



BOTTOM UP PARSERS / SHIFT REDUCE PARSERS

1

Build the parse tree from leaves to root. Bottom-up parsing can be defined as an attempt to reduce the input string w to the axiom of the grammar by tracing out the rightmost derivations of an input in reverse.

SHIFT REDUCE PARSERS

- Syntax analysis is a common part of language implementation
- A lexical analyzer is a pattern matcher that isolates small-scale parts of a program
 - Detects syntax errors
 - Produces a parse tree
- A recursive-descent parser is an LL parser
 - EBNF
- Parsing problem for bottom-up parsers: **find the substring of current sentential form**



EXAMPLE

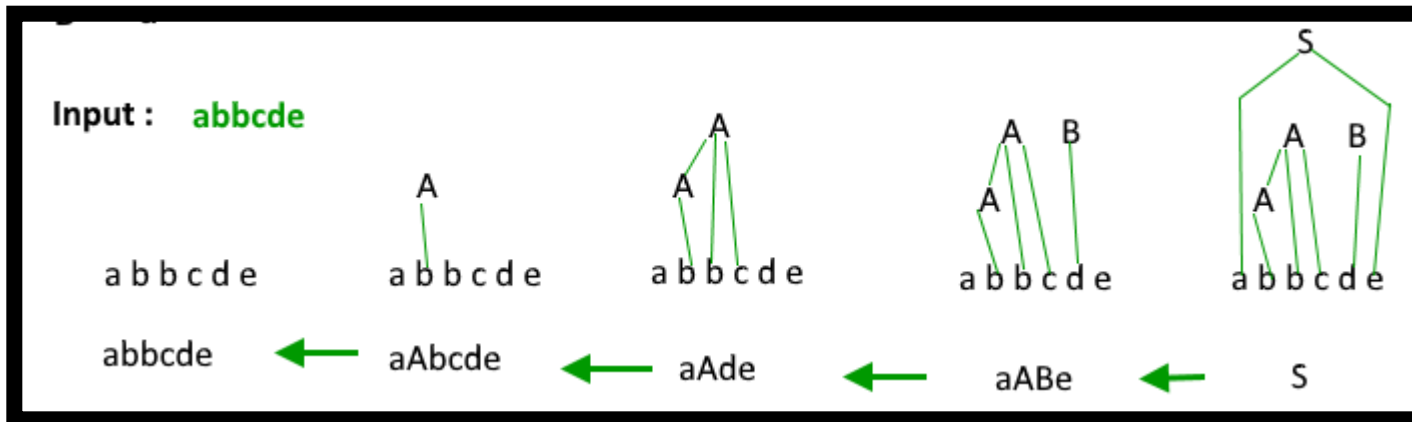
1. $S \rightarrow \mathbf{aABe}$

2. $A \rightarrow Abc$

3. $\quad \quad | b$

4. $B \rightarrow d$

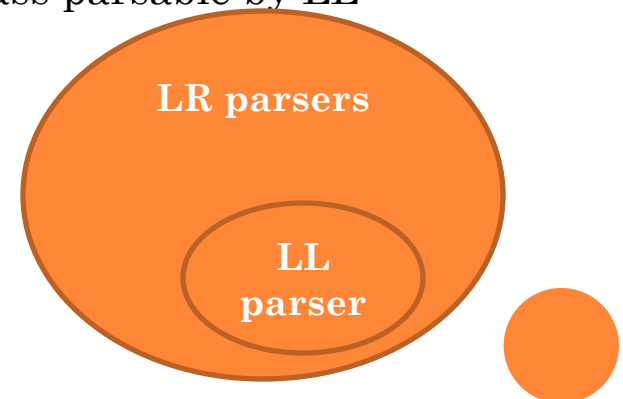
Try to construct the parse tree of the input



Rules are applied in the following order:
3, 2, 4, 1

SR ALGORITHMS

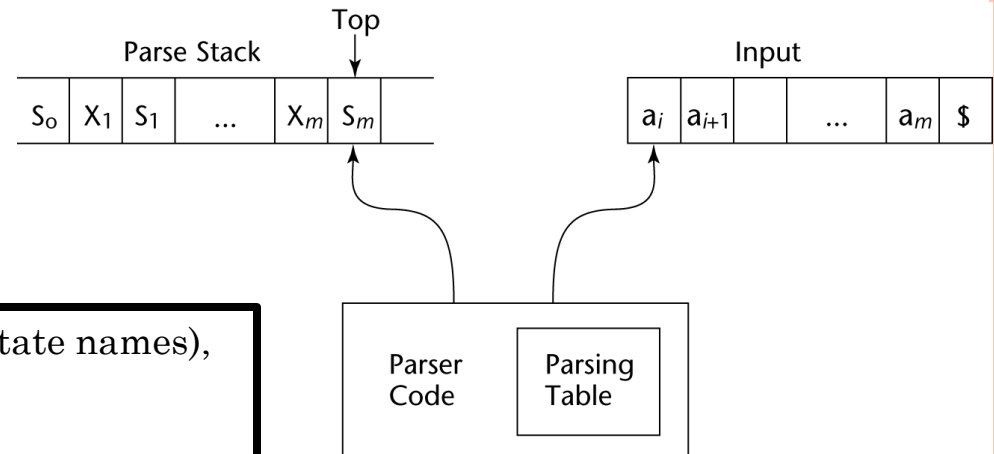
- SR Algos
 - **Shift** is the action of moving the next token to the top of the parse stack
 - **Reduce** is the action of replacing the sentential form on the top of the parse stack with its corresponding LHS
- **Advantages of LR/SR parsers:**
 - They will work for nearly all grammars that describe programming languages.
 - 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- - STRUCTURE OF AN LR PARSER

