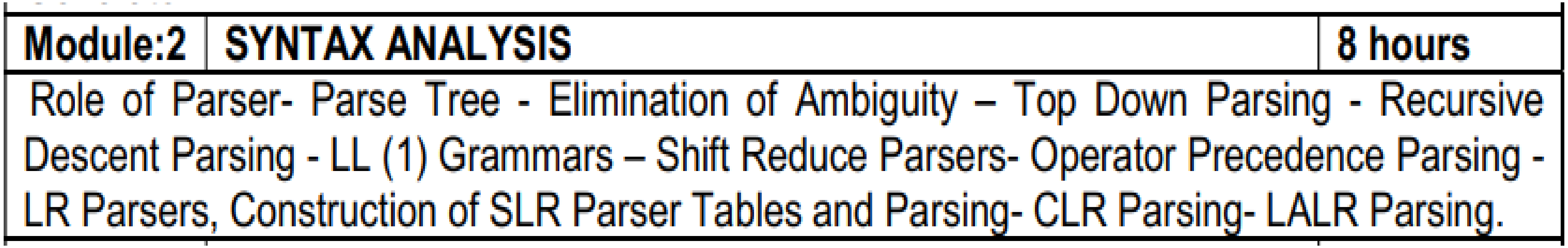
BCSE307L – COMPILER DESIGN

TEXT BOOK:

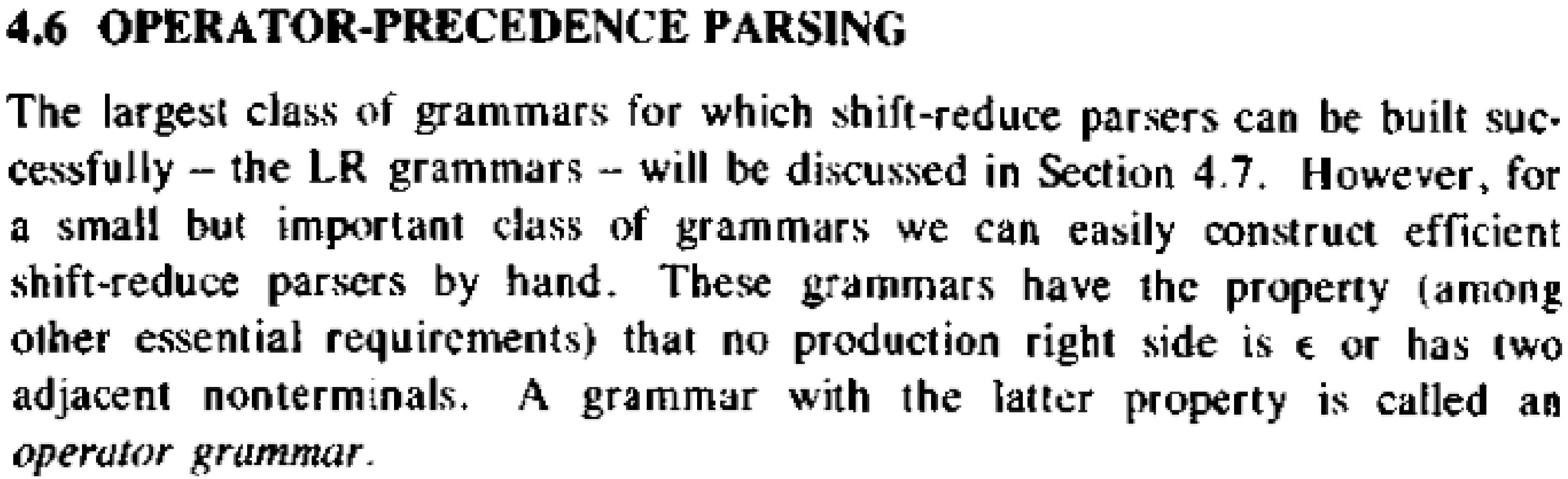
1. Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, "Compilers: Principles, Techniques and Tools", Second Edition, Pearson Education Limited, 2014.



Module:2	**SYNTAX ANALYSIS**	**8 hours**
Role of Parser- Parse Tree - Elimination of Ambiguity – Top Down Parsing - Recursive Descent Parsing - LL (1) Grammars – Shift Reduce Parsers- Operator Precedence Parsing - LR Parsers, Construction of SLR Parser Tables and Parsing- CLR Parsing- LALR Parsing.		

