

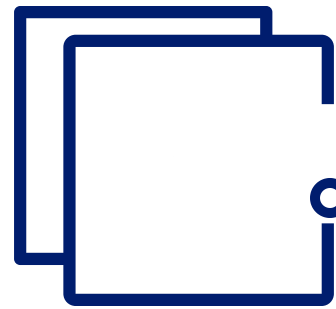


LEXICAL SYNTAX

ANALYSIS



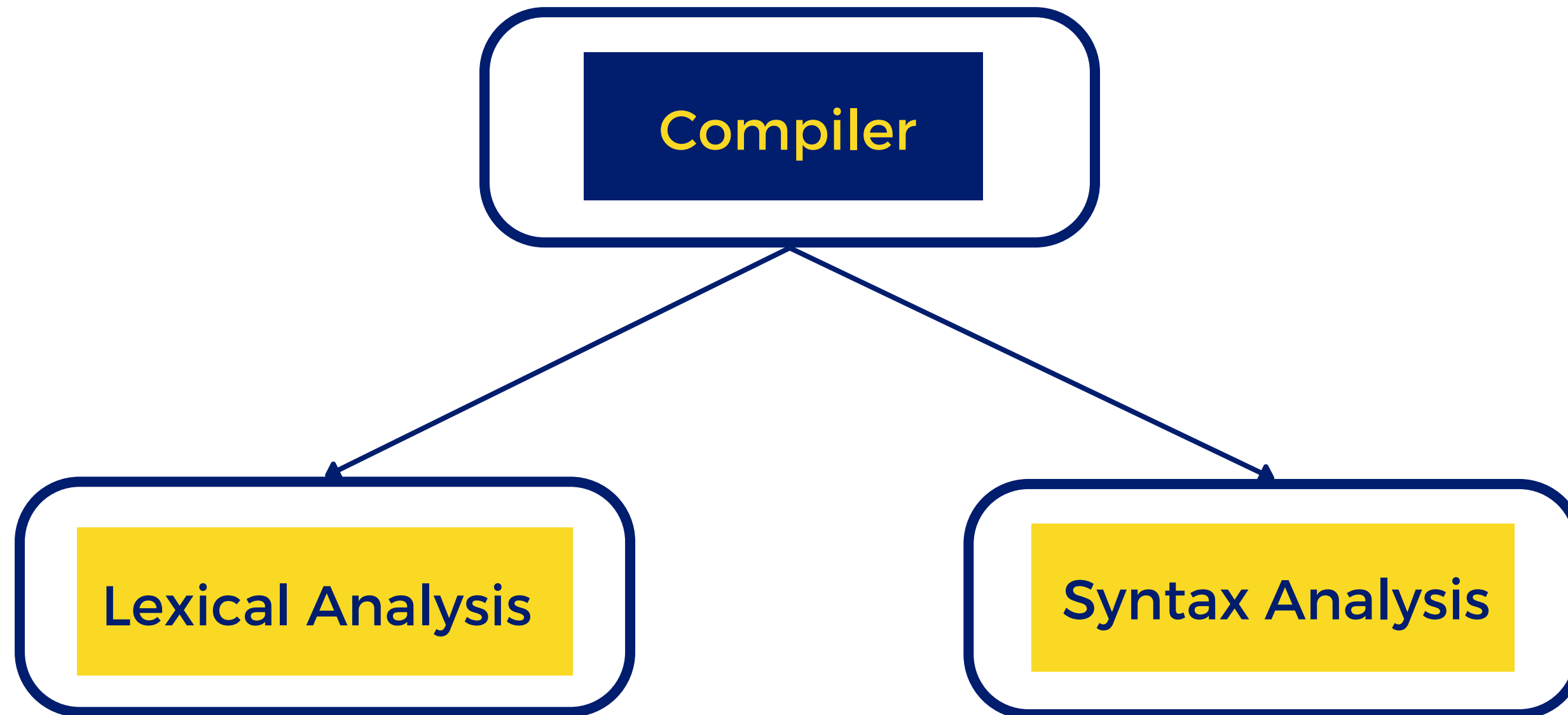
Monzales, Samson, and Tejada



CONTENT

- INTRODUCTION
- LEXICAL ANALYSIS
- THE PARSING PROBLEM
- RECURSIVE-DESCENT PARSING
- BOTTOM-UP PARSING
- SUMMARY

INTRODUCTION



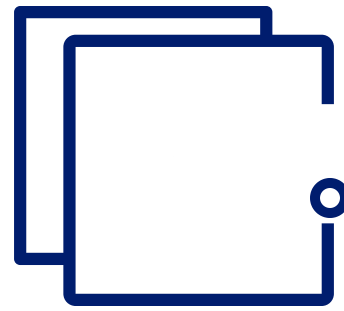


INTRODUCTION



Why do we separate Lexical and Syntax Analysis?

- Simplicity - lexical analysis is less complex
- Efficiency - lexical analysis can be optimized since it requires more compilation time
- Portability - lexical analyzer is platform dependent whereas syntax analyzer is platform independent



LEXICAL ANALYSIS

Lexical Analyzer

- is a **pattern matcher** for **character strings**
- is a “**front-end**” for the **parser**
- Identifies **substrings** of the source program that belong together - **lexemes**
 - Lexemes match a character pattern, which is associated with a lexical category called a **token**



LEXICAL ANALYSIS

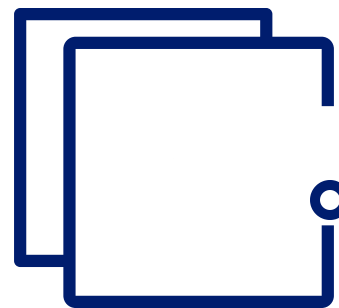
Lexical Analyzer

- Scans the Pure HLL (High-Level Language) code **line by line**
- Takes **Lexemes** as **input** and produces **Tokens** as **output**
- Removes **comments** and **whitespaces** from the Pure HLL code

LEXICAL ANALYSIS

Lexical Analyzer





LEXICAL

`x = a+b*c;`



Lexical Analysis



Lexemes	Tokens
x	identifier
=	operator
a	identifier
+	operator
b	identifier
*	operator
c	identifier

Monzales, Samson, and Tejada



LEXICAL ANALYSIS

Three approaches to building a lexical analyzer:

- Write a formal description of the tokens and use a software tool that constructs table-driven lexical analyzers given such a description
- Design a state diagram that describes the tokens and write a program that implements the state diagram
- Design a state diagram that describes the tokens and hand-construct a table-driven implementation of the state diagram

LEXICAL ANALYSIS

C-Tokens:

- **if:**



- **Identifier:**

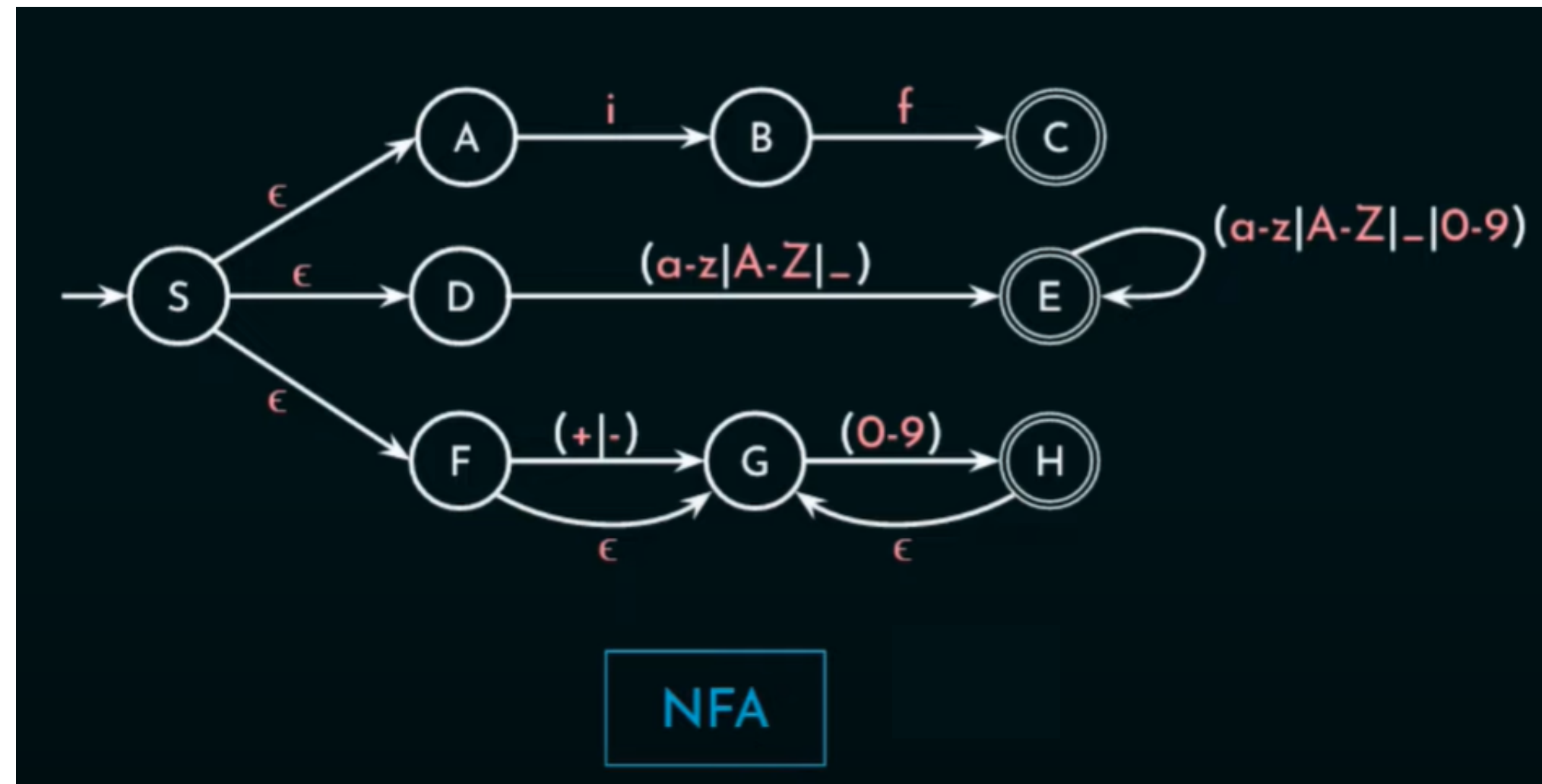


- **Integer:**



LEXICAL ANALYSIS

State diagram: Non-deterministic Finite Automata (NFA)





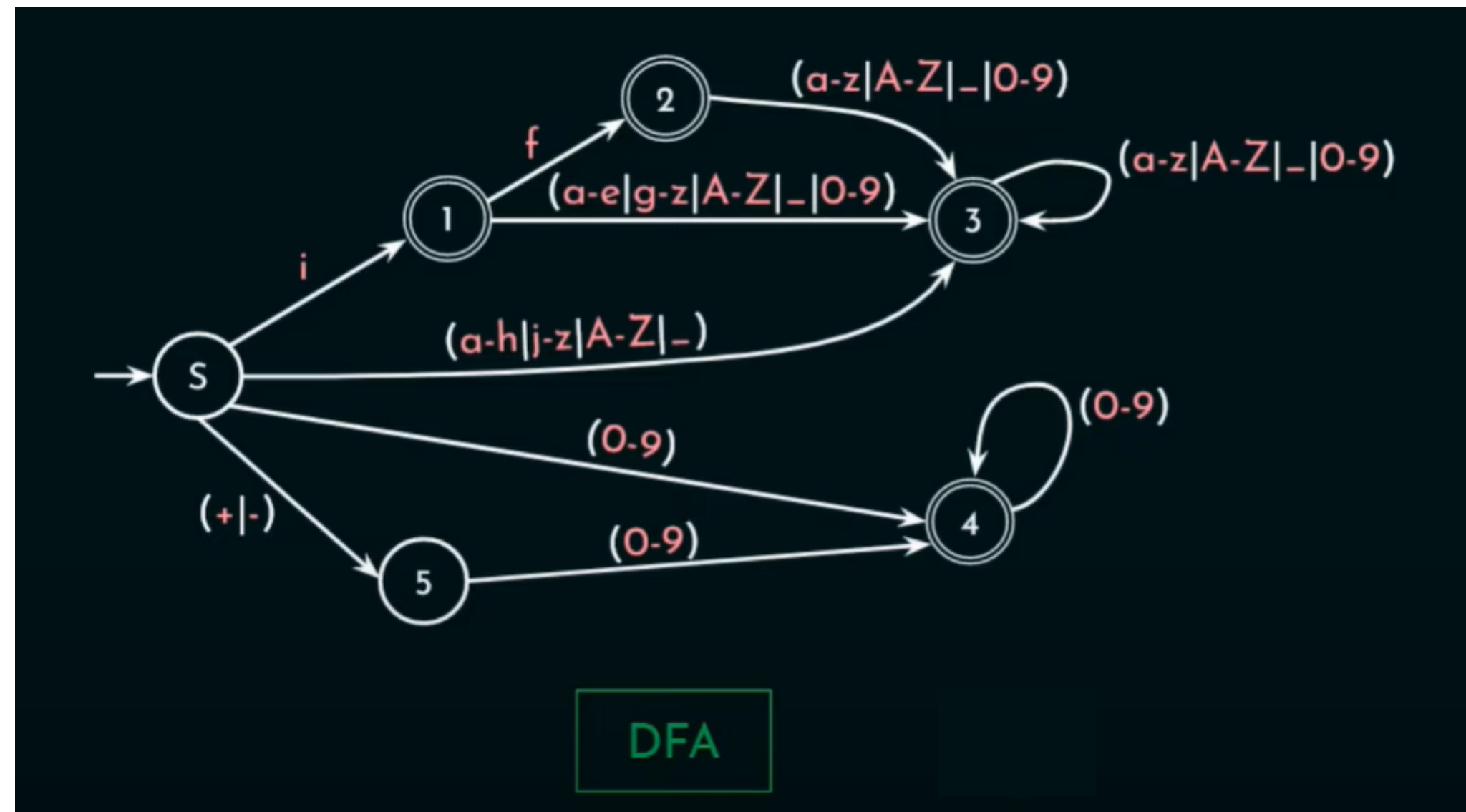
LEXICAL ANALYSIS

State Diagram

- **NFA is purely conceptual, so it cannot be implemented.**
- **Hence, conversion to DFA is necessary.**

LEXICAL ANALYSIS

State diagram: Deterministic Finite Automata (DFA)