An LR configuration stores the state of an LR parser

$(S_0X_1S_1X_2S_2...X_mS_m, a_ia_{i+1}...a_n\$)$



LR parser is a **table driven**(rows are state names), it has 2 entries:

ACTION specifies either shift, reduce end of parsing or error, “given the parser state and the next token what to do?”, columns are terminals and \$.

GOTO specifies which state to put on top of the stack after a reduction action is done. Columns are NTs

BOTTOM-UP PARSING

- Initial configuration: $(S_0, a_1 \dots a_n \$)$
- Parser actions:
 - **For a Shift**, the next symbol of input is pushed onto the stack, along with the state symbol (Item) that is part of the Shift specification in the Action table
 - **For a Reduce**, remove the sentential from the stack, along with its state symbols (items). Push the LHS of the adequate production. Push the state symbol from the GOTO table, using the state symbol just below the new LHS in the stack and the RHS of the new production as the row and column into the GOTO table

All empty entries denotes **ERROR**

Parsing ends:

- For an **Accept**, the parse is complete and no errors were found.
- For an **Error**, the parser calls an error-handling routine.



LR PARSING TABLE- ARITHMETIC GRAMMAR EXAMPLE

	Action						Goto		
State	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			





SHIFT/REDUCE PARSERS

1. LR(0),
2. SLR(1),
3. LR(1),
4. LALR(1) : (Bison/ yacc)

EXAMPLE

Grammar: $E \rightarrow E + n \mid n$

input: $n+n\$$ (\$ denotes the input end, also initial stack symbol)

stack	input	action
\$	$n+n\$$	shift n
$\$n$	$+n\$$	reduce $E \rightarrow n$
$\$E$	$+n\$$	shift $+$
$\$E+$	$n\$$	shift n
$\$E+n$	$\$$	reduce $E \rightarrow E+n$
$\$E$	$\$$	accept

Left recursion is not a problem in bottom-up parser

DECISION PROBLEMS IN BOTTOM –UP PARSING (PARSING CONFLICTS):

- Shift-Reduce conflicts: almost come from ambiguities, and almost the right disambiguating **rule is to shift** (dangling-else).
- Reduce-Reduce conflicts are more difficult; bottom-up parsers try to resolve them using Follow contexts as we will see later.

SHIFT-REDUCE CONFLICT EXAMPLE

1. $S \rightarrow I$
2. $| o$
3. $I \rightarrow i S$
4. $| i S e S$

Input: i i o e o \$

stack	input	action
\$	i i o e o \$	shift i
\$i	i o e o \$	shift i
\$ii	o e o \$	shift o
\$ ii o	e o \$	reduce $S \rightarrow o$
\$ii S	e o \$	SHIFT /REDUCE Shift e or reduce $I \rightarrow iS$ (shift)
\$iiSe	o \$	shift o
\$iiSeo	\$	reduce $S \rightarrow o$
\$iiSeS	\$	reduce $I \rightarrow i S e S$
\$il	\$	reduce $S \rightarrow I$
\$iS	\$	reduce $l \rightarrow i S$
\$l	\$	reduce $S \rightarrow I$
\$S	\$	accept