Parsing Techniques

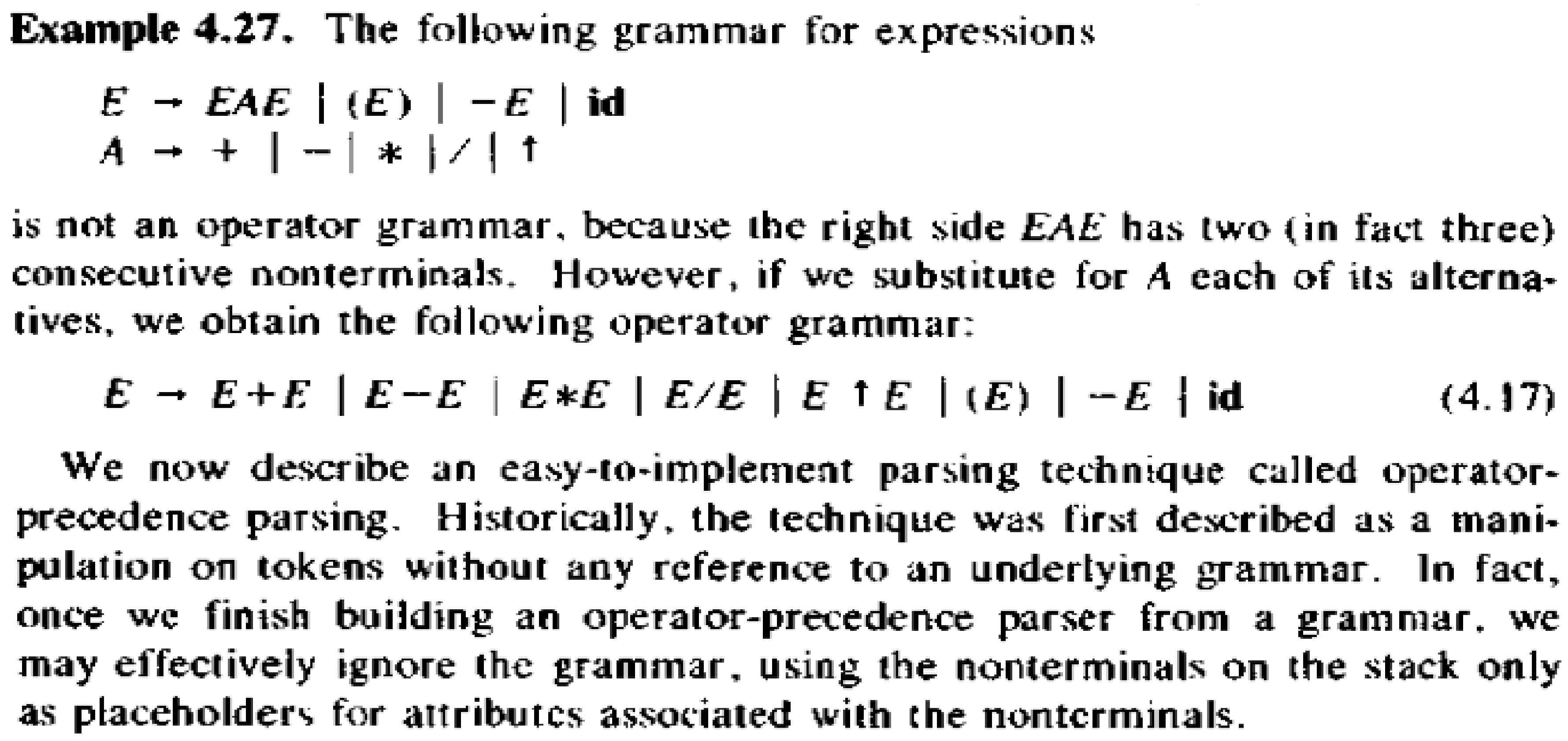
Operator Precedence



4.6 OPERATOR-PRECEDENCE PARSING

The largest class of grammars for which shift-reduce parsers can be built successfully – the LR grammars – will be discussed in Section 4.7. However, for a small but important class of grammars we can easily construct efficient shift-reduce parsers by hand. These grammars have the property (among other essential requirements) that no production right side is ϵ or has two adjacent nonterminals. A grammar with the latter property is called an *operator grammar*.

