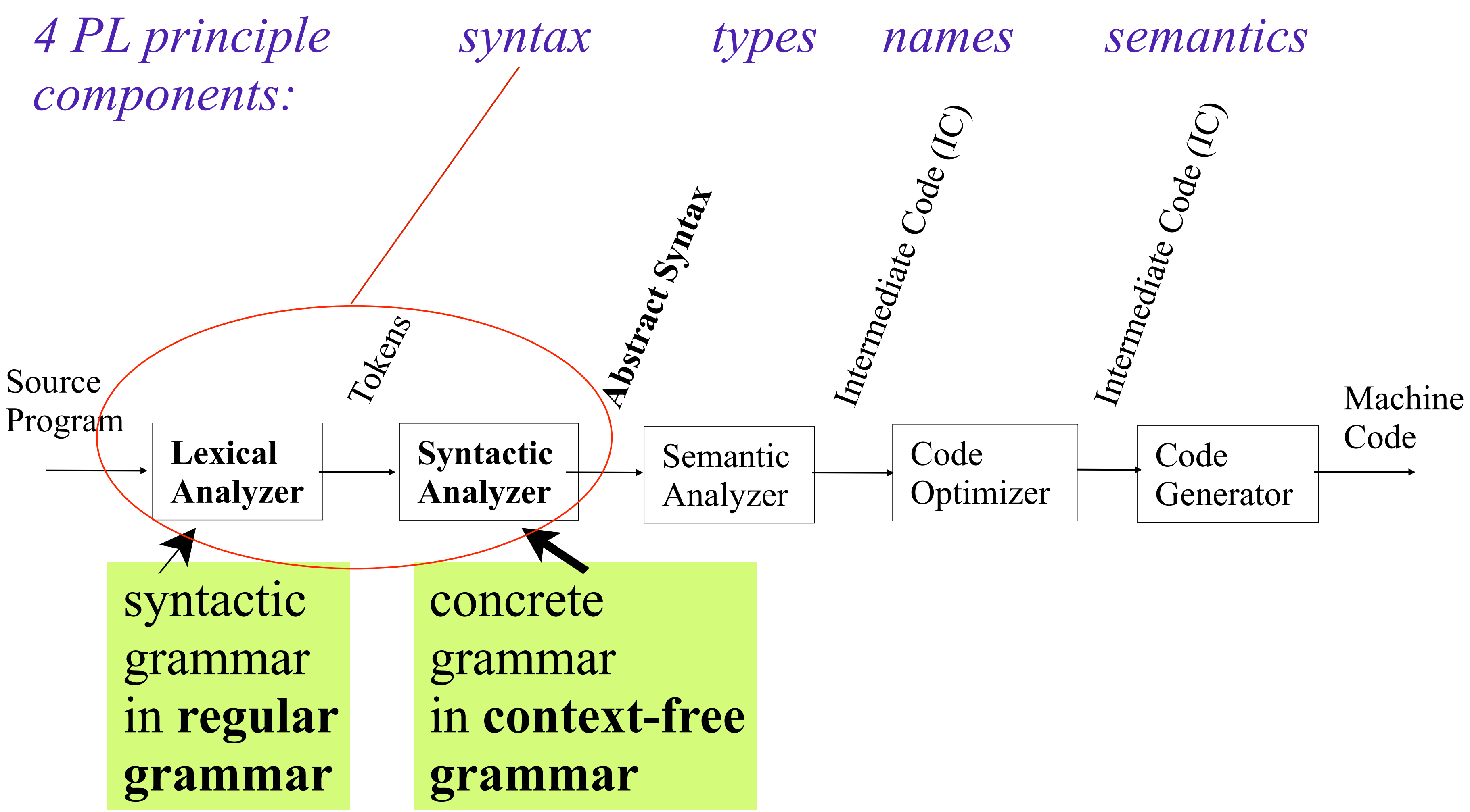
Review

Introduction

- What is a PL?
- Why to learn it?
- Four principled properties of a PL
  - Syntax: on grammar correctness
  - Names: for variables, functions, types, etc.
  - Types: collection of values and operations on them
  - Semantics: on meaning of a program
- Four main paradigms of PL
  - Imperative: program is a seq of commands
  - OO: a collection of objects that interact
  - Functional: a collection of mathematical functions
  - Logic: what to solve

- What makes a successful PL
  - Simplicity and readability
  - Clarity about binding
  - Reliability
  - Support
  - Abstraction
  - Orthogonality
  - Efficient implementation

Compiler and Interpreter



Regular Grammar

Simplest; least powerful in **Chomsky Hierarchy**

Equivalent to:

