Chapter 2 (continued)

More metalanguages
Grammar categories

Extended BNF (EBNF)

Extensions

[ ] used to enclose *optional* syntax
{ } used to enclose syntax that can be repeated *zero or more times* (like Kleene star)

EBNF

Examples

<Real_Number> → [+ | -] *<Unsigned_Real>*
<Unsigned_Real> → *<Integer>* [. *<Integer>*]
<Integer> → *<Digit>* {*<Digit>*}
<Expr> → *<Term>* { (+ | -) *<Term>* }

C-style EBNF

C-style EBNF lists alternatives on separate lines and uses $_{opt}$ to signify optional parts.

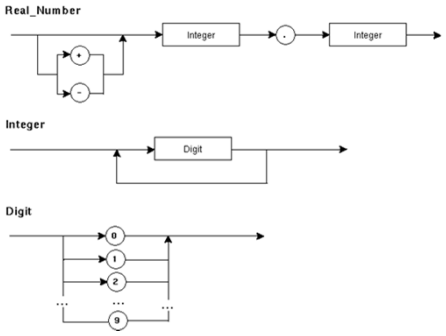
IfStatement:
`if` (*Expression*) *Statement* *ElsePart*$_{opt}$

ElsePart:
`else` *Statement*

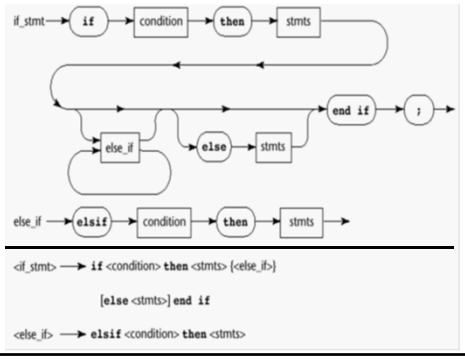
Syntax Graphs

A graphic method for representing EBNF grammar rules
Uses directed graphs with nodes that are terminals (ellipses) and nonterminals (rectangles)

Syntax Graphs



Ada **if** statement
(syntax graph and EBNF)



The Chomsky Hierarchy

Each class of grammar is differentiated by the set of productions that are permitted

- Type 3: Regular Grammars
- Type 2: Context-Free Grammars
- Type 1: Context-Sensitive Grammars
- Type 0: Unrestricted Grammars

Regular Grammars

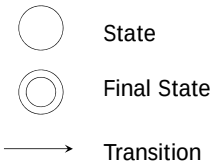
Sentences can be recognized by **Finite State Automata**

Have no memory, actions are determined by the current state and next input

Production rules have the form

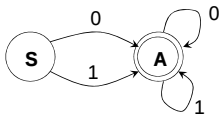
$\langle A \rangle \rightarrow a$
$\langle A \rangle \rightarrow a\langle B \rangle$

Finite State Automata (FSA)



FSA

An FSA for the binary number grammar:



Regular Expressions

Can be used to represent regular grammars, rather than production rules

Used by Unix utilities for string search & replace operations (grep, awk, vi, sed)

Also part of Perl and Ruby

Supported in Java and C# libraries

Regular Expressions

Pattern:	*Matches:*
ring	Diamond ring, string, stringent

. Matches any single character

Pattern:	*Matches:*
.ing	sling, ping

Regular Expressions

[ ] – Defines a class of characters that matches any single character in the brackets

Pattern:	*Matches:*
[bB]ill	billion, Bill Gates
Num [1-6]	Num 2, Num 49

Pattern:	*Matches:*
a.bc	axbc, aabc, !a5bczz
t[aeiou].k	talking, storks, teak

Regular Expressions

^ - Complement of a character class in [ ]
(Note: ^ also has another meaning in regexp's)

Pattern:	*Matches:*
[^a-zA-Z]	7, }, stop!

Regular Expressions

* Matches zero or more occurrences of the preceding item

Pattern:	*Matches:*
ab*c	ac, abc, abbc, abbbc
ab.*c	abc, abxc, abxyz123c
.*	9, (@$%!), qwerty2#

\+ Matches one or more occurrences

? Matches zero or one occurrence

Context-Free Grammars

Allows the definition of nested syntactic units

Sentences recognized by **push-down automata**

Productions have the form

$<A> \rightarrow B$

where B consists of zero or more terminals and/or non-terminals

BNF expresses context-free grammar rules

Most programming language constructs can be defined using context-free grammar rules

Context-Sensitive Grammars

Recognized by **linear-bounded automata**

Productions have the form

$A \rightarrow B$

where A and B are strings of terminals and non-terminals, and length of B $\geq$ length of A

Context-Sensitive Grammars

Example:

$\mathbf{w}\langle X\rangle\mathbf{y} \rightarrow \mathbf{wzy}$

(can only replace $\langle X\rangle$ by $\mathbf{z}$ in the *context* $\mathbf{w}\langle X\rangle\mathbf{y}$)

Example:

In a procedure call, the number of actual parameters must match the number of formal parameters

Unrestricted Grammars

Recognized by **Turing machines**

No restrictions on productions