Abstract Machines

Turing, FSM, SECD, Semantics

Yegor Bugayenko

Lecture #5 out of 10
90 minutes

Chapter #1:

Who Are Abstract Machines?

Definition

An abstract machine is a theoretical model of computation.

Similar to a function, a machine receives inputs and produces outputs based on predefined rules.

Abstract machines are "machines" because they allow step-by-step execution of programmes. (really?)

They are "abstract" because they ignore many aspects of actual (hardware) machines.

An abstract machine is an intermediate language with a small-step operational semantics.

Purpose

"The implementation of a programming language consists of two stages. The implementation of the compiler and the implementation of the abstract (virtual?) machine. This is a typical divide-and-conquer approach. From a pedagogical point of view, this simplifies the presentation and teaching of the principles of programming language implementations. From a software engineering point of view, the introduction of layers of abstraction increases maintainability and portability." (1999)

Virtual Machines

An abstract machine implemented in software is termed a virtual machine, and one implemented in hardware is called simply a "machine."

JVM (for Java) and CLR (for .NET) are among most notable examples of virtual machines.

IR (intermediate representation) is used internally by a compiler or virtual machine to represent source code. An intermediate language is the language of an abstract machine.

LLVM

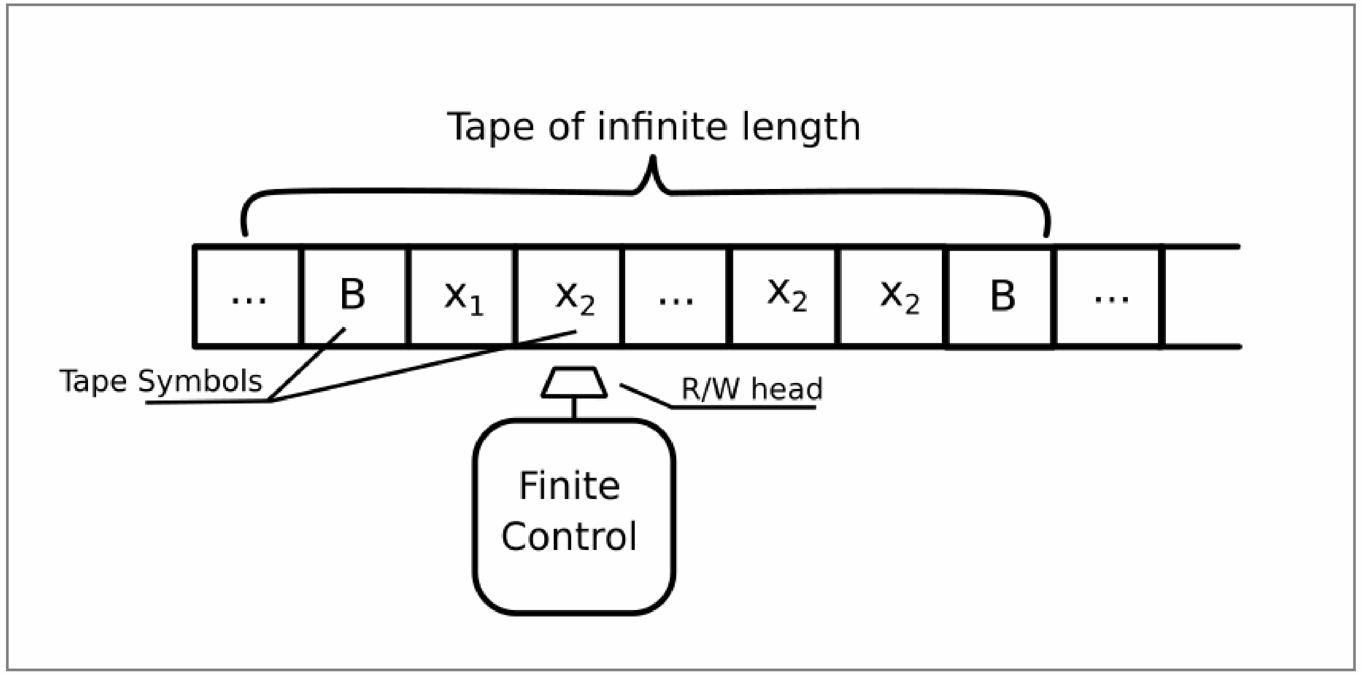
LLVM (Low Level Virtual Machine) is a standard de-facto.

```
@.str = internal constant [14 x i8] c"hello, world\0A\00"

declare i32 @printf(ptr, ...)

define i32 @main(i32 %argc, ptr %argv) nounwind {
entry:
    %tmp1 = getelementptr [14 x i8], ptr @.str, i32 0, i32 0
    %tmp2 = call i32 (ptr, ...) @printf( ptr %tmp1 ) nounwind
    ret i32 0
}
```

