

Formal Languages

- Syntax
- Regular expressions
- Backus Naur Form (BNF)
- Introduction to Grammars
 - Regular grammars

Formally

- A PL is a set of strings, called *sentences*, over some finite alphabet of symbols, called *terminals*
 - Not necessarily a finite set
- Rules describe how to combine the terminals into well-formed sentences in the PL - **syntax**
- PLs are categorized by the complexity of these rules
 - Regular expressions used to describe tokens (atomic bits) of PLs
 - BNF used to describe *context-free languages*
 - Most PLs fall in this category

Formal Language Theory

- **Offers a way to describe computation problems formulated as language recognition problems**
 - Enables proofs of relative difficulty of certain computational problems
- **Provides a mechanism to aid description of programming language constructs**
 - Regular expressions ~ PL tokens (e.g., keywords)
 - Finite state automata (FSAs)
 - Context-free grammars ~ PL statements

Formal Language Theory

- **Recognizers for languages are more complex as the languages themselves become more complex**
 - Simple constructs correspond to FSAs
 - Keywords, numerical constants
 - More complex constructs correspond to Push-down Automata
 - If statements, looping statements, declarations
 - Even more complex constructs correspond to more complex automata
 - Type checking of use with declared type

Regular Expressions

- Formalism for describing simple PL constructs
 - reserved words
 - identifiers
 - numbers
- Simplest sort of structure
- Recognized by a finite state automaton
- Defined recursively

Regular Expressions

<u>PL construct</u>	<u>RE Notation</u>	<u>Language</u>
	an empty RE	{ }
symbol a	a	{ a }
null symbol	ϵ	{ ϵ }
R, S regular exprs	$R \mid S$	$L_R \cup L_S$
<i>a, b terminals</i>	<i>a/b (alternation)</i>	<i>{a, b}</i>
R, S regular exprs	RS	$L_R L_S$
<i>a, b terminals</i>	<i>ab (concatenation)</i>	<i>{ab}</i>

RE's for PLs

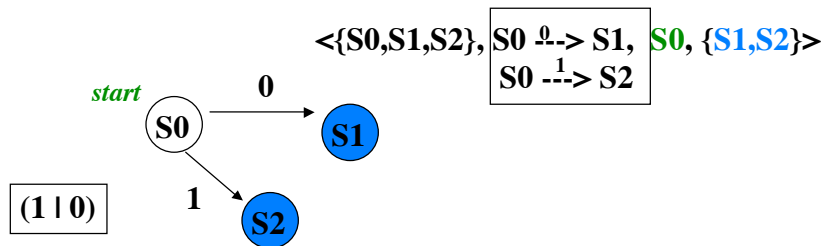
- Let *letter* stand for a|b|c|...|z and *digit* stand for 0|1|2|3|4|5|6|7|8|9
 - *letter (letter | digit) ** is identifier
 - *digit +* is an integer constant
 - *digit * . digit +* is real number
- Which identifiers are described by
 - *letter (letter | digit) ** ?
ABC OC B% X1

Examples

- Which of the following are legal real numbers described by
 - *digit * . digit +* ? .5 1.5 2 4. 6.3 0.2
- Can see that simple PL constructs can be defined as regular expressions
 - Can you define a number in scientific notation as an RE? (e.g., 1.25e+20)

Finite State Automaton (FSA)

- Recognizer of the language generated by a regular expression
- Described by $\langle \text{set of states, labelled transitions, start state, final state(s)} \rangle$

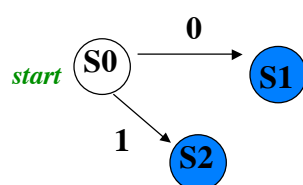


Formal-1, CS314 Fall 2007 © BG Ryder

11

FSA

- FSA *accepts or recognizes* an input string iff there is a path from its start state to a final state such that the labels on the path are the terminals in that string
 - Empty transitions signify illegal moves; can think of FSA going to a sink error state



