Lexer

COP-3402 Systems Software

Paul Gazzillo

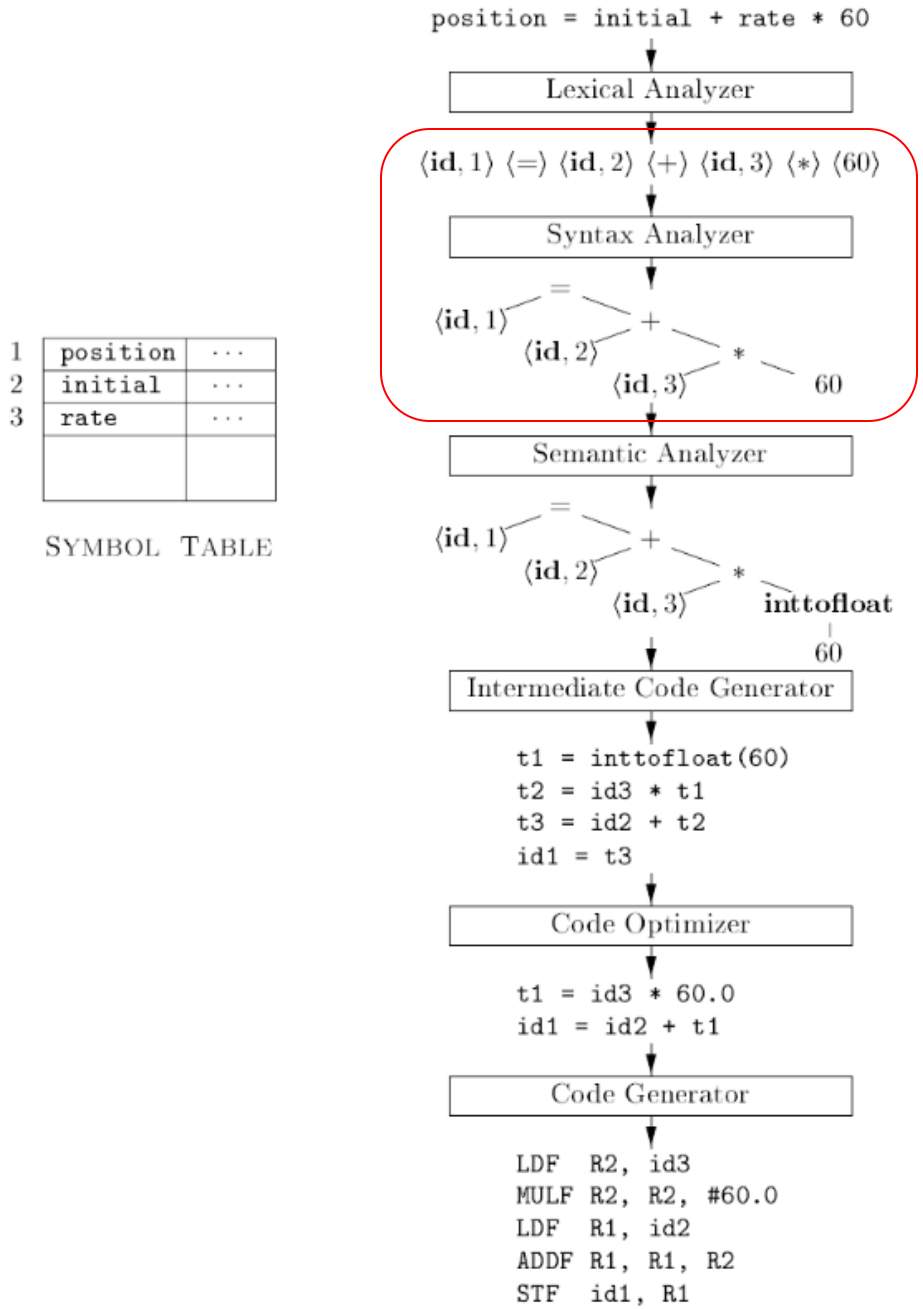


Figure 1.7: Translation of an assignment statement

Core Idea: Recognize Syntax of our Programs

- Lexer recognizes words (lexemes) of the language
- Languages have nested structure
  - e.g., if (x) { while (!y) { … } }
- Regular expressions cannot express this syntax
- Need more powerful type of language: context-free languages

