Today

- Toni Boggan on Senior Design
- PL Basics
- Language implementation overview

Assignments

- HW1 (out, due Tues) ... see piazza for submission instructions
What is a Programming Language?

One type of definition of a PL (from wikipedia)

“A programming language is a formal language designed to communicate instructions to a machine, particularly a computer.”

“A programming language is a notation for writing programs, which are specifications of a computation or algorithm.”

Another definition: **Turing Complete**

**Turing Machines:** Every turing machine consists of:

1. an infinite tape of (blank) memory cells with input string
2. a read/write head
3. a transition function: state, value → new-value, move (L,R), new-state

   • alphabet of symbols (plus blank symbol)
   • set of “states” with a designated start state

**Simple Example** (assume start one cell to left of first input cell)

   • alphabet: \{a, b\}
   • states: s (start), s₁, h (halt)
   • transitions:

```
<table>
<thead>
<tr>
<th>state</th>
<th>value</th>
<th>new-value</th>
<th>move</th>
<th>new-state</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>⊥</td>
<td>⊥</td>
<td>R</td>
<td>s₁</td>
</tr>
<tr>
<td>s₁</td>
<td>a</td>
<td>a</td>
<td>R</td>
<td>s₁</td>
</tr>
<tr>
<td>s₁</td>
<td>b</td>
<td>a</td>
<td>R</td>
<td>s₁</td>
</tr>
<tr>
<td>s₁</td>
<td>⊥</td>
<td>⊥</td>
<td>L</td>
<td>h</td>
</tr>
</tbody>
</table>
```
Another example (subtract 1 from a binary number)

- alphabet: \{0, 1\} (binary digits)
- states: \(s\) (start), \(s_1\) (go to end), \(s_2\) (find first 1), \(s_3\) (write 1’s), \(h\) (halt)
- transitions:

<table>
<thead>
<tr>
<th>state</th>
<th>value</th>
<th>new-value</th>
<th>move</th>
<th>new-state</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s)</td>
<td>⊥</td>
<td>⊥</td>
<td>R</td>
<td>(s_1)</td>
</tr>
<tr>
<td>(s_1)</td>
<td>0</td>
<td>0</td>
<td>R</td>
<td>(s_1)</td>
</tr>
<tr>
<td>(s_1)</td>
<td>1</td>
<td>1</td>
<td>R</td>
<td>(s_1)</td>
</tr>
<tr>
<td>(s_1)</td>
<td>⊥</td>
<td>⊥</td>
<td>L</td>
<td>(s_2)</td>
</tr>
<tr>
<td>(s_2)</td>
<td>0</td>
<td>0</td>
<td>L</td>
<td>(s_2)</td>
</tr>
<tr>
<td>(s_2)</td>
<td>⊥</td>
<td>⊥</td>
<td>R</td>
<td>(h)</td>
</tr>
<tr>
<td>(s_2)</td>
<td>1</td>
<td>0</td>
<td>R</td>
<td>(s_3)</td>
</tr>
<tr>
<td>(s_3)</td>
<td>0</td>
<td>1</td>
<td>R</td>
<td>(s_3)</td>
</tr>
<tr>
<td>(s_3)</td>
<td>⊥</td>
<td>⊥</td>
<td>L</td>
<td>(h)</td>
</tr>
</tbody>
</table>

A language is “Turing Complete” if it can simulate any Turing Machine

- A TM captures a specific algorithm (defines one “machine”)
- Every algorithm can be captured as a TM (Church-Turing thesis)
- A Universal TM is a TM that runs TMs (like computers run programs)

A language is a “programming language” if it is Turing Complete

- making it possible to express all computations / algorithms (according to the Church-Turing thesis)
- implying the language is computationally “powerful”
Examples of languages that are not Turing Complete:

- Markup languages: HTML, XML, JSON, ...
- Many “domain-specific” languages: SQL, regular expressions

Not necessarily tied to specific constructs

- imperative languages with conditional branching (if-goto, while loops) and arbitrary mem access (# of variables, arrays/lists)
- whereas Haskell and LP languages use recursion (no goto, no loops)

“Languages” that are (accidentally) Turing Complete

- Musical Notation (requires human to be the memory/tape)
- Excel spreadsheets w/ formulas (no macros, etc.)
- Apache rewrite rules
- Magic The Gathering card game (human selects moves)
- PowerPoint animations (requires human to follow links)
Compilation (typically) moves in increasingly abstract steps

1. identify language “tokens” in source (more later)
2. ensure syntax is correct (“parse”) 
3. ensure source code is “correct” (nothing used before defined, no type errors)
4. generate an intermediate representation (for optimization)
5. improve performance of code (optimization)
6. generate executable code (machine code)

Example of “separation of concerns” (engineering design strategy)

- too complex to do “all at once” (similarly, single vs multi-pass)
- easier to manage, maintain, update/improve