OPERATOR PRECEDENCE PARSING

CSE 340 FALL 2021

Rida Bazzi

Notes based on the "Dragon Book"

Parsing Operator Grammars

The grammar we have see for expressions does not include the operator minus ('-').

This is not an oversight!

We can write the following grammar

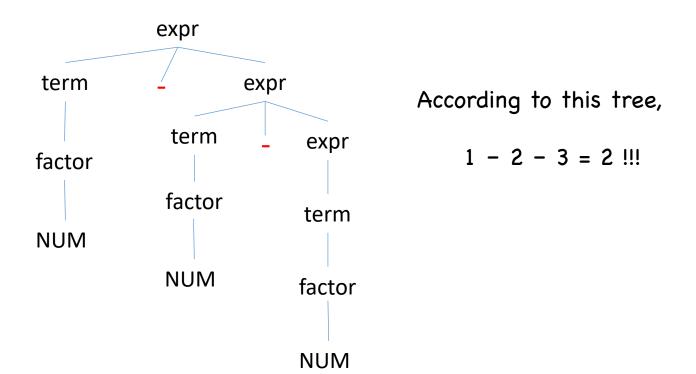
Expr -> term - Expr Expr -> term + Expr Expr -> term

but that would not work!

How do we parse the following?

$$1 - 2 - 3$$

According to the grammar above, we get



Parsing expressions with minus

The issue is that minus is left associative and the grammar treats minus as right-associative

Left associative grouping (correct)

$$1 - 2 - 3$$

 $(1 - 2) - 3$
 $((1-2) - 3)$

Right associative grouping (wrong)

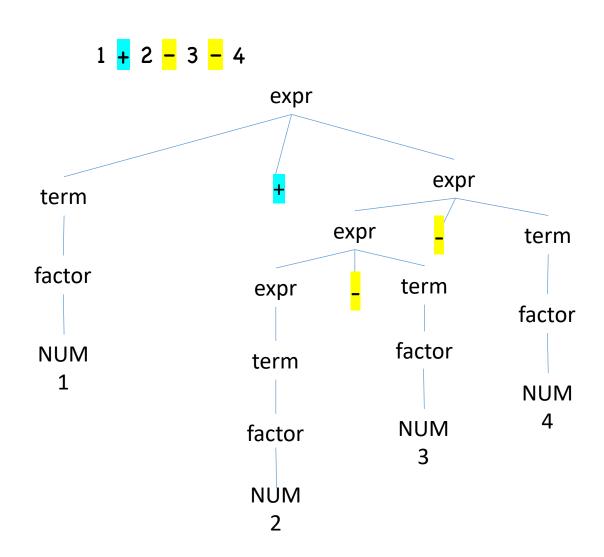
$$1 - 2 - 3$$

 $1 - (2 - 3)$
 $(1 - (2 - 3))$

Parsing expressions with minus

We can attempt to fix the problem by using the following grammar

This grammar would give the following parsing for



Parsing expressions with minus

We can attempt to fix the problem by using the following grammar

```
expr -> expr - term
expr -> term + expr
expr -> term
```

We cannot parse this grammar with a recursive descent parser!

```
parse_expr()
{
      // expr -> expr - term
      ....
      parse_expr() // infinite loop !!
```

We need another way to parse such expressions!

A NEAT TRICK FROM FORTRAN COMPILER!

```
a + b * c - d
  add ( ( at the beginning
  replace every + with ))) + (((
  replace every - with ))) - (((
  replace every * with )) * ((
  replace every ^ with ) ^ (
  add ) ) ) at the end
We get
(((a)))+(((b))*((c)))-(((d)))
(((a)))+(((b))*((c)))-(((d)))
```

Always works!