REDUCE-REDUCE CONFLICT EXAMPLE

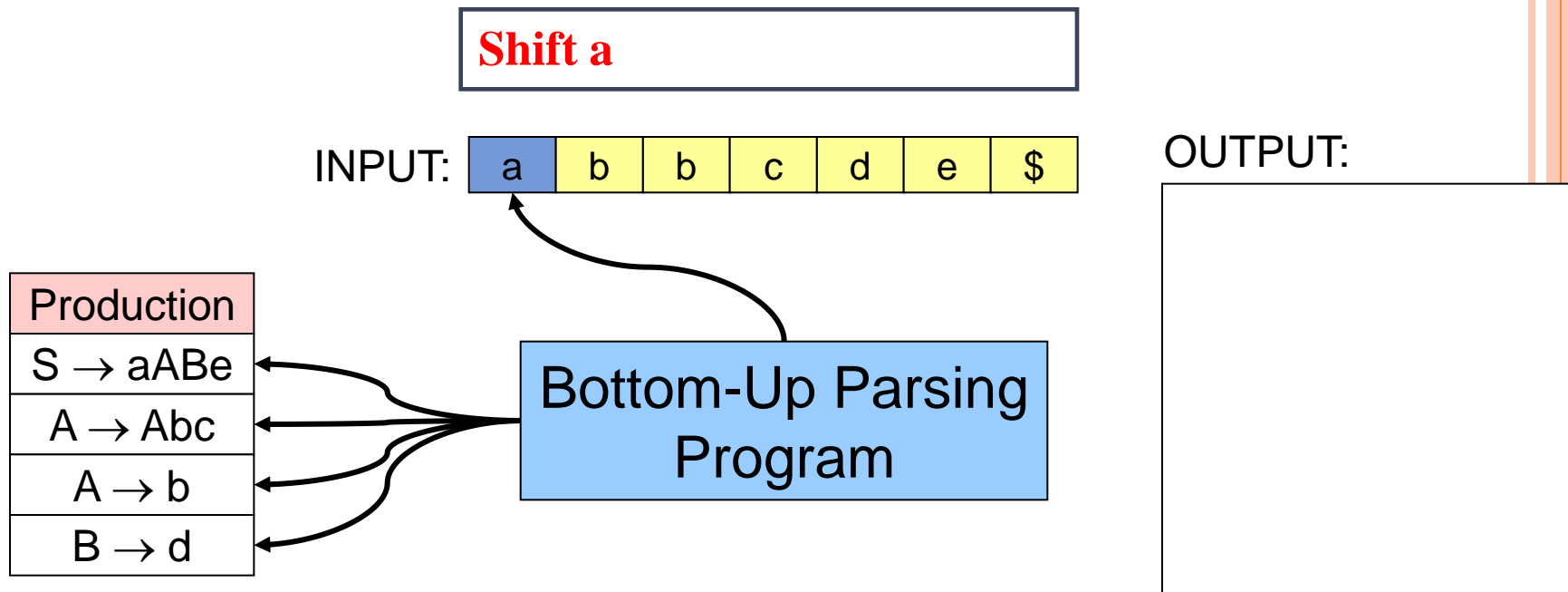
- $S \rightarrow AB$
- $A \rightarrow x$
- $B \rightarrow x$

input: x x \$

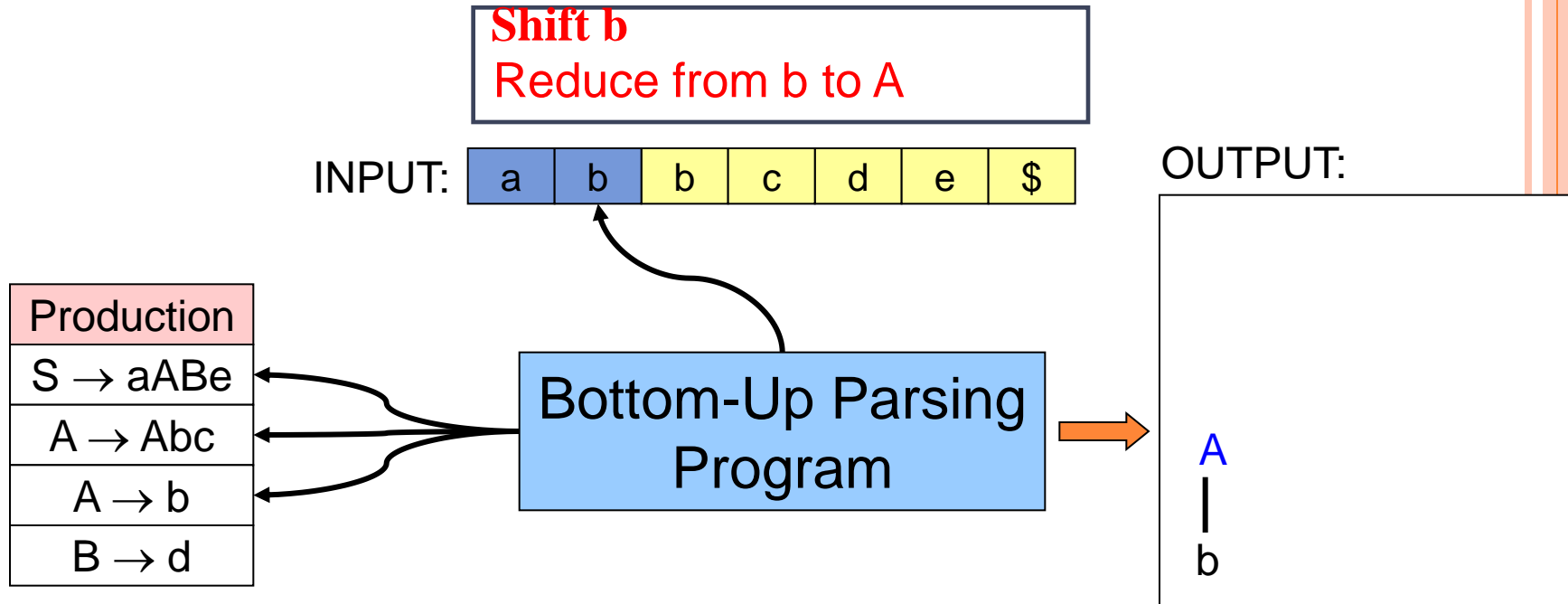
stack	input	action
\$	x x\$	shift x
\$x	x\$	reduce A \rightarrow x (reduce B \rightarrow x)
\$A	x\$	shift x
\$Ax	\$	(reduce A \rightarrow x) reduce B \rightarrow x
\$AB	\$	reduce S \rightarrow AB
\$S	\$	accept

Reduce Reduce
conflict

BOTTOM-UP PARSER EXAMPLE



....



