readable form. - Kenneth Louden

- So, what do we use them for?
- What does it mean to be computable?
Turing Machine

• Abstract machine
• Consists of:
  – A tape
  – A set of symbols
  – A read/write head
  – A finite set of instructions
  – A state register
Turing Machine

• One state is the start state
• Instructions are pentuplets:
  \[ q_c, a_c \rightarrow q_n, a_n, d \]
  – q is the state
  – a is the symbol read or written
  – d is the direction the head moves
• The tape is infinite (one- or two-ended)
Turing “programs”

- See [http://ironphoenix.org/tril/tm/](http://ironphoenix.org/tril/tm/) for a nice simulator from Suzanne Britton

<table>
<thead>
<tr>
<th>Hello World</th>
<th>Unary Subtraction</th>
<th>Now you try:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, _ 2, H, &gt;</td>
<td>1, _ 1, _, &gt;</td>
<td>• Unary addition</td>
</tr>
<tr>
<td>2, _ 3, E, &gt;</td>
<td>1, 1 1, 1, &gt;</td>
<td>• Copy a string</td>
</tr>
<tr>
<td>3, _ 4, L, &gt;</td>
<td>1, - 1, -,-, &gt;</td>
<td>• Reverse a string</td>
</tr>
<tr>
<td>4, _ 5, L, &gt;</td>
<td>1, = 2, _, &lt;</td>
<td></td>
</tr>
<tr>
<td>5, _ 6, O, &gt;</td>
<td>2, 1 3, =, &lt;</td>
<td></td>
</tr>
<tr>
<td>6, _ H, !, &gt;</td>
<td>2, - H, _, &lt;</td>
<td></td>
</tr>
</tbody>
</table>

Now you try:
- Unary addition
- Copy a string
- Reverse a string
Church-Turing Thesis

- Alonso Church was also exploring what was computable, using lambda calculus
- The Turing Machine can simulate the logic of any computer algorithm
  - May be less efficient
  - May use more instructions/memory
  - But, no less powerful
- Anything that is *effectively calculable* can be decided by a Turing Machine
Turing Complete

• The Turing machine is *universal*
  – It can be used to implement a Turing Machine
• Any language that can be used to simulate a Turing Machine is *Turing complete*
  – So, HTML might be a language, but it is not a *programming language*
Halting Problem

• There are problems that are not decidable
• It is not possible to write a program that can always determine if a given program (along with some input) would halt
  – I could sketch a proof here, but I’d be plagiarising and there are some nice discussions out there
  – See http://www.cgl.uwaterloo.ca/~csk/halt/ , from Prof. Kaplan at Waterloo
Busy Beaver

• A game
• Given $n$ states, write a Turing program that makes a maximum number of marks on the tape, but still halts
  – Number gets large quickly
  – Undecidable (for some $n$)
• If we could compute the busy beaver value, we could solve the halting problem