We can then parse with a simple parser that only has to worry about matching parentheses

Operator Grammar

A grammar is called an operator grammar if

- 1. there is no righthand side of a rule which has two adjacent non-terminal
- 2. there is no rule of the form A -> ϵ

is not an operator grammar because of E A E has three adjacent non-terminals

Example 2
$$E \rightarrow E + E | E - E | E * E | E / E$$

 $| E ^ E | (E) | - E | ID$

is an operator grammar

OPERATOR PRECEDENCE RELATIONSHIPS

To parse operator grammar, we first define parsing precedence relationships between the terminals of the grammar

We also introduce a new symbol \$

parsing precedence relationships

- ∀ yields precedence to
- > takes precedence over
- ≐ has the same precedence as

These are not the same as the operator precedence levels.

These are used in guiding the parsing

You can think of $\langle \cdot \rangle$ as matching parentheses that group what appears between them (this should become clearer with the examples)

There is a theory to determine these relationship for an unambiguous operator grammar. We will only look at heuristics for common expressions.

The parsing algorithm assumes that we already have a table that defines these relationships

		ı		•			•	•	-
+	•	·	•	,	,	•	•	·	·
-	•	·>	`	`	`	`	`	•	· ·
	•	·>	•	•	•	,	,	•	•
/	·	·>	·	·	·	·	·	·	·
^	•	·>	·	·	,	•	•	·	·
id		·>							·>
(•	<∙	•	·	·	•			
)		·>						·>	·>
\$	<.	<.	<.	<.	<.	<.	<.		

Parsing Algorithm

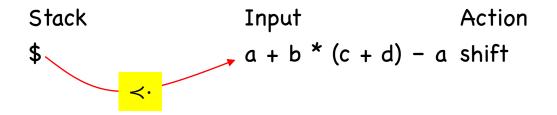
```
Input
            w $
Output
            parse tree with E in all internal nodes
Initially
            stack contains $, scanning starts at the start of w
repeat
            if $ is on top of the stack and lexer.peek() = $ // EOF
                         return;
            else
            {
                         t = lexer.peek(); b = t.type;
                                                                    // next token from w
                                                                     // terminal at the top of stack
                         a = stack.terminalpeek().type;
                                                                     // or just below if top is non-terminal
                         if (table[a][b] == \checkmark \checkmark) | ( table[a][b] = \checkmark \doteq \checkmark)
                                                                            // shift
                                      t = lexer.getToken();
                                      stack.push(t)
                         else if (table[a][b] == \cdot \cdot >')
                                                                            // reduce
                                      RHS = an empty stack
                                      repeat
                                                  s = stack.pop()
                                                                            // pop terminals and
                                                                            // non-terminals
                                                  if s is a terminal
                                                       last_popped_term = s
                                                  RHS.push(s)
                                      until ( ( is_a_terminal(stack.peek() ) and
                                             ( table[stack.terminalpeek()][last_popped_term] == `<' ))</pre>
                                      if E -> RHS rule exists// RHS calculated above
                                                   reduce E -> RHS
                                                                            // we can think of E as the
                                                   push E
                                                                            // root of subtree for E -> RHS
                                      else
                                                  syntax_error()l
                         }
                         else
                                      syntax_error();
            }
```

Note:

- stack.peek() peeks at the symbol at the top of the stack, which could be a terminal or a non-terminal.
- stack.terminalpeek() peeks at the terminal closest to the top of the stack.

		ı		•			•	•	-
+	•	·	•	,	,	•	•	·	·
-	•	·>	`	`	`	`	`	•	· ·
	•	·>	•	•	•	,	,	•	•
/	·	·>	·	·	·	·	·	·	·
^	•	·>	·	·	,	•	•	·	·
id		·>							·>
(•	<∙	•	·	·	•			
)		·>						·>	·>
\$	<.	<.	<.	<.	<.	<.	<.		