states:	inputs:	
	0	1
S0	S1	S2
S1	---	---
S2	---	---

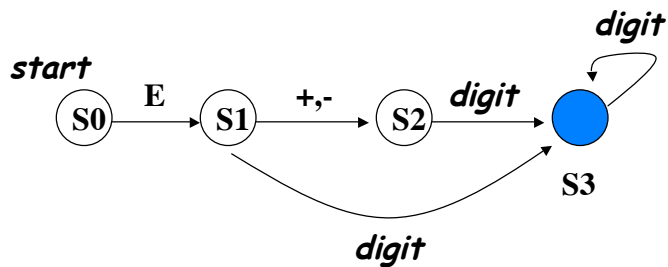
transition table

Formal-1, CS314 Fall 2007 © BG Ryder

12

Example

Exponent in scientific notation:
 $E (+ \mid -) \textit{digit}^+ \mid E \textit{digit}^+$



Backus Naur Form (BNF)

- Metasymbols $\langle \rangle ::= |$
- Terminal symbols of the PL
 - e.g., keywords, operators
- Nonterminal symbols

$\langle \textit{while_stmt} \rangle ::= \textit{while} \langle \textit{expr} \rangle \textit{do} \langle \textit{stmt} \rangle$
 $\langle \textit{identifier} \rangle ::= \langle \textit{letter} \rangle |$
 $\quad \langle \textit{identifier} \rangle \langle \textit{digit} \rangle | \langle \textit{identifier} \rangle \langle \textit{letter} \rangle$

BNF Examples

- **letter** (*letter* | *digit*)^{*}
 $\langle id \rangle ::= \langle letter \rangle | \langle id \rangle \langle letter \rangle | \langle id \rangle \langle digit \rangle$
- **digit**^{*}
 $\langle integer \rangle ::= \langle integer \rangle \langle digit \rangle | \langle digit \rangle | \epsilon$
- **Strings of 1's and 0's where all 1's come before all 0's, that is, 1*0***
 $\langle str \rangle ::= \langle one \rangle \langle zero \rangle$
 $\langle one \rangle ::= 1 \langle one \rangle | 1 | \epsilon$
 $\langle zero \rangle ::= 0 \langle zero \rangle | 0 | \epsilon$
- **If statement**
 $\langle if_stmt \rangle ::= if \langle expr \rangle then \langle statement \rangle else \langle statement \rangle$

Formal-1, CS314 Fall 2007 © BG Ryder

15

Extended BNF (EBNF)

- Nonterminals begin with capital letters or are shown in a different font
- {...} means repeat the enclosed 0 or more times
- [...] means the enclosed is optional
- (...) is used for grouping, usually with the alternation symbol |
- If { }, [], or () are terminals in the PL being defined, then when they are used as terminals they must be underlined
{ } terminals, { } metasympols

Formal-1, CS314 Fall 2007 © BG Ryder

16

EBNF Examples

Identifier ::= Letter { LetterorDigit }

LetterorDigit ::= Letter | Digit

Expr ::= [Expr -] Subexpr

IfStmt ::= if LogicExpr then Stmt [else Stmt]

CompoundStmt ::= begin Stmt {; Stmt} end

WhileStmt ::= while (LogicExpr) Stmt {; Stmt}

ArrayElement ::= Identifier [Identifier]

Grammar

- A formalism to describe the sentences of a PL
- <set of terminals, set of nonterminals, productions (rules), special symbol>
 - terminals are alphabet symbols (e.g., +)
 - nonterminals represent PL constructs (e.g., Stmt)
 - productions are rules for forming syntactically correct constructs
 - special symbol tells where to start applying the rules

Example

```
<letter> ::=  
    a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z  
<digit> ::= 0|1|2|3|4|5|6|7|8|9  
<identifier> ::= <letter> | <identifier> <letter> |  
    <identifier> <digit>  
<assign_stmt> ::= <identifier> = 0 //terminals;
```

```
//nonterminals are  
    {<letter>, <digit>, <assign_stmt>, <identifier>}  
//special symbol is <assign_stmt>
```

Regular PLs

- Describe the simple constructs in real PLs
- Form of rules
 - Each righthandside is length ≤ 2 symbols
 - A terminal or nonterminal
 - A nonterminal followed by a terminal
- All PLs describable by REs can be written as regular grammars

```
e.g.,  $1 2^* | 0^+$   $N ::= X | Y$   
       $X ::= 1 | X 2$   
       $Y ::= 0 | Y 0$ 
```