603AA - Principles of Programming Languages [PLP-2015]

Andrea Corradini
Department of Computer Science, Pisa
Academic Year 2015/16
Admins

• http://www.di.unipi.it/~andrea/Didattica/PLP-15/
• 9 CFU/ECTS
• Students enrolled till AY 2013/14 have to integrate the course with a 3 CFU activity
  – To be agreed upon with me
• Office Hours: (was Monday, 15:30-17:30)
Evaluation

• 2 midterms
  – November 2-6, 2015
  – December 16-18, 2015

• Written proof

• Oral examination

• Pre-evaluation:
  – Starter kit test: Thu October 1 at 17:00
Course Topics and Goals

• The course presents principles and techniques for the implementation and usage of programming languages.

• First part:
  – formal definition of the syntax of programming languages
  – main phases of a compiler with emphasis on the lexical, syntactical and semantical analysis phases of the front-end.

• Second part:
  – main topics of the structure of programming languages from the viewpoint of the runtime support of its abstract machine and of the expressiveness of the supported linguistic constructs
  – focus on constructs of imperative, functional, object-oriented, and scripting languages
Textbooks

• [Scott] Programming Language Pragmatics
  by Michael L. Scott, 3rd edition

• [ALSU] Compilers: Principles, Techniques, and Tools
  by Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman, 2nd edition

• [GM] Programming Languages: Principles and Paradigms
  by Maurizio Gabbrielli and Simone Martini

• [Mitchell] Concepts in Programming Languages
  by John C. Mitchell
Credits

• Slides freely taken and elaborated from a number of sources:
  – Marco Bellia (DIP)
  – Gianluigi Ferrari (DIP)
  – Robert A. van Engelen (Florida State University)
  – Gholamreza Ghassem-Sani (Sharif University of Technology)
Abstract Machines
Abstract Machine for a Language $L$

• Given a programming language $L$, an **Abstract Machine $M_L$ for $L$** is a collection of data structures and algorithms which can perform the storage and execution of programs written in $L$

• An abstraction of the concept of hardware machine

• Structure of an abstract machine:

<table>
<thead>
<tr>
<th>Memory</th>
<th>Interpreter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programs</td>
<td>Operations and Data Structures for:</td>
</tr>
<tr>
<td>Data</td>
<td>• <em>Primitive Data processing</em></td>
</tr>
<tr>
<td></td>
<td>• <em>Sequence control</em></td>
</tr>
<tr>
<td></td>
<td>• <em>Data Transfer control</em></td>
</tr>
<tr>
<td></td>
<td>• <em>Memory management</em></td>
</tr>
</tbody>
</table>
General structure of the Interpreter

Sequence control
- Fetch next instruction
- Decode
- Choose
- Execute op₁
- Execute op₂
- ... Execute opₙ
- Execute HALT
- Store the result

Data control

Operations

stop
The Machine Language of an AM

- Given and Abstract machine \( M \), the machine language \( L_M \) of \( M \)
  - includes all programs which can be executed by the interpreter of \( M \)
- Programs are particular data on which the interpreter can act
- The components of \( M \) correspond to components of \( L_M \), eg:
  - Primitive data types
  - Control structures
  - Parameter passing and value return
  - Memory management
- Every Abstract Machine has a unique Machine Language
- A programming language can have several Abstract Machines
An example: the Hardware Machine

- The language?
- The memory?
- The interpreter?
- Operations and Data Structures for:
  - Primitive Data processing?
  - Sequence control?
  - Data Transfer control?
  - Memory management?
Implementing an Abstract Machine

• Each abstract machine can be implemented in **hardware** or in **firmware**, but if it is high-level this is not convenient in general
• An abstract machine \( M \) can be implemented over a **host machine** \( M_0 \), which we assume is already implemented
• The components of \( M \) are realized using data structures and algorithms implemented in the machine language of \( M_0 \)
• Two main cases:
  – The interpreter of \( M \) coincides with the interpreter of \( M_0 \)
    • \( M \) is an **extension** of \( M_0 \)
    • other components of the machines can differ
  – The interpreter of \( M \) is different from the interpreter of \( M_0 \)
    • \( M \) is **interpreted** over \( M_0 \)
    • other components of the machines may coincide
Hierarchies of Abstract Machines

- Implementation of an AM with another can be iterated, leading to a hierarchy (onion skin model)
- Example:

```
Hardware machine
Firmware machine
Operating System machine
Intermediate machine (Java Bytecode)
HL machine (Java)
Web machine (browser etc.)
Web Service machine (languages for web services)
E-Business machine (on-line commerce applications)
```

In the literature on networks, one often speaks of a stack rather than, more correctly, of a hierarchy.
Implementing a Programming Language

• **L**  high level programming language
• **M_L**  abstract machine for **L**
• **M_o**  host machine

**Pure Interpretation**
- **M_L** is interpreted over **M_o**
- Not very efficient, mainly because of the interpreter (fetch-decode phases)

**Pure Compilation**
- Programs written in **L** are translated into equivalent programs written in **L_o**, the machine language of **M_o**
- The translated programs can be executed directly on **M_o**
  - **M_L** is not realized at all
  - Execution more efficient, but the produced code is larger

• Two limit cases that almost never exist in reality