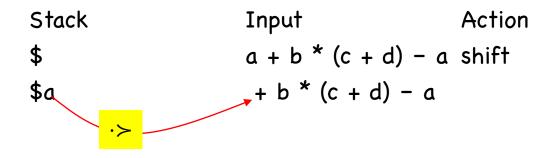
			Input										
		+	-	*	/	٨	id	()	\$			
	+	·>	·≻	<.	<.	∀ ·	∀ ·	÷	Ý	·>			
	-	·>	·>	≺·	<.	<.	≺·	<.		·>			
	*	·>	·>	·>	·>	<∙	≺·	≺·	·>	·>			
쏭	1	·>	·>	·>	·>	<.	<.	<.	·>	·>			
stack	٨	·>	·>	·>	·>	<∙	<∙	<∙	·>	·>			
	id	·>	·>	·>	·>	·>			·>	·>			
	(<.	<∙	<∙	<.	<∙	<.	<∙	≐				
)	·>	·>	·>	·>	·≻				·>			
	\$	<.	≺·	<.	<.	<.	<.	<∙					



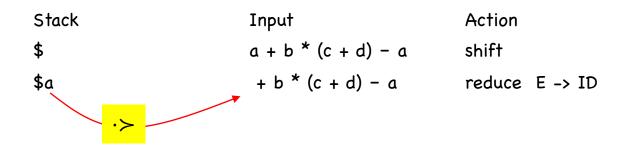
	+	-	*	/	۸	id	()	\$
+	·≻	·≻	∀	⊹	<.	<.	<.	·>	·>
-	·>	·>	<∙	<∙	<∙	<∙	<∙	·>	·>
*	·>	·>	·>	·>	<.	<.	<.	·>	·>
/	·≻	·≻	·≻	·≻	≺·	≺·	≺·	·>	·>
^	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
id	·≻	·>	·≻	·≻	·>			·>	·>
(≺·	<.	<.	<.	≺·	≺·	≺·	≐	
)	·>	·≻	·>	·>	·>			·>	·>
\$	<∙	<∙	<∙	<∙	<∙	<.	<∙		

Stack Input Action a + b * (c + d) - a shift b * (c + d) - a

	+	ı	*	/	٨	id	()	\$
+	·>	·≻	<.	∀	⊹	<.	<.	·>	·>
-	·>	·>	<∙	<.	<.	<∙	<∙	· >	·>
*	·>	·>	·>	·≻	<∙	<∙	<∙	·>	·>
/	·>	·>	·>	·>	<∙	<∙	<∙	·>	·>
٨	·>	·>	·>	·>	<.	<∙	<∙	·>	·>
id		·>	·>					·>	·>
(<.	<.	<∙	<.	<.	<∙	<∙	≐	
)	$\cdot \succ$	·>	·>					·>	·>
\$	<.	<.	<∙	<.	<.	<∙	<.		

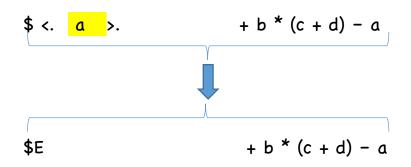


	+	ı	*	/	٨	id	()	\$
+	·>	·≻	<.	∀	⊹	<.	<.	·>	·>
-	·>	·>	<∙	<.	<.	<∙	<∙	· >	·>
*	·>	·>	·>	·≻	<∙	<∙	<∙	·>	·>
/	·>	·>	·>	·>	<∙	<∙	<∙	·>	·>
٨	·>	·>	·>	·>	<∙	<∙	<.	·>	·>
id	·	·>	·>					·>	·>
(<.	<.	<∙	<.	<.	<∙	<∙	≐	
)	$\cdot \succ$	·>	·>					·>	·>
\$	<.	<.	<∙	<.	<.	<∙	<.		