Formal Languages

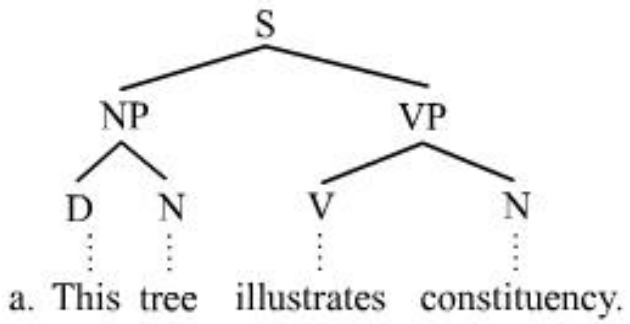
- An *alphabet* is a **finite** set of symbols
  - e.g., ASCII characters
- A *string* over an alphabet is a **finite** sequence of symbols
  - e.g., tokens in PL/0
- ε is the empty string
- A *language* is a **possibly infinite** set of strings over an alphabet
  - Ø is the empty language

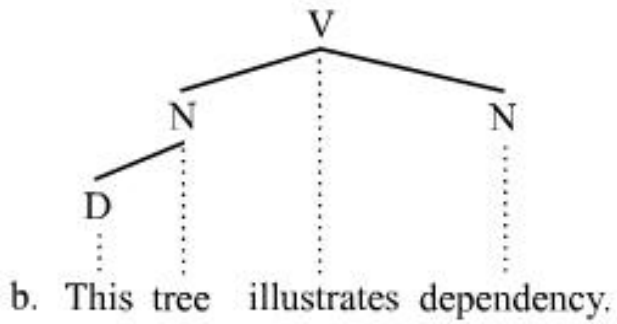
Regular Expressions are Limited

- No counting
- No nesting
- Example: matching a's and b's or matching parentheses
  - L = { a^n b^n | for all n = 0, 1, 2, … }
  - Intuition: need infinite number of states for all combinations of a and b
- Proving regularity
  - Pumping lamma
  - Myhill-Nerode
  - (Take big discrete)

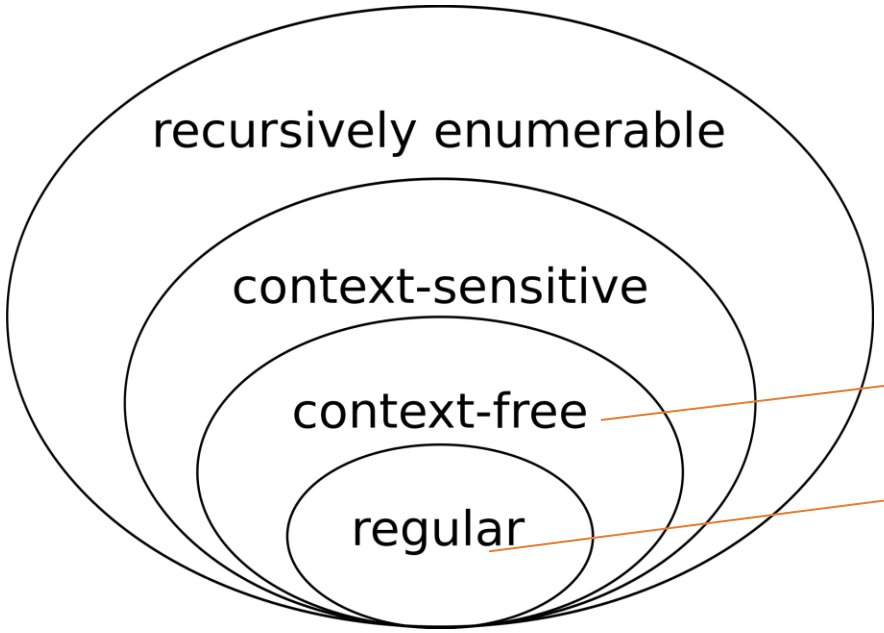
Enter Context-Free Languages (CFLs)

- Superset of regular languages
- Captures nested structure of language, syntax
- Expressed with *context-free grammars* (CFGs)
  - Think elementary school grammar
  - Think sentence diagramming