...

Shift A

INPUT:

a	A	b	c	d	e	\$
---	---	---	---	---	---	----

OUTPUT:

A
b

Production
$S \rightarrow aABe$
$A \rightarrow Abc$
$A \rightarrow b$
$B \rightarrow d$

Bottom-Up Parsing
Program

...

Shift b

INPUT:

a	A	b	c	d	e	\$
---	---	---	---	---	---	----

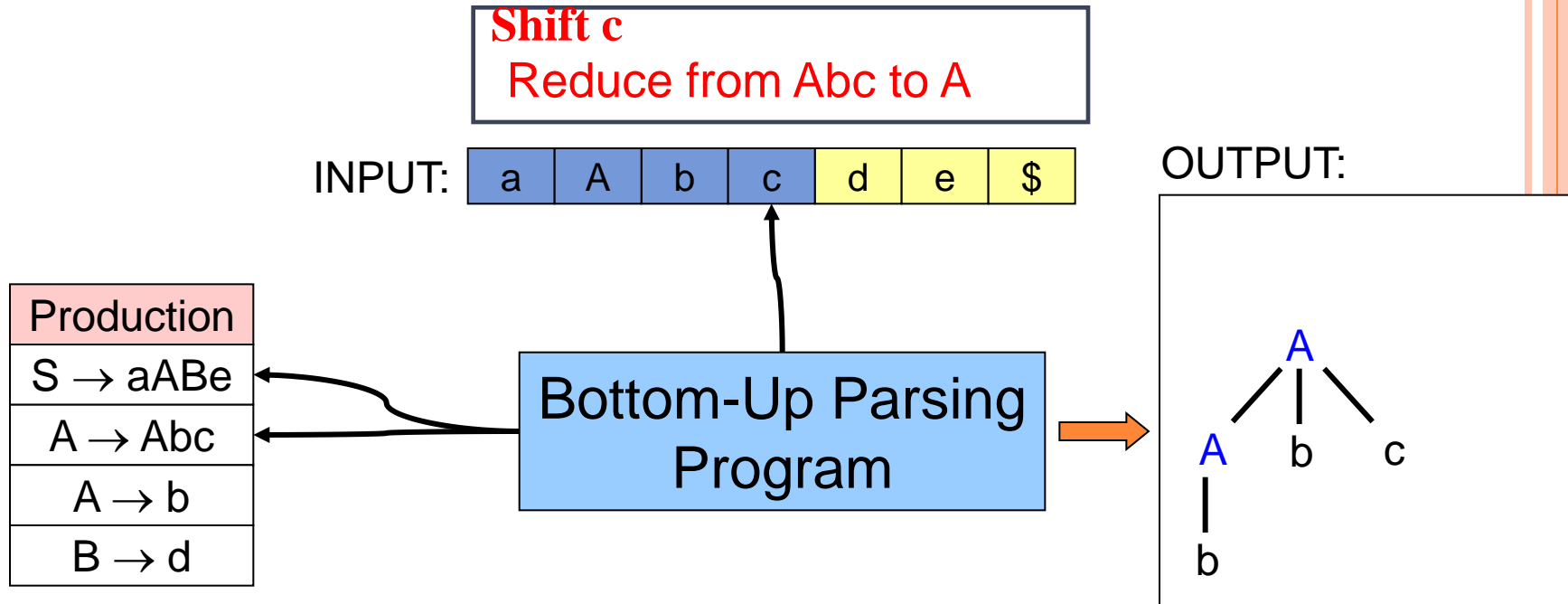
OUTPUT:

A
b

Production
$S \rightarrow aABe$
$A \rightarrow Abc$
$A \rightarrow b$
$B \rightarrow d$

Bottom-Up Parsing
Program

...



...

Shift A

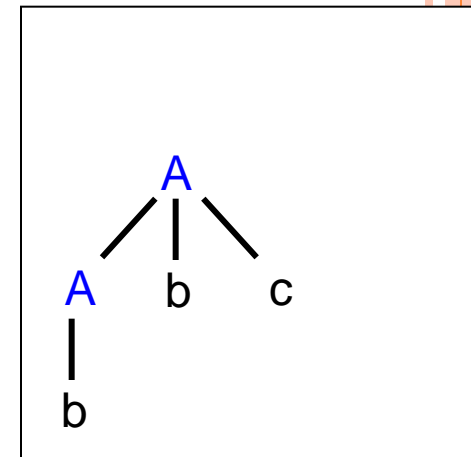
INPUT:

a	A	d	e	\$
---	---	---	---	----

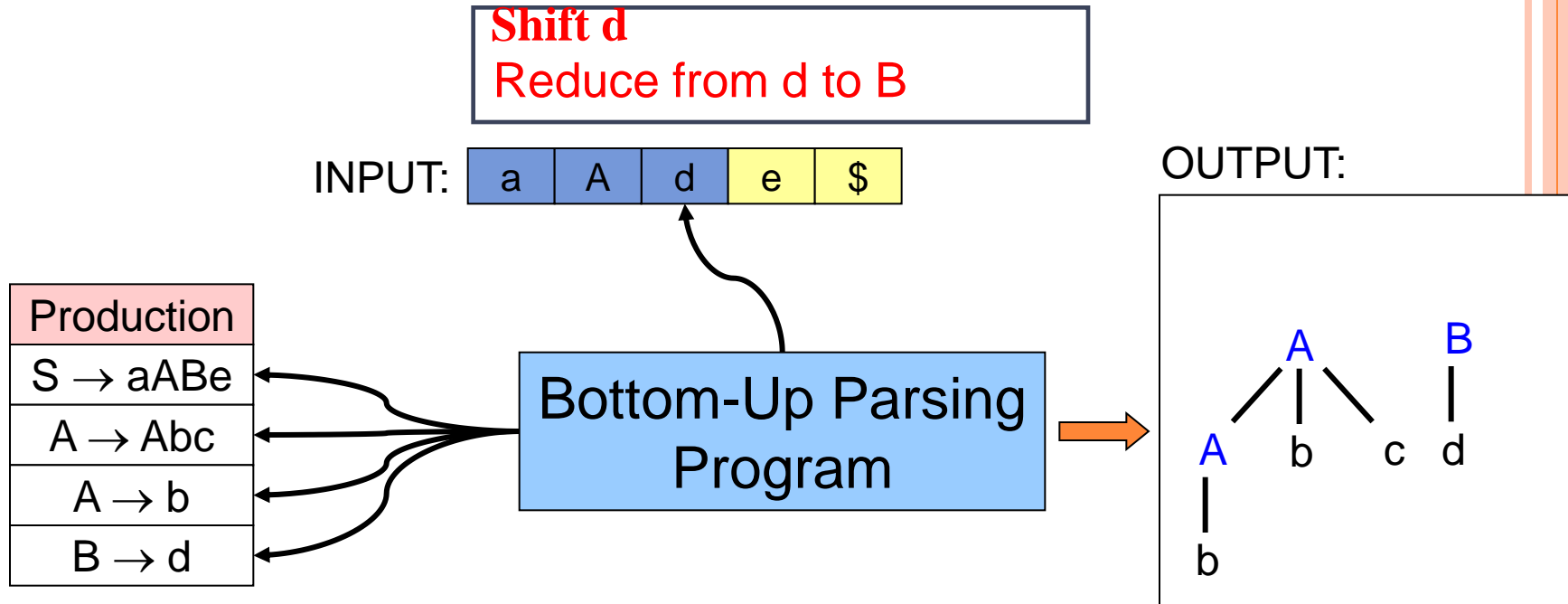
Production
$S \rightarrow aABe$
$A \rightarrow Abc$
$A \rightarrow b$
$B \rightarrow d$

Bottom-Up Parsing Program

OUTPUT:



...



...

Shift B

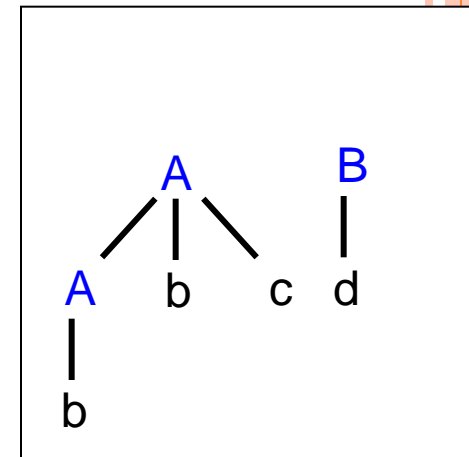
INPUT:

a	A	B	e	\$
---	---	---	---	----

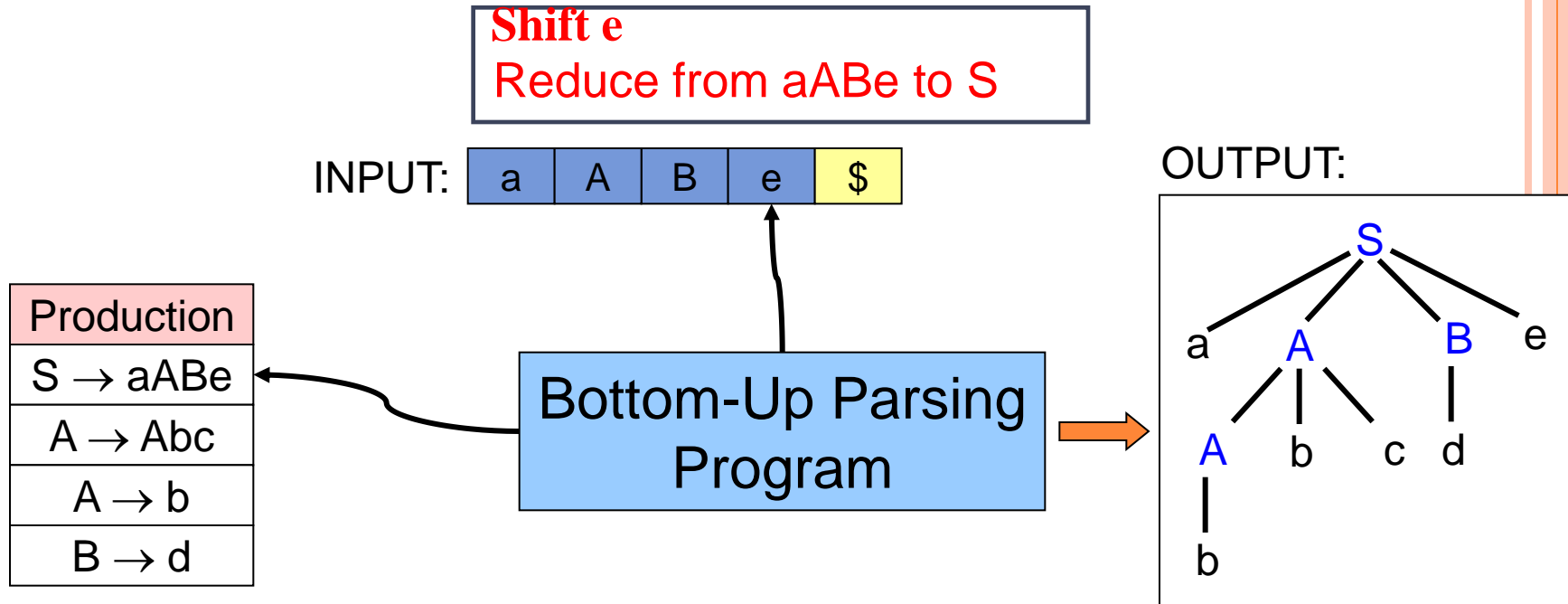
Production
$S \rightarrow aABe$
$A \rightarrow Abc$
$A \rightarrow b$
$B \rightarrow d$

Bottom-Up Parsing Program

OUTPUT:



...



...

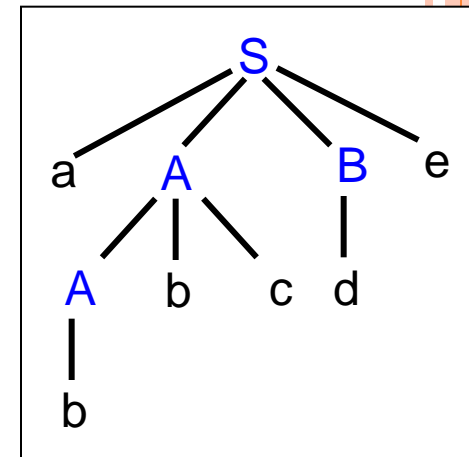
Shift S
Hit the target \$

INPUT: S \$

Production
$S \rightarrow aABe$
$A \rightarrow Abc$
$A \rightarrow b$
$B \rightarrow d$

Bottom-Up Parsing
Program

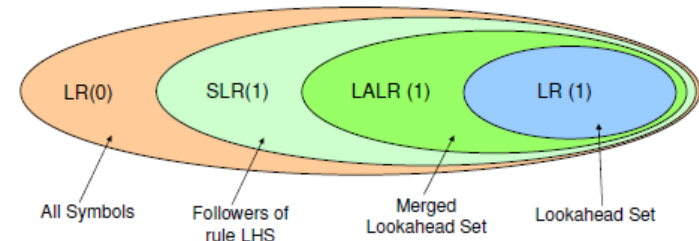
OUTPUT:



This parser is known as an **LR Parser** because it scans the input from **Left to right**, and it constructs a **Rightmost derivation** in reverse order.

SHIFT-REDUCE PARSERS

Comparing Parsers: conditioning reduce actions



- LR(0) no lookahead.
- SLR(1) uses *Follow sets*.
- LR(1) uses context to *split Follow sets into subsets for different parsing paths* (huge, inefficient parsers).
- LALR(1) like LR(1) but coarser subsets are used (achieves most of the benefit, but much smaller and faster).

AUGMENTED GRAMMAR

- SR parsers have **trouble figuring out when to accept**, so acceptance is turned into a **reduction by a new rule $S' \rightarrow S$ (unit production)** with a new start symbol S' .
- Add this rule to the grammar:

augmented grammar


1. $S' \rightarrow S$

2. $S \rightarrow AB$

3. $A \rightarrow x$

4. $B \rightarrow x$

input: $x\ x\$$

stack	input	action
\$	$x\ x\$$	shift x
$\$x$	$x\$$	reduce $A \rightarrow x$ (reduce $B \rightarrow x$)
$\$A$	$x\$$	shift x
$\$Ax$	$\$$	reduce $B \rightarrow x$ (reduce $A \rightarrow x$)
$\$AB$	$\$$	reduce $S \rightarrow AB$
$\$S$	$\$$	reduce $S' \rightarrow S$ 
$\$S'$	$\$$	accept