Operator Precedence Parsing



Example 4.27. The following grammar for expressions

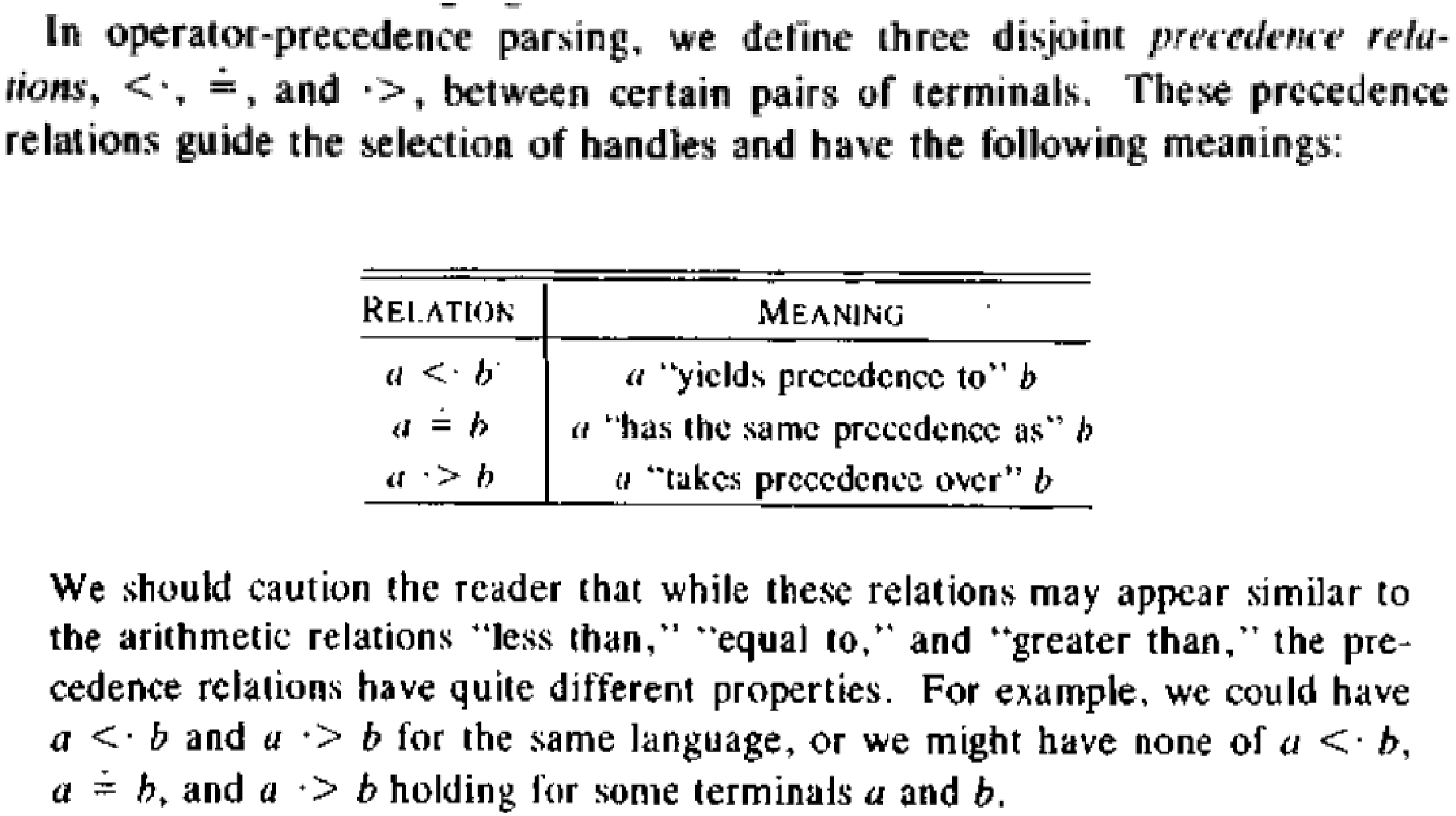
$$E \to EAE \mid (E) \mid -E \mid \textbf{id}$$
$$A \to + \mid - \mid * \mid / \mid \uparrow$$

is not an operator grammar, because the right side *EAE* has two (in fact three) consecutive nonterminals. However, if we substitute for *A* each of its alternatives, we obtain the following operator grammar:

$$E \to E+E \mid E-E \mid E*E \mid E/E \mid E \uparrow E \mid (E) \mid -E \mid \textbf{id} \qquad (4.17)$$

We now describe an easy-to-implement parsing technique called operator-precedence parsing. Historically, the technique was first described as a manipulation on tokens without any reference to an underlying grammar. In fact, once we finish building an operator-precedence parser from a grammar, we may effectively ignore the grammar, using the nonterminals on the stack only as placeholders for attributes associated with the nonterminals.

Operator Precedence Parsing



In operator-precedence parsing, we define three disjoint *precedence relations*, $<\cdot$, $\doteq$, and $\cdot>$, between certain pairs of terminals. These precedence relations guide the selection of handles and have the following meanings:

RELATION	MEANING
$a <\cdot b$	a "yields precedence to" b
$a \doteq b$	a "has the same precedence as" b
$a \cdot> b$	a "takes precedence over" b

We should caution the reader that while these relations may appear similar to the arithmetic relations "less than," "equal to," and "greater than," the precedence relations have quite different properties. For example, we could have $a <\cdot b$ and $a \cdot> b$ for the same language, or we might have none of $a <\cdot b$, $a \doteq b$, and $a \cdot> b$ holding for some terminals a and b.

Operator Precedence Parsing

Suppose that between a_i and a_{i+1} exactly one of the relations $<\cdot$, $\doteq$, and $\cdot>$ holds. Further, let us use \$ to mark each end of the string, and define $\$ <\cdot b$ and $b \cdot> \$$ for all terminals b. Now suppose we remove the nonterminals from the string and place the correct relation $<\cdot$, $\doteq$, or $\cdot>$, between each pair of terminals and between the endmost terminals and the \$'s marking the ends of the string. For example, suppose we initially have the right-sentential form **id** + **id** $*$ **id** and the precedence relations are those given in Fig. 4.23. These relations are some of those that we would choose to parse according to grammar (4.17).