	+	-	*	/	٨	id	()	\$
+	Ý	Ý	∀ ·	÷	÷	÷	÷	·>	·>
-			<.	<.	<.	<.	<.	·>	·>
*	·>	·≻	·>	·>	<.	≺·	≺·	·>	·>
/	·≻	·≻	·>	·≻	<.	≺·	≺·	·>	·>
^	·>	·≻	·>	·≻	<.	≺·	≺·	·>	·>
id	÷	·≻	·>	·≻	·>			·>	·>
(≺·	<.	≺·	<.	<.	≺·	≺·	≐	
)		·≻	·	·	·			·>	·>
\$	<.	<∙	<∙	<.	<.	<∙	<∙		

Stack Input Action a + b * (c + d) - a shift b * (c + d) - a reduce b = -b ID



	+	ı	*	/	٨	id	()	\$
+	·	÷	∀ ·	÷	÷	÷	÷	·>	·>
-	·>	·>	<∙	<∙	<∙	<∙	<∙	·>	·>
*	·>	·>	·>	·>	<∙	≺·	<.	·>	·>
/	·>	·≻	·≻	·>	<.	≺·	≺·	·>	·>
^	·>	·>	·>	·>	<.	≺·	≺·	·>	·>
id	·>	·≻	·≻	·>	·≻			·>	·>
(≺·	<.	<.	≺·	<.	≺·	≺·	≐	
)	·>	·≻	·	·>				·>	·>
\$	≺·	<.	<∙	<∙	<.	<∙	<.		

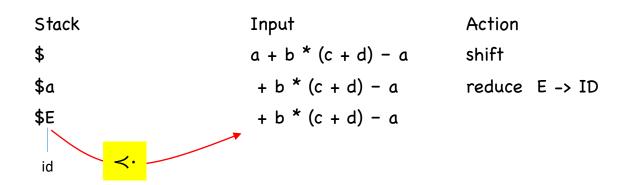
Stack \$ \$a \$E id Input Action

a + b * (c + d) - a shift

+ b * (c + d) - a reduce E -> ID

+ b * (c + d) - a

	+	•	*	/	۸	id	()	\$
+	Ý	·≻	∀ ·	÷	÷	÷	√·	÷	·>
-	·>	·>	<.	<∙	<.	<.	<∙	·>	·>
*	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
/	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
٨	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
id	·>	·>	·>	·>	·>			·>	·>
(≺·	<.	≺·	≺·	≺·	≺·	≺·	≐	
)	·>	·>	·≻	·>	·>			·>	·>
\$	<∙	<∙	<∙	<∙	<∙	<∙	<∙		



	+	-	*	/	۸	id	()	\$
+	·>	·	<.	<.	<.	<.	<.	·>	·>
-	·>	·>	<∙	<∙	<.	<.	<.	·>	·>
*	·>	·>	·>	·>	≺·	≺·	<.	·>	·>
1	·>	·>	·>	·>	≺·	≺·	<.	·>	·>
٨	·>	·>	·>	·>	≺·	≺·	<.	·>	·>
id	·>	·≻	·>	·>	·>			·>	·>
(<.	≺·	≺·	≺·	≺·	≺·	<.	÷	
)	·>	·>	·>	·>	·>			·>	·>
\$	<.	<∙	<∙	<∙	<∙	<∙	<∙		

Stack Input Action $a + b * (c + d) - a \qquad shift$ $b * (c + d) - a \qquad reduce E -> ID$ $b * (c + d) - a \qquad shift$

	+	-	*	/	۸	id	()	\$
+	·>	·≻	<.	<.	<.	<.	≺·	·>	•>
-			<.	<.	<.	<.	<∙	·>	·>
*	·>	·≻	·>	·>	≺·	≺·	≺·	·>	·>
/	·≻	·≻	·>	·≻	≺·	≺·	≺·	·>	·>
^	·>	·≻	·>	·>	≺·	≺·	≺·	·>	·>
id	·≻	·≻	·≻	·>	·>			·>	·>
(≺·	<.	≺·	≺·	≺·	≺·	≺·	≐	
)	·≻	·≻	·	·≻	·>			·>	·>
\$	<.	<∙	<∙	<∙	<∙	<∙	<∙		

