02 - Languages, Abstract Machines, Runtime systems

Programming Languages

- Syntax: form of the program
- Semantics: meaning of well-formed program
- Pragmatics: conventions, guidelines, use cases

Paradigms

Classification of programming languages, based on features.

Some paradigms (where some are included in others):

- Imperative;
- Object-oriented;
- Concurrent;
- Functional;
- Logic.

Abstract Machine

An Abstract Machine is a collection of data structures and algorithms which can perform the storage ad execution of programs written in a specific programming language. Vice versa, each Abstract Machine defines a language including all programs which can be executed by the interpreter of M.

Composition

- Memory: contains programs and data;
- Interpreter:
  - Memory management: how and where to store data;
  - Primitive Data Processing: executes easy primitive operations;
  - Sequence control: handles conditional jumps, returns...
  - Data transfer control: handles parameter passing and values returns.

Implementation

Abstract Machine M can be either implemented:

- In hardware or in firmware: it is difficult because they have different abstraction standpoints (e.g. calling a function in AM is easy, on hardware is really complex);
- Over an already implemented host machine M_0. The interpreters of the two machines can coincide (so M is an extension of M_0), or differ (so M is interpreted over M_0).

To implement M over M_0 we have two different approaches:

- Pure interpretation: real time translation, not efficient (fetch-decode phases, no code optimizations);
- Pure compilation: beforehand translation of entire code.

Generally, compilation leads to better performance, while interpretation facilitates interactive debugging and testing. Modern implementations is to go for both approaches: compilation into intermediate program, interpreted

through Virtual Machine. Several advantages:

- Portability: once a program is compiled to the intermediate program, it can be executed on any platform equipped with the VM;
- Interoperability: can create a new language which is compiled for an existing VM;

This kind of implementation can be iterated, leading to a hierarchy.

Runtime Systems

The execution model of a programming language defines the sequence of steps that are taken to process the code and produce the desired output (fetch-decode-execute).

Runtime systems implement part of the execution model, providing runtime support. Needed both by interpreted and by compiled programs.

Generally made of:

- Compiler/Interpreter/VM;
- Code generated by the compiler;
- Code running in other threads during program execution (e.g. garbage collector);
- Language libraries;
- OS functionalities;

Needed for:

- Memory management: stack (push/pop activation records), heap (allocation, garbage collection);
- Interaction with runtime environment, which are data needed by the program in execution but not part of the program itself (e.g. environment variables, I/O, threads communication, dynamic type checking/binding...);
- Debugging, monitoring, ecc...