Semantics

Principles of Programming Languages

Fatma Başak Aydemir

basak.aydemir@boun.edu.tr

Semantics

- BNF is a formal and sufficient notation for syntax
- To capture semantics we need formal methods as well to prevent ambiguity, but it is difficult.
- No standard, or widely accepted method
- In Algol 60 BNF was used for the first time and the semantics was given in English

Semantics

- Natural language is inherently ambiguous!
  - A user must understand the exact meaning of language constructs to write correct programs
  - The meaning must be the same to all implementors to prevent different interpretations of the same language
- Questions we try to answer
  - What does a program do?
  - What is the effect of a programming construct?

Semantics

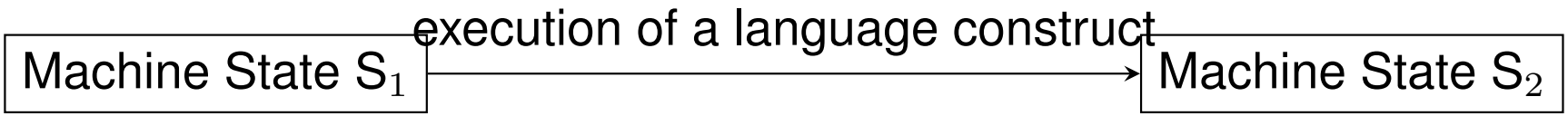
- Formal semantics is also used to prove the correctness of the program!
  - Saves time for we catch mistakes early

Methods for describing the semantics of programming languages

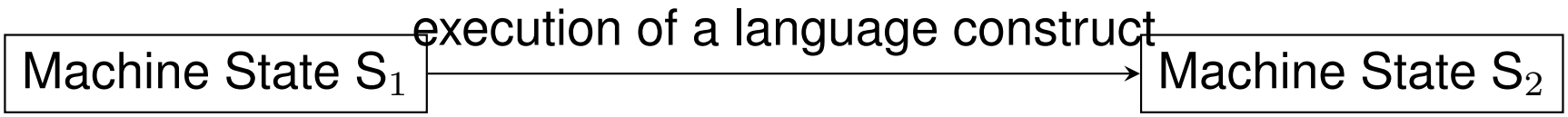
- Operational (imperative) semantics
- Axiomatic semantics
- Denotational semantics
- Algebraic semantics (we will not cover this)

Operational Semantics

- Simplest semantics
- Algorithmic approach rather than a formal one



Operational Semantics



- State: Memory contents, registers, everything whose value can be changed by the execution
- The idea is similar to an automaton but it is more complex. Automatons will be studied during CmpE 350.

How to do this?

- Implement an interpreter, install on a machine, execute a program there and observe the state changes on the machine
  - It is difficult to keep track of changes on a physical device!
  - It is only useful if you use the same machine, not general.
- A low-level language and storage locations are defined and the program constructs are defined using this low-level language
  - It is like simulating a simple machine
  - General
  - We need a translator to translate the construct to the low-level language and a program that simulates the execution

Operational Semantics: Low-level Language

$$\langle \text{program} \rangle \rightarrow \texttt{\{[}\langle \text{label} \rangle \texttt{:]}\langle \text{stmt} \rangle \texttt{\}}$$

$$\langle \text{stmt} \rangle \rightarrow \langle \text{ident} \rangle = \langle \text{var} \rangle \mid \langle \text{ident} \rangle = \langle \text{ident} \rangle \texttt{(+|-)1} \mid \texttt{goto}\langle \text{label} \rangle \mid \texttt{if}\langle \text{var} \rangle \langle \text{relop} \rangle \langle \text{var} \rangle \texttt{goto}\langle \text{label} \rangle$$

<label> The address of a word in RAM, where the following statement begins

goto <label> Replaces the contents of the program counter by the address <label>

<ident> An identifier which refers to the address of a word in RAM or the stack

<var> An identifier or a constant

<relop> is an operator $=, <, >, \geq, \leq$

Example: For loop

```
for(expr1;expr2;expr3;){statements;}
```

```
    loop: if expr2=0 goto out
          statements
          expr3
          goto loop
    out:  ...
```

Example: Another for

```
for i in min...max {statements;}
```

```
        i=min
loop:   statements
        if i=max goto out
        i=i+1
        goto loop
out:    ...
```

Operational Semantics

- Useful for implementors
- Not very useful for users