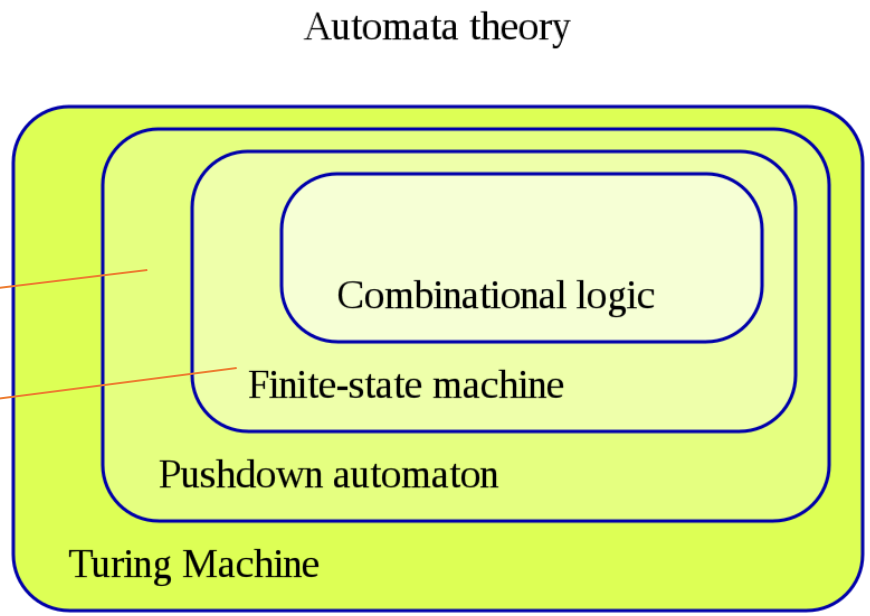




Language Correspond to Machines



Chomsky language hierarchy



Expressing Languages

- *Regular languages* described with *regular expressions*, e.g., (a|b)*abb
- *Context-free languages* described with *context-free grammars* (CFGs)

Context-Free Grammars (CFG) Express Syntax

- CFGs let us express language syntax
  - Infinite language
  - Finite representation
- Captures hierarchical nature of languages
- Intuition: syntactically correct, even if not meaningful
  - "Colorless green ideas sleep furiously."
    VS.
  - "Furiously sleep ideas green colorless."

  (Chomsky, Syntactic Structures)

Context-Free Grammars

- Elements of language, "sentence", "subject", "blue"
- Broken down into smaller elements
  - Subject is an article and a noun
- Until we get to words
  - Lexical analysis gives us these!
- Elements have multiple substitutions
  - Several types of sentences

sentence -> subject verb object
sentence -> subject verb
subject -> article noun
subject -> article adjective noun
article -> "the"
article -> "an"
adjective -> "blue"
adjective -> "nine"
...

Formal Definition of Context-Free Grammars

- Symbols are either *terminals* (words) or *nonterminals* (structures)
- Grammar rules are *productions* with a *starting symbol*
- CFGs defined by
  - Terminals
  - Nonterminals
  - Productions
    - Denoted by "->"
  - Starting symbol
    - Nonterminal of first production

$$
\begin{aligned}
\textit{expression} &\rightarrow \textit{expression} + \textit{term} \\
\textit{expression} &\rightarrow \textit{expression} - \textit{term} \\
\textit{expression} &\rightarrow \textit{term} \\
\textit{term} &\rightarrow \textit{term} * \textit{factor} \\
\textit{term} &\rightarrow \textit{term} \;/\; \textit{factor} \\
\textit{term} &\rightarrow \textit{factor} \\
\textit{factor} &\rightarrow (\; \textit{expression} \;) \\
\textit{factor} &\rightarrow \mathbf{id}
\end{aligned}
$$

Figure 4.2: Grammar for simple arithmetic expressions

All Regular Languages are also Context-Free

- But not vice-versa
- Concatenation, e.g., ab

  A -> a b
- Union, e.g., a|b

  A -> a
  A -> b
- Closure, e.g., a*

  A -> A a

  A -> ε

Derivations

Demo

Parse Trees

Demo

Ambiguous Grammars

Demo

$$
\begin{aligned}
\textit{expression} &\rightarrow \textit{expression} + \textit{term} \\
\textit{expression} &\rightarrow \textit{expression} - \textit{term} \\
\textit{expression} &\rightarrow \textit{term} \\
\textit{term} &\rightarrow \textit{term} * \textit{factor} \\
\textit{term} &\rightarrow \textit{term} \;/\; \textit{factor} \\
\textit{term} &\rightarrow \textit{factor} \\
\textit{factor} &\rightarrow (\; \textit{expression} \;) \\
\textit{factor} &\rightarrow \mathbf{id}
\end{aligned}
$$