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

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

$A \rightarrow \omega B$

$A \rightarrow \omega$

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

BNF Grammar

$$A \to \omega$$

Set of *productions*: P $\quad A \in N \qquad \omega \in (N \cup T)^*$

terminal symbols: T

nonterminal symbols: N

start symbol: $S \in N$

Associativity and Precedence

Ambiguous Grammars

Extended BNF (EBNF)

EBNF: additional metacharacters

- **{ }** for a series of zero or more
- **( )** for a list, must pick one
- **[ ]** for an optional list; pick none or one

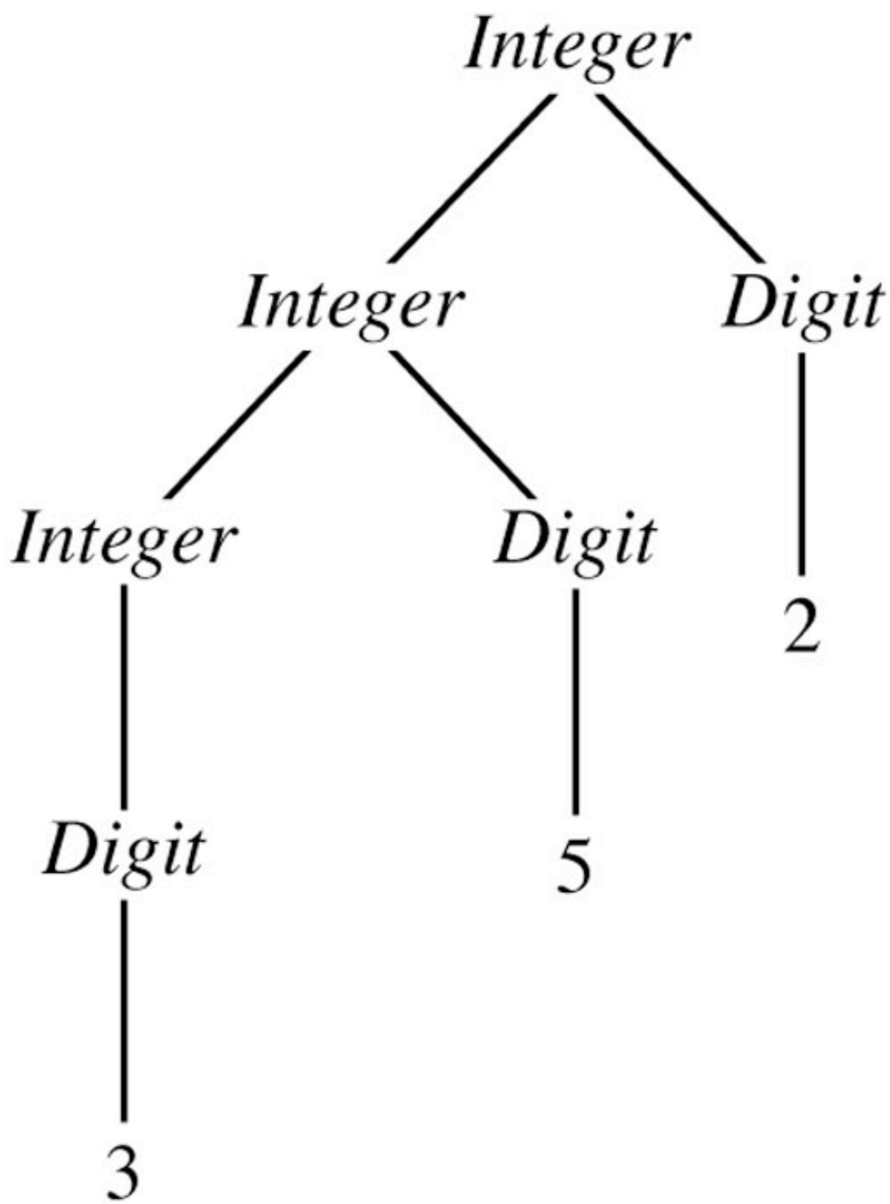
Parse Trees

A *parse tree* is a graphical representation of a derivation.

