Syntax

CMPSC 461
Programming Language Concepts
Penn State University
Fall 2016

Midterms

Sep. 30 (Friday) 6:30PM – 7:45PM

010 Sparks Building

Nov. 7 (Monday) 6:30PM – 7:45PM

112 Kern Building

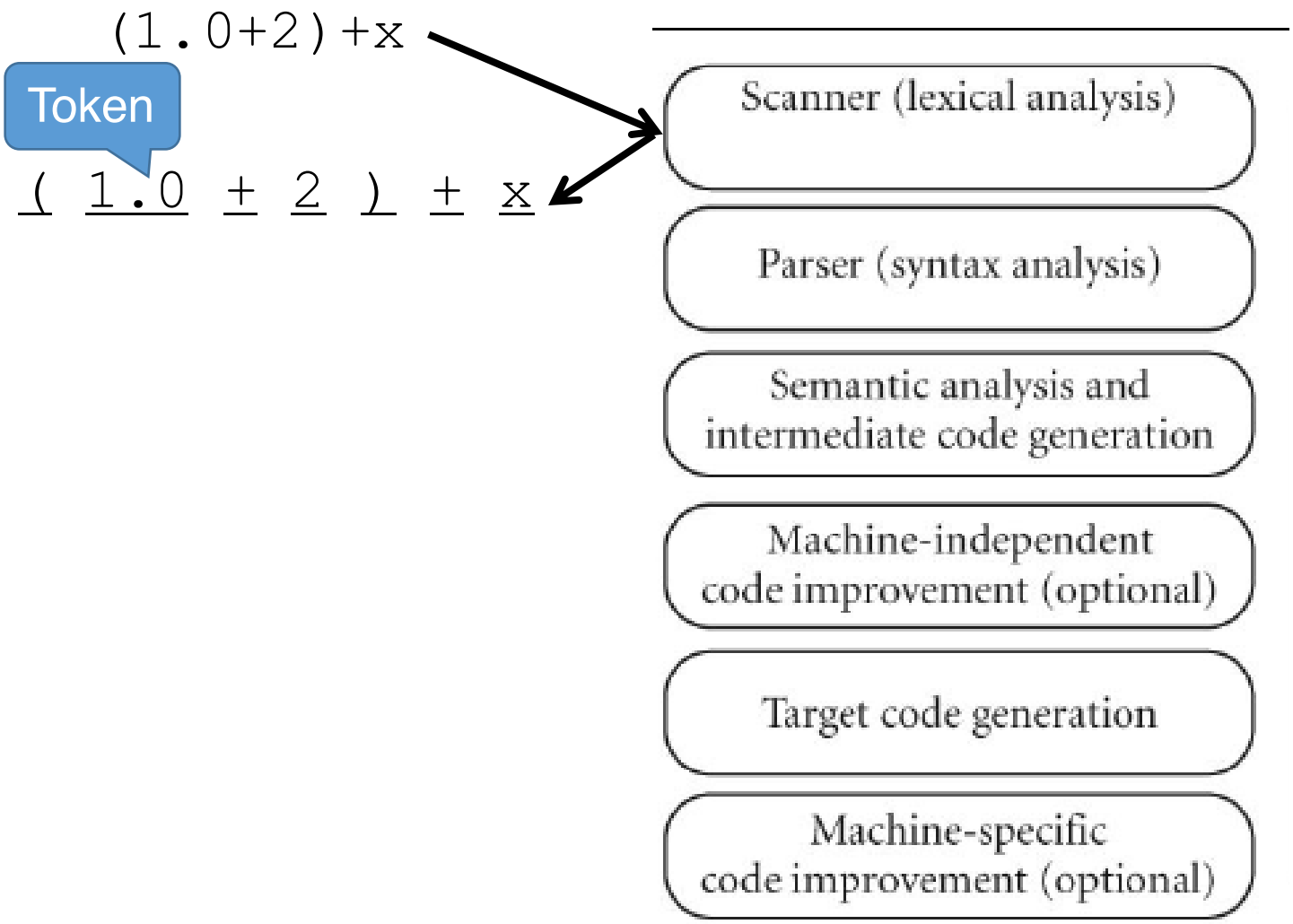
Formal Languages

Language: a set of (legal) strings

Goal: a concise & precise notation for specifying a language

Four levels of languages [Chomsky]:

1. Regular
2. Context-Free
3. Context-Sensitive
4. Unrestricted


(1.0+2)+x
Token
(1.0 + 2) + x
Scanner (lexical analysis)
Parser (syntax analysis)
Semantic analysis and intermediate code generation
Machine-independent code improvement (optional)
Target code generation
Machine-specific code improvement (optional)

Language of tokens (C)

Identifier: letters, digits and underscore '_' only. The first character must be an underscore or a letter

literals: digits, decimal point, suffix such as “l”, “u”

operators: + - * / …

keywords : if, while, for, int, …

punctuation: { } [ ] ; …

How can we specify these tokens (sets of strings)?

Regular Expression

Definition:

- A character
- Empty string (ε)
- Concatenation of two RE (e.g., (ab))
- Alternation of two RE, separated by “|” (e.g., (a|b))
- Closure (Kleene star) (e.g.(a*))

Examples

RE	Meaning
a	“a”
ac	“ac”
a|c	“a” or “c”
a*	“” or “a” or “aa” or …
(a|b)(c|d)	“ac” or “ad” or “bc” or “bd”
(ab)|(cd)	“ab” or “cd”
((a|b)c)*	“” or “ac” or “bc” or “acac” or “bcbc” or …

More Formally…

Regular expression defines ***a set of strings*** (aka. a language)

$L(R)$: the language defined by RE R

- A character x: $L(x) = \{"x"\}$
- Empty string ε: $L(\varepsilon) = \{""\}$
- Concatenation: $L(RS) = \{r.s | r \in L(R), s \in L(S)\}$
- Alternation: $L(R|S) = L(R) \cup L(S)$
- Kleene star: $L(R^*) = \{""\} \cup L\{R\} \cup L\{RR\} \cup \cdots$

String concatenation

Examples

RE	L(RE)
a	{“a”}
ac	{“ac”}
a|c	{“a”, “c”}
a*	{“”, “a”, “aa”, …}
(a|b)(c|d)	{“ac”, “ad”, “bc”, “bd”}
(ab)|(cd)	{“ab”, “cd”}
((a|b)c)*	{“”, “ac”, “bc”, “acac”, “bcbc”, …}

Precedence of RE

The order is (high to low)

- Closure (*), then
- Concatenation, then
- Alternation

Analogy in arithmetic:

- Exponentiation
- Multiplication
- Addition

ablcd	(ab)l(cd)
albc*d	(a)l(b(c*)d)

UNIX Extensions to RE

Extension	Core RE
[abcd]	a|b|c|d
. (wild cast)	a|b|c|…(all char. except new line)
a?	ε | a
a+	a(a*)
[a-z]	a|b|c|…|z
\[\] \.	[ ] .

Examples

bit	0|1
4 bits	(0|1)(0|1)(0|1)(0|1)
bits	(0|1)*
Even # of bits	((0|1)(0|1))*
frag: Num bet. 0 and 255	[0-9] | [1-9][0-9] 1 [0-9][0-9] 2 [0-4][0-9] | 25[0-5]
IP Address	Use the RE above
bits with equal 0's and 1's	Not a regular language!

Grammar

$$A \rightarrow \omega$$

nonterminal symbol

Lang. definition in meta language

Language of tokens (C)

Identifier: letters, digits and underscore '_' only. The first character must be an underscore or a letter

numbers: digits, decimal point, suffix such as “l”, “u”

operators: + - * / …

keywords : if, while, for, int, …

punctuation: { } [ ] ; …

Grammar

$identifier$ → [_a-zA-z][_a-zA-z0-9]∗

$number$ → [1-9][0-9]∗(\. [0-9]+)? (l?lu?)

$operator$ → + | - | ∗ | / | …

$keyword$ → if | while | for | int | …

$punctuation$ → { | } | \[| \] | ; | …

Scanning with RE

Read one character at a time and then

- output a token
- ignore character
- wait to see next character

```
// hello world
main()  /* main */
{for(;;)
 {printf ("Hello World!\n");}
}
```

```
ident(“main”) lparen rparen lbrace for lparen semi semi
rparen lbrace printf lparen string(“Hello World!\n”)
rparen semi rbrace rbrace
```