Stack
\$
\$a
\$E
\$id

Input Action a + b * (c + d) - a shift + b * (c + d) - a reduce E -> ID + b * (c + d) - a shift b * (c + d) - a

	+	ı	*	/	٨	id	()	\$
+	·≻	·	<.	∀	⊹	<.	<.	·>	·>
-	·>	·>	<∙	<.	<.	<∙	<∙	· >	·>
*	·>	·≻	·>	·≻	<∙	<∙	<∙	·>	·>
/	·>	·>	·>	·>	<∙	<∙	<∙	·>	·>
٨	·>	·>	·>	·>	<∙	<∙	<∙	·>	·>
id			·>					·>	·>
(<∙	<∙	<∙	<∙	<.	<∙	<∙	≐	
)	$\cdot \succ$		·>					·>	·>
\$	<.	<.	<∙	<.	<.	<∙	<.		

Stack

Input a + b * (c + d) - ashift b * (c + d) - areduce $E \rightarrow ID$ b * (c + d) - ashift b * (c + d) - a b * (c + d) - a

	+	-	*	/	۸	id	()	\$
+	÷	·	<.	<.	<.	<.	<.	·>	·>
-	·>	·>	<∙	<∙	<∙	∀ ·	<∙	·>	·>
*	·>	·>	·>	·>	<.	<.	≺·	·>	·>
/	·≻	·≻	·≻	·>	≺·	≺·	<∙	·>	·>
^	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
id	·≻	·>	·≻	·>	·>			·>	·>
(≺·	<.	≺·	≺·	≺·	≺·	≺·	≐	
)	·>	·≻	·	·>	·>			·>	·>
\$	<∙	<∙	<∙	<.	<∙	<∙	<∙		

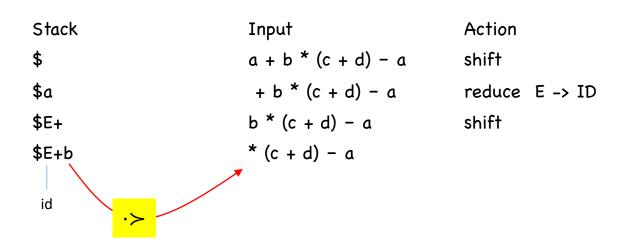
Input Stack Action a + b * (c + d) - a\$ shift \$a + b * (c + d) - a reduce E -> ID + b * (c + d) - a\$E shift \$E+ b * (c + d) - a shift id

	+	-	*	/	٨	id	()	\$
+	·≻	·≻	∀	⊹	∀	<.	<.	·>	·>
-	·>	·>	<.	<.	<.	√·	<∙	·>	·>
*	·>	·>			<.	<∙	<∙		·>
/	·>	·>	·>	·>	<∙	<∙	<∙	·>	·>
٨	·>	·>	·>	·>	<.	<∙	<∙	·>	·>
id	·≻	·>						·>	·>
(<.	<.	<∙	<.	<.	<∙	<∙	≐	
)	·>	·≻	·≻	·≻	·≻			·>	·>
\$	<.	<.	<.	<.	<.	<.	<.		

Stack \$ \$a \$E+ \$E+b

Input Action a + b * (c + d) - a shift + b * (c + d) - a reduce E -> ID b * (c + d) - a shift * (c + d) - a

	+	-	*	/	۸	id	()	\$
+	Ý	Ý	∀ ·	÷	÷	÷	÷	·>	·>
-		·>	<.	<∙	<.	<.	<.	·>	·>
*	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
/	·≻	·≻	·>	·>	≺·	≺·	≺·	·>	·>
^	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
id	·≻	·≻	·>	·>	·>			·>	·>
(≺·	≺·	≺·	≺·	≺·	≺·	≺·	≐	
)	·≻	·>	·	·>	·>			·>	·>
\$	<∙	<∙	<∙	<∙	<∙	<∙	<∙		