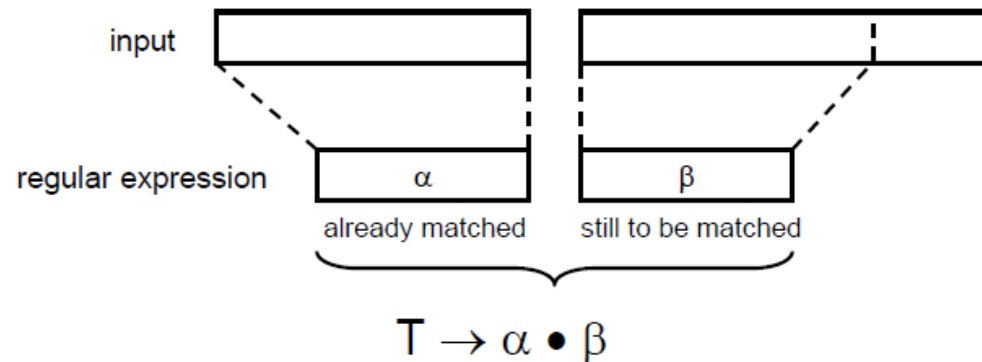


LR(0) : SR PARSER WITH NO LOOKAHEAD SYMBOL

25

DOTTED RULE: ITEM DENOTES STATE SYMBOL

- We represent a configuration of an LR(0) parser with a data structure called an LR(0) item
- keep trace of matched characters in a token description: $T \rightarrow \alpha\beta$



Types of items

❑ **SHIFT** item: dot in front of a Terminal

$A \rightarrow \bullet ab$ // shift a

$A \rightarrow a \bullet b$ //shift b

❑ **REDUCE** item: dot at the end

$A \rightarrow ab \bullet$ //reduce ab into A

❑ **non-basic (GOTO)** item: dot in front of a NT

$A \rightarrow X \bullet Y$

LR(0) PARSER

1. Construct transition relation between states

- Use **item set** and **Next item (GOTO) set**
- States are LR(0) items
- Shift items of the form $P \rightarrow \alpha \bullet \underline{a} \beta$ (there is a terminal after dot)
- Reduce items of the form $P \rightarrow \alpha \bullet$ (no more character after dot)

2. Construct parsing table

- If every state contains **no conflicts** \rightarrow use LR(0) parsing algo
- Otherwise
 - Rewrite grammar productions or
 - Resolve conflict or
 - Use **stronger parsing technique** SLR(1), LR(1) or LALR(1)

BASIC OPERATIONS: CLOSURE & GOTO

- To build a set of states the 2 basic routines must be defined, they are:
 - `closure(I)` and `goto(I, X)`, where `I` is a set of items, `X` is a grammar symbol, (terminal or nonterminal).
- The closure routine adds an item to the set of items, the productions, which have the nonterminal in the left part, are added.
- It expands dots in front of non-terminals

```
closure (I)  {
do  {
initially I = {[A → w • Xv]}
  for(each item [A → w • Xv] in I)  {
    for (each grammar production X → u) {
      I+= [X → • u]; //Operation += adds an element to a set
    }
  }
} while (I is changed);
return I;
}
```

BASIC OPERATIONS: CLOSURE & GOTO

- The goto routine moves the dot behind the symbol $X \in NT \cup T$. That means that transition is performed from one state over X .

```
goto (I, X){  
  J={};  
  for (each item [A→ w•Xv] from I)  
    J+= [A → wX • v];  
  
  return closure (J);  
}
```

LR(0) PARSING TABLE

Grammar rules must be numbered

State	Action	goto
	terminals + \$	non terminals
items number states		

- 1) **SHIFT_{state}** : IF $[A \rightarrow x \bullet a w]$ is in state m for input symbol a , AND An item $X \rightarrow x a \bullet w$ is in state n , THEN enter S_n (Shift n) at $\text{Table}[m, \underline{a}]$.
- 2) **REDUCE_{state}**: IF $[A \rightarrow w \bullet]$ is in state n , THEN enter R_i at $\text{Table}[n, a]$ where i is the production $i : A \rightarrow w$ for each terminal (include \$) symbol
- 3) IF $[S' \rightarrow S \bullet]$ is in State n , THEN enter "Accept" at $\text{Table}[n, \$]$
- 4) **GOTO** : IF $[A \rightarrow x \bullet B w]$ is in State m (B is a NT), AND $[A \rightarrow x B \bullet w]$ is in State n , THEN enter n at $\text{Table}[m, B]$.

If this algo generates conflict actions (Shift-Reduce or Reduce-Reduce) \Rightarrow the grammar is not LR(0)

EXERCISE1: COMPUTE LR(0) ITEMS FOR THE FOLLOWING EXPRESSION GRAMMAR

1. $S \rightarrow E \$$
2. $E \rightarrow E + T$
3. $E \rightarrow T$
4. $T \rightarrow i$
5. $T \rightarrow (E)$

Note that

- *small characters denote terminals*
- *the grammar is already augmented*

$I_0 = \text{closure}([S \rightarrow \bullet E]) = \{$
 $[S \rightarrow \bullet E \$]$
 $[E \rightarrow \bullet E + T]$
 $[E \rightarrow \bullet T]$
 $[T \rightarrow \bullet i]$
 $[T \rightarrow \bullet (E)]$
 $\}$

EXERCISE1 (RESOLVED)

1. Continue the set of items (9 items) by completing the transition diagram for the LR(0) automaton
2. Construct LR(0) parsing table
3. Parse the input $i+i\$$

