Want to Write a Compiler?

Pat Morin
COMP 3002
What is a Compiler?

• From Day 1:
  – A compiler is a program that translates
  – from a source language S
  – into a target language T
  – while preserving semantics

• Often (but not always)
  – S is a programming language
  – T is a machine language
Want to Write a Compiler?

• A compiler has 3 main parameter
  – Source language (S)
    • What kind of input does the compiler take?
      • C, C++, Java, Python, …
  – Implementation language (I)
    • What language is the compiler written in?
      • C, Java, i386, x86_64
  – Target language (T)
    • What is the compiler's target language
      • i386, x86_64, PPC, MIPS, …
Source Language Issues

• Complexity
  – Is a completely handwritten compiler feasible?

• Stability
  – Is the language definition still changing?

• Novelty
  – Do there already exist compilers for this language?

• Complicated, or still-changing languages promote the use of compiler generation tools
Target Language Issues

• Novelty
  – Is this a new architecture?
  – Are there similar architectures/instruction sets?

• Available tools
  – Is there an assembler for this language?
  – Are there other compilers for this language?
Performance criteria

- **Speed**
  - Does it have to be a fast compiler?
  - Does it have to be a small compiler?
  - Does it have to generate fast code?

- **Portability**
  - Should the compiler run on many different architectures (*rehostability*)
  - Should the compiler generate code for many different architectures (*retargetability*)
**Possible Workarounds**

- **Rewrite an existing front end**
  - when the source is new
  - reuse back (code generation) end of the compiler

- **Rewrite an existing back end**
  - when the target architecture is new
  - retarget an existing compiler to a new architecture

- **What happens when both the source language and target language are new?**
  - Write a compiler from scratch?
  - Do we have other options?
Composing Compilers

- Compilers can be composed and used to compile each other
- Example:
  - We have written a Java to JVM compiler in C and we want to make it to run on two different platforms i386 and x86_64
  - both platforms have C compilers

```
+----------+----------+----------+----------+----------+
<table>
<thead>
<tr>
<th>Java</th>
<th>JVM</th>
<th>Java</th>
<th>JVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>C</td>
<td>i386</td>
<td>i386</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i386</td>
<td></td>
</tr>
</tbody>
</table>
+----------+----------+----------+----------+
<table>
<thead>
<tr>
<th>Java</th>
<th>JVM</th>
<th>Java</th>
<th>JVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>C</td>
<td>x64</td>
<td>i386</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x64</td>
<td></td>
</tr>
</tbody>
</table>
+----------+----------+----------+----------+```
Example

• Assignment 3:

• Assignment 4:
Example

• Show how to
  – To take your PRM compiler and make it faster
  – To take your Jasmin optimizer and make it faster
Bootstrapping by cross-compiling

- Sometimes the source and implementation language are the same
  - E.g. A C compiler written in C
- In this case, cross compiling can be useful

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1</td>
<td>x64</td>
<td>C</td>
<td>C</td>
<td>3</td>
<td>x64</td>
<td>x64</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>2</td>
<td>i386</td>
<td>i386</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i386</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bootstrapping Cont'd

• Bootstrapping by reduced functionality
  – Implement, in machine language, a simplified compiler
    • A subset of the target language
    • No optimizations
  – Write a compiler for the full language in the reduced language
Bootstrapping for Self-Improvement

• If we are writing a good optimizing compiler with I=S then
  – We can compile the compiler with itself
  – We get a fast compiler

• gcc does this (several times)
Summary

• When writing a compiler there are several techniques we can use to leverage existing technology
  – Reusing front-ends or back ends
  – Cross-compiling
  – Starting from reduced instruction sets
  – Self-compiling