- *Each internal node of the tree corresponds to a step in the derivation.*
- *The children of a node represents a right-hand side of a production.*
- *Each leaf node represents a symbol of the derived string, reading from left to right.*

Parse Tree for 352 as an *Integer*

Figure 2.1



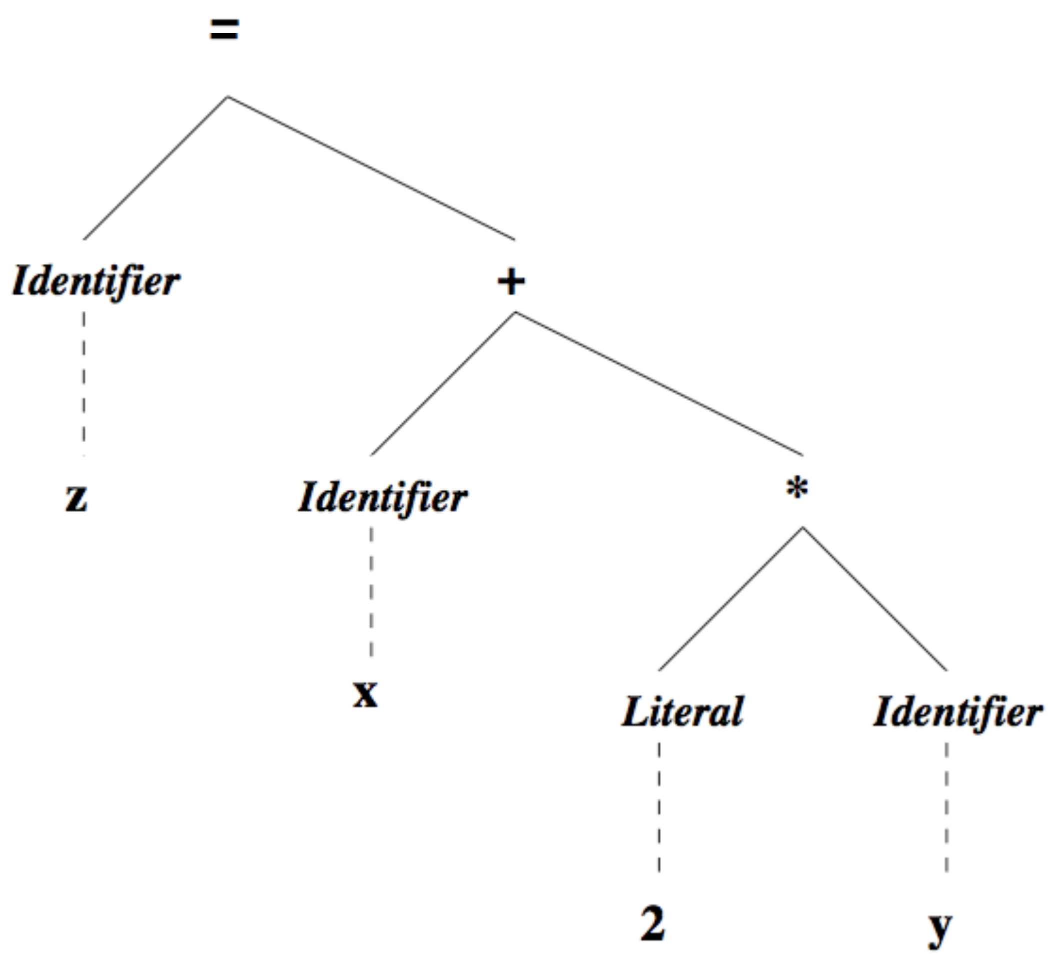
Abstract Syntax Tree

Output: parse tree is inefficient

- One nonterminal per precedence level

Shape of parse tree that is important

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



Syntactic Analysis (Parser)

- *LL & LR Grammar and Parser*
  - A-> A w ?
- *Recursive Descent Parser*
  - Each non-terminal has a corresponding function
  - FirstSet and its calculation
    - algorithm for computing the nullable sets

Computer FirstSet

For w = X1 ... Xn V ...

First(w) = First(X1) ∪ ... ∪ First(Xn) ∪ First(V)

where X1, ..., Xn are nullable

and V is not nullable

A is *nullable* if it derives the empty string.

Nullable algorithm.

Recursive Descent Parser

- Grammar rewriting for convenience of parser development
- Apply FirstSet
- Parser development with an example
  - Basic coding
  - Output abstract syntax tree

Predictive Parsing

Basic idea

Given $A \rightarrow \alpha \mid \beta$, the parser should be able to choose between α & β

FIRST sets

For some rhs $\alpha \in G$, define FIRST(α) as the set of tokens that appear as the first symbol in some string that derives from α

That is, $\underline{x} \in$ FIRST(α) iff $\alpha \Rightarrow^* \underline{x}\, \gamma$, for some γ

The LL(1) Property

If $A \rightarrow \alpha$ and $A \rightarrow \beta$ both appear in the grammar, we would like

$$\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \varnothing$$

This would allow the parser to make a correct choice with a lookahead of exactly one symbol !

This is almost correct
See the next slide

LL and LR Parsers