Chapter #2:

Turing Machine

Turing Machine was the first (1936) … but not the simplest.



For example, Emil Post's Machine is simpler.

Proof

The Church-Turing thesis: Anything that can be computed can be computed by some Turing machine.

There **has never been a proof**, but the evidence for its validity comes from the fact that every realistic model of computation, yet discovered, has been shown to be equivalent. — here.

Chapter #3:

λ-calculus

Abstraction:

$$(\lambda x.t) \qquad \text{e.g. } f = \lambda x.\sqrt{x}$$

Application:

$$\lambda(ts) \qquad \text{e.g. } (f\ 16) = 4$$

In lambda calculus, functions are taken to be "first class values," so functions may be used as the inputs, or be returned as outputs from other functions.

Chapter #4:

Finite-State Machine

Chapter #5:

SECD Machine(s)

There are SECD (**s**tack, **e**nvironment, **c**ontrol, **d**ump), CESK, CEK, CS, and maybe other abstract machines.

I like the CRM (**c**ontrol stack, **r**esult stack, **m**emory) machine explained by Michael Pradel in his YouTube course about program analysis: $\langle c, r, m \rangle$.

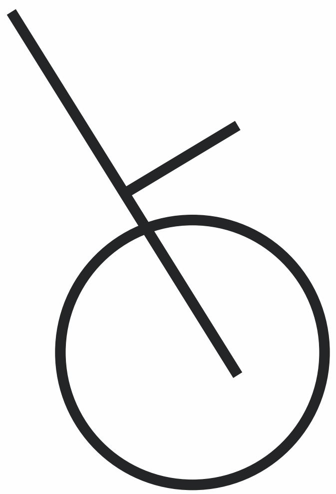
$$
\begin{aligned}
\langle \mathtt{x} := 2 \times 3, \mathtt{nil}, \{\} \rangle &\longrightarrow \langle \mathtt{x} \circ 2 \times 3 \circ :=, \mathtt{nil}, \{\} \rangle \\
&\longrightarrow \langle 2 \times 3 \circ :=, \mathtt{x} \circ \mathtt{nil}, \{\} \rangle \\
&\longrightarrow \langle :=, 6 \circ \mathtt{x} \circ \mathtt{nil}, \{\} \rangle \\
&\longrightarrow \langle \mathtt{nil}, \mathtt{nil}, \{\mathtt{x} \mapsto 6\} \rangle
\end{aligned}
$$

Chapter #6:

Semantic

This is our programming language that helps us draw on a canvas:

```
L 10, 20, 15, 23;
C 13, 13, 35;
L 5, 28, 15, 12;
```



Its semantic may be explained by the abstract machine with the following instruction set, which semantic is **obvious** to a reader:

```
DRAW x, y;
LOOP; IF t THEN BREAK; END LOOP;
x > y; x + y; x - y; x / y;
x := y;
1600; 900.
```

This is what "`L x1, y1, x2, y2`" means:

```
dx := x2 - x1;
dx := dx / 1600;
dy := y2 - y1;
dy := dy / 900;
LOOP;
DRAW x1, y1;
IF x1 > x2 THEN BREAK;
IF y1 > y2 THEN BREAK;
x1 := x1 + dx;
y1 := y1 + dy;
END LOOP;
```