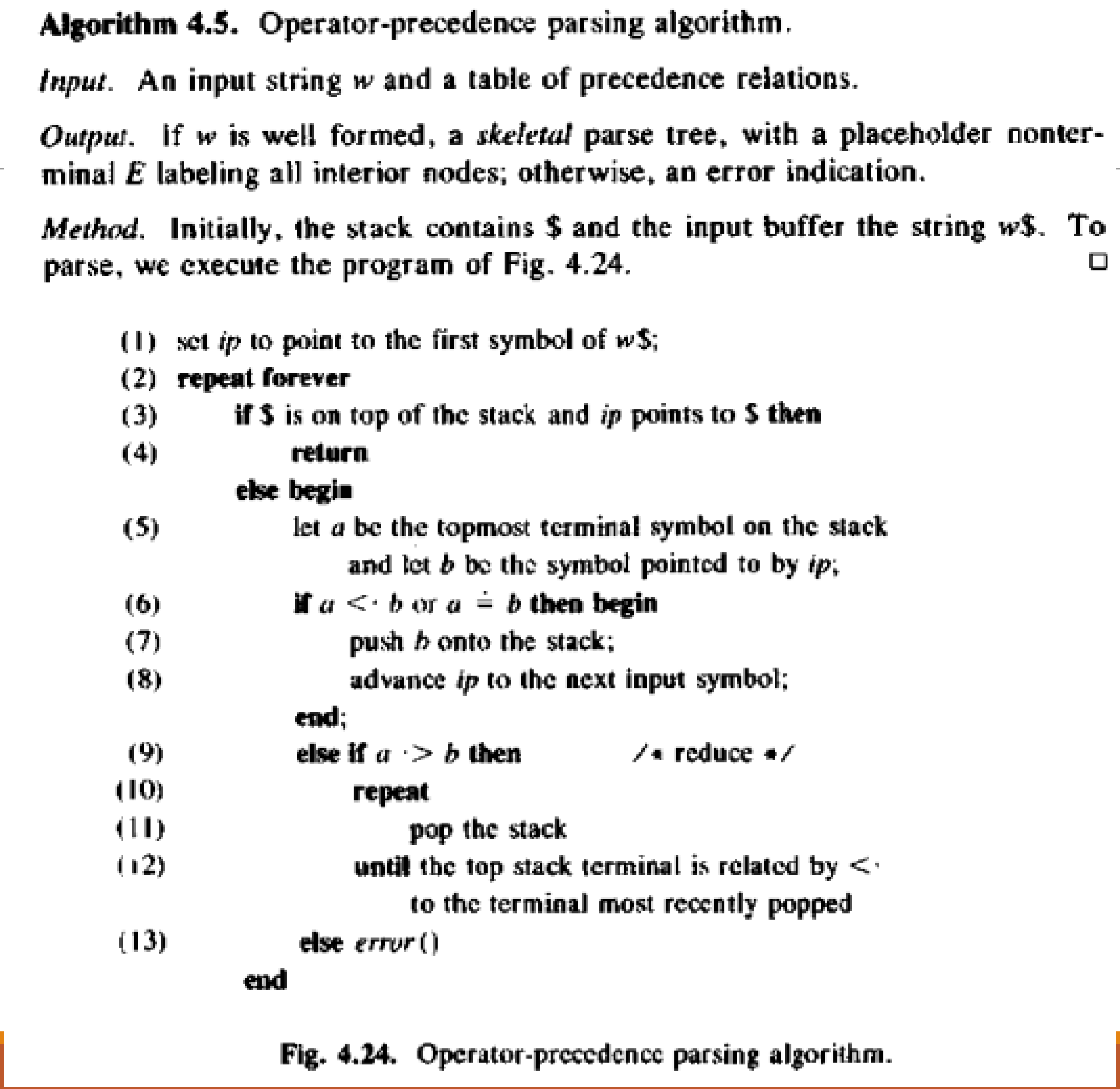
	id	+	*	\$
id		$\cdot>$	$\cdot>$	$\cdot>$
+	$<\cdot$	$\cdot>$	$<\cdot$	$\cdot>$
*	$<\cdot$	$\cdot>$	$\cdot>$	$\cdot>$
\$	$<\cdot$	$<\cdot$	$<\cdot$	

Fig. 4.23. Operator-precedence relations.

Then the string with the precedence relations inserted is:

$$\$ <\cdot \textbf{id} \cdot> + <\cdot \textbf{id} \cdot> * <\cdot \textbf{id} \cdot> \$ \qquad (4.18)$$

Operator Precedence Parsing



Algorithm 4.5. Operator-precedence parsing algorithm.

Input. An input string w and a table of precedence relations.