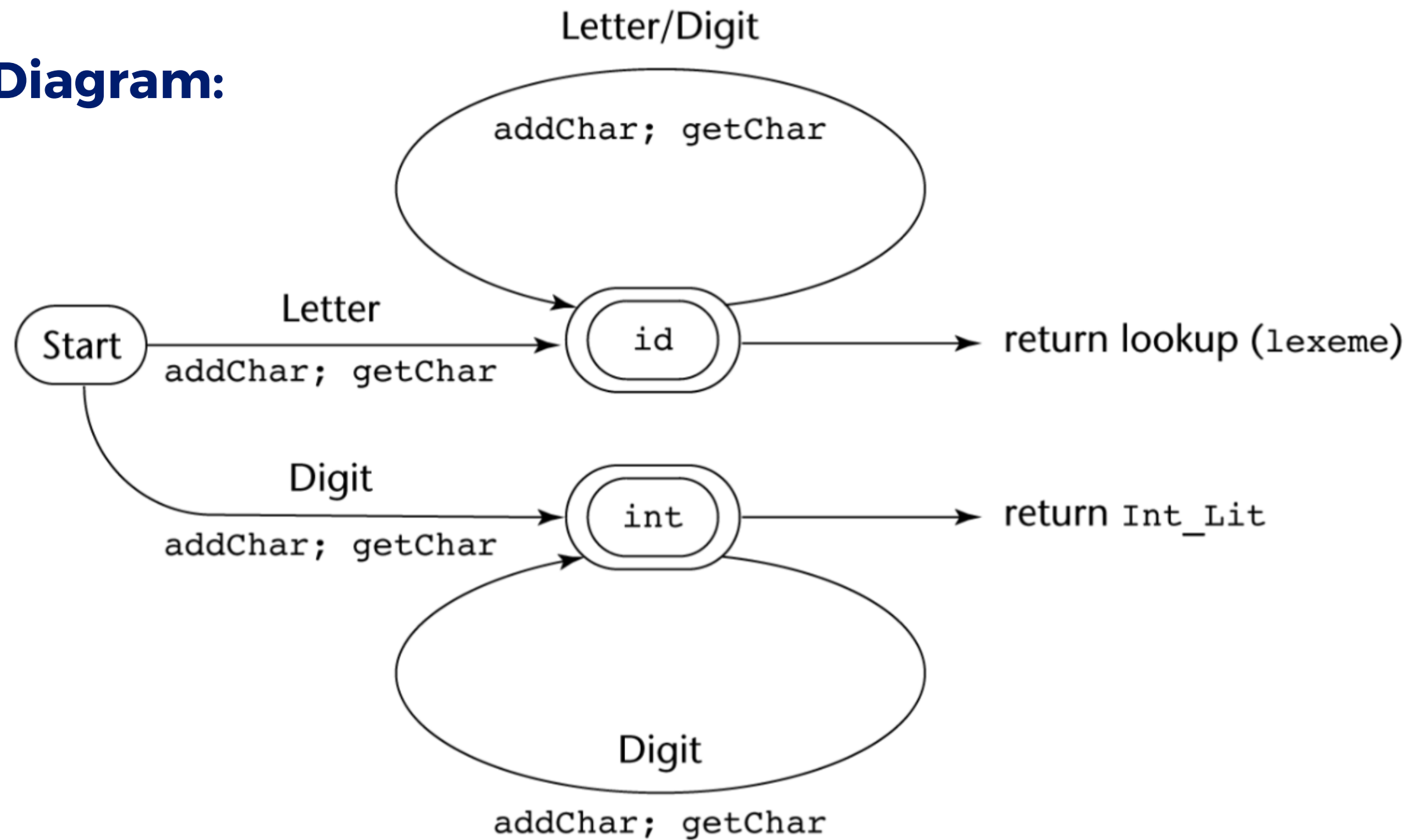
LEXICAL ANALYSIS

Convenient utility subprograms:

- **getChar** - gets the next character of input, puts it in
- **nextChar**, determines its class and puts the class in charClass
- **addChar** - puts the character from nextChar into the place the lexeme is being accumulated, lexeme
- **lookup** - determines whether the string in lexeme is a reserved word (returns a code)

LEXICAL ANALYSIS

State Diagram:





LEXICAL ANALYSIS

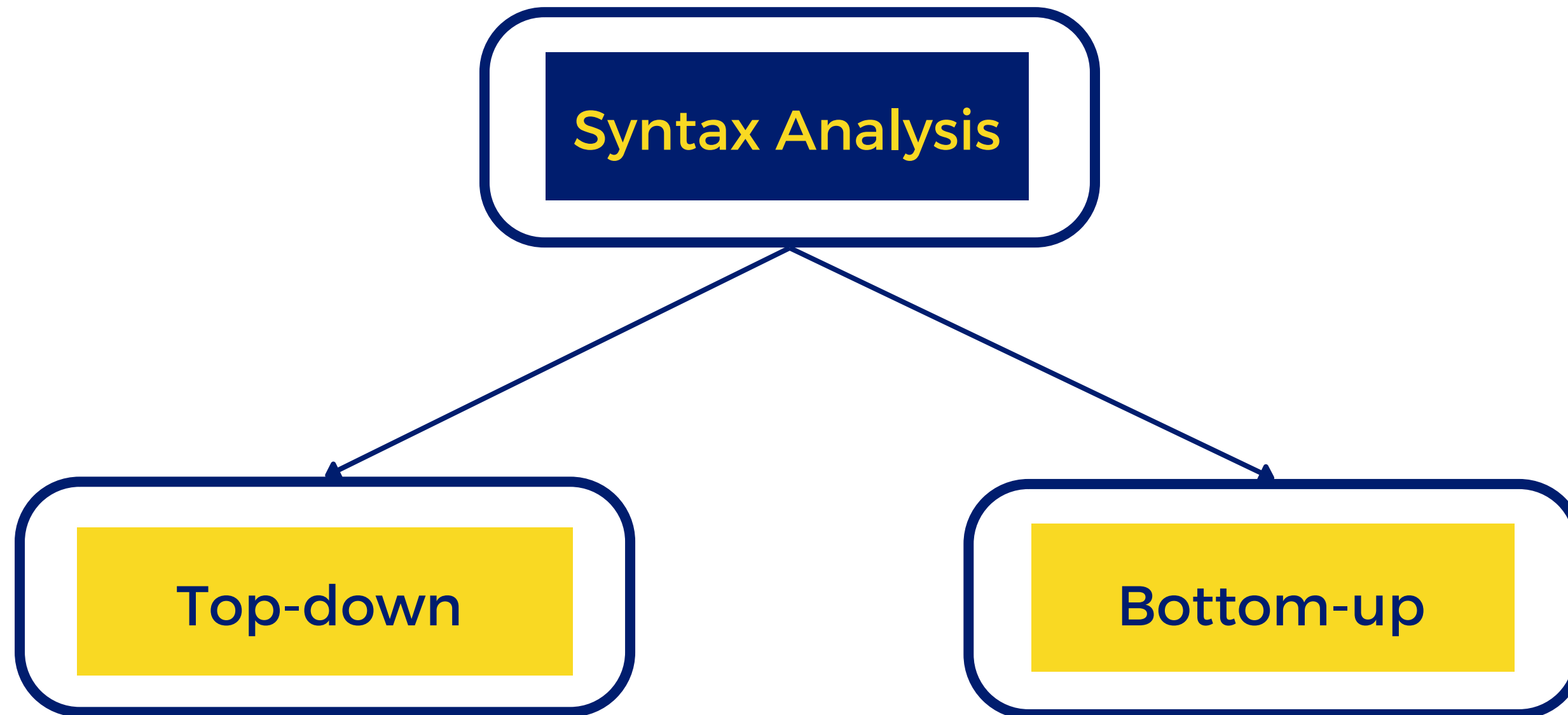
Implementation:

SHOW front.c (pp. 176-181)

Following is the output of the lexical analyzer offront.c when used on **(sum + 47) / total**

Next token is: 25 Next lexeme is (
Next token is: 11 Next lexeme is sum
Next token is: 21 Next lexeme is +
Next token is: 10 Next lexeme is 47
Next token is: 26 Next lexeme is)
Next token is: 24 Next lexeme is /
Next token is: 11 Next lexeme is total
Next token is: -1 Next lexeme is EOF

THE PARSING PROBLEM





THE PARSING PROBLEM



Two distinct goals of syntax analysis:

1. Check for syntax errors and produce a diagnostic message and recover
2. Produce a complete parse tree, or at least trace the structure of the complete parse tree

