CSCI312 Principles of Programming Languages

Chapter 3

Regular Expression and Lexer

Xu Liu

Recap

Clite: Lexical Syntax

Input: a stream of characters from the ASCII set, keyed by a programmer.

Output: a stream of *tokens* or basic symbols, classified as follows:

- *Identifiers* e.g., Stack, x, i, push
- *Literals* e.g., 123, 'x', 3.25, true
- *Keywords* bool char else false float if int main true while
- *Operators* = || && == != < <= > >= + - * / !
- *Punctuation* ; , { } ( )

Clite: Concrete Syntax

Based on a parse of its *Tokens*

; is a statement terminator

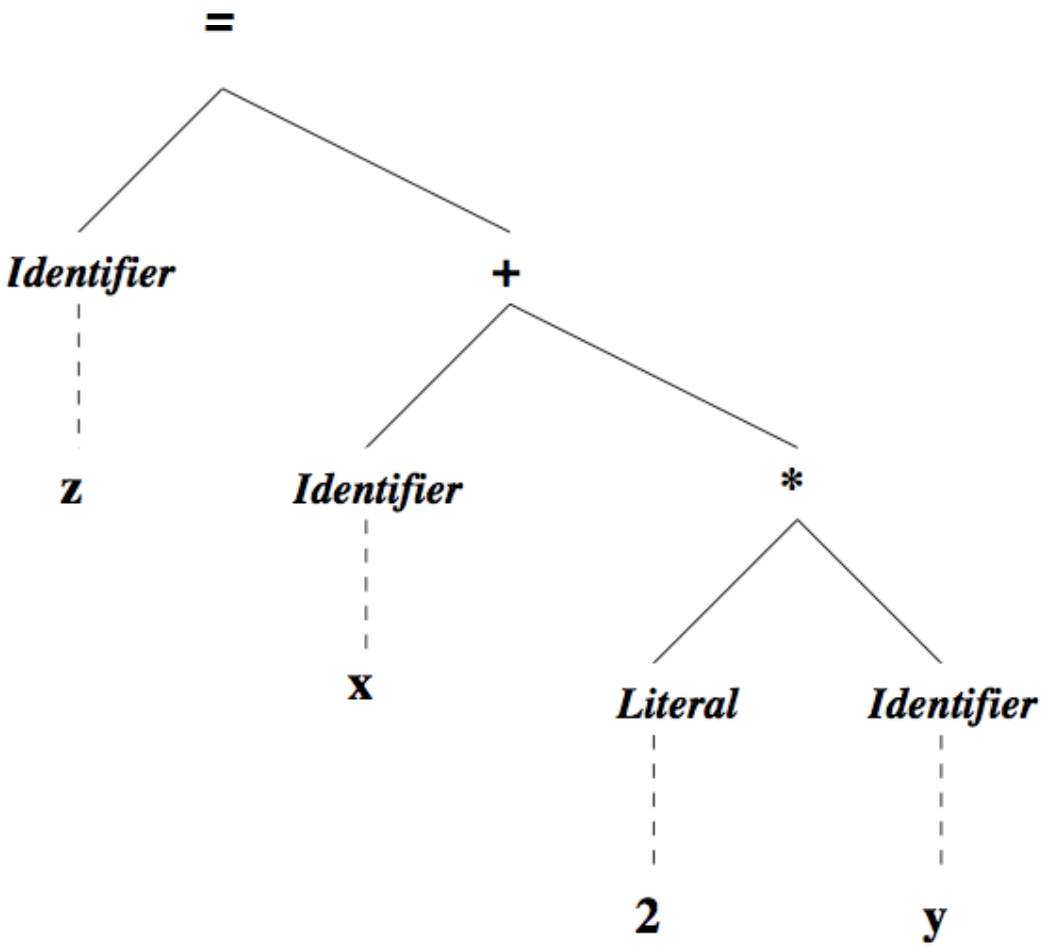
(Algol-60, Pascal use ; as a separator)

Rule for *IfStatement* is ambiguous:

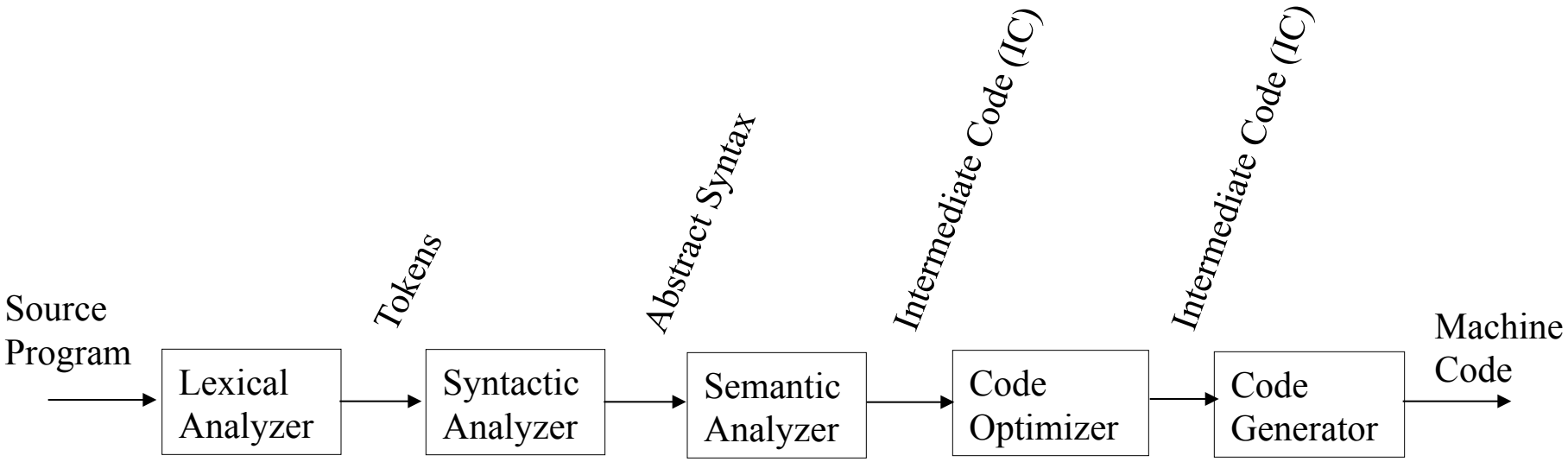
"The else ambiguity is resolved by connecting an **else** with the last encountered else-less if."

[Stroustrup, 1991]

Abstract Syntax Tree for
z = x + 2*y;
Fig. 2.10



Compilers and Interpreters



Contents

3.1 Chomsky Hierarchy

Regular grammar -- least powerful

Context-free grammar (BNF)

Context-sensitive grammar

Unrestricted grammar

Regular Grammar

Simplest; least powerful

Equivalent to:

- *Regular expression*
- *Finite-state automaton*

Right regular grammar: $\omega \in T^*$, $B \in N$

$A \rightarrow \omega B$

$A \rightarrow \omega$

Example

$$\textit{Integer} \rightarrow 0\ \textit{Integer} \mid 1\ \textit{Integer} \mid \ldots \mid 9\ \textit{Integer} \mid$$

$$0 \mid 1 \mid \ldots \mid 9$$

Regular Grammars

Left regular grammar: equivalent

Used in construction of tokenizers

Less powerful than context-free grammars

Not a regular language

$$A = a\ A\ b \mid \varepsilon$$

$\{ a^n b^n \mid n \geq 1 \}$

i.e., cannot balance: ( ), { }, begin end

$$A = a\ B\ b$$
$$B = a\ B\ b \mid \varepsilon$$

Context-free Grammars

BNF a stylized form of CFG

Equivalent to a pushdown automaton

For a wide class of unambiguous CFGs, there are table-driven, linear time parsers

Context-Sensitive Grammars

Production:

$\alpha \rightarrow \beta \qquad |\alpha| \leq |\beta|$

$\alpha, \beta \in (N \cup T)^*$

i.e., lefthand side can be composed of strings of terminals and nonterminals

Undecidable Properties of CSGs

Given a string ω and grammar G: $\omega \in L(G)$

$L(G)$ is non-empty

Defn: *Undecidable* means that you cannot write a computer program that is guaranteed to halt to decide the question for all $\omega \in L(G)$.