	+	1	*	/	<	id	()	\$
+	·≻	Ý	∀ ·	÷	÷	√·	√·	·>	·>
-	·>	·>	<.	<∙	<.	<∙	<∙	·>	·>
*	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
/	·>	·>	·>	·>	≺·	<.	<∙	·>	·>
^	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
id	·>	·>	·	·>	·>			·>	·>
(≺·	≺·	<.	≺·	≺·	≺·	≺·	≐	
)	·>	·>	·>	·>	·>			·>	·>
\$	<∙	<∙	<∙	<∙	<∙	<∙	<∙		

Input Stack Action a + b * (c + d) - a\$ shift \$a + b * (c + d) - a reduce E -> ID b * (c + d) - a\$E+ shift * (c + d) - a \$E+b reduce E -> ID id

	+	1	*	/	<	id	()	\$
+	·≻	Ý	∀ ·	÷	÷	√·	√·	·>	·>
-	·>	·>	<.	<∙	<.	<∙	<∙	·>	·>
*	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
/	·>	·>	·>	·>	≺·	≺·	<∙	·>	·>
^	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
id	·>	·>	·	·>	·>			·>	·>
(≺·	≺·	<.	≺·	≺·	≺·	≺·	≐	
)	·>	·>	·>	·>	·>			·>	·>
\$	<∙	<∙	<∙	<∙	<∙	<∙	<∙		

Stack

\$

\$a

\$E+

\$E+b

\$E+E

id id

Input

a + b * (c + d) - a

+ b * (c + d) - a reduce E -> ID

b * (c + d) - a

* (c + d) - a

*(c + d) - a

Action

shift

shift

reduce E -> ID

	+	-	*	/	۸	id	()	\$
+	Ý	Ý	<.	<.	<.	<.	<.	·>	·>
-	·>	·>	<.	<.	≺·	≺·	<.	·>	·>
*	·>	·>	·>	·>	≺·	≺·	<.	·>	·>
/	·>	·>	·>	·>	<∙	<∙	<.	·>	·>
۸	·>	·>	·>	·>	<∙	<∙	<.	·>	·>
id	·>	·>	·>	·>	·>			·>	·>
(<∙	<∙	<∙	<∙	<∙	<∙	<.	÷	
)	·>	·>	·>	·>	·>			·>	·>
\$	<∙	<∙	<∙	<∙	<∙	<∙	<∙		

Action Stack Input a + b * (c + d) - a\$ shift + b * (c + d) - a \$a reduce E -> ID \$E+ b * (c + d) - ashift * (c + d) - a\$E+b reduce E -> ID * (c + d) - a\$E+E id id

	+	1	*	/	<	id	()	\$
+	·	÷	÷	÷	÷	÷	÷	·>	·>
-	·>	·>	√·	<∙	<∙	<∙	<∙	·>	·>
*	·>	·>	·>	·>	<.	<.	≺·	·>	·>
/	·>	·≻	·>	·>	≺·	≺·	≺·	·>	·>
^	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
id	·>	·≻	·>	·>	·>			·>	·>
(≺·	<.	<∙	<.	≺·	≺·	≺·	≐	
)	·>	·>	·>	·>	·>			·>	·>
\$	≺·	<.	<.	<∙	<∙	<.	≺·		

Stack

Input

\$ a + b * (c + d) - a

\$ a + b * (c + d) - a

\$ E+ b * (c + d) - a

\$ E+b * (c + d) - a

\$ E+E * (c + d) - a

\$ (c + d) - a

\$ (c + d) - a

\$ (c + d) - a

Action
d) - a shift
d) - a reduce E -> ID
a shift
reduce E -> ID
shift

	+	1	*	/	۸	id	()	\$
+	Ý	Ý	Ÿ	÷	÷	÷	÷	·>	·>
-	·>	·>	÷	<∙	<.	<.	<.	·>	·>
*	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
/	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
^	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
id	·>	·>	·>	·>	·>			·>	·>
(≺·	≺·	≺·	≺·	≺·	≺·	≺·	≐	
)	·>	·>	·>	·>	·>			·>	·>
\$	<.	<.	<∙	<.	<.	<.	<.		