THE PARSING PROBLEM

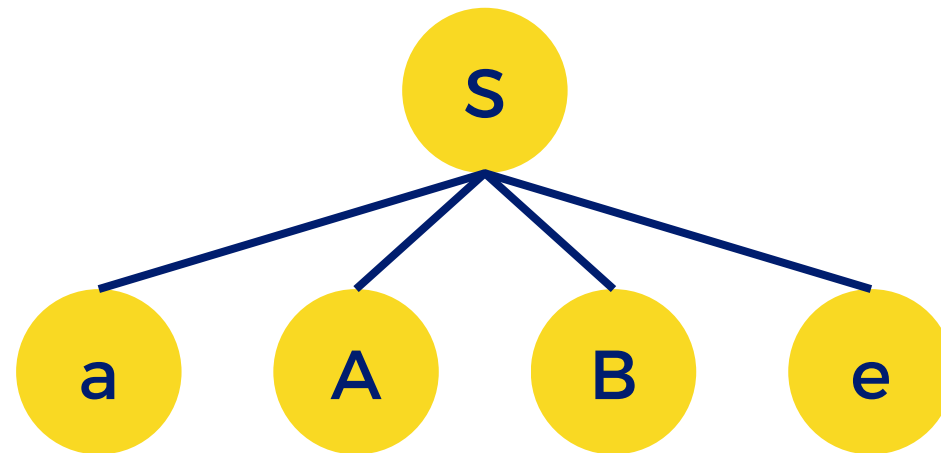
S

Top-down Approach

$S \rightarrow aABe, A \rightarrow Abc \mid a, B \rightarrow d$

aabcde

THE PARSING PROBLEM

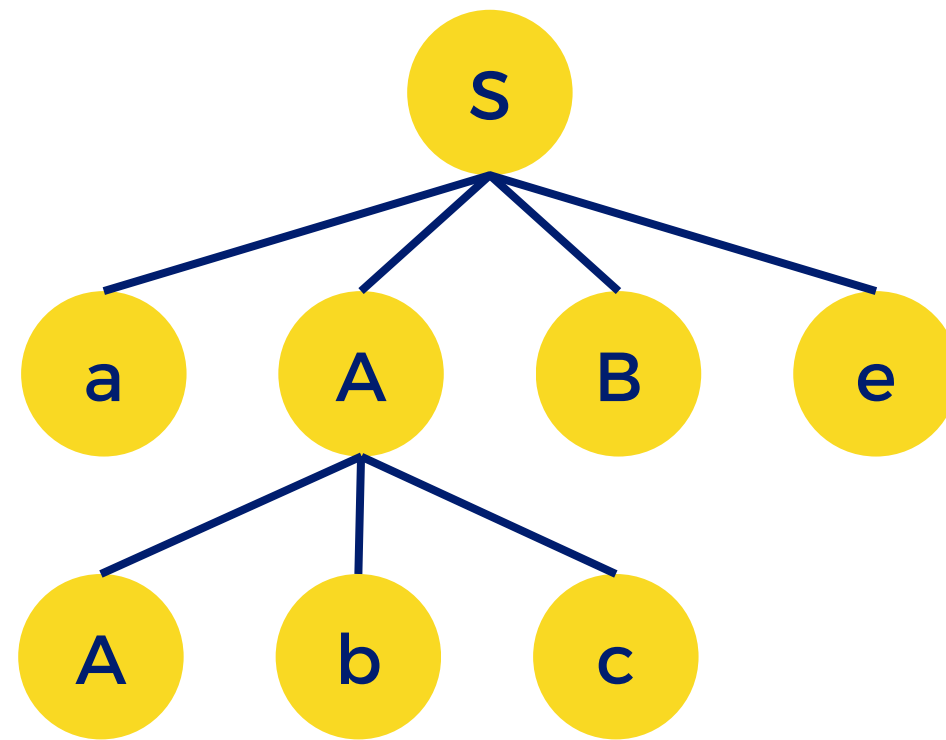


Top-down Approach

$S \rightarrow aABe, A \rightarrow Abc \mid a, B \rightarrow d$

aabcde

THE PARSING PROBLEM

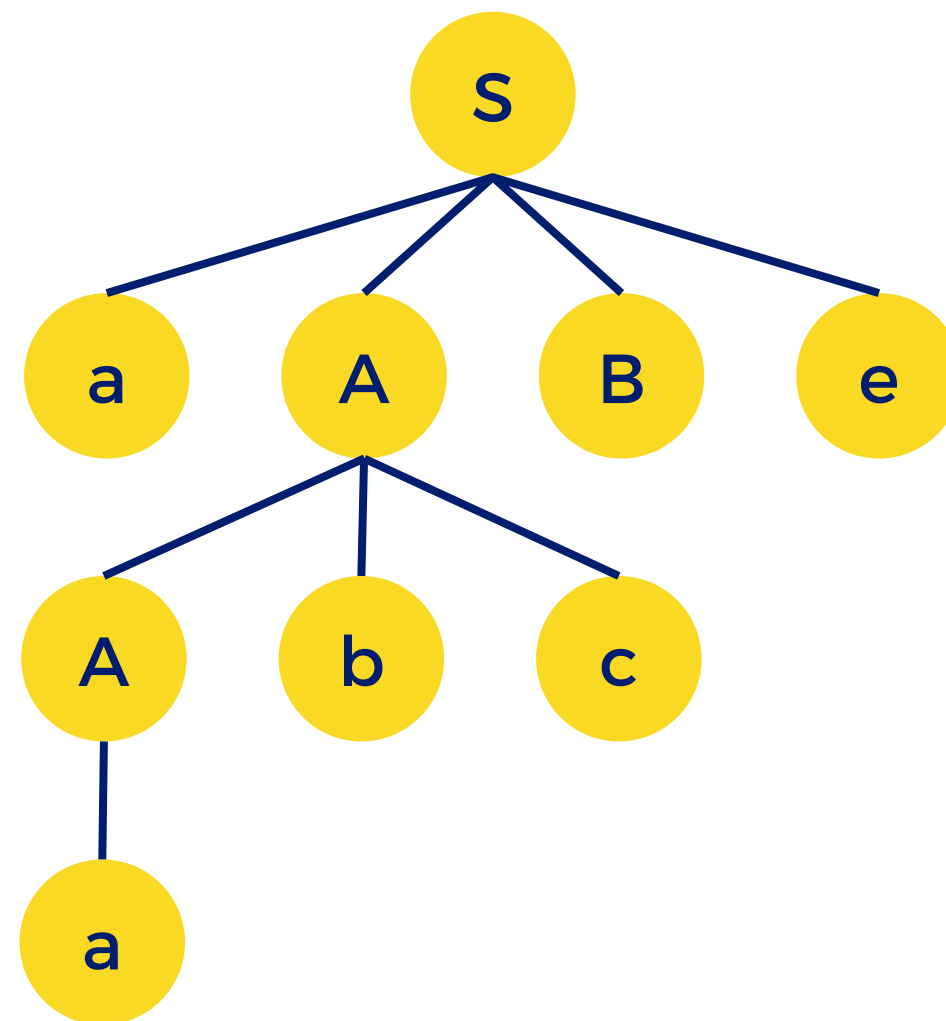


Top-down Approach

$S \rightarrow aABe, A \rightarrow Abc \mid a, B \rightarrow d$

aabcde

THE PARSING PROBLEM

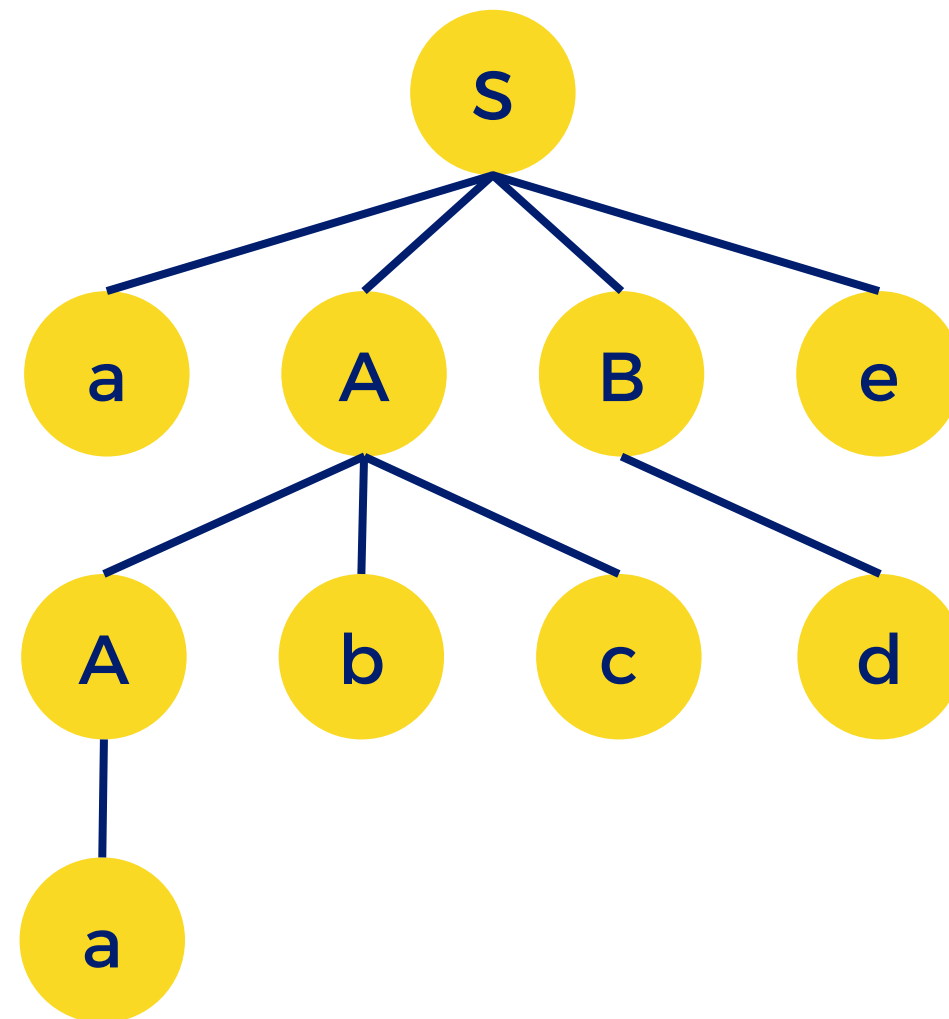


Top-down Approach

$S \rightarrow aABe, A \rightarrow Abc \mid \boxed{a}, B \rightarrow d$

aabcde

THE PARSING PROBLEM

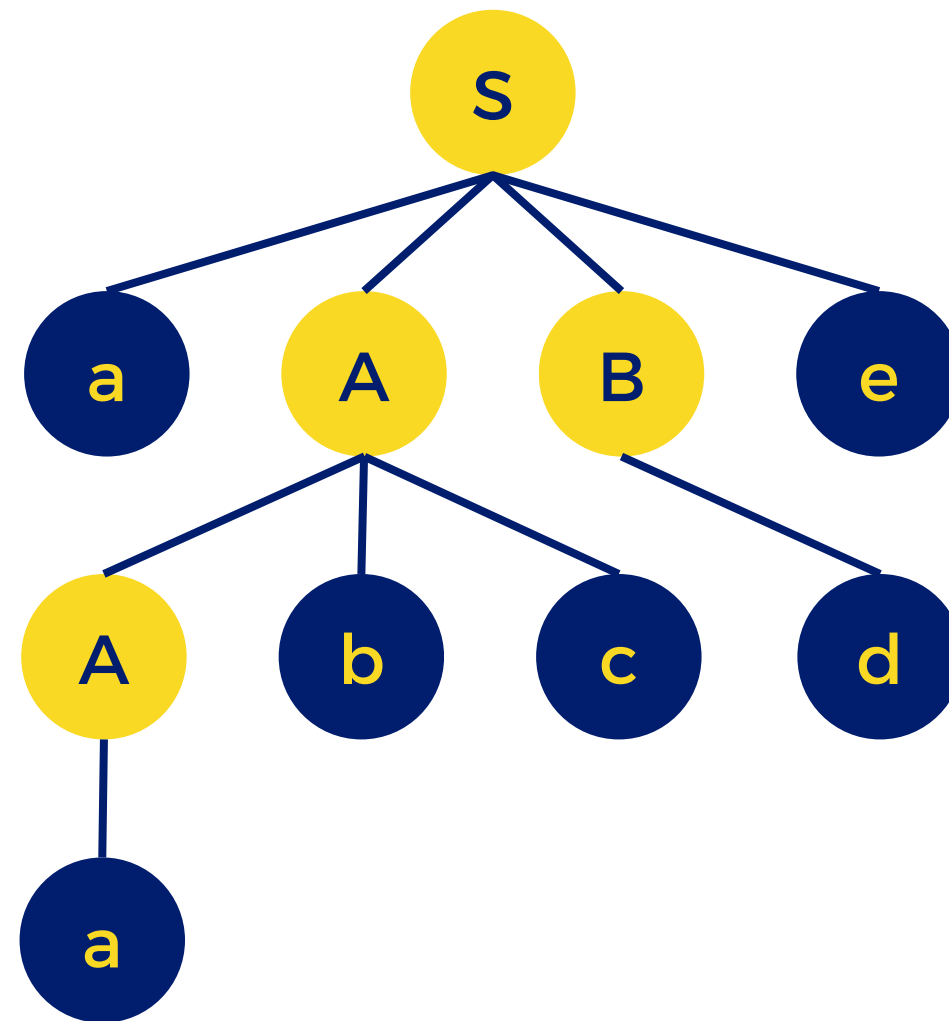


Top-down Approach

$S \rightarrow aABe$, $A \rightarrow Abc \mid a$, $B \rightarrow d$

aabcde

THE PARSING PROBLEM



Top-down Approach

$S \rightarrow aABe, A \rightarrow Abc \mid a, B \rightarrow d$

aabcde

THE PARSING PROBLEM

Bottom-up Approach

$S \rightarrow aABe, A \rightarrow Abc \mid a, B \rightarrow d$

aabcde

a

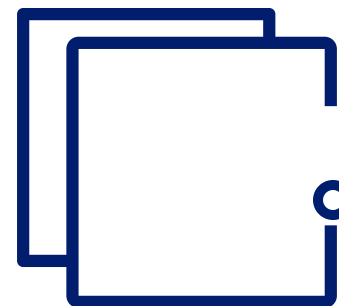
a

b

c

d

e

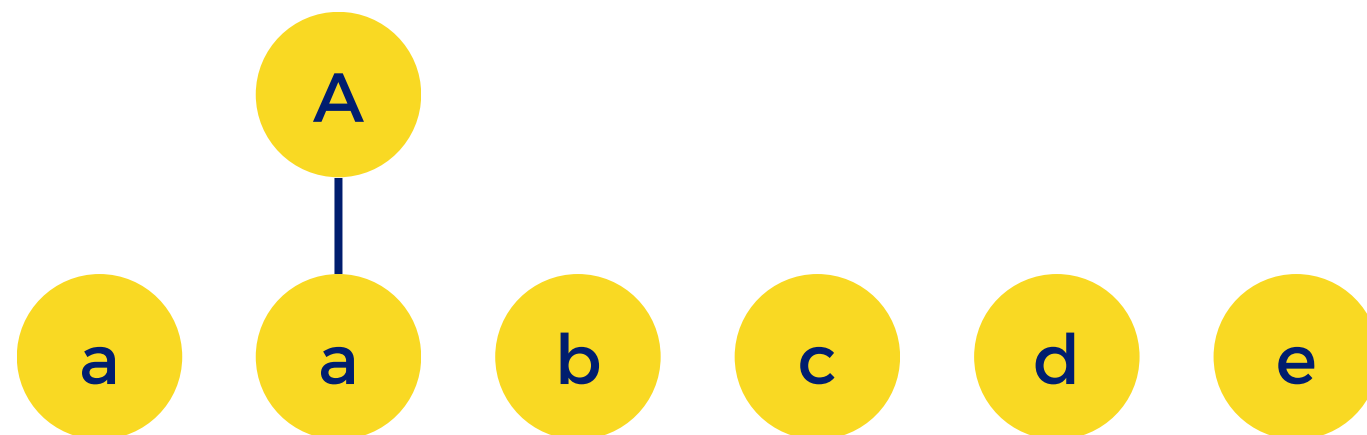


THE PARSING PROBLEM

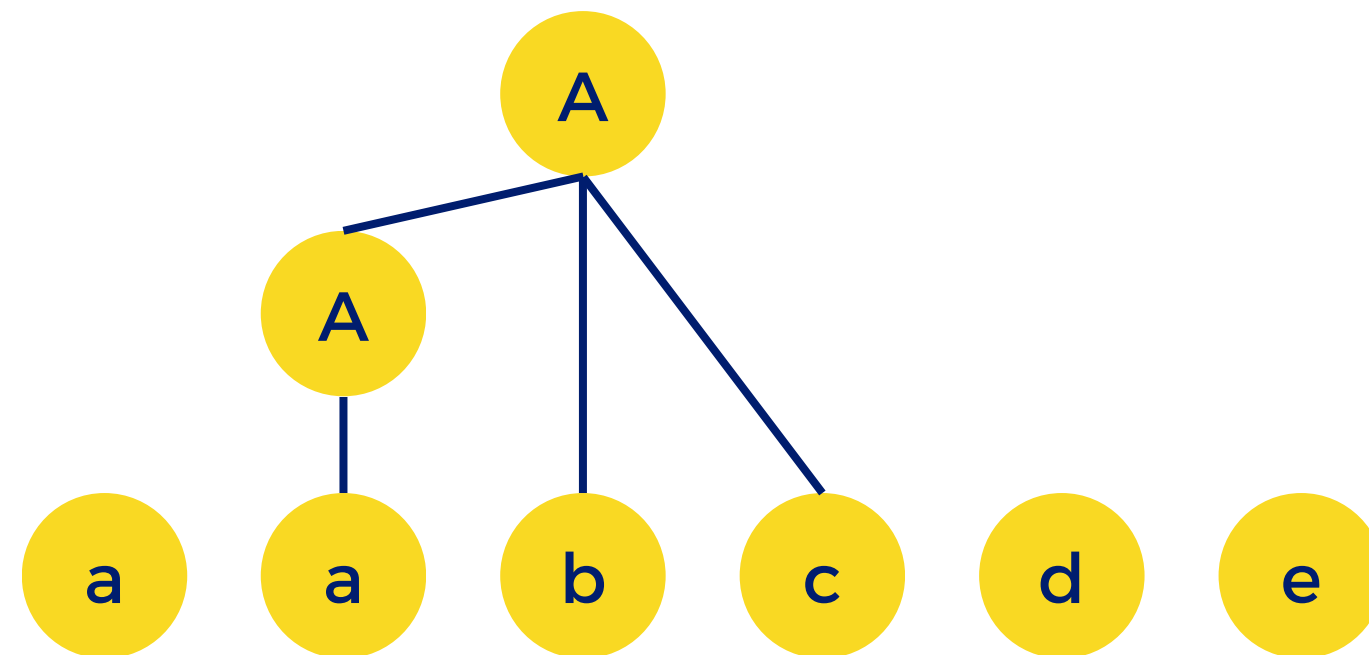