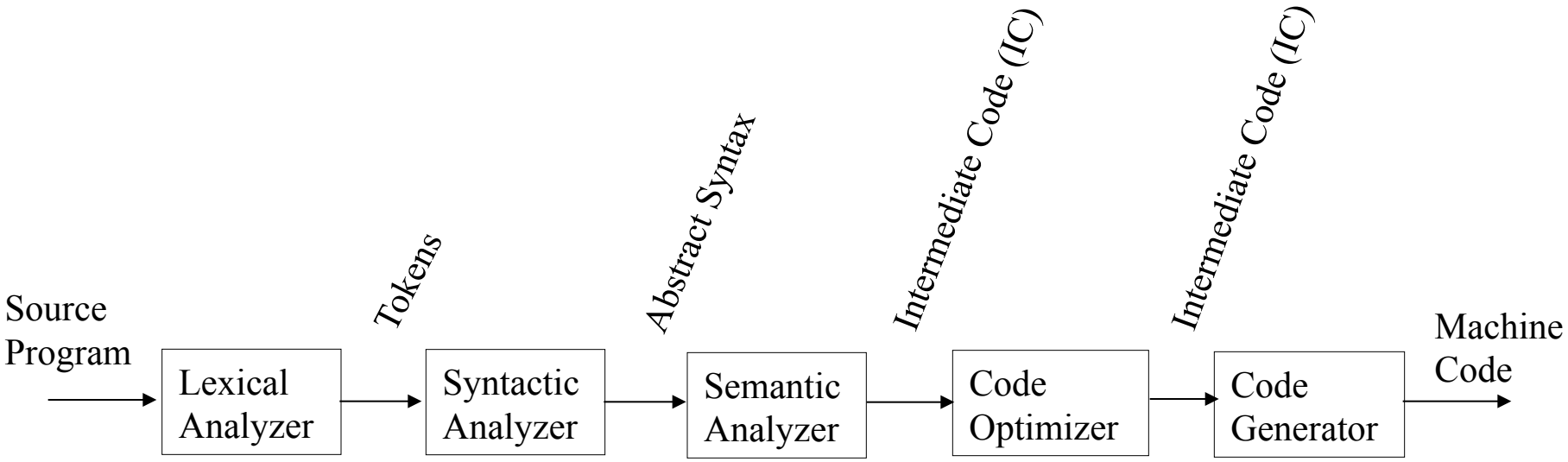
Unrestricted Grammar

Equivalent to:

- *Turing machine*
- *von Neumann machine*
- *C++, Java*

That is, can compute any computable function.

Review: Compilers and Interpreters



Lexical Analysis

Purpose: transform program representation

Input: printable ASCII characters

Output: tokens

Discard: whitespace, comments

Defn: A token is a logically cohesive sequence of characters representing a single symbol.

Example Tokens

Identifiers

Literals: 123, 5.67, 'x', true

Keywords: bool char ...

Operators: + - * / ...

Punctuation: ; , ( ) { }

Other Sequences

Whitespace: space tab

Comments

// any-char end-of-line*

End-of-line

End-of-file

Why a Separate Phase?

Simpler, faster machine model than parser

75% of time spent in lexer for non-optimizing compiler

Differences in character sets

End of line convention differs

Regular Expressions

RegExpr	Meaning
x	a character x
\x	an escaped character, e.g., \n
{ name }	a reference to a name
M | N	M or N
M N	M followed by N
M*	zero or more occurrences of M

RegExpr	Meaning
M+	One or more occurrences of M
M?	Zero or one occurrence of M
[aeiou]	the set of vowels
[0-9]	the set of digits
.	Any single character

Clite Lexical Syntax

Category	Definition
anyChar	[-~]
Letter	[a-zA-Z]
Digit	[0-9]
Whitespace	[\t]
Eol	\n
Eof	\004

Category	Definition
Keyword	bool | char | else | false | float | if | int | main | true | while
Identifier	{Letter}({Letter} | {Digit})*
integerLit	{Digit}+
floatLit	{Digit}+\.{Digit}+
charLit	‘{anyChar}’

Category	Definition
Operator	= | || | && | == | != | < | <= | > | >= | + | - | * | / |! | [|]
Separator	: | . | { | } | (|)
Comment	// ({anyChar} | {Whitespace})* {eol}

Generators

Input: usually regular expression

Output: table (slow), code

C/C++: Lex, Flex

Java: JLex

Finite State Automata

Set of states: representation – graph nodes

Input alphabet + unique end symbol

State transition function

Labelled (using alphabet) arcs in graph

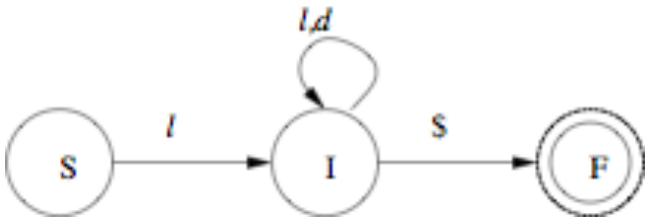
Unique start state

One or more final states

Deterministic FSA

Defn: A finite state automaton is *deterministic* if for each state and each input symbol, there is at most one outgoing arc from the state labeled with the input symbol.

A Finite State Automaton for Identifiers



What is a non-deterministic FSA?

Definitions

A *configuration* on an fsa consists of a state and the remaining input.

