# Describe a programming language

Based on the slides of Maurizio Gabbrielli

# Description of a language (artificial)

- The description of a language is on three dimensions (Morris, 1938):
  - Syntax

    Grammar rules
    When a sentence is correct
    Relationship between symbols
  - Semantics

    Attribution of Meaning
    What does a correct sentence mean
    Relationship between signs and meaning
  - Pragmatic

    How correct and sensible sentences are used
    Relationships between signs, meaning and user

# A fourth level

- For an executable language
  - Implementation

Execute a correct sentence respecting its semantics

# Tools to describe languages

- Syntax
  - Formal: Generative Grammars (Chomsky), BNF(Backus Naur Form)
  - Quasi-formal: typing, contextual constraints
- Semantics
  - Informal: Natural language, Manual
  - Formal: denotational, operational (axiomatic, algebraic)
- Pragmatic
  - Informal: examples
  - Semi-formal: programming methodologies
- Implementation
  - Derivation from semantics
  - Abstract machine

# Syntax

- It tells us how to build the correct language phrases:

| | |
|---|---|
| **var A: integer;** | **if a > 0 then C else C** |
| **var A: intgr;** | **if a > 0 else C then C;** |
| Error *Lexical* | Error *Grammar* |

- Same constructs expressed differently in different languages
  - Ex: vector of 10 elements
    - In C: **int V [10]**
    - In Pascal: **V: Array [0.. 9] of Integer**
  - Presumably these two objects have the same meaning (the same semantics).

# Syntax

- How to **define and recognize** the correct language constructs?
  This is an essential problem when compiling a program
  - Enumerating all well-formed programs? ... a little long
  - Various tools:
    - **Generative grammars**: rules for building language sentences
      - Regular grammars
      - Context-free grammars
    - **Automata**: formalisms to recognize the constructs of a language
    - **BNF**: Backus Naur Form first used for Algol in 1958 by Naur (Chair of ALGOL Committee) and Backus (Secretary)
      - Same thing as context-free grammars

# BNF

- The form of Backus and Naur (BNF) is a notation for presenting free grammars, used for the first time in the syntax definition ALGOL 60.

  - <W> notation is used to indicate a symbol
  - We use the notation

    <A>:: = <B> | <C>

    To indicate two rules <A>:: = <B>  And  <A>:: = <C>
  - In the extended version the symbol <A>* (Kleene star) has the meaning that A is repeated 0 or more times

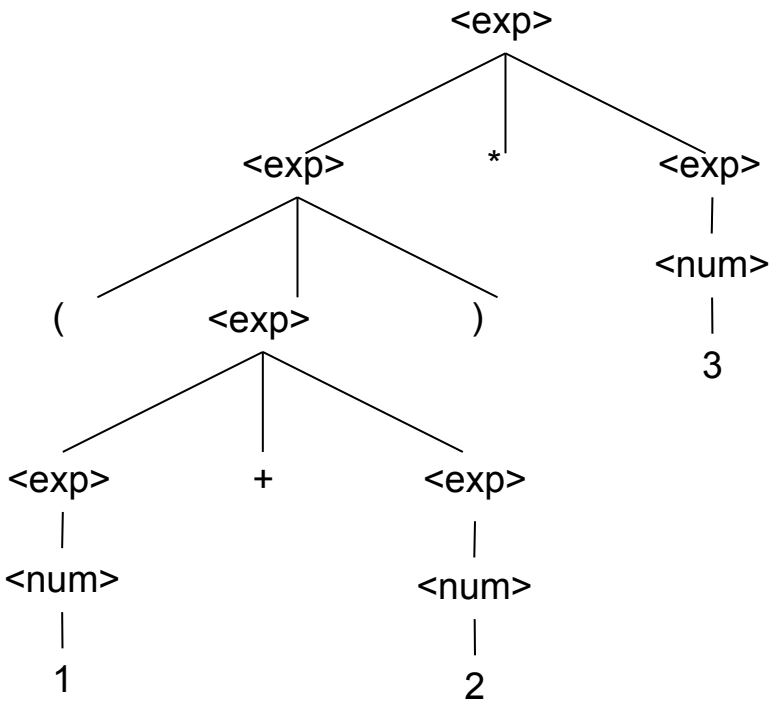
# Examples of derivations

<exp>:: = <num> | - <exp>

( <exp> ) | <exp> + <exp> | <exp> * <exp>

<num> ::= 0| 1 | 2 | … | 9

<exp> ⇒ <exp> * <exp>
→ <exp> * <num>
→ <exp> * 3
→ (<exp>) * 3
→ (<exp> + <exp>) * 3 → …
→ (1 + <exp>) * 3 → …
→ (1 + 2) * 3



# A simple imperative language example

$$
\begin{array}{lcl}
Num & ::= & 1 \mid 2 \mid 3 \mid \ldots \\
Var & ::= & X_1 \mid X_2 \mid X_3 \mid \ldots \\
 & & \\
AExp & ::= & Num \mid Var \mid AExp + AExp \mid AExp - AExp \\
BExp & ::= & \mathbf{tt} \mid \mathbf{ff} \mid AExp == AExp \mid \neg BExp \mid BExp \wedge BExp \\
 & & \\
Com & ::= & \mathbf{skip} \mid Var := AExp \mid Com; Com \mid \\
 & & \mathbf{if}\ BExp\ \mathbf{then}\ Com\ \mathbf{else}\ Com \mid \mathbf{while}\ BExp\ \mathbf{do}\ Com
\end{array}
$$

# The *Dangling Else*

Grammar are ambiguous sometimes

Example in Java:

```
if (door-open)
 if (none-in-view)
    return "nobody?";
       else ring-bell();
```

which "if" the "else" is paired?

We look at the official definition of Java...

## So...

The correct reading is:

```
if (door-open)
    if (None-in-view)
        return "Nobody?";
        else Play-camp ();
```

- "The Java programming language, like C and C++ and many programming languages before them, arbitrarily decided that an else clause belongs to the innermost if to which it might possibly belong"

# Semantics

- Need to be able to define exactly "What does" a program
- The formal (semi) definition of semantics is part of a language standard definition of languages since the 80'
- Various ways to define the semantics of a programming language
  - Algebraic
  - Axiomatic
  - Operational
  - Denotational
- There is no universally accepted standard methodology

# Operational semantics

- Obtained by defining an Interpreter of the language on a guest machine which components are mathematically described
  - State (describes altogether the system)
  - Transition (specify a computation step that changes the state and/or the program)
  - Computation (sequence of transitions)
- Especially useful because it directly provides an implementation model
- You can define it in a formal and syntax-driven way (SOS)

# Example

$$\langle \mathbf{skip}, \sigma \rangle \to \sigma$$

$$\langle \mathbf{if\ tt\ then}\ c_1\ \mathbf{else}\ c_2, \sigma \rangle \to \langle c_1, \sigma \rangle$$

$$\frac{\langle c_1, \sigma \rangle \to \langle c_1', \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \to \langle c_1'; c_2, \sigma' \rangle}$$