Compiler Design

Fatemeh Deldar

Isfahan University of Technology

1402-1403

Converting a Regular Expression Directly to a DFA

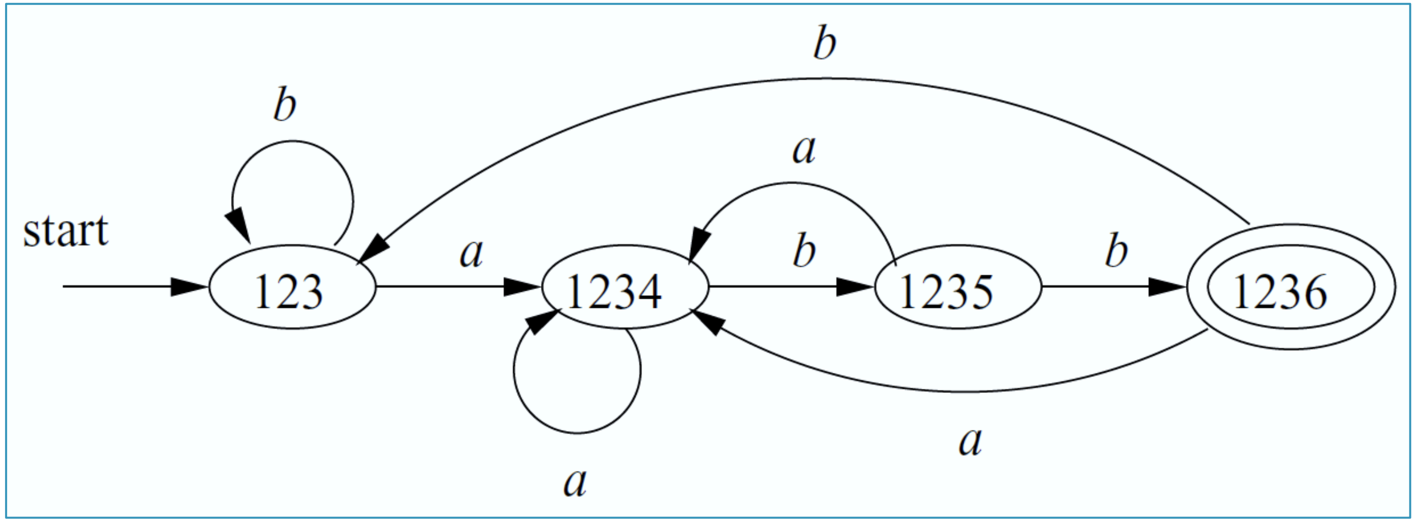
- **Computing followpos**
  1. If n is a cat-node with left child c_1 and right child c_2, then for every position i in *lastpos*(c_1), all positions in *firstpos*(c_2) are in *followpos*(i)
  2. If n is a star-node, and i is a position in *lastpos*(n), then all positions in *firstpos*(n) are in *followpos*(i)

- **Example**

POSITION n	*followpos*(n)
1	$\{1, 2, 3\}$
2	$\{1, 2, 3\}$
3	$\{4\}$
4	$\{5\}$
5	$\{6\}$
6	$\emptyset$

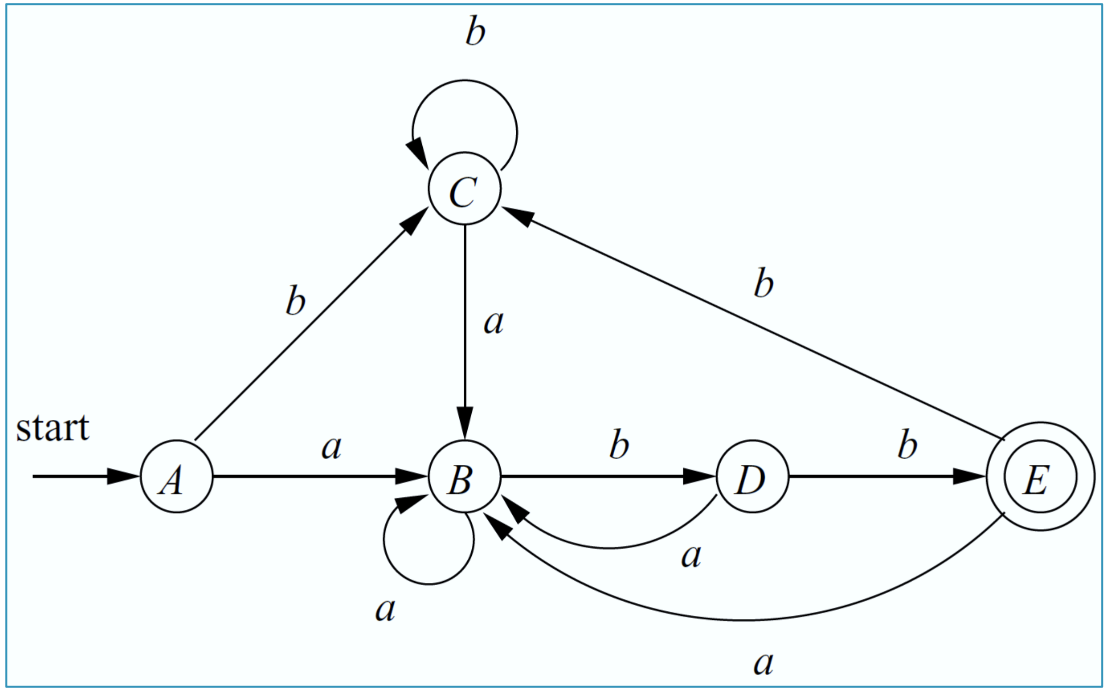
Converting a Regular Expression Directly to a DFA

