Language paradigms

- imperative (C, FORTRAN)
- functional (Scheme, ML, Haskell)
- object-oriented (C++, Java, C#)
- logical (Prolog)

Each group has its own "model of computation" – a one sentence description of the main internal mechanism.

How many languages have ever been developed?

- 2500

How many languages were used a lot?

- 200

How many languages are commonly used today?

- 20

How many languages claim to be "universal"

~5

The course is about semantics – answers the question "what is the meaning of a program". Largely we will be considered input/output behavior.

To answer the question we can use the concept of a virtual machine. (VM)

The real machine (hardware) interprets machine code instruction to produce I/O behavior.

A virtual machine interprets a HLL (high-level language) – anything "above" assembler.

The VM contains structures and operations on them that support the constructs in the HLL.

A VM can be described in words, or can be implemented in software, i.e. an interpreter.

A very simple example $x = 3$

The VM needs a model of memory – where to store the values of variables. The assignment then alters the memory to associate x with the value 3.

We could implement memory as a hash table, with identifier names as the key and numbers as the associated values.

hash table function

$x \xrightarrow{\Theta} 3$

$y \rightarrow 22$

$z \rightarrow 17$

The best way to describe the semantics of a language is to use mathematics.

If we can do this, we can talk about program behavior independently of any machine, virtual or otherwise.

There are 3 approaches

1. operational semantics
2. denotational semantics
3. axiomatic semantics

Operational semantics

- has a formal of memory - function that maps identifiers to values - program "state"
- all constructs are expressed in logical form

e.g. assignment $I = E$

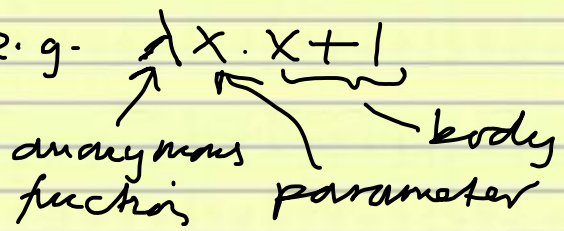
State is a function σ. The rule is:

$$\frac{[E, \sigma] \to v}{[I = E, \sigma] \to \sigma[I \mapsto v]}$$

This is read is as: <u>if</u> the value of expression E in state σ is v, <u>then</u> the execution of $I = E$ in state σ is a state in which I is now mapped to v

Denotational semantics

- each construct is represented by a function in the lambda calculus e.g. $\lambda x.\, x+1$



- the model of memory is the state, just as in operational semantics

e.g. assignment

$$C[\![I = E]\!] = \lambda s.\, s[M[\![I]\!] \mapsto E[\![E]\!]s]$$

Axiomatic semantics

- effects of operations as logical expressions involving state variables

e.g. $\underline{\{x = 0\}}\ \underline{x = 3}\ \underline{\{x > 0\}}$

precondition — PL construct — post-condition

This read as: If $x = 0$ is true and we execute $x = 3$, then $x > 0$ is true after the execution finishes

If we link many such statements, we can express the meaning of a whole program. In fact we can prove the programs are correct.