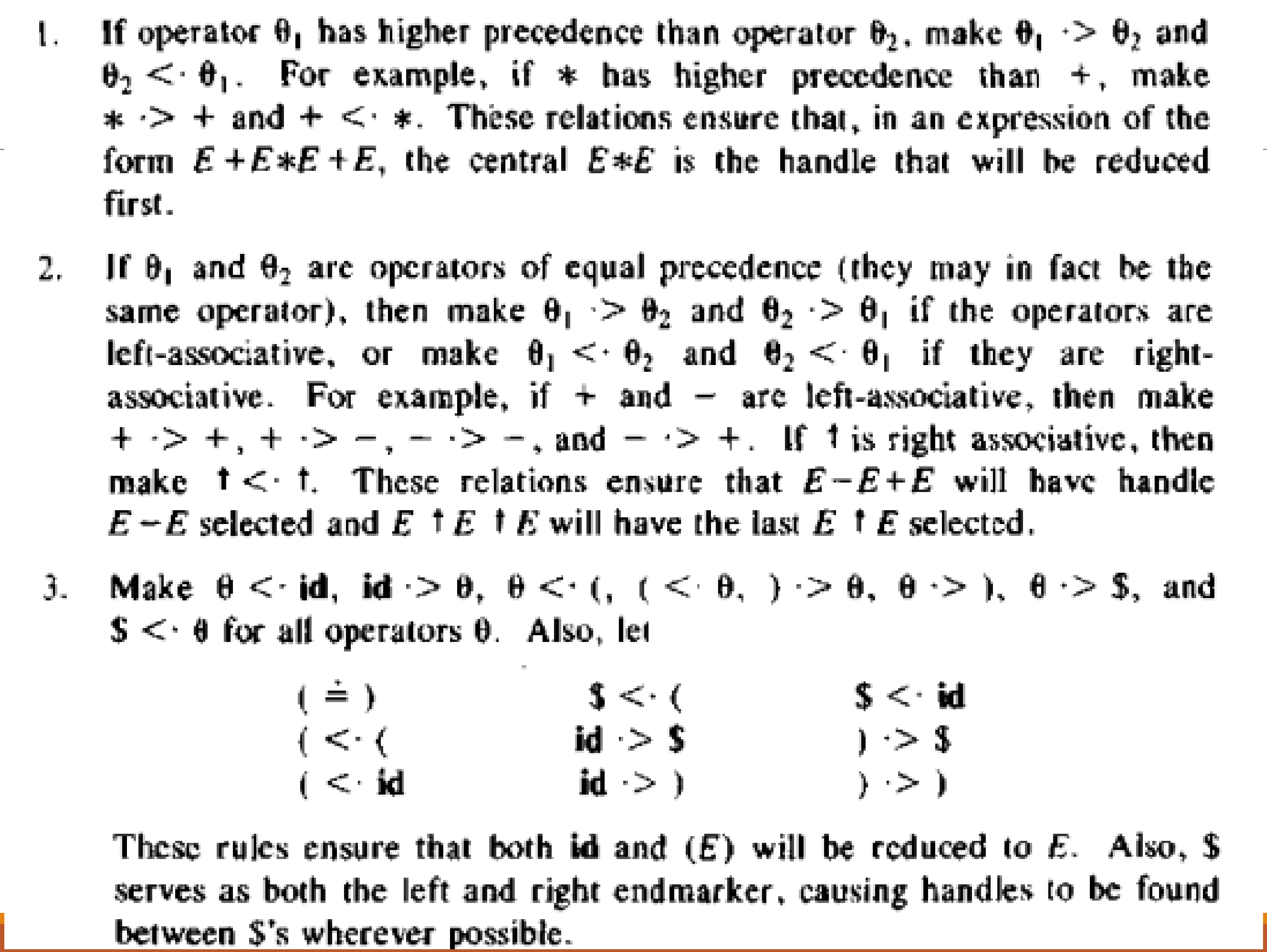
Output. If w is well formed, a *skeletal* parse tree, with a placeholder nonterminal E labeling all interior nodes; otherwise, an error indication.

Method. Initially, the stack contains \$ and the input buffer the string w\$. To parse, we execute the program of Fig. 4.24. □

```
(1)  set ip to point to the first symbol of w$;
(2)  repeat forever
(3)      if $ is on top of the stack and ip points to $ then
(4)          return
         else begin
(5)          let a be the topmost terminal symbol on the stack
                 and let b be the symbol pointed to by ip;
(6)          if a <· b or a ≐ b then begin
(7)              push b onto the stack;
(8)              advance ip to the next input symbol;
             end;
(9)          else if a ·> b then              /* reduce */
(10)             repeat
(11)                 pop the stack
(12)             until the top stack terminal is related by <·
                     to the terminal most recently popped
(13)         else error()
         end
```

Fig. 4.24. Operator-precedence parsing algorithm.

Operator Precedence Parsing



1. If operator θ_1 has higher precedence than operator θ_2, make $\theta_1 \cdot\!> \theta_2$ and $\theta_2 <\!\cdot\; \theta_1$. For example, if $*$ has higher precedence than $+$, make $* \cdot\!> +$ and $+ <\!\cdot\; *$. These relations ensure that, in an expression of the form $E+E*E+E$, the central $E*E$ is the handle that will be reduced first.