Figure 4.2: Grammar for simple arithmetic expressions

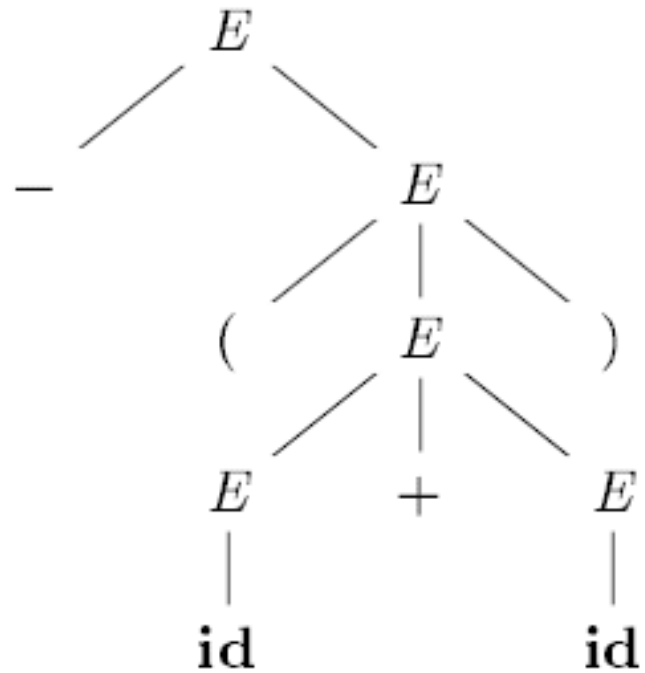


Figure 4.3: Parse tree for $-(\mathbf{id} + \mathbf{id})$