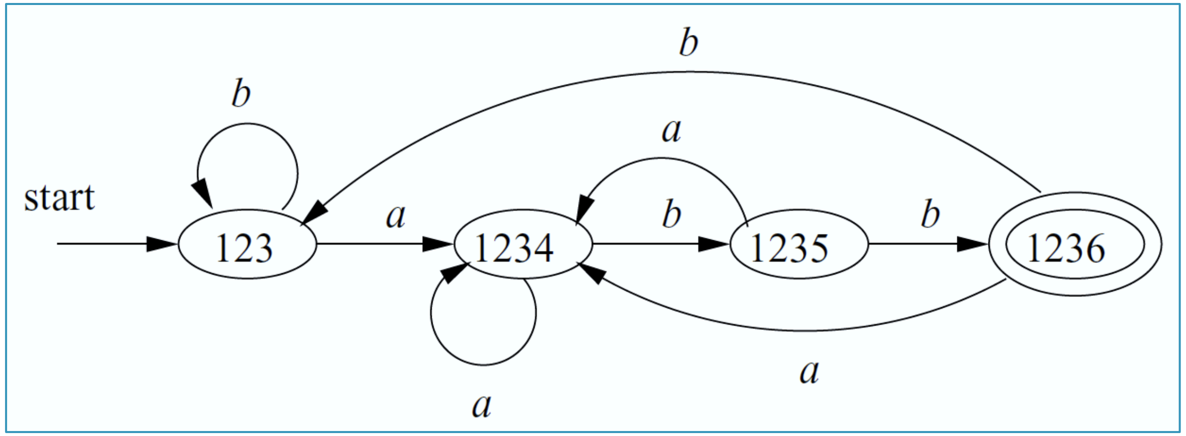
- **Example**



Minimizing the Number of States of a DFA

- There can be many DFAs that recognize the same language
- **Example:** $L((a|b)^*abb)$





Minimizing the Number of States of a DFA

- There is always a unique minimum-state DFA for any regular language
- **Algorithm**
  1. Start with an initial partition Π with two groups, F and $S - F$, the accepting and nonaccepting states of D
  2. initially, let $\Pi_{\text{new}} = \Pi$;

     for (each group G of Π) {

     partition G into subgroups such that two states s and t

     are in the same subgroup if and only if for all

     input symbols a, states s and t have transitions on a

     to states in the same group of Π;