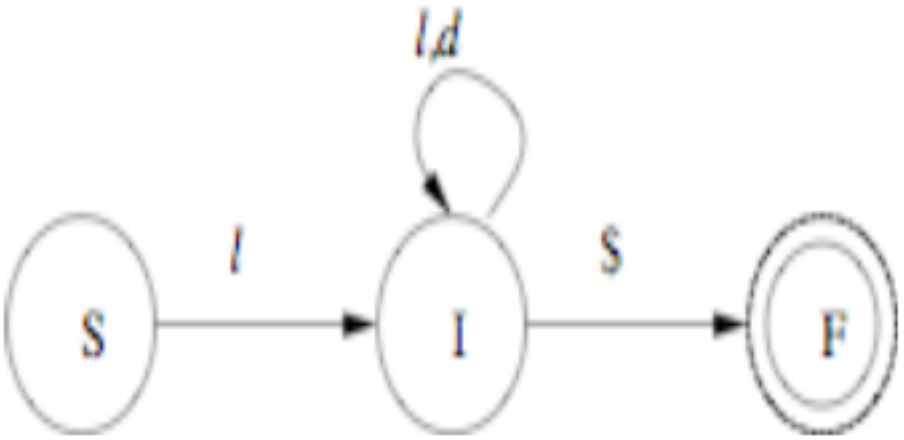
A *move* consists of traversing the arc exiting the state that corresponds to the leftmost input symbol, thereby consuming it. If no such arc, then:

- *If no input and state is final, then accept.*
- *Otherwise, error.*

An input is *accepted* if, starting with the start state, the automaton consumes all the input and halts in a final state.

Example

(S, a2i$) ⊢ (I, 2i$)
⊢ (I, i$)
⊢ (I, $)
⊢ (F,)



Thus: (S, a2i$) ⊢* (F,)

Chomsky Hierarchy

Regular grammar – least powerful

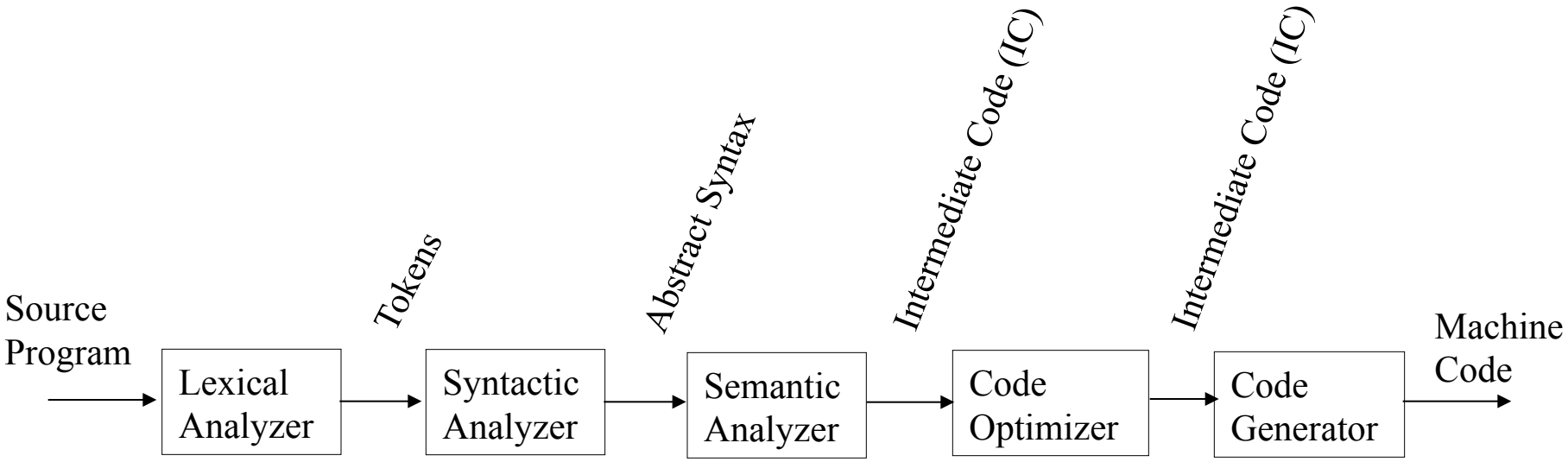
Context-free grammar (BNF)

Context-sensitive grammar

Unrestricted grammar

Contents

Review: Compilers and Interpreters



Syntactic Analysis

Phase also known as: parser

Purpose is to recognize source structure

Input: tokens

Output: parse tree or abstract syntax tree

A recursive descent parser is one in which each nonterminal in the grammar is converted to a function which recognizes input derivable from the nonterminal.