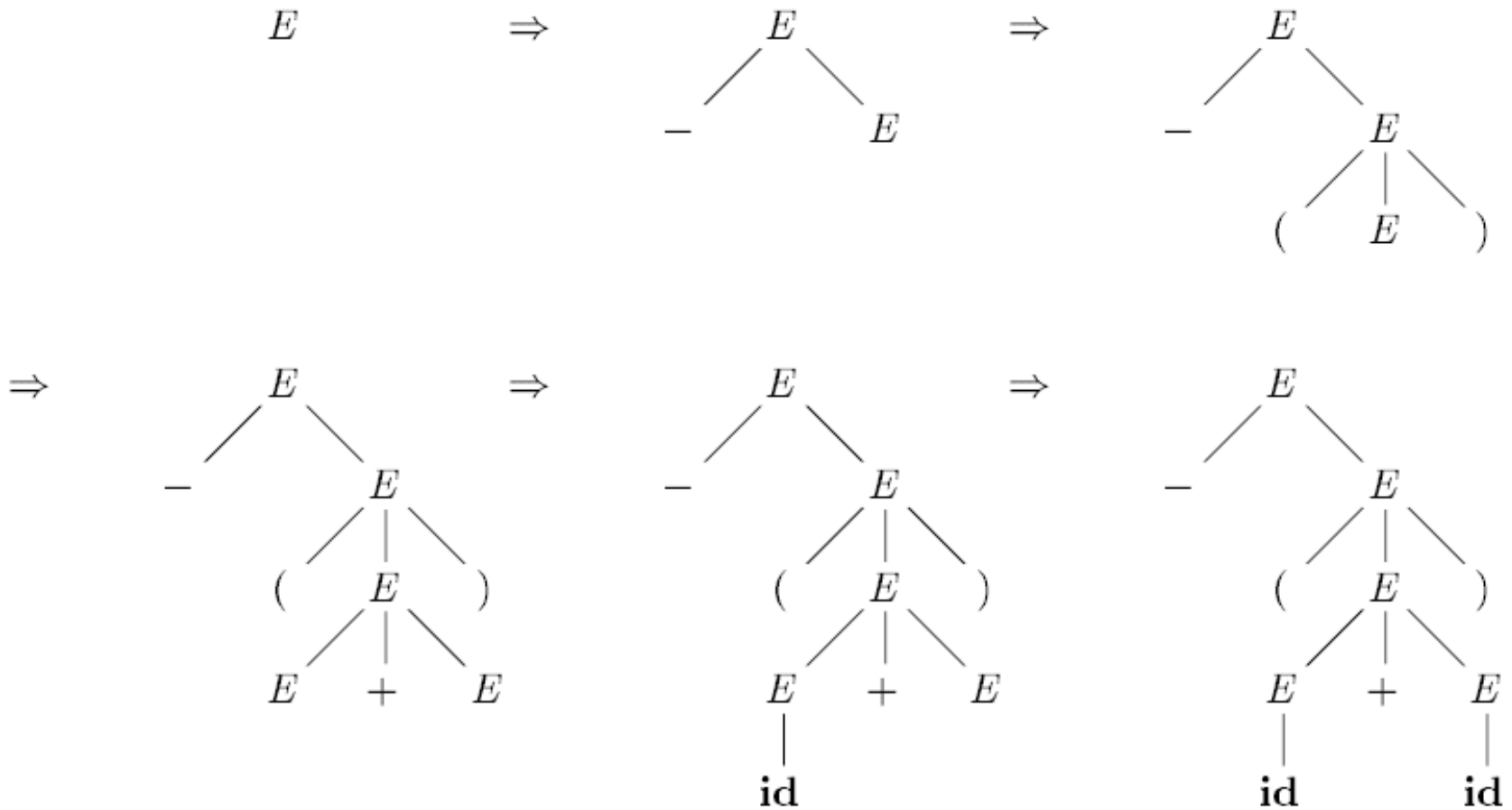


Figure 4.4: Sequence of parse trees for derivation (4.8)

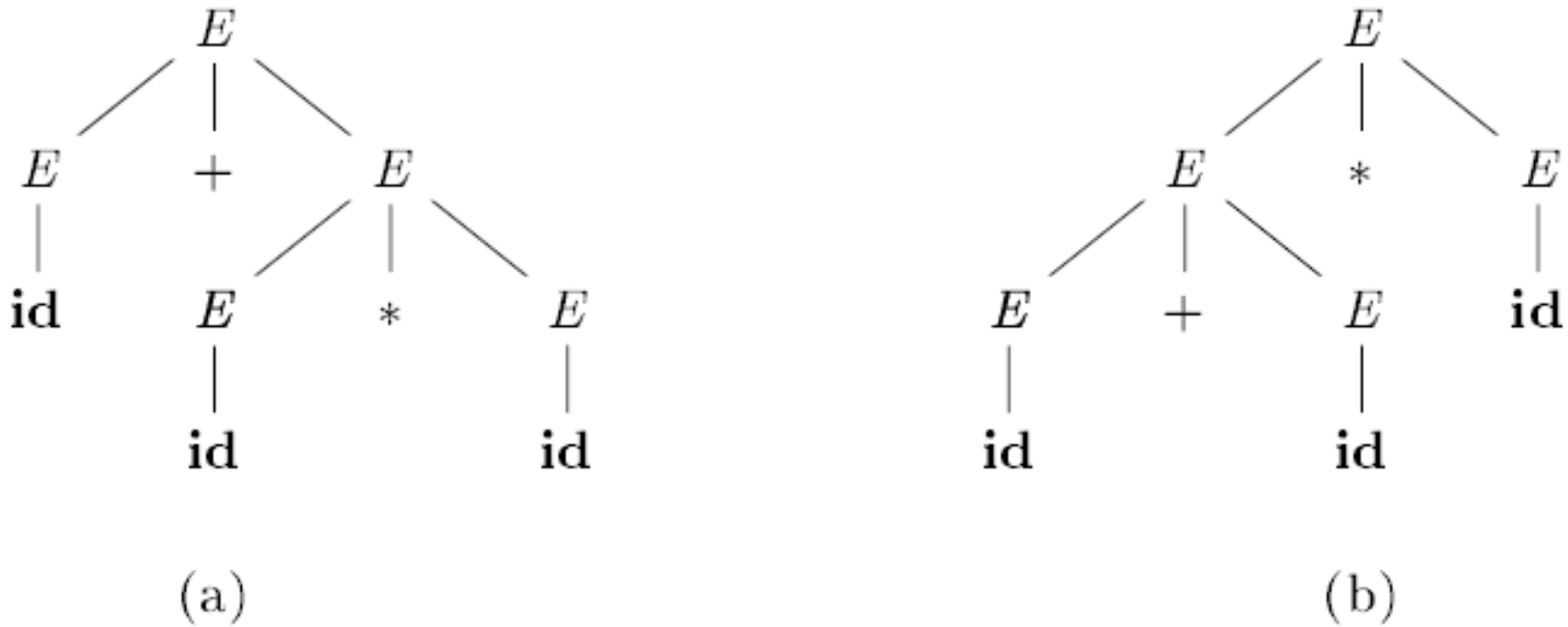


Figure 4.5: Two parse trees for **id+id*id**