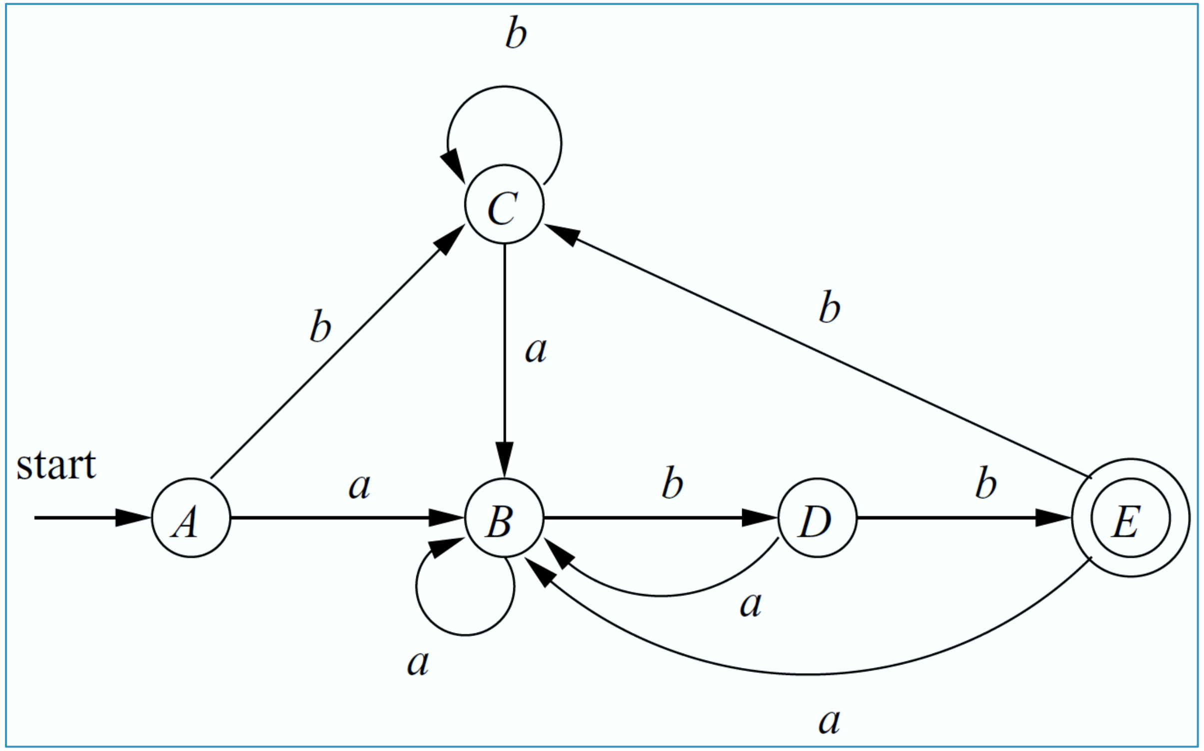
     /* at worst, a state will be in a subgroup by itself */

     replace G in Π_{new} by the set of all subgroups formed;

     }
  3. If $\Pi_{new} != \Pi$, repeat step (2) with Π_{new} in place of Π

Minimizing the Number of States of a DFA

- **Example**
  - $\{A, B, C, D\}\{E\}$
  - $\{A, B, C\}\{D\}\{E\}$
  - $\{A, C\}\{B\}\{D\}\{E\}$