ITEMS SET:

```
I0= closure([S → • E])= {  
[S → • E$]  
[E → • E+T]  
[E → • T]  
[T → • i]  
[T → • (E)]  
}  
  
I1=goto(I0,E)= {  
[S → E • $]//acceptance state  
[E → E • +T]  
}  
  
I2=goto(I0,T)= {[E → T • ]}  
  
I3=goto(I0,i)= {[T → i • ]}  
  
I4= goto(I0,())= {  
[T → (•E)]  
[E → • E+T]  
[E → • T]  
[T → • i]  
[T → • (E)]  
}  
  
I5=goto(I1,+)= {  
[E → E+ • T]  
[T → • i]  
[T → • (E)]  
}
```

```
I6= goto(I4,E)= {  
[T → (E • )]  
[E → E • +T]  
}  
  
goto(I4,T)=I2  
goto(I4,i)=I3  
goto(I4,())=I4  
  
I7=goto(I5,T)= {[E → E+T • ]}  
goto(I5,i)=I3  
goto(I5,())=I4  
  
I8= goto(I6,())= {[T → (E) • ]}  
goto(I6,+)=I5
```

PARSING TABLE

State	Terminals Shift-reduce actions					Non-terminals GOTO	
	i	+	()	\$	E	T
0							
1							
2							
3							
4							
5							
6							
7							
8							

PARSING TABLE:

$I_0 = \text{closure}([S \rightarrow \cdot E]) = \{$
 $[S \rightarrow \cdot E\$]$
 $[E \rightarrow \cdot E+T]$
 $[E \rightarrow \cdot T]$
 $[T \rightarrow \cdot i]$
 $[T \rightarrow \cdot (E)]\}$

$I_1 = \text{goto}(I_0, E) = \{$
 $[S \rightarrow E \cdot \$] // \text{acceptance state}$
 $[E \rightarrow E \cdot +T]\}$

$I_2 = \text{goto}(I_0, T) = \{[E \rightarrow T \cdot]\}$

$I_3 = \text{goto}(I_0, i) = \{[T \rightarrow i \cdot]\}$

$I_4 = \text{goto}(I_0, () = \{$

$[T \rightarrow (\cdot E)]$

$[E \rightarrow \cdot E+T]$

$[E \rightarrow \cdot T]$

$[T \rightarrow \cdot i]$

$[T \rightarrow \cdot (E)]\}$

$I_5 = \text{goto}(I_1, +) = \{$

$[E \rightarrow E+ \cdot T]$

$[T \rightarrow \cdot i]$

$[T \rightarrow \cdot (E)]\}$

$I_6 = \text{goto}(I_4, E) = \{$

$[T \rightarrow (E \cdot)]$

$[E \rightarrow E \cdot +T]\}$

$I_7 = \text{goto}(I_5, T) = \{[E \rightarrow E+T \cdot]\}$

$I_8 = \text{goto}(I_6,) = \{[T \rightarrow (E) \cdot]\}$

State	Terminals Shift-reduce actions					Non-terminals GOTO	
	i	+	()	\$	E	T
0	S3		S4			1	2
1		S5			ACC		
2	R3	R3	R3	R3	R3		
3	R4	R4	R4	R4	R4		
4	S3		S4			6	2
5	S3		S4				7
6		S5		S8			
7	R2	R2	R2	R2	R2		
8	R5	R5	R5	R5	R5		

No multiple entries in LR(0) table of items then the grammar is LR(0)

EXERCISE1 CONT'D

PARSE THE INPUT I+I \$

1. $S \rightarrow E$
2. $E \rightarrow E + T$
3. $E \rightarrow T$
4. $T \rightarrow i$
5. $T \rightarrow (E)$

Stack	input	Action: shift S, reduce R
\emptyset	i+i\$	S3 push i goto3
<u>0</u> i 3	+i\$	R4: <u>I</u> →i ; reduce i goto 2 ([<u>0</u> , <u>I</u>]=2)
<u>0</u> T 2	+i\$	R3: <u>E</u> → T ; reduce i goto 1 ([<u>0</u> , <u>E</u>]=1)
\emptyset E 1	+i\$	S5 push + goto5
\emptyset E 1 + 5	i\$	S3 push i goto3
\emptyset E 1 + <u>5</u> i 3	\$	R4: <u>I</u> →i ; [<u>5</u> , <u>I</u>]=7
\emptyset E 1 + <u>5</u> T 7	\$	R2: <u>E</u> →E+T; [<u>0</u> , <u>E</u>]=1
\emptyset E 1	\$	ACCEPT

	Terminals Shift-reduce actions					NTs GOTO	
	i	+	()	\$	E	T
0	S3		S4			1	2
1		S5			ACC		
2	R3	R3	R3	R3	R3		
3	R4	R4	R4	R4	R4		
4	S3		S4			6	2
5	S3		S4				7
6		S5		S8			
7	R2	R2	R2	R2	R2		
8	R5	R5	R5	R5	R5		

NEXT WEEK RECORDED SESSION

Handling LR(0) conflicts

solution: use a one-token look-ahead

two-dimensional ACTION table [state, token]

- ❑ different construction of ACTION table
- ❑ SLR(1) – Simple LR
- ❑ LR(1)
- ❑ LALR(1) – Look-Ahead LR