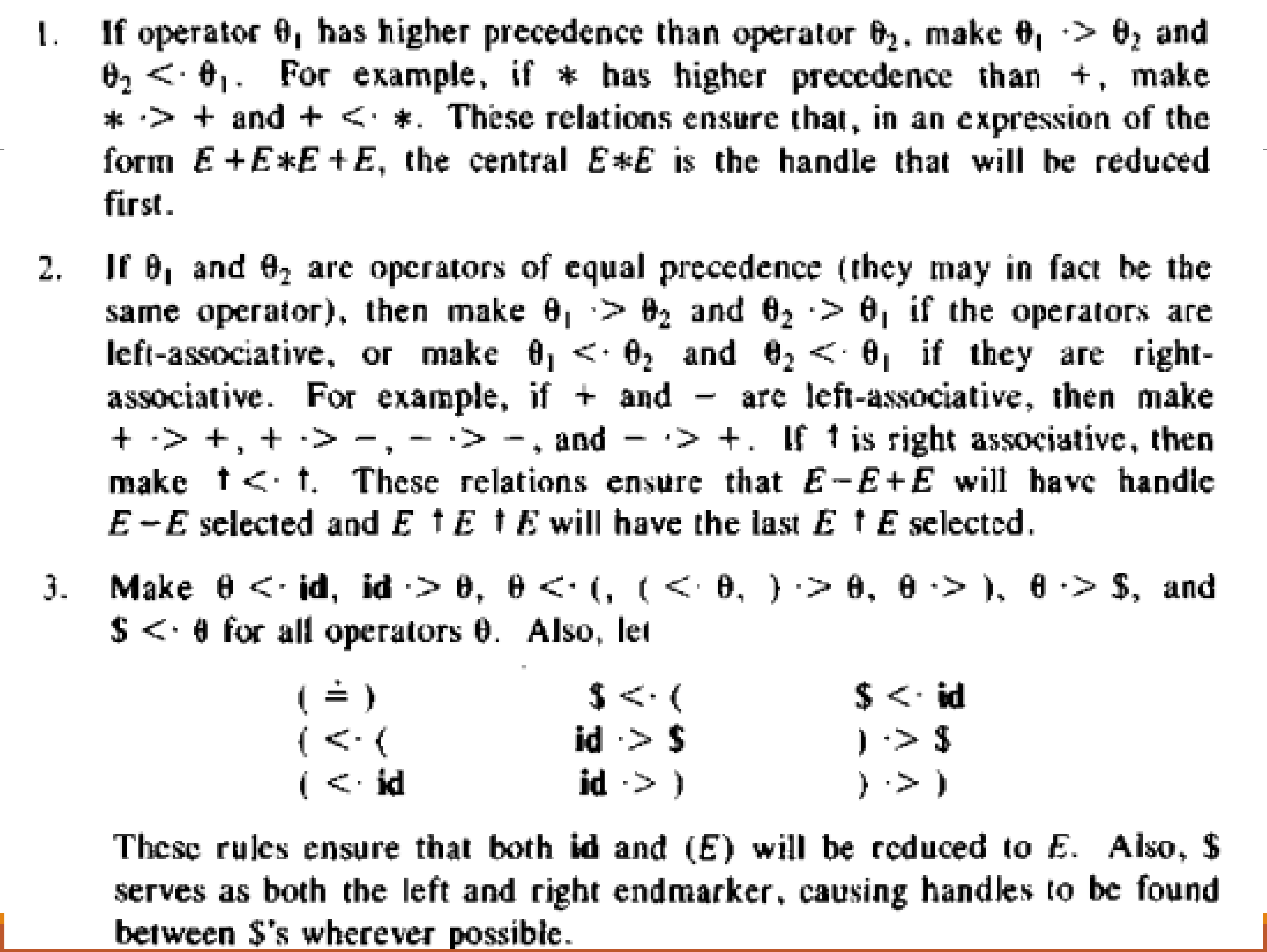
2. If θ_1 and θ_2 are operators of equal precedence (they may in fact be the same operator), then make $\theta_1 \cdot\!> \theta_2$ and $\theta_2 \cdot\!> \theta_1$ if the operators are left-associative, or make $\theta_1 <\!\cdot\; \theta_2$ and $\theta_2 <\!\cdot\; \theta_1$ if they are right-associative. For example, if $+$ and $-$ are left-associative, then make $+ \cdot\!> +$, $+ \cdot\!> -$, $- \cdot\!> -$, and $- \cdot\!> +$. If $\uparrow$ is right associative, then make $\uparrow <\!\cdot\; \uparrow$. These relations ensure that $E-E+E$ will have handle $E-E$ selected and $E \uparrow E \uparrow E$ will have the last $E \uparrow E$ selected.

3. Make $\theta <\!\cdot\; \textbf{id}$, $\textbf{id} \cdot\!> \theta$, $\theta <\!\cdot\; ($, $(<\!\cdot\; \theta$, $) \cdot\!> \theta$, $\theta \cdot\!>)$, $\theta \cdot\!> \$$, and $\$ <\!\cdot\; \theta$ for all operators θ. Also, let

$(\doteq)$	$\$ <\!\cdot\; ($	$\$ <\!\cdot\; \textbf{id}$
$(<\!\cdot\; ($	$\textbf{id} \cdot\!> \$$	$) \cdot\!> \$$
$(<\!\cdot\; \textbf{id}$	$\textbf{id} \cdot\!>)$	$) \cdot\!>)$

These rules ensure that both **id** and (E) will be reduced to E. Also, \$ serves as both the left and right endmarker, causing handles to be found between \$'s wherever possible.

Operator Precedence Parsing



1. If operator θ_1 has higher precedence than operator θ_2, make $\theta_1 \cdot\!> \theta_2$ and $\theta_2 <\!\cdot\, \theta_1$. For example, if $*$ has higher precedence than $+$, make $* \cdot\!> +$ and $+ <\!\cdot *$. These relations ensure that, in an expression of the form $E+E*E+E$, the central $E*E$ is the handle that will be reduced first.