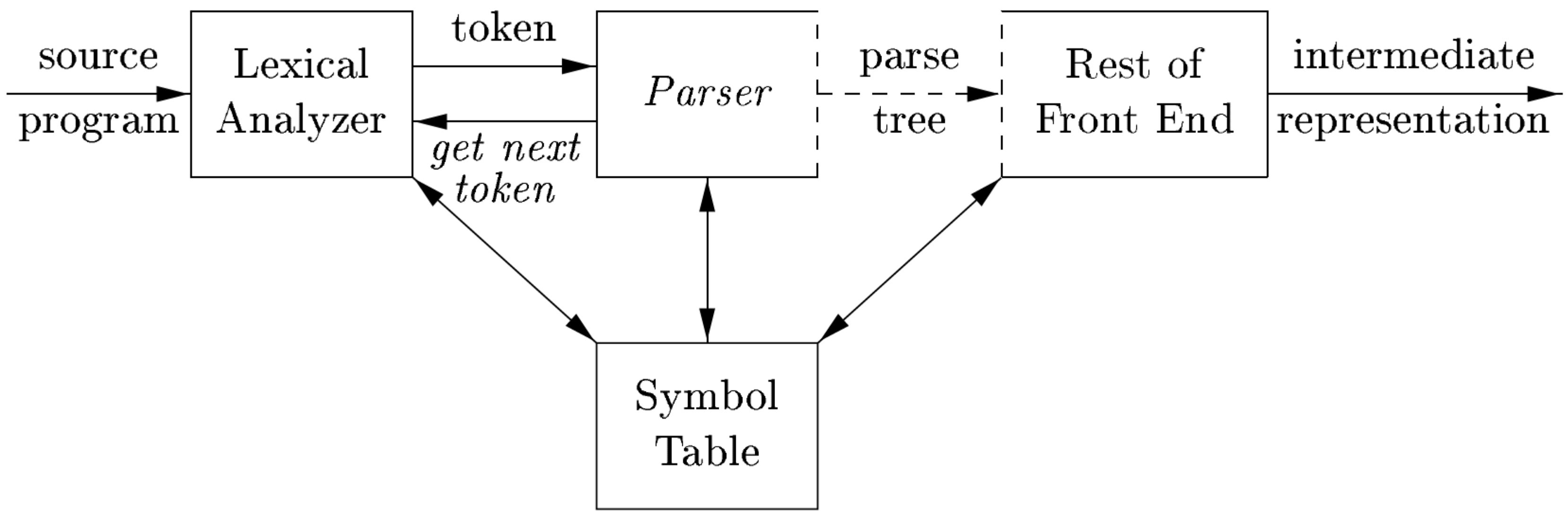


Syntax Analysis

Introduction

- Every programming language has precise rules that prescribe the syntactic structure of well-formed programs
- The syntax of programming language constructs can be specified by **context-free grammars**
- The parser **obtains a string of tokens from the lexical analyzer** and **verifies that the string of token names can be generated by the grammar for the source language**
- The parser constructs a parse tree and passes it to the rest of the compiler for further processing

Introduction



Introduction

- The methods commonly used in compilers for parser can be classified as:
  - **Top-down**
    - Top-down methods build parse trees from the top (root) to the bottom (leaves)
  - **Bottom-up**
    - Bottom-up methods start from the leaves and work their way up to the root
- The input to the parser is scanned from left to right, one symbol at a time

Context-Free Grammars

- **A context-free grammar consists of:**
  - **Terminals**
    - Terminals are the basic symbols from which strings are formed
  - **Nonterminals**
    - Nonterminals are syntactic variables that denote sets of strings
  - **Start symbol**
    - One nonterminal is distinguished as the start symbol
  - **Productions**
    - The productions of a grammar specify the manner in which the terminals and nonterminals can be combined to form strings
    - **A production consists of:**
      - A nonterminal called the head or left side of the production
      - The symbol $\rightarrow$
      - A body or right side consisting of zero or more terminals and nonterminals

Context-Free Grammars

- **Example**
  - Terminal: id + - * / ( )
  - Nonterminals: expression, term, factor

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

- **Example**

$$
\begin{aligned}
E &\rightarrow E + T \mid E - T \mid T \\
T &\rightarrow T * F \mid T \,/\, F \mid F \\
F &\rightarrow (\ E\ ) \mid \mathbf{id}
\end{aligned}
$$

Derivations

- **Example** $$E \rightarrow E + E \mid E * E \mid - E \mid (E) \mid \mathbf{id}$$
  - A derivation of -(**id**) from E is: $$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(\mathbf{id})$$
- If $S \overset{*}{\Rightarrow} \alpha$, where S is the start symbol of a grammar G, we say that α is a **sentential form** of G
  - A sentential form may contain both terminals and nonterminals
  - The strings E, -E, -(E), -(id + id) are all sentential forms
- A **sentence** of G is a sentential form with no nonterminals
- **The language generated by a grammar is its set of sentences**
- **Example**
$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(\mathbf{id} + E) \Rightarrow -(\mathbf{id} + \mathbf{id})$$