Bottom-up Approach

$S \rightarrow aABe, A \rightarrow Abc \parallel a, B \rightarrow d$

aabcde



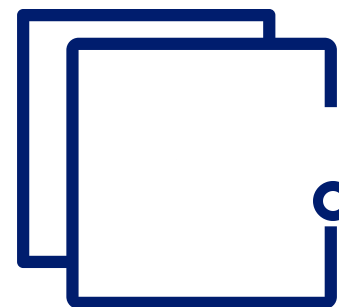
THE PARSING PROBLEM



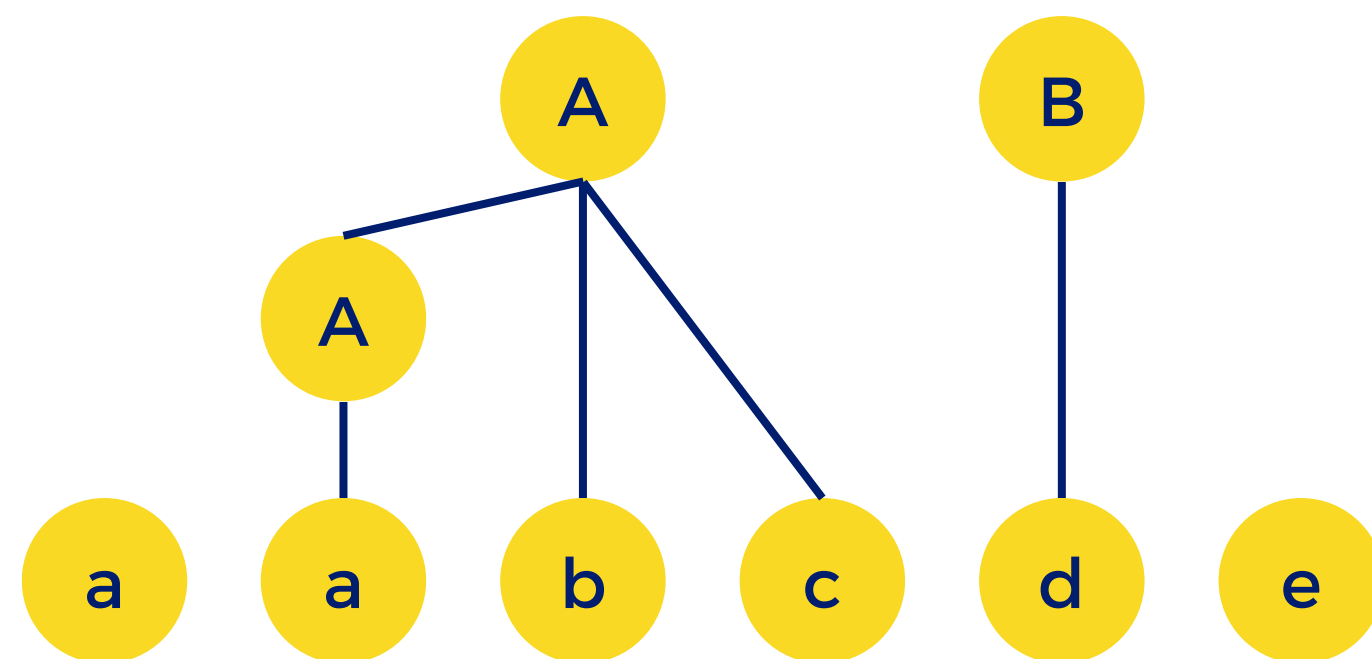
Bottom-up Approach

$S \rightarrow aABe$, $A \rightarrow Abc$, a , $B \rightarrow d$

aabcde



THE PARSING PROBLEM

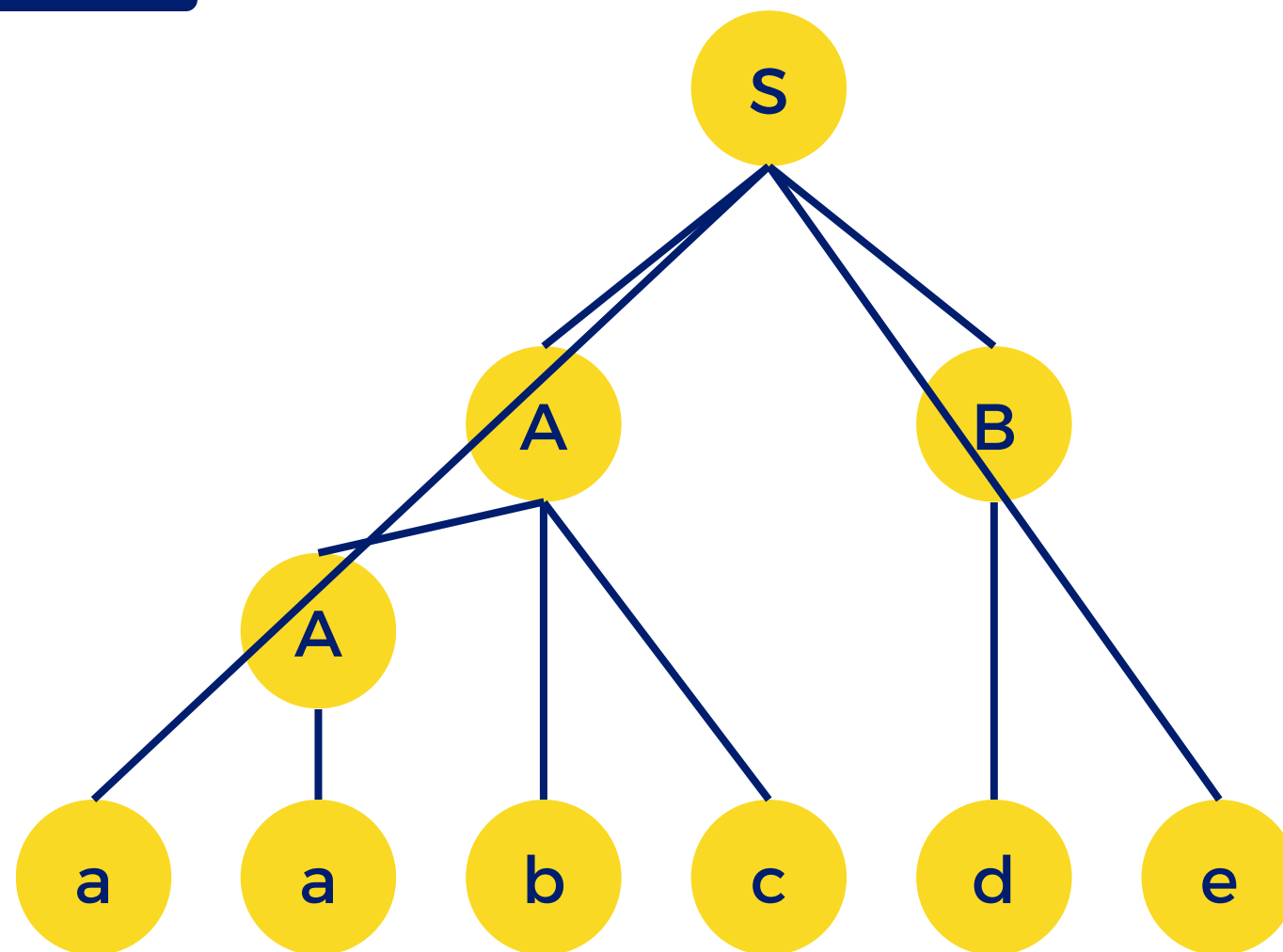


Bottom-up Approach

$S \rightarrow aABe, A \rightarrow Abc \mid a, B \rightarrow d$

aabcde

THE PARSING PROBLEM

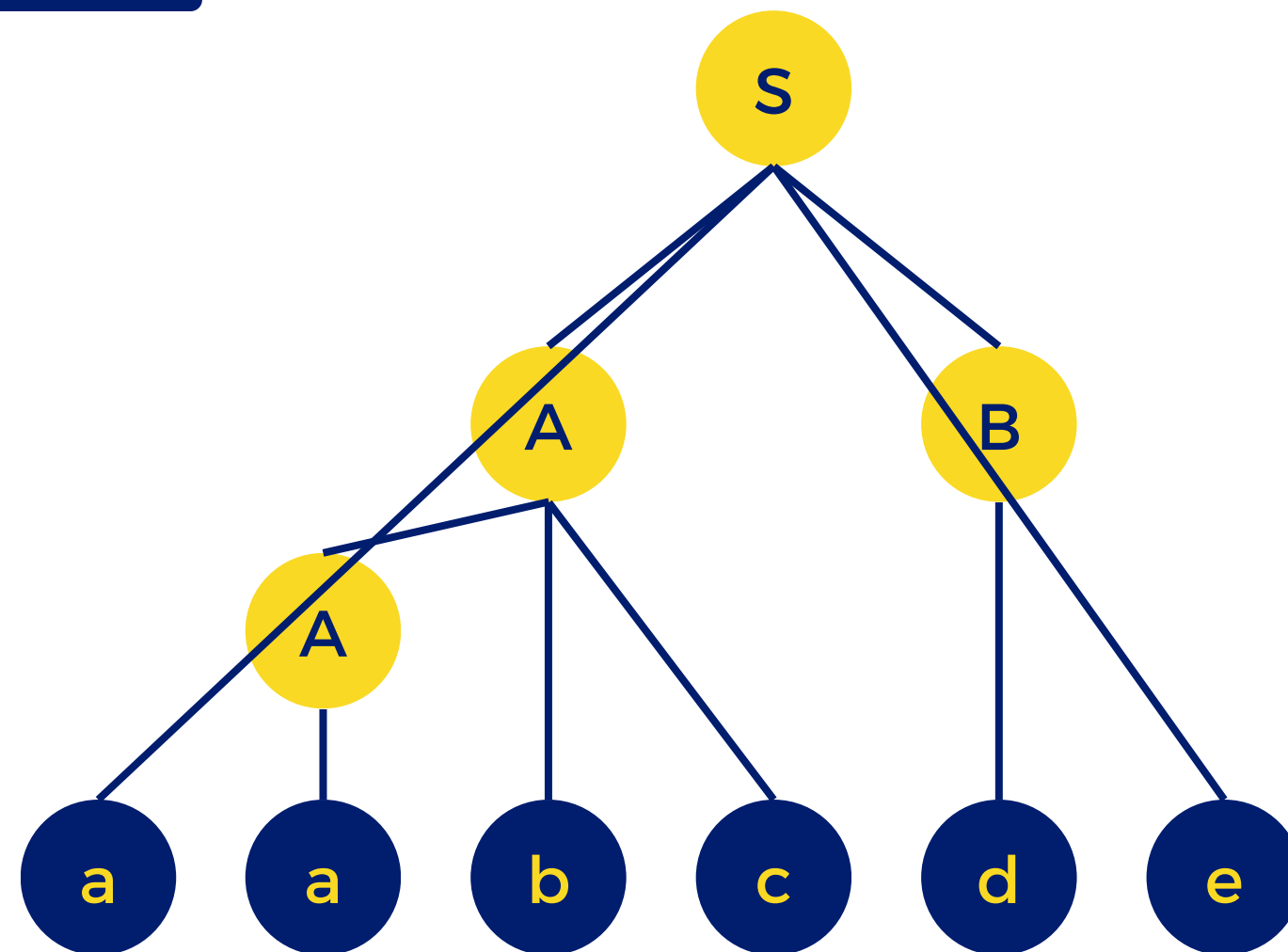


Bottom-up Approach

$S \rightarrow aABe, A \rightarrow Abc \mid a, B \rightarrow d$

aabcde

THE PARSING PROBLEM



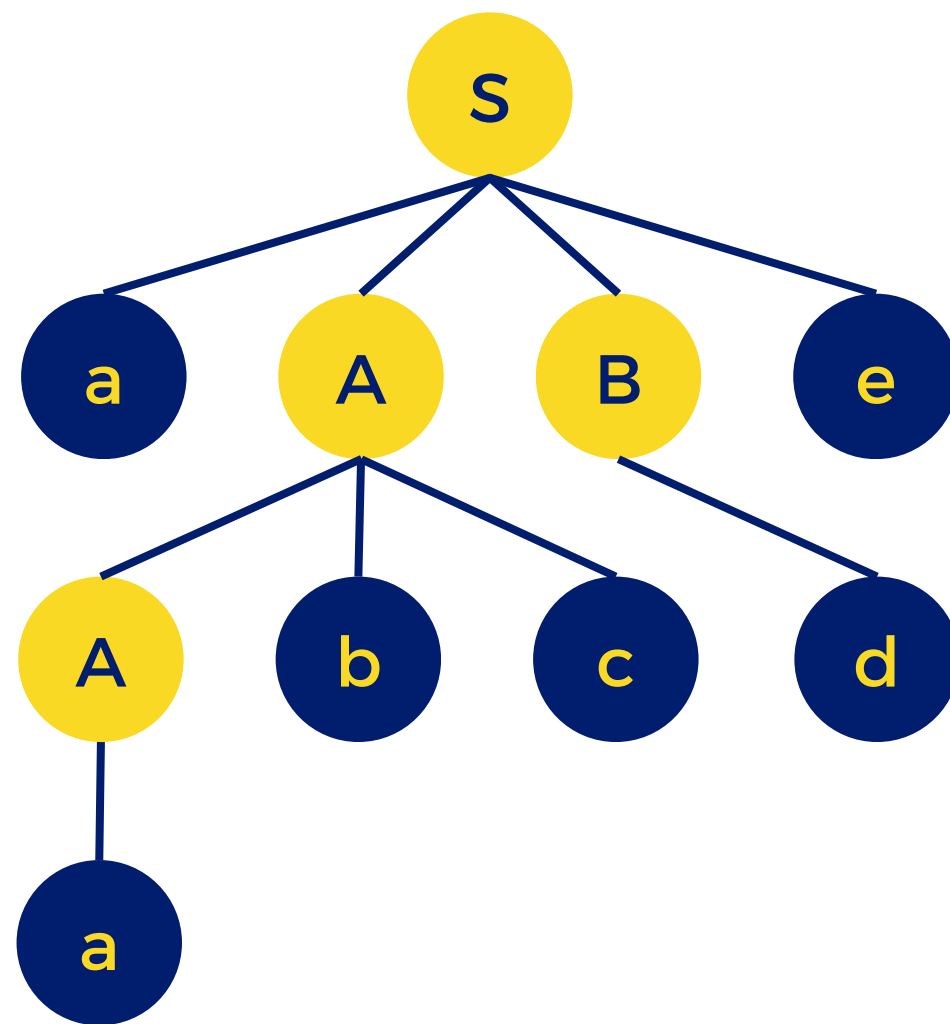
Bottom-up Approach

$S \rightarrow aABe, A \rightarrow Abc \mid a, B \rightarrow d$

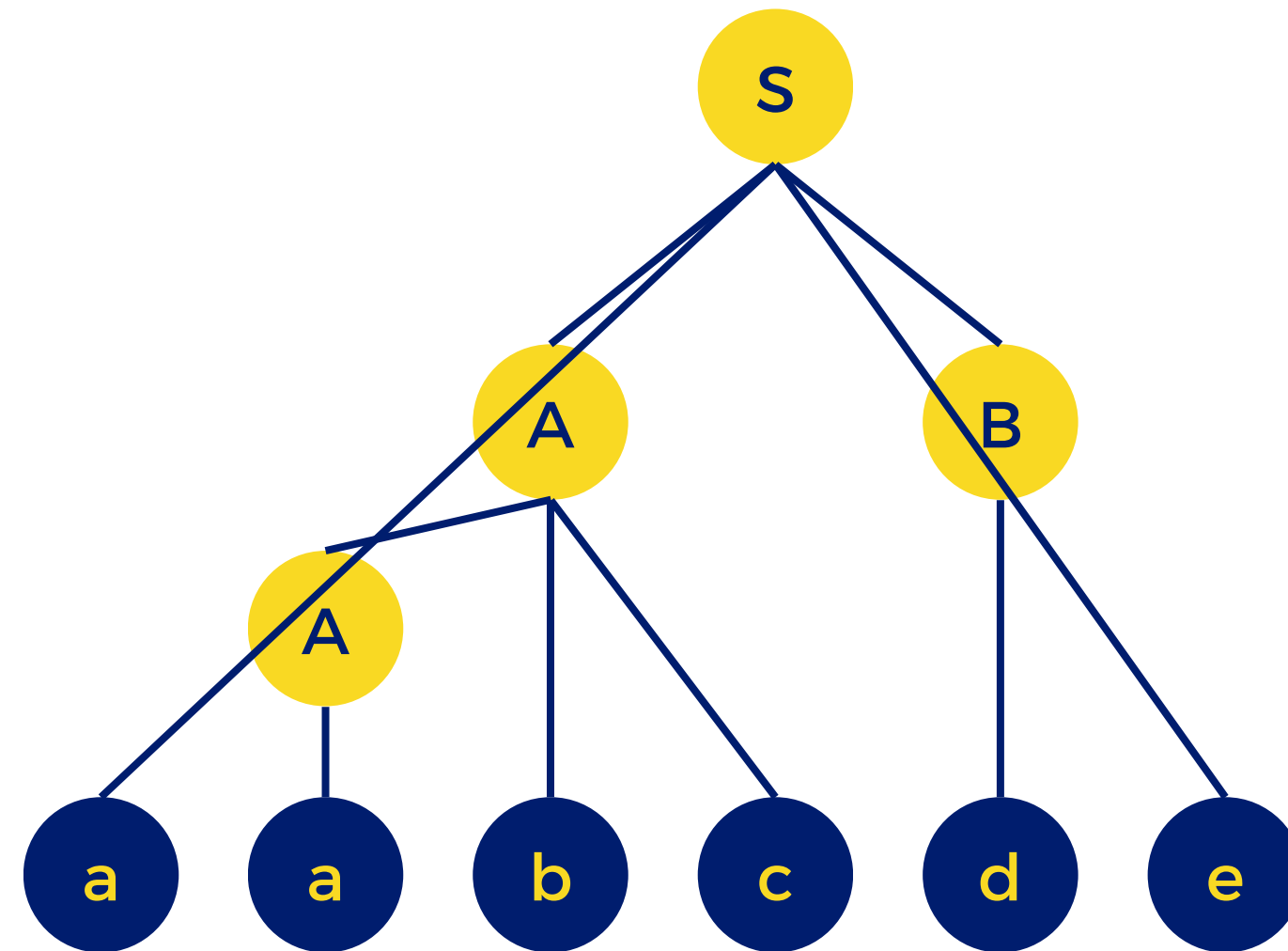
aabcde

THE PARSING PROBLEM

Top-down Approach



Bottom-up Approach





THE PARSING PROBLEM



Time Complexity

Unambiguous grammar = $O(n^3)$

Commercial Compilers = $O(n)$



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing (Top-Down)

There is a **subprogram for each nonterminal** in the **grammar**, which can parse sentences that can be generated by that nonterminal value.