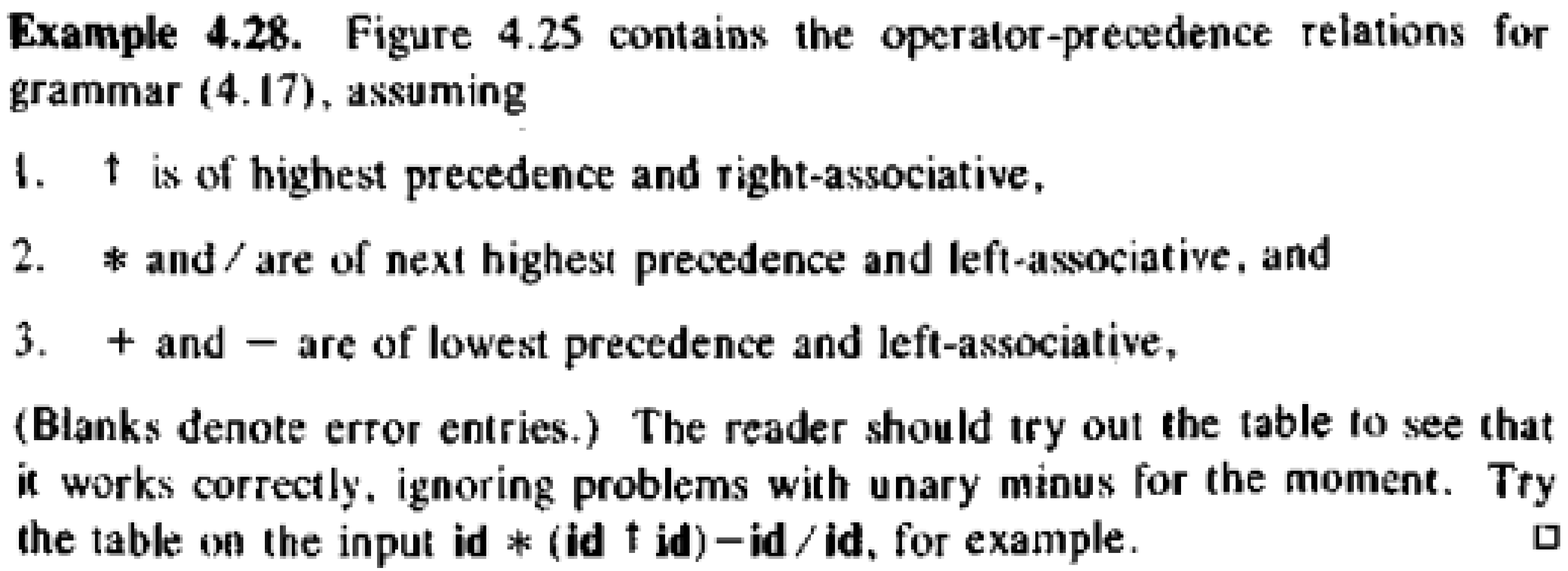
2. If θ_1 and θ_2 are operators of equal precedence (they may in fact be the same operator), then make $\theta_1 \cdot\!> \theta_2$ and $\theta_2 \cdot\!> \theta_1$ if the operators are left-associative, or make $\theta_1 <\!\cdot\, \theta_2$ and $\theta_2 <\!\cdot\, \theta_1$ if they are right-associative. For example, if $+$ and $-$ are left-associative, then make $+ \cdot\!> +$, $+ \cdot\!> -$, $- \cdot\!> -$, and $- \cdot\!> +$. If $\uparrow$ is right associative, then make $\uparrow <\!\cdot \uparrow$. These relations ensure that $E-E+E$ will have handle $E-E$ selected and $E \uparrow E \uparrow E$ will have the last $E \uparrow E$ selected.

3. Make $\theta <\!\cdot\, \mathbf{id}$, $\mathbf{id} \cdot\!> \theta$, $\theta <\!\cdot\, ($, $(<\!\cdot\, \theta$, $) \cdot\!> \theta$, $\theta \cdot\!>)$, $\theta \cdot\!> \$$, and $\$ <\!\cdot\, \theta$ for all operators θ. Also, let

$(\doteq)$	$\$ <\!\cdot\, ($	$\$ <\!\cdot\, \mathbf{id}$
$(<\!\cdot\, ($	$\mathbf{id} \cdot\!> \$$	$) \cdot\!> \$$
$(<\!\cdot\, \mathbf{id}$	$\mathbf{id} \cdot\!>)$	$) \cdot\!>)$

These rules ensure that both **id** and (E) will be reduced to E. Also, $\$$ serves as both the left and right endmarker, causing handles to be found between $\$$'s wherever possible.

