Interpreters and Compilers
Von Neumann machine

IAP@LMAI

Simone Martini

a.a. 2019-2020
Part I

Von Neumann Architecture
Ordered sequence of cells. Each cell may store a certain number of bits (this is a word). Each cell is uniquely identified by an address (a kind of index of the cell in the sequence).

- 8 bits = 1 byte.
- Word = one multiple of bytes (1, 2, 4, 8).
- Volatile.
- There is no a priori way to know if the cell contains data, or programs, or non significant values.
Why binary digits?

- **Technological reasons**
- **Bi-stable circuits are simpler, more economical, and more reliable**

- Multiples and powers of 2; logarithms in base 2

- **bit** = 1 binary digit
- **byte** = 8 bits

- **K (kilo)** = prefix for $2^{10} = 1024$, of the order of $10^3$
- **M (mega)** = prefix for $2^{20}$, of the order of $10^6$
- **G (giga)** = prefix for $2^{30}$, of the order of $10^9$
- **T (tera)** = prefix for $2^{40}$, of the order of $10^{12}$
- **P (peta)** = prefix for $2^{50}$, of the order of $10^{15}$
Part II

Interpreters and compilers
Abstract machines

- An abstract machine is an executor for a computation.
- It has its own machine language.
- If the machine is reasonably expressive: a programming language.
- The machine realizes ("is") an interpreter for its own language.
Interpreter: the crucial part of an AM which effectively implements the computation

By exploiting another AM!

IDLE presents the same language

But it is implemented on different machines: Windows, Linux, Mac OSX

And each OS is implemented on different hardware...
Interpreter: the crucial part of an AM which effectively implements the computation

By exploiting another AM!

- IDLE presents the same language
  - But it is implemented on different machines: Windows, Linux, Mac OSX
  - And each OS is implemented on different hardware...
Interpreter: the crucial part of an AM which effectively implements the computation

By exploiting another AM!

IDLE presents the same language

But it is implemented on different machines: Windows, Linux, Mac OSX

And each OS is implemented on different hardware...
Interpreter?

- Interpreter: the crucial part of an AM which effectively implements the computation
- By exploiting another AM!

- IDLE presents the same language
- But it is implemented on different machines: Windows, Linux, Mac OSX
- And each OS is implemented on different hardware...
Interpreter: the crucial part of an AM which effectively implements the computation

By exploiting another AM!

- IDLE presents the same language
- But it is implemented on different machines: Windows, Linux, Mac OS X
- And each OS is implemented on different hardware...
Machine $M_i$:
- uses the language $L_{M_{i-1}}$
  "is written in $L_{M_{i-1}}"$
- to implement (realize) its own language $L_i$
- it hides (up to a point) the machine $M_{i-1}$

At level $i$ we don’t know (and we don’t need to know) which is level 0 (hw)
For executing language $L$ on machine $MO$ different from my own
I may interpret $L$:
write (program) a machine for $L$, using (the language of) $MO$:

Programma scritto in $L$

Dati in Input

Interprete per $L$ scritto in $LO$

Dati di output

Esecuzione su $MO$

$MO$
For executing language $L$ on machine $MO$

different from my own

I may compile $L$:
translate $L$ into $LO$, and execute the result on $MO$:
A v C from L into Lo:
- it translates a program written in L into an equivalent one written in Lo
- it is a program itself
- written (programmed) in a (other?) language

The production cycle:
1. Write P
2. Compile P, obtaining $P_o$
3. Link $P_o$ with some libraries (auxiliary modules)
4. Send $P_o$ into execution
Interpreter or compiler?

Interpreter:
- less efficiency
- flexibility and portability
- easy interaction at run-time (e.g. debugging)

Compiler:
- greater efficiency
- less flexibility and portability
- loss of information on the structure of the source program
Interpreter or compiler?

- Any language may be interpreted
- Any language may be compiled

- When we say that *Python is interpreted*, or that *C is compiled*...
  - It only means that:
    - The usual Python implementation uses an interpreter;
    - The usual C implementation uses a compiler.
- There exist Python compilers
- There exist C interpreters
Any language may be interpreted
Any language may be compiled

When we say that *Python is interpreted*,
or that *C is compiled*...

It only means that:
The usual Python implementation uses an interpreter;
The usual C implementation uses a compiler.

There exist Python compilers
There exist C interpreters
In reality the two techniques *always* coexist:
The hierarchy of machines/languages
bottom-up

1. Absolute binary code
2. Relocatable binary code
3. Assembly (mnemonics with a limited facility for names)
4. High level languages: e.g., C
5. Interpretative implementation of high level languages: e.g., Python

Canonical implementation:
- from 4 into 1: compilation (direct, or in sequence).
- 5: interpreted on 4;
  Or: compiled into some intermediate language, which is then interpreted on 4.
The hierarchy of machines/languages
bottom-up

1. Absolute binary code
2. Relocatable binary code
3. Assembly (mnemonics with a limited facility for names)
4. High level languages: e.g., C
5. Interpretative implementation of high level languages: e.g., Python

Canonical implementation:
- from 4 into 1: compilation (direct, or in sequence).
- 5: interpreted on 4;
  Or: compiled into some intermediate language, which is then interpreted on 4.