EBNF is ideally suited for being the basis for a recursive-descent parser, because **EBNF minimizes the number of nonterminals**

***EBNF - Extended Backus-Naur form**



RECURSIVE-DESCENT PARSING

- Assume we have a lexical analyzer named `lex`, which puts the next token code in `nextToken`
- The coding process when there is only one right-hand side (RHS):
 - For each terminal symbol in the RHS, compare it with the next input token; if they match, continue, else there is an error
 - For each nonterminal symbol in the RHS, call its associated parsing subprogram



RECURSIVE-DESCENT PARSING

A nonterminal that has more than one RHS requires an initial process to determine which RHS it is to parse

- The correct RHS is chosen on the basis of the next token of input (the lookahead)
- The next token is compared with the first token that can be generated by each RHS until a match is found
- If no match is found, it is a syntax error



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

Given grammar:

$A \rightarrow abC \mid aBd \mid aAD$

$B \rightarrow bB \mid \epsilon$

$C \rightarrow d \mid \epsilon$

$D \rightarrow a \mid b \mid \epsilon$

*Capital characters - non-terminal

Input: aaba

RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a a b a



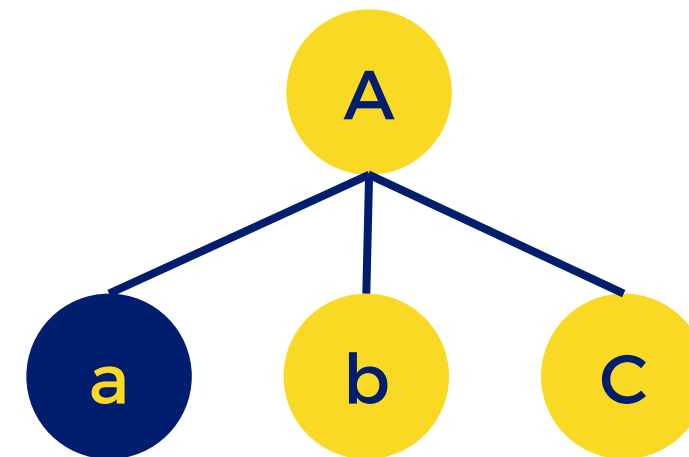
Given grammar:

A \rightarrow **a****b****C** | **a****B****d** | **a****A****D**

B \rightarrow **b****B** | ϵ

C \rightarrow **d** | ϵ

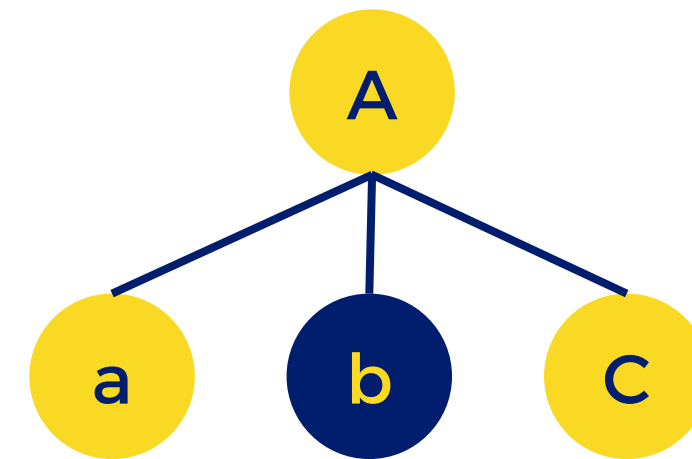
D \rightarrow **a** | **b** | ϵ



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a **a** b a
↑



Given grammar:

$A \rightarrow abC \mid aBd \mid aAD$

$B \rightarrow bB \mid \epsilon$

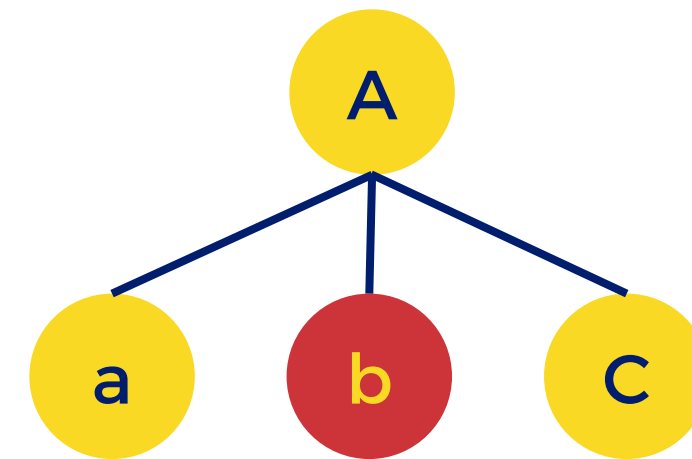
$C \rightarrow d \mid \epsilon$

$D \rightarrow a \mid b \mid \epsilon$

RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a **a** b a
↑



Given grammar:

$A \rightarrow \mathbf{abC} \mid \mathbf{aBd} \mid \mathbf{aAD}$

$B \rightarrow \mathbf{bB} \mid \epsilon$

$C \rightarrow \mathbf{d} \mid \epsilon$

$D \rightarrow \mathbf{a} \mid \mathbf{b} \mid \epsilon$

RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a a b a



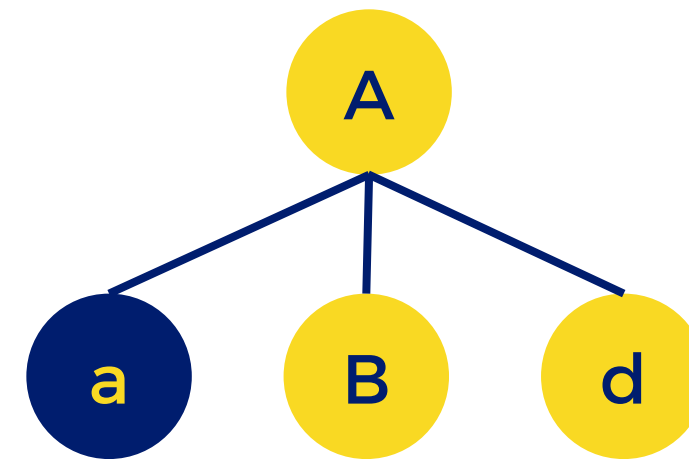
Given grammar:

A \rightarrow abC | **aBd** | aAD

B \rightarrow bB | ϵ

C \rightarrow d | ϵ

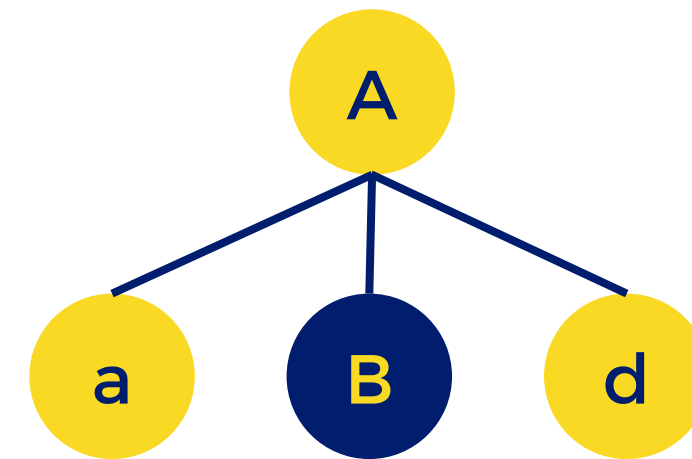
D \rightarrow a | b | ϵ



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a **a** b a
↑



Given grammar:

$A \rightarrow abC \mid \mathbf{aBd} \mid aAD$

$B \rightarrow bB \mid \epsilon$

$C \rightarrow d \mid \epsilon$

$D \rightarrow a \mid b \mid \epsilon$

RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a **a** b a



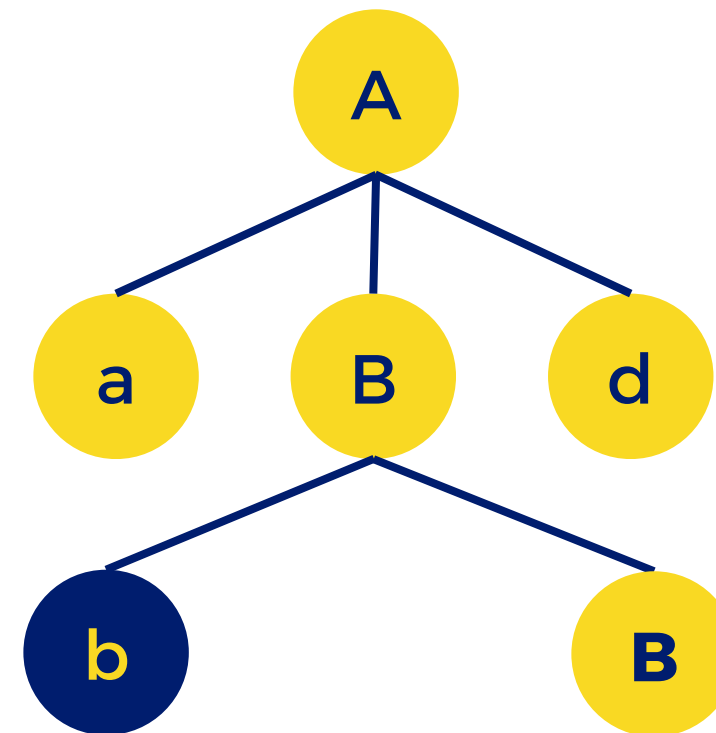
Given grammar:

$A \rightarrow abC \mid \mathbf{aBd} \mid aAD$

$B \rightarrow \mathbf{bB} \mid \epsilon$

$C \rightarrow d \mid \epsilon$

$D \rightarrow a \mid b \mid \epsilon$



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a **a** b a



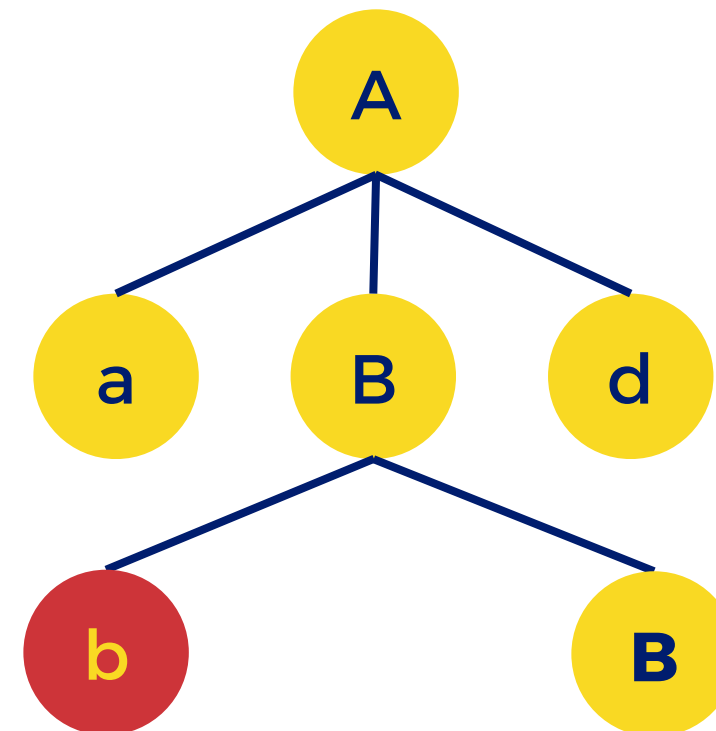
Given grammar:

$A \rightarrow abC \mid \mathbf{aBd} \mid aAD$

$B \rightarrow \mathbf{bB} \mid \epsilon$

$C \rightarrow d \mid \epsilon$

$D \rightarrow a \mid b \mid \epsilon$



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a **a** b a



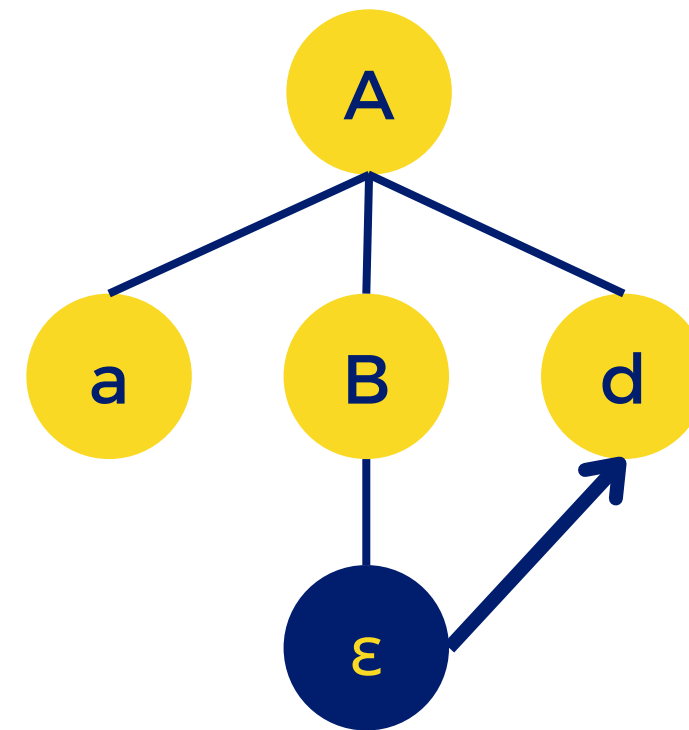
Given grammar:

$A \rightarrow abC \mid \mathbf{aBd} \mid aAD$

$B \rightarrow bB \mid \epsilon$

$C \rightarrow d \mid \epsilon$

$D \rightarrow a \mid b \mid \epsilon$



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a **a** b a
↑

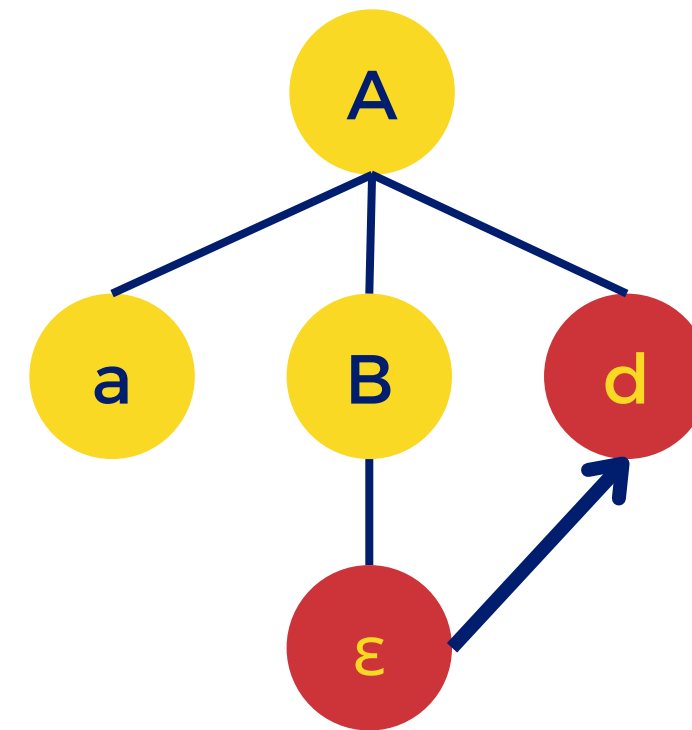
Given grammar:

$A \rightarrow abC \mid \mathbf{aBd} \mid aAD$

$B \rightarrow bB \mid \epsilon$

$C \rightarrow d \mid \epsilon$

$D \rightarrow a \mid b \mid \epsilon$



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a a b a



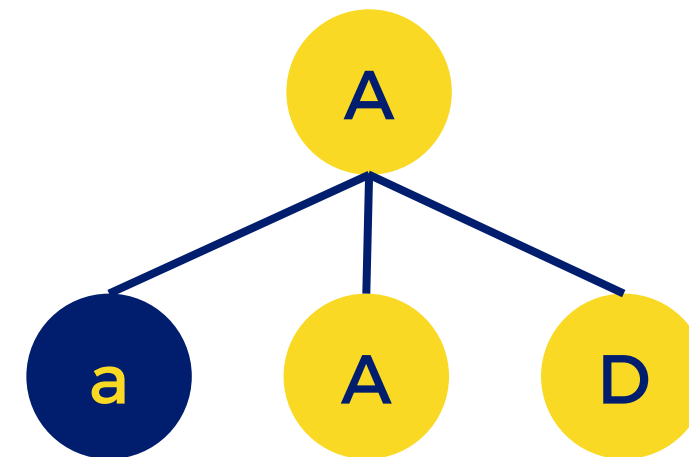
Given grammar:

A \rightarrow abC | aBd | **aAD**

B \rightarrow bB | ϵ

C \rightarrow d | ϵ

D \rightarrow a | b | ϵ



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a a b a



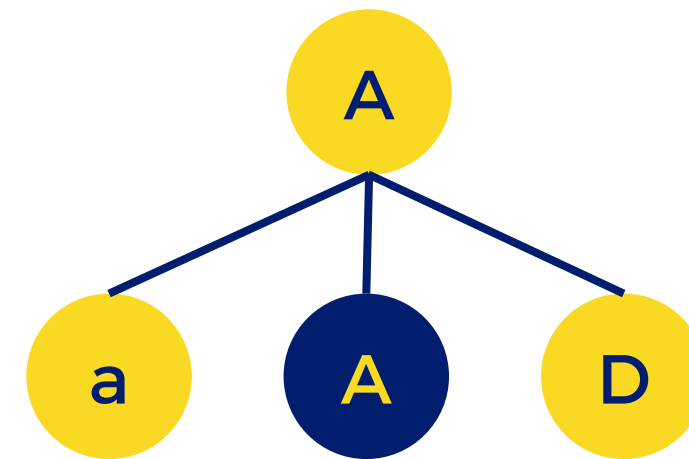
Given grammar:

$A \rightarrow abC \mid aBd \mid aAD$

$B \rightarrow bB \mid \epsilon$

$C \rightarrow d \mid \epsilon$

$D \rightarrow a \mid b \mid \epsilon$



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a a b a



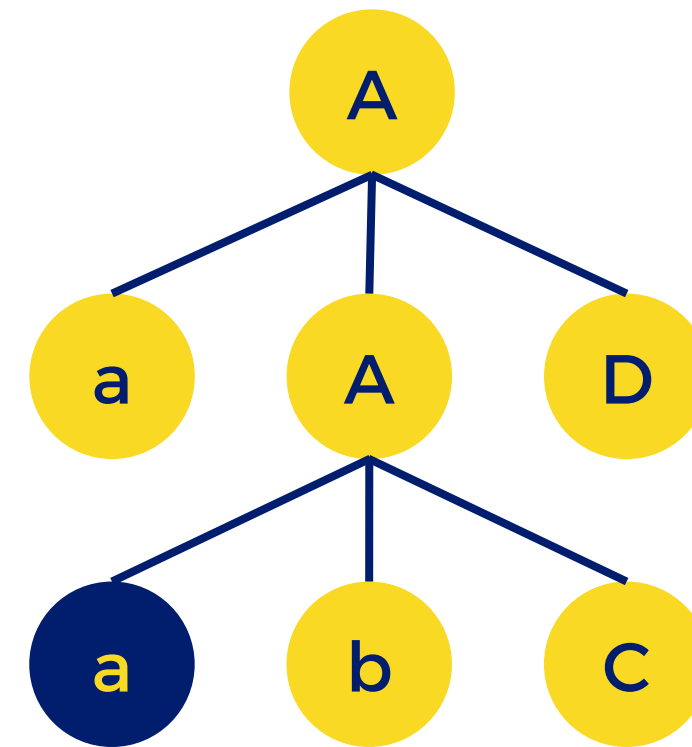
Given grammar:

$A \rightarrow abC \mid aBd \mid aAD$

$B \rightarrow bB \mid \epsilon$

$C \rightarrow d \mid \epsilon$

$D \rightarrow a \mid b \mid \epsilon$



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a a **b** a



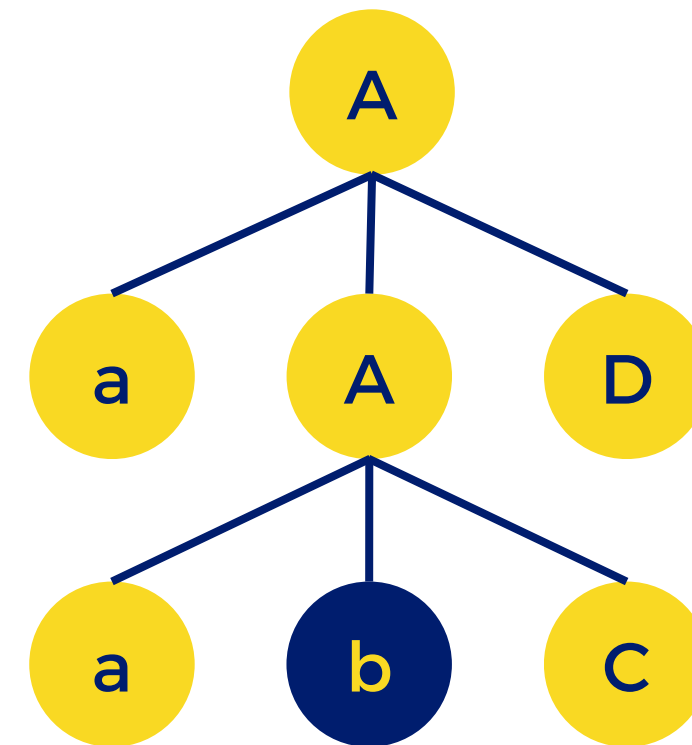
Given grammar:

A \rightarrow **a****b****C** | **a****B****d** | **a****A****D**

B \rightarrow **b****B** | ϵ

C \rightarrow **d** | ϵ

D \rightarrow **a** | **b** | ϵ



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a a b a



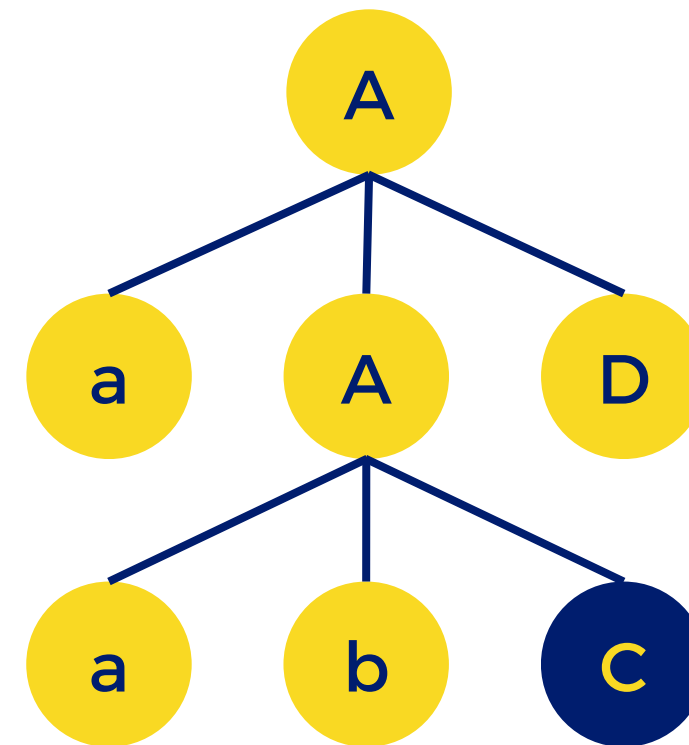
Given grammar:

$A \rightarrow abC \mid aBd \mid aAD$

$B \rightarrow bB \mid \epsilon$

$C \rightarrow d \mid \epsilon$

$D \rightarrow a \mid b \mid \epsilon$



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a a b a



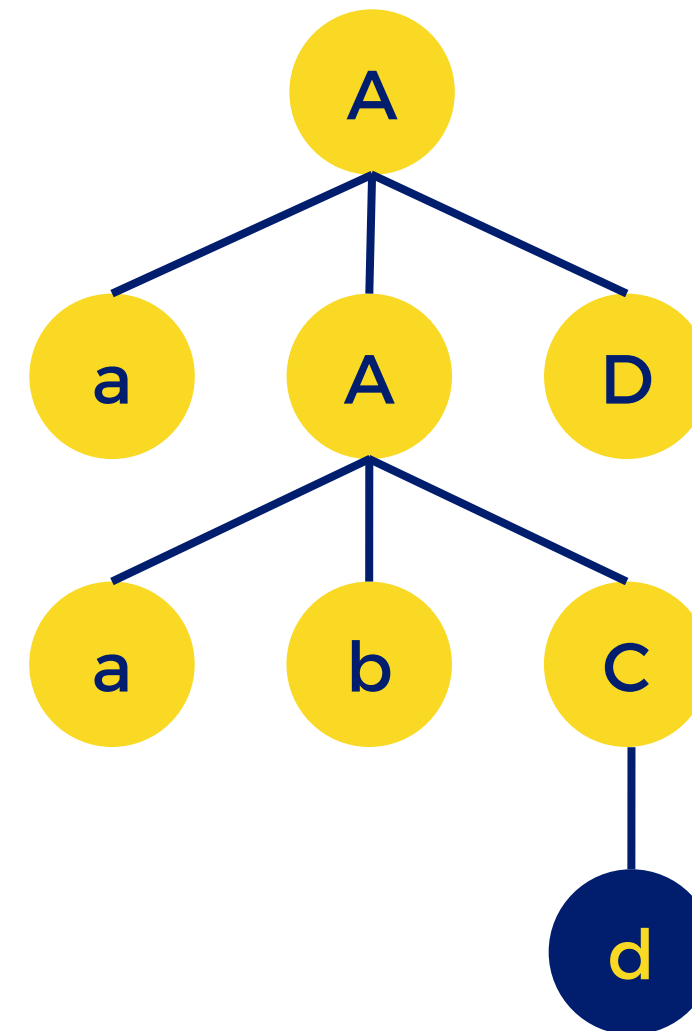
Given grammar:

$A \rightarrow abC \mid aBd \mid aAD$

$B \rightarrow bB \mid \epsilon$

$C \rightarrow d \mid \epsilon$

$D \rightarrow a \mid b \mid \epsilon$



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a a b **a**



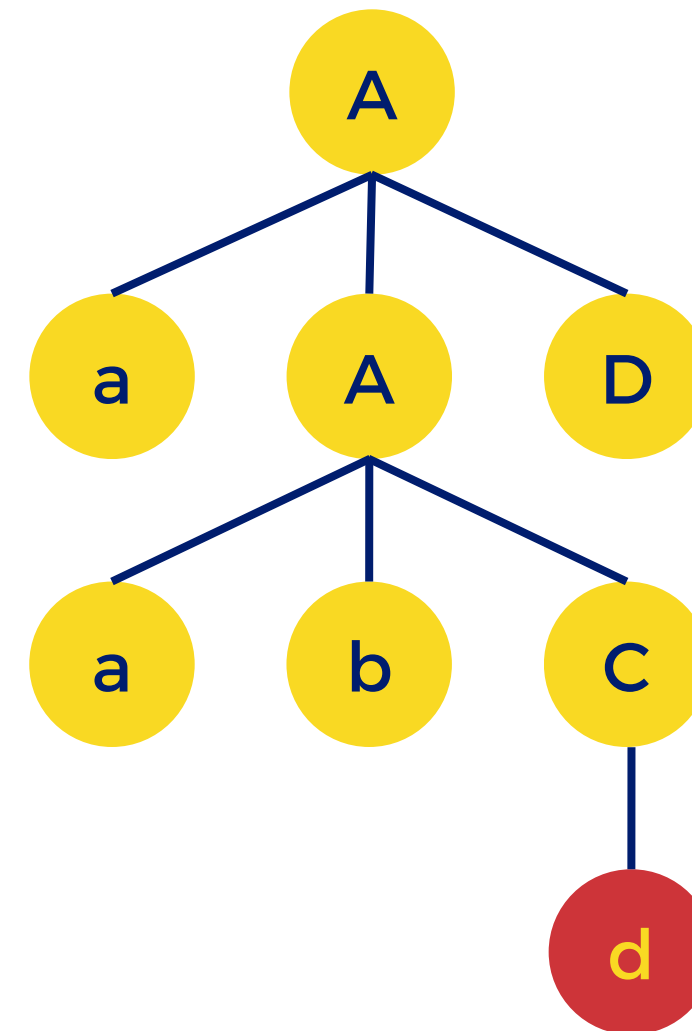
Given grammar:

A \rightarrow **a****b****C** | **a****B****d** | **a****A****D**

B \rightarrow **b****B** | ϵ

C \rightarrow **d** | ϵ

D \rightarrow **a** | **b** | ϵ



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a a b a



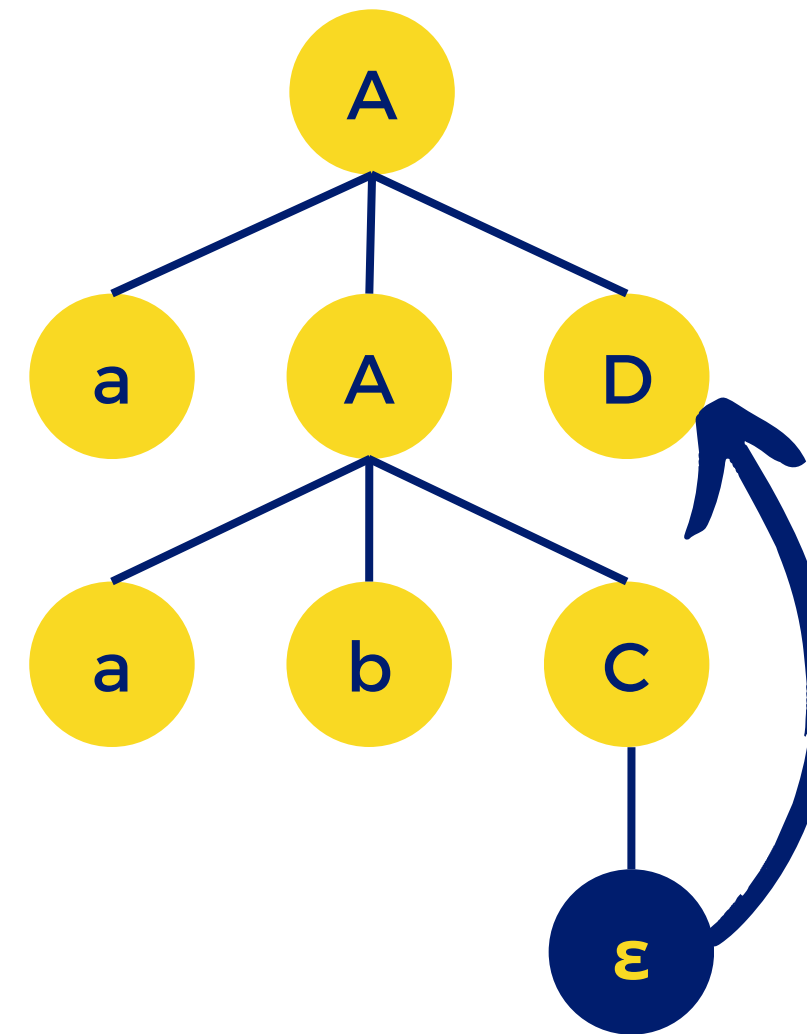
Given grammar:

A \rightarrow **a****b****C** | **a****B****d** | **a****A****D**

B \rightarrow **b****B** | ϵ

C \rightarrow **d** | ϵ

D \rightarrow **a** | **b** | ϵ



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a a b a



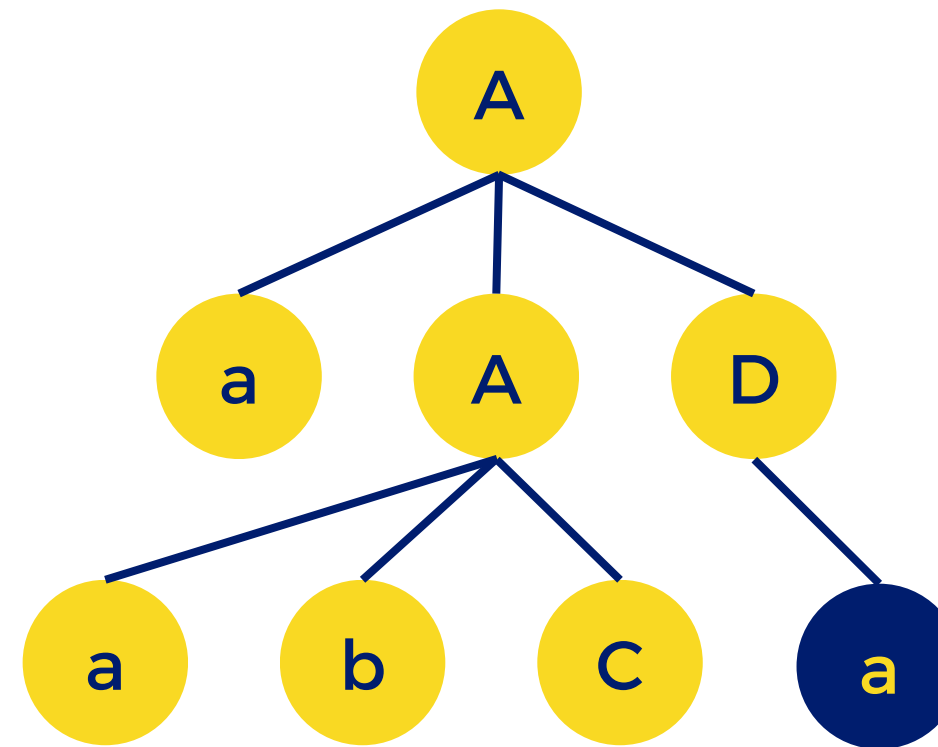
Given grammar:

A \rightarrow **a****b****C** | **a****B****d** | **a****A****D**

B \rightarrow **b****B** | ϵ

C \rightarrow **d** | ϵ

D \rightarrow **a** | **b** | ϵ



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a a b a

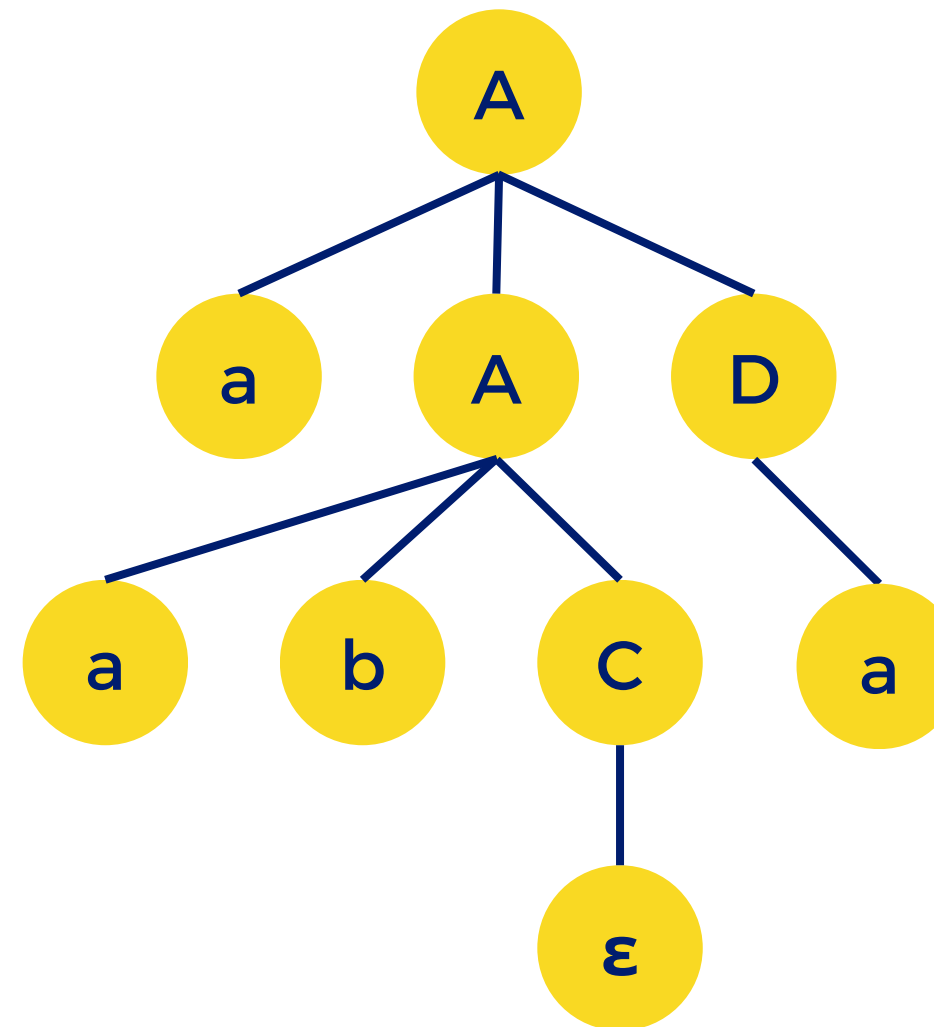
Given grammar:

$A \rightarrow abC \mid aBd \mid \mathbf{aAD}$

$B \rightarrow bB \mid \epsilon$

$C \rightarrow d \mid \epsilon$

$D \rightarrow a \mid b \mid \epsilon$



RECURSIVE-DESCENT PARSING

Recursive-Descent Parsing

a a b a

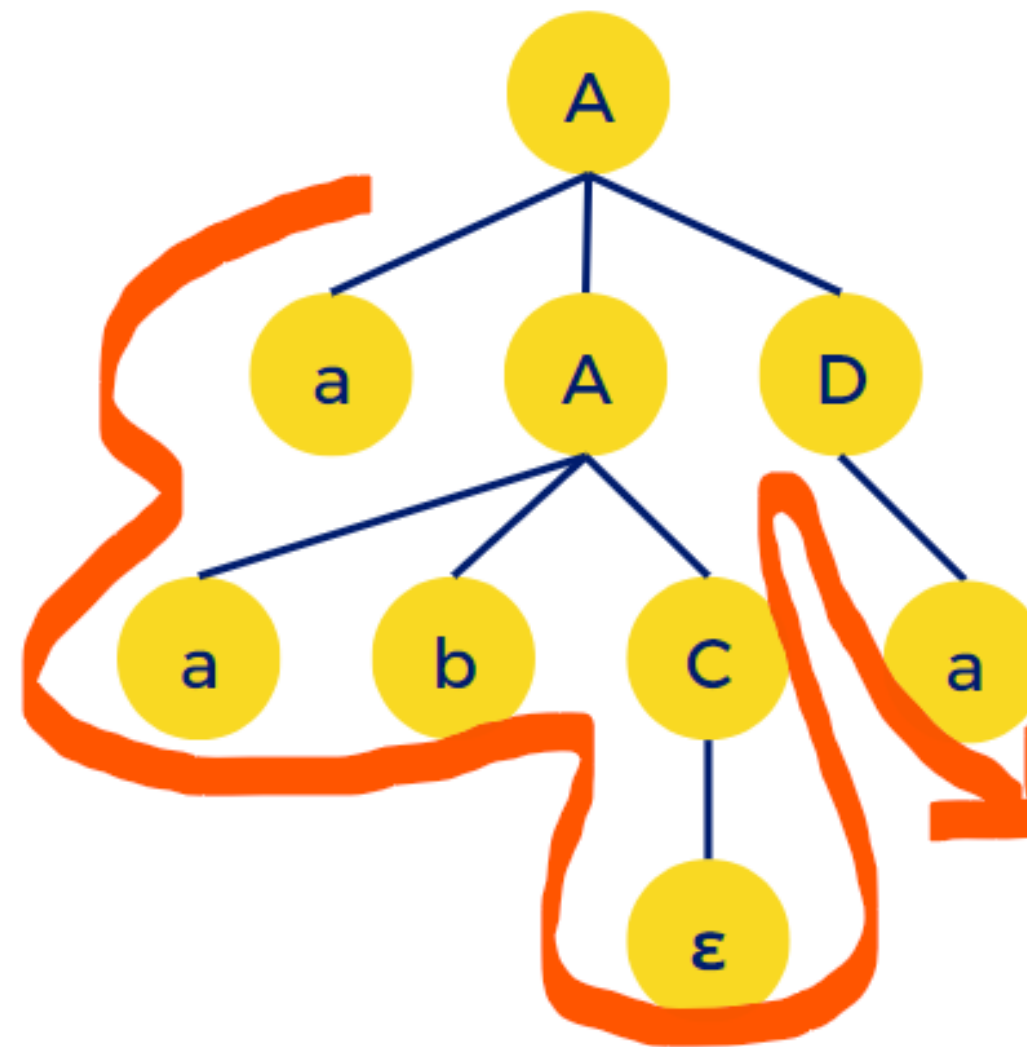
Given grammar:

$A \rightarrow abC \mid aBd \mid \mathbf{aAD}$

$B \rightarrow bB \mid \epsilon$

$C \rightarrow d \mid \epsilon$

$D \rightarrow a \mid b \mid \epsilon$





RECURSIVE-DESCENT PARSING

Problem with Left Recursion:

If a left recursion is present in any grammar then, during parsing in the syntax analysis part of compilation, there is a chance that the grammar will create an infinite loop. This is because, at every time of production of grammar, A will produce another A without checking any condition.

RECURSIVE-DESCENT PARSING

- The other characteristic of grammars that disallows top-down parsing is the lack of pairwise disjointness
 - The inability to determine the correct RHS on the basis of one token of lookahead
 - Def: $\text{FIRST}(\alpha) = \{a \mid \alpha \Rightarrow^* a\beta\}$
(If $\alpha \Rightarrow^* \epsilon$, ϵ is in $\text{FIRST}(\alpha)$)

RECURSIVE-DESCENT PARSING

- Pairwise Disjointness Test:
 - For each nonterminal, A , in the grammar that has more than one RHS, for each pair of rules, $A \rightarrow \alpha_i$ and $A \rightarrow \alpha_j$, it must be true that
$$\text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \varnothing$$

RECURSIVE-DESCENT PARSING

Example 1: Consider the following grammar

$A : a B$

$A : b A b$

$A : B b$

$B : c B$

$B : d$

The FIRST sets for the RHS of A-rules are: $\text{FIRST}(aB) = \{ a \}$, $\text{FIRST}(bAb) = \{ b \}$, and $\text{FIRST}(Bb) = \{ c, d \}$. These are disjoint and hence PASS the pairwise disjoint test.

The FIRST sets for the RHS of B-rules are: $\text{FIRST}(cB) = \{ c \}$ and $\text{FIRST}(d) = \{ d \}$. These are disjoint and hence PASS the pairwise disjoint test.

RECURSIVE-DESCENT PARSING

Example 2: Consider the following grammar

A : a B
A : B A b
B : a B
B : b

The FIRST sets for the RHS of A-rules are: $\text{FIRST}(aB) = \{ a \}$ and $\text{FIRST}(BA b) = \{ a, b \}$. These are not disjoint and hence FAIL the pairwise disjoint test.

The FIRST sets for the RHS of B-rules are: $\text{FIRST}(aB) = \{ a \}$ and $\text{FIRST}(b) = \{ b \}$. These are disjoint and hence PASS the pairwise disjoint test.

So, the grammar as a whole fails the pairwise disjoint test and hence cannot be parsed using top-down parsers!

RECURSIVE-DESCENT PARSING

- Left factoring can resolve the problem

Replace

$\langle \text{variable} \rangle \rightarrow \text{identifier} \mid \text{identifier} [\langle \text{expression} \rangle]$

with

$\langle \text{variable} \rangle \rightarrow \text{identifier} \langle \text{new} \rangle$

$\langle \text{new} \rangle \rightarrow \varepsilon \mid [\langle \text{expression} \rangle]$

or

$\langle \text{variable} \rangle \rightarrow \text{identifier} [[\langle \text{expression} \rangle]]$

(the outer brackets are metasympols of EBNF)

RECURSIVE-DESCENT PARSING

$A \rightarrow a\alpha1 / a\alpha2 / a\alpha3$

Grammar
with
common prefixes

Left Factoring

$A \rightarrow aA'$
 $A' \rightarrow \alpha1 / \alpha2 / \alpha3$

Left Factored Grammar

RECURSIVE-DESCENT PARSING

Problem-01:

Do left factoring in the following grammar-

$$S \rightarrow iEtS / iEtSeS / a$$

$$E \rightarrow b$$

RECURSIVE-DESCENT PARSING

Solution-

The left factored grammar is-

$$S \rightarrow iEtSS' / a$$

$$S' \rightarrow eS / \epsilon$$

$$E \rightarrow b$$



RECURSIVE-DESCENT PARSING



Problem-02:

Do left factoring in the following grammar-

$$A \rightarrow aAB / aBc / aAc$$

RECURSIVE-DESCENT PARSING

Step-01:

$$A \rightarrow aA'$$

$$A' \rightarrow AB / Bc / Ac$$

Again, this is a grammar with common prefixes.

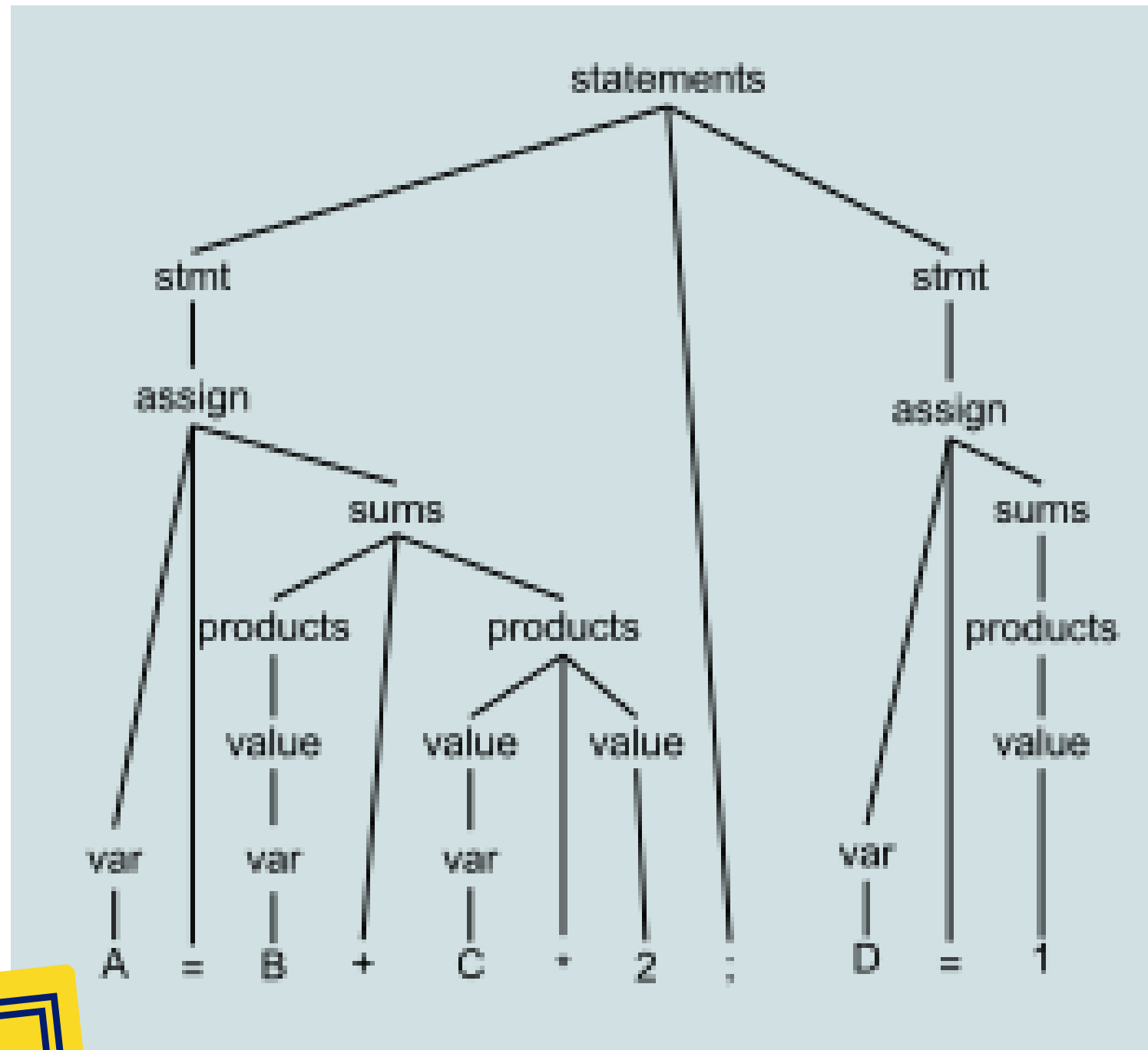
Step-02:

$$A \rightarrow aA'$$

$$A' \rightarrow AD / Bc$$

$$D \rightarrow B / c$$

BOTTOM UP PARSING



The parsing problem is finding the correct RHS in a right-sentential form to reduce to get the previous right-sentential form in the derivation.