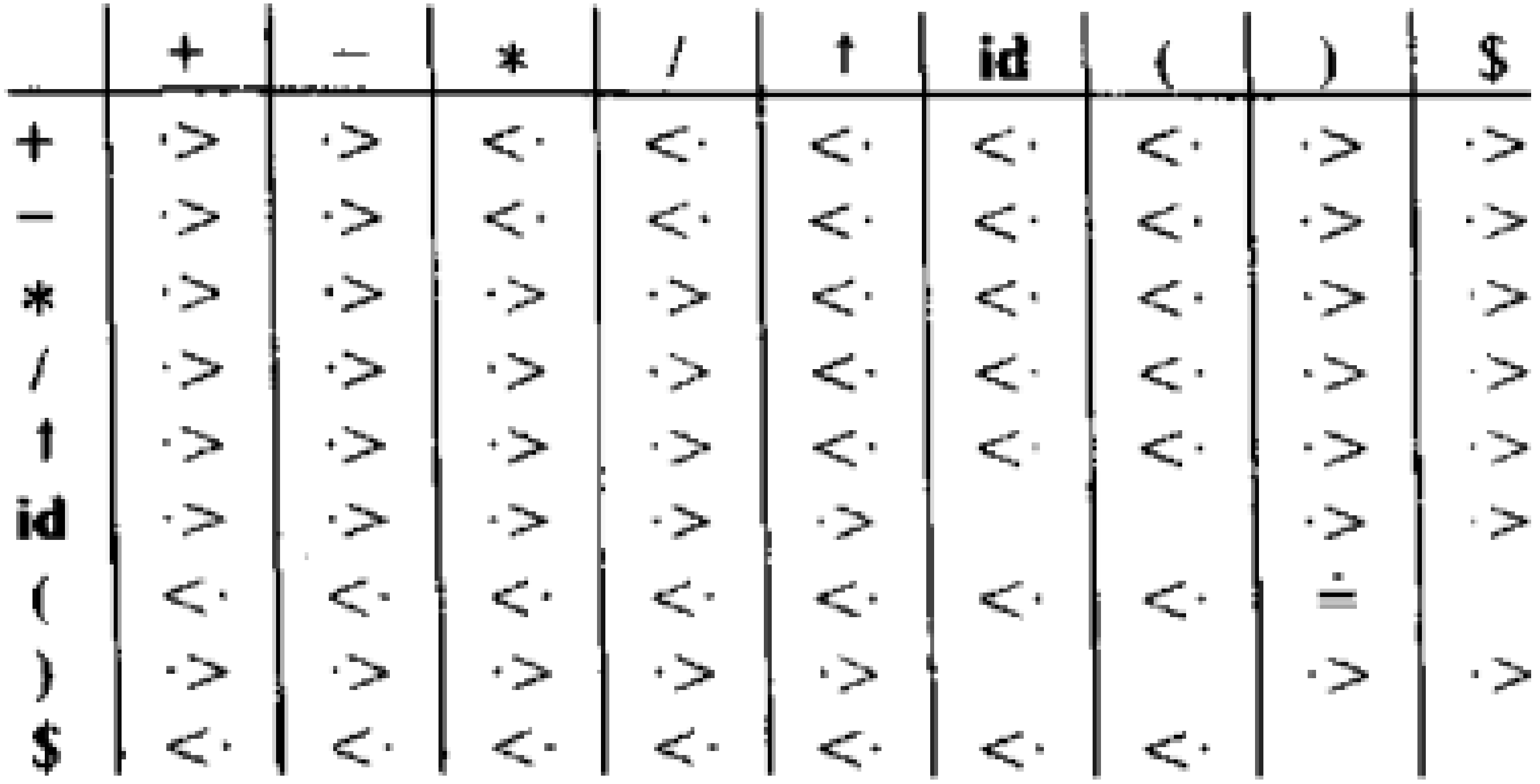
Operator Precedence Parsing


Example 4.28. Figure 4.25 contains the operator-precedence relations for grammar (4.17), assuming

1. ↑ is of highest precedence and right-associative,
2. * and / are of next highest precedence and left-associative, and
3. + and − are of lowest precedence and left-associative,

(Blanks denote error entries.) The reader should try out the table to see that it works correctly, ignoring problems with unary minus for the moment. Try the table on the input **id** * (**id** ↑ **id**) − **id** / **id**, for example. □

Operator Precedence Parsing



	+	−	*	/	↑	id	(	)	$
+	$\cdot>$	$\cdot>$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$\cdot>$	$\cdot>$
−	$\cdot>$	$\cdot>$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$\cdot>$	$\cdot>$
*****	$\cdot>$	$\cdot>$	$\cdot>$	$\cdot>$	$<\cdot$	$<\cdot$	$<\cdot$	$\cdot>$	$\cdot>$
/	$\cdot>$	$\cdot>$	$\cdot>$	$\cdot>$	$<\cdot$	$<\cdot$	$<\cdot$	$\cdot>$	$\cdot>$
↑	$\cdot>$	$\cdot>$	$\cdot>$	$\cdot>$	$<\cdot$	$<\cdot$	$<\cdot$	$\cdot>$	$\cdot>$
id	$\cdot>$	$\cdot>$	$\cdot>$	$\cdot>$	$\cdot>$			$\cdot>$	$\cdot>$
(	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$\doteq$	
)	$\cdot>$	$\cdot>$	$\cdot>$	$\cdot>$	$\cdot>$			$\cdot>$	$\cdot>$
$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$		

Fig. 4.25. Operator-precedence relations.

Thank You