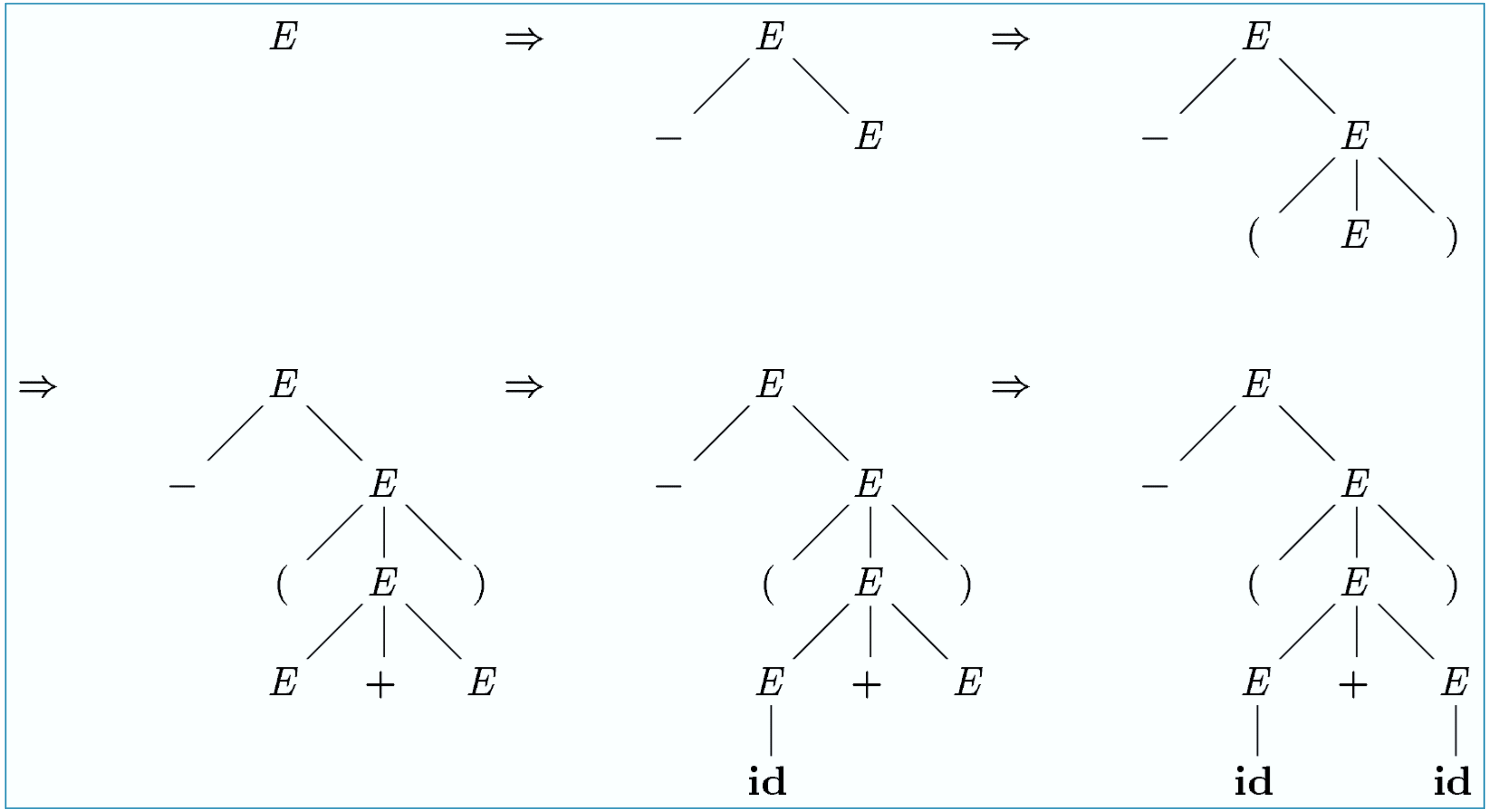
Derivations

- **Leftmost derivations** $\boxed{\alpha \underset{lm}{\Rightarrow} \beta}$
  - The leftmost nonterminal in each sentential is always chosen
- **Rightmost derivations** $\boxed{\alpha \underset{rm}{\Rightarrow} \beta}$
  - The rightmost nonterminal in each sentential is always chosen
- **Example**

$$\boxed{E \underset{lm}{\Rightarrow} -E \underset{lm}{\Rightarrow} -(E) \underset{lm}{\Rightarrow} -(E+E) \underset{lm}{\Rightarrow} -(\mathbf{id}+E) \underset{lm}{\Rightarrow} -(\mathbf{id}+\mathbf{id})}$$

Parse Trees

- A parse tree is a graphical representation of a derivation that filters out the order in which productions are applied to replace nonterminals

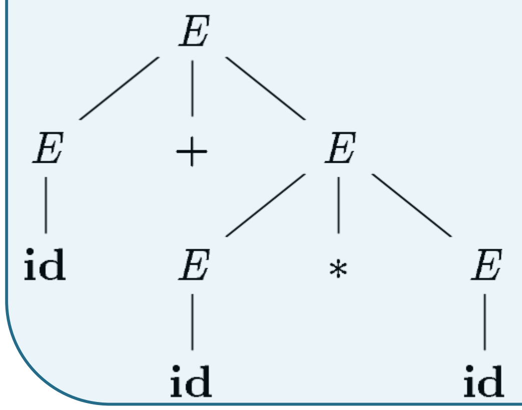


Ambiguity

- A grammar that produces more than one parse tree for some sentence is said to be **ambiguous**
  - An ambiguous grammar is one that produces more than one leftmost derivation or more than one rightmost derivation for the same sentence

$$E \to E + E \mid E * E \mid -E \mid (E) \mid \mathbf{id}$$

$$\begin{aligned} E &\Rightarrow E + E \\ &\Rightarrow \mathbf{id} + E \\ &\Rightarrow \mathbf{id} + E * E \\ &\Rightarrow \mathbf{id} + \mathbf{id} * E \\ &\Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id} \end{aligned}$$



$$\begin{aligned} E &\Rightarrow E * E \\ &\Rightarrow E + E * E \\ &\Rightarrow \mathbf{id} + E * E \\ &\Rightarrow \mathbf{id} + \mathbf{id} * E \\ &\Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id} \end{aligned}$$