

Announcements

- **If you're looking for a SP#**
 - today after class (6 pm), CoRE 310
- **Midterm is Friday, March 5 (2:50-4:10pm)**
- **Lecture notes, readings, etc... will be on the course web page shortly**
<http://remus.rutgers.edu/cs314>
- **Book in SERC: Aho, Sethi, Ullman “*Compilers: Principles, Practices and Techniques*”**
- **About attendance...**
- **Office hours: Thursdays, 1:30-3:00 pm**

1

Formal Languages

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

2

Formally

- A programming language (PL) is a set of strings (called *sentences*) over some finite alphabet of symbols, called *terminals*
- 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

3

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

4

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**

5

Regular Expressions

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

6

Regular Expressions

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

7

Regular Expressions

<u>PL construct</u>	<u>RE Notation</u>	<u>Language</u>
<i>R,S</i> regular exprs	<i>R</i> [*]	{ ϵ } $\cup L_R \cup L_R L_R \cup L_R L_R L_R \dots$
<i>a</i>	<i>a</i> [*]	{ $\epsilon, a, aa, aaa, \dots$ }
<i>R,S</i> regular exprs	<i>R</i> ⁺	$L_R \cup L_R L_R \cup L_R L_R L_R \dots$
<i>a</i>	<i>a</i> ⁺	{ <i>a, aa, aaa, ...</i> }

Note: $\epsilon a = a \epsilon = a$

Precedence is

{ [*] +}	<i>high</i>
concatenation	
	<i>low</i>

(all are left associative operators)

8

RE Examples

$1 2$	$\{1,2\}$
$1^* 2$	$\{2,\epsilon,1,11,111,\dots\}$
$1 2^*$	$\{1, 12, 122, 1222, \dots\}$
$1 2^* 0^+$	$\{0,00,000,\dots,1,12,122,\dots\}$
$(0 1)^*$	Binary numbers: $\{\epsilon,0,1,00,01,10,11,\dots\}$
$(0 1)^*1$	Odd binary numbers (end in 1)
$(11)^*$	An even number of 1's: $\{\epsilon,11,1111, \dots\}$

9

RE's for PLs

- Let *letter* stand for $a|b|c|\dots|z$ and *digit* stands for $0|1|2|3|4|5|6|7|8|9$ Louden uses $[0-9]$
 - *letter* ($letter | digit$)^{*} is identifier $[a-z]([a-z] | [0-9])^*$
 - *digit*⁺ is an integer constant $[0-9]^+$
 - *digit*^{*}.*digit*⁺ is real number $[0-9]^* \backslash . [0-9]^+$
- Which identifiers are described by
 - *letter* ($letter | digit$)^{*} ? ABC 3C B% X1

10

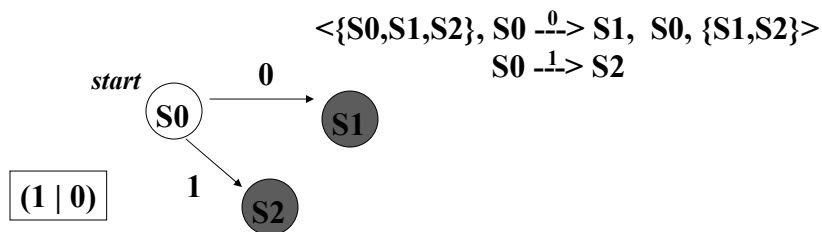
Examples

- Which of the following are legal real numbers described by: $digit^* \cdot digit^+$
.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., $1.25e+20$)

11

Finite State Automaton (FSA)

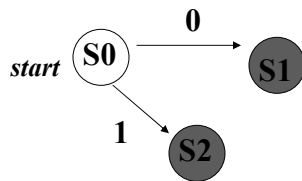
- Recognizer of the language generated by a regular expression
- Described by
<set of states, labeled transitions, start state, final state(s)>



12

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	---	---

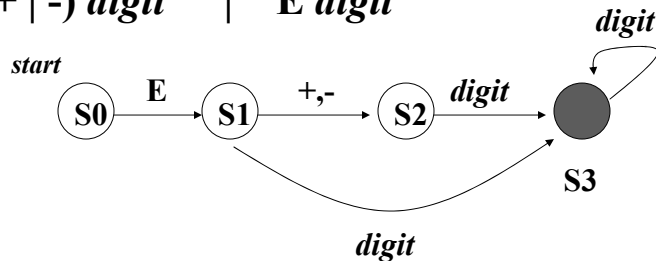
transition table

13

Example

Exponent in scientific notation:

$E (+ | -) digit^+ | E digit^+$



14

Backus-Naur form (BNF)

- Metasymbols < > ::= |
 - Terminal symbols of the PL
 - e.g., keywords, operators
 - Nonterminal symbols
-

<while_stmt> ::= while <expr> do <stmt>
<identifier> ::= <letter> | <identifier> <digit> |
<identifier> <letter>

15

BNF Examples

- *letter (letter | digit) **
<id> ::= <letter> | <id> <letter> | <id> <digit>
- *digit**
<integer> ::= <integer><digit> | <digit> | ϵ
- **Strings of 1's and 0's where all 1's come before all 0's, that is, 1^*0^***
<str> ::= <one> <zero>
<one> ::= 1 <one> | 1 | ϵ
<zero> ::= 0 <zero> | 0 | ϵ
- **If statement**
<if_stmt> ::= if <expr> then <statement> else <statement>

16