Programming Language Definition

- Syntax
  - To describe what its programs look like
  - Specified using **regular expressions** and **context-free grammars**
- Semantics
  - To describe what its programs mean
  - Specified using axiomatic semantics, operational semantics, or denotational semantics

1

Regular Expressions

- A regular expression is one of the following:
  - A character
  - The empty string, denoted by ε
  - Two or more regular expressions concatenated
  - Two or more regular expressions separated by | (or)
  - A regular expression followed by the Kleene star (concatenation of zero or more strings)

2

Regular Expressions

- The pattern of numeric constants can be represented as:

$$
\begin{aligned}
\textit{digit} &\longrightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
\textit{unsigned\_integer} &\longrightarrow \textit{digit}\ \textit{digit}* \\
\textit{unsigned\_number} &\longrightarrow \textit{unsigned\_integer}\ (\ (\ \texttt{.}\ \textit{unsigned\_integer}\ )\mid \epsilon\ ) \\
&\qquad (\ (\ (\ \texttt{e} \mid \texttt{E}\ )\ (\ \texttt{+} \mid \texttt{-} \mid \epsilon\ )\ \textit{unsigned\_integer}\ ) \mid \epsilon\ )
\end{aligned}
$$

3

What is the meaning of following expressions ?

- [0-9a-f]+
- b[aeiou]+t
- a*(ba*ba*)*

4

Define Regular Expressions

- Match strings only consisting of 'a', 'b', or 'c' characters
- Match only the strings "Buy more milk", "Buy more bread", or "Buy more juice"
- Match identifiers which contain letters and digits, starting with a letter

5

Lexeme, Token, & Pattern [1]

- Lexeme
  - A sequence of characters in the source program with the lowest level of syntactic meanings
    - E.g., sum, +, -
- Token
  - A category of lexemes
  - A lexeme is an instance of token
  - The basic building blocks of programs

6

Token Examples

Token	Informal Description	Sample Lexemes
keyword	All keywords defined in the language	if else
comparison	<, >, <=, >=, ==, !=	<=, !=
id	Letter followed by letters and digits	pi, score, D2
number	Any numeric constant	3.14159, 0, 6
literal	Anything surrounded by "'s, but exclude "	"core dumped"

7

Lexeme, Token, & Pattern [1]

- Pattern
  - A description of the form that the lexemes of a token may take
  - Specified with regular expressions

8

Context-Free Grammars

- Using the notation Backus-Naur Form (BNF)
- A context-free grammar consists of [1]
  - A set of *terminals* *T*
  - A set of *non-terminals* *N*
  - A *start symbol* *S* (a non-terminal)
  - A set of *productions* *P*

9

Terminals T

- The basic symbols from which strings are formed
- Terminals are tokens
  - if, foo, ->, 'a'

10

Non-terminals N

- Syntactic variables that denote sets of strings or classes of syntactic structures
  - expr, stmt
- Impose a hierarchical structure on the language

11

Start Symbol S

- One nonterminal
- Denote the language defined by the grammar

12

Production P

- Specify the manner in which terminals and nonterminals are combined to form strings
- Each production has the format

  nonterminal -> a string of nonterminals and terminals
- One nonterminal can be defined by a list of nonterminals and terminals

13

Production

- Nonterminal symbols can have more than one distinct definition, representing all possible syntactic forms in the language

  <if_stmt> -> if <logic_expr> then <stmt>

  <if_stmt> -> if <logic_expr> then <stmt> else <stmt>

Or

<if_stmt> -> if <logic_expr> then <stmt>
| if <logic_expr> then <stmt> else <stmt>

14

Backus-Naur Form

- Invented by John Backus and Peter Naur to describe syntax of Algol 58/60
- Used to describe the context-free grammars
- A **meta-language**: a language used to describe another language

15

BNF Rules

- A rule has a left-hand side(LHS), one or more right-hand side (RHS), and consists of **terminal** and **nonterminal** symbols
- For a nonterminal, when there is more than one RHS, there are multiple alternative ways to expand/replace the nonterminal
  - E.g., <stmt> -> <single_stmt>
    | begin <stmt_list> end

16

BNF Rules

- Rules can be defined using recursion

```
<ident_list> -> ident
              | ident, <ident_list>
```

- Two types of recursion
  - Left recursion:
    - id_list_prefix -> id_list_prefix, id | id
  - Right recursion
    - The above example

17

How does BNF work?

- It is like a mathematical game:
  - You start with a symbol **S**
  - You are given rules (**P**s) describing how you can replace the symbol with other symbols (**T**s or **N**s)
  - The language defined by the BNF grammar is the set of all terminal strings you can produce by following these rules

18

Derivation

- By repeatedly applying rules to nonterminals, we end up with strings containing only terminal symbols (**sentences**)
- All derived strings compose the language defined by the grammar

19