BOTTOM UP PARSING



Right sentential form a sentential form that occurs in the rightmost derivation of some sentence.

The process of deriving a string by expanding the rightmost non-terminal at each step is called as **rightmost derivation**.

BOTTOM UP PARSING

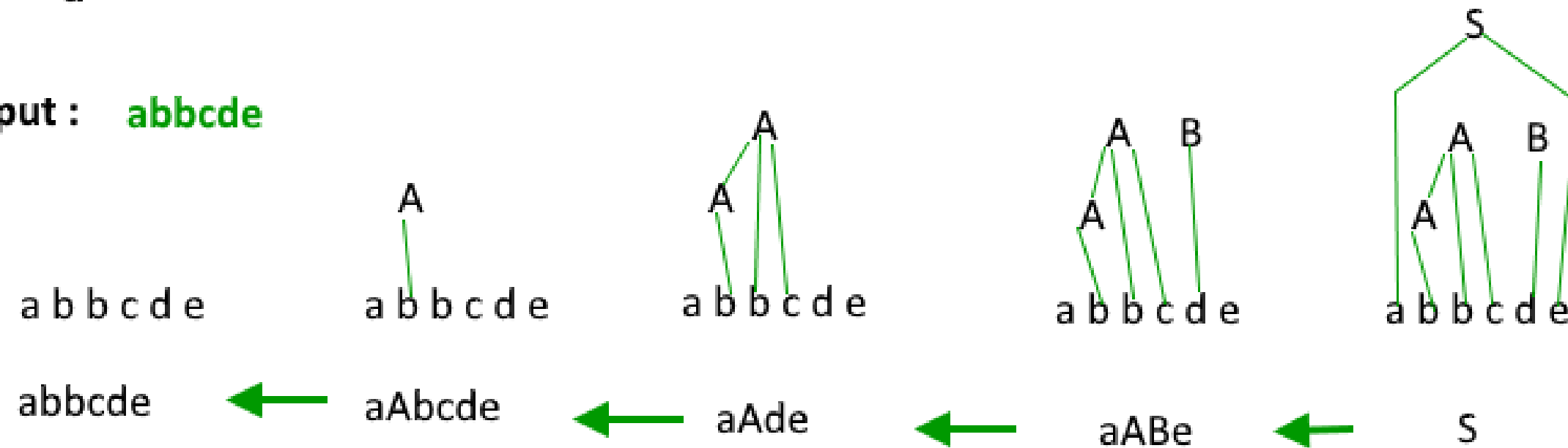
Handle - string of symbols to be replaced at each stage of parsing

$S \rightarrow aABe$

$A \rightarrow Abc/b$

$B \rightarrow d$

Input: **abbcde**



BOTTOM UP PARSING

- Intuition about handles:

- Def: β is the *handle* of the right sentential form $\gamma = \alpha\beta w$ if and only if $S \Rightarrow_{rm}^* \alpha A w \Rightarrow_{rm} \alpha\beta w$
- Def: β is a *phrase* of the right sentential form γ if and only if $S \Rightarrow^* \gamma = \alpha_1 A \alpha_2 \Rightarrow + \alpha_1 \beta \alpha_2$
- Def: β is a *simple phrase* of the right sentential form γ if and only if $S \Rightarrow^* \gamma = \alpha_1 A \alpha_2 \Rightarrow \alpha_1 \beta \alpha_2$



BOTTOM UP PARSING

- The handle of a right sentential form is its leftmost simple phrase
- Given a parse tree, it is now easy to find the handle
- Parsing can be thought of as handle pruning



BOTTOM UP PARSING

LR Parsing Algorithm

L - left to right scanning of input string

R - start with rightmost derivation



BOTTOM UP PARSING

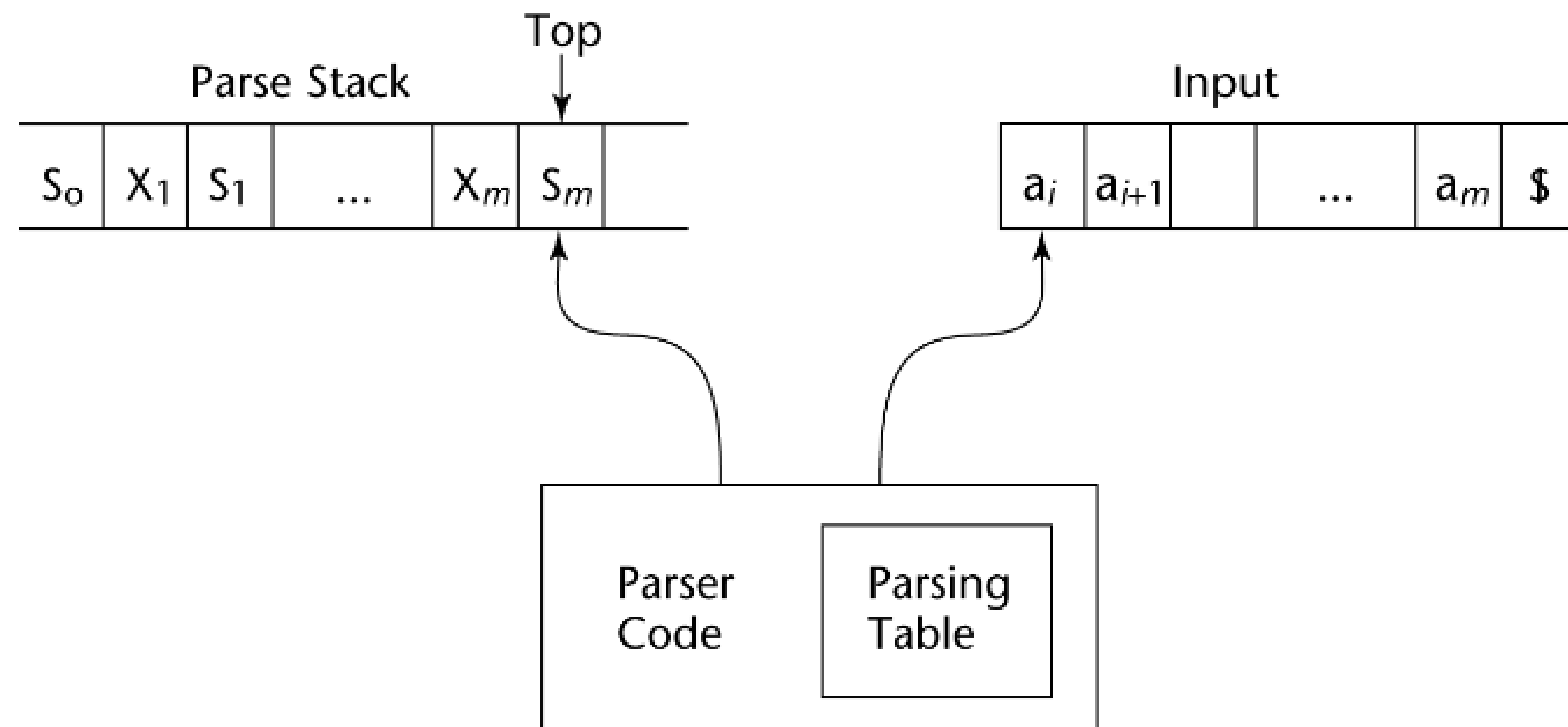
- Advantages of LR parsers:
 - They will work for nearly all grammars that describe programming languages.
 - They work on a larger class of grammars than other bottom-up algorithms, but are as efficient as any other bottom-up parser.
 - They can detect syntax errors as soon as it is possible.
 - The LR class of grammars is a superset of the class parsable by LL parsers.



BOTTOM UP PARSING

- LR parsers must be constructed with a tool
- Knuth's insight: A bottom-up parser could use the entire history of the parse, up to the current point, to make parsing decisions
 - There were only a finite and relatively small number of different parse situations that could have occurred, so the history could be stored in a parser state, on the parse stack

BOTTOM UP PARSING





BOTTOM UP PARSING

Bottom up parsers make use of Shift-Reduce Algorithms

- Shift-Reduce Algorithms
 - Reduce is the action of replacing the handle on the top of the parse stack with its corresponding LHS
 - Shift is the action of moving the next token to the top of the parse stack



BOTTOM UP PARSING

- LR parsers are table driven, where the table has two components, an ACTION table and a GOTO table
 - The ACTION table specifies the action of the parser, given the parser state and the next token
 - Rows are state names; columns are terminals
 - The GOTO table specifies which state to put on top of the parse stack after a reduction action is done
 - Rows are state names; columns are nonterminals

BOTTOM UP PARSING

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

BOTTOM UP PARSING

- Initial configuration: $(S_0, a_1 \dots a_n \$)$
- Parser actions:
 - If $\text{ACTION}[S_m, a_i] = \text{Shift } S$, the next configuration is:
 $(S_0 X_1 S_1 X_2 S_2 \dots X_m S_m a_i S, a_{i+1} \dots a_n \$)$
 - If $\text{ACTION}[S_m, a_i] = \text{Reduce } A \rightarrow \beta$ and $S = \text{GOTO}[S_{m-r}, A]$, where $r = \text{the length of } \beta$, the next configuration is
 $(S_0 X_1 S_1 X_2 S_2 \dots X_{m-r} S_{m-r} AS, a_i a_{i+1} \dots a_n \$)$
 - If $\text{ACTION}[S_m, a_i] = \text{Accept}$, the parse is complete and no errors were found.
 - If $\text{ACTION}[S_m, a_i] = \text{Error}$, the parser calls an error-handling routine.

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

id + id * id \$

0						lexeme
						state

shift

BOTTOM UP PARSING

Action:

Shift

$$\text{ACTION}[\emptyset, \text{id}] = S5$$

Push leftmost
element of input
string into the
parse stack

Push state
number from
ACTION into the
stack

BOTTOM UP PARSING

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0	id					lexeme
						state

BOTTOM UP PARSING

Action:

Shift

$$\text{ACTION}[\emptyset, id] = S5$$

Push leftmost
element of input
string into the
parse stack

Push state
number from
ACTION into the
stack

BOTTOM UP PARSING

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0	id					lexeme
	5					state

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0	id					lexeme
	5					state

BOTTOM UP PARSING

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0	id					lexeme
	5					state

reduce

BOTTOM UP PARSING

Action:

Reduce

$\text{ACTION}[5, +] = \text{R6}$

Pop the symbol/s on the topmost of the stack (with its corresponding states) that is in the RHS of the rule number indicated in ACTION

Push the RHS of the rule indicated in ACTION

Push the state number indicated in the GOTO

BOTTOM UP PARSING

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0	id					lexeme
	5					state

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0						lexeme
						state

BOTTOM UP PARSING

Action:

Reduce

$\text{ACTION}[5, +] = \text{R6}$

Pop the symbol/s on the topmost of the stack (with its corresponding states) that is in the RHS of the rule number indicated in ACTION

Push the RHS of the rule indicated in ACTION

Push the state number indicated in the GOTO

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0	F					lexeme
						state

GOTO[0,F].

BOTTOM UP PARSING

Action:

Reduce

$$\text{GOTO}[\emptyset, F] = 3$$

Pop the symbol/s on the topmost of the stack (with its corresponding states) that is in the RHS of the rule number indicated in ACTION

Push the RHS of the rule indicated in ACTION

Push the state number indicated in the GOTO

BOTTOM UP PARSING

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0	F					lexeme
	3					state

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0	F					lexeme
	3					state

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0	F					lexeme
	3					state

reduce (rule
4)

Monzales, Samson, and Tejada

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0	T					lexeme
						state

GOTO[0,T].

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0	T					lexeme
	2					state

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0	T					lexeme
	2					state

reduce (rule
2)

Monzales, Samson, and Tejada

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0	E					lexeme
						state

GOTO[0, E]

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

± id * id \$

0	E					lexeme
	1					state

shift

BOTTOM UP PARSING

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

id * id \$

0	E	+				lexeme
	1	6				state

shift

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

* id \$

0	E	+	id			lexeme
	1	6	5			state

reduce (rule
6)

Monzales, Samson, and Tejada

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

* id \$

0	E	+	F			lexeme
	1	6				state

GOTO[6,F].

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

* id \$

0	E	+	F			lexeme
	1	6	3			state

reduce (rule
6)

Monzales, Samson, and Tejada

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

* id \$

0	E	+	T			lexeme
	1	6				state

GOTO[6,T].

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

* id \$

0	E	+	T			lexeme
	1	6	9			state

shift

BOTTOM UP PARSING

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

id \$

0	E	+	T	*		lexeme
	1	6	9	7		state

shift

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

\$.

0	E	+	T	*	id	lexeme
	1	6	9	7	5	state

reduce (rule
6).

Monzales, Samson, and Tejada

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

\$

0	E	+	T	*	F	lexeme
	1	6	9	7		state

GOTO[7,F].

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

\$.

0	E	+	T	*	F	lexeme
	1	6	9	7	10	state

reduce (rule
3).

Monzales, Samson, and Tejada

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

\$

0	E	+	T			lexeme
	1	6				state

GOTO[6,T].

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

\$.

0	E	+	T			lexeme
	1	6	9			state

reduce (rule
1).

Monzales, Samson, and Tejada

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

\$

0	E					lexeme
						state

GOTO[0, E]

BOTTOM UP PARSING

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

State	Action						Goto		
	id	+	*	()	\$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

\$

0	E					lexeme
	1					state

accept



SUMMARY

Syntax analysis is normally based on a formal syntax description of the language being implemented, commonly using BNF.

Parts of Syntax Analysis:

- Lexical Analysis
- Syntax Analysis

Reasons for Separating Lexical and Syntax Analysis:

- Simplicity
- Efficiency
- Portability



SUMMARY

- **Lexical Analyzer** is a pattern matcher that isolates **lexemes**, which is the basic lexical unit of a language.
- **Lexemes** are categorized by **tokens**.

Goals of Syntax Analysis:

- detect syntax errors and provide diagnostic message if an error exists
- produce a parse tree which would be used for code generation

Approaches to Syntax Analysis:

- Top-Down Approach
- Bottom Up Approach



SUMMARY

- **Top Down Approach:** Given a sentential form, xAq , the parser must choose the correct A-rule to get the next sentential form in the leftmost derivation, using only the first token produced by A. Commonly uses Recursive Descent parsing algorithm. Subprogram driven.
- **Bottom Up Approach:** The parsing problem is finding the correct RHS in a right-sentential form to reduce to get the previous right-sentential form in the derivation. Table driven.



**THANK YOU VERY
MUCH FOR LISTENING**



Monzales, Samson, and Tejada