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

* (c + d) - a

*(c + d) - a

(c + d) - a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

	+	-	*	/	^	id	()	\$
+	Ÿ	÷	<.	<.	<.	<.	<.	·>	·>
-	·>	·>	<.	<.	<.	<.	<.	·>	·>
*	·>	·>	·>	·>	<.	<.	<.	·>	·>
/	·>	·>	·>	·>	<∙	<.	<.	·>	·>
٨	·>	·>	·>	·>	<.	<.	<.	·>	·>
id	·≻	·>	·>	·>	·>			·>	·>
(<∙	<∙	<∙	<.	<.	<.	<.	÷	
)	·≻	·>	·>	·>	·>			·>	·>
\$	<.	<∙	≺·	<∙	<.	<.	<.		

Stack Input Action a + b * (c + d) - a\$ shift + b * (c + d) - a\$a reduce E -> ID \$E+ b * (c + d) - ashift *(c + d) - a\$E+b reduce E -> ID * (c + d) - a\$E+E shift (c + d) - a\$E+E* id id

	+	1	*	/	<	id	()	\$
+	Ý	Ý	∀ ·	÷	÷	√·	√·	·>	·>
-	·>	·>	<∙	<∙	<∙	<∙	<∙	·>	·>
*	·>	·>	·>	·>	<.	≺·	<∙	·>	·>
/	·≻	·≻	·>	·>	≺·	≺·	≺·	·>	·>
^	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
id	·≻	·≻	·>	·>	·>			·>	·>
(≺·	<.	<.	≺·	≺·	≺·	≺·	≐	
)	·≻	·≻	·	·>	·>			·>	·>
\$	<.	<.	<.	<∙	<.	<∙	≺·		

Stack Input Action a + b * (c + d) - a\$ shift + b * (c + d) - a\$a reduce E -> ID \$E+ b * (c + d) - ashift *(c + d) - a\$E+b reduce E -> ID * (c + d) - a\$E+E shift (c + d) - a\$E+E* shift id id

	+	-	*	/	<	id	()	\$
+	·	·>	∀ ·	√ ·	÷	÷	÷	·>	·>
-	·>	·>	<∙	<.	<∙	<∙	<∙	·>	·>
*	·≻	·>	·>	·>	<∙	<∙	<∙	·>	·>
/	·>	·>	·>	·>	<∙	<∙	≺·	·>	·>
^	·>	·>	·>	·>	<∙	≺·	≺·	·>	·>
id	·>	·>	·>	·>	·>			·>	·>
(≺·	<.	≺·	<.	<.	≺·	≺·	≐	
)	·>	·>	·>	·>	·>			·>	·>
\$	<∙	<∙	<∙	<∙	<∙	<∙	<∙		

Stack Input Action a + b * (c + d) - a\$ shift + b * (c + d) - a\$a reduce E -> ID b * (c + d) - a\$E+ shift *(c + d) - a\$E+b reduce E -> ID \$E+E *(c + d) - ashift (c + d) - a\$E+E* shift id id

	+	1	*	/	<	id	()	\$
+	·≻	Ý	∀ ·	÷	÷	÷	√·	·>	·>
-	·>	·>	<.	<∙	<.	<.	<∙	·>	·>
*	·>	·>	·>	·>	≺·	≺·	<.	·>	·>
/	·>	·≻	·>	·>	≺·	≺·	≺·	·>	·>
^	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
id	·>	·≻	·>	·>	·>			·>	·>
(≺·	≺·	≺·	≺·	≺·	≺·	≺·	≐	
)	·>	·≻	·≻	·>	·>			·>	·>
\$	<.	<.	<.	<∙	<∙	<∙	<∙		

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

	+	-	*	/	٨	id	()	\$
+	·>	·>	<.	<.	<.	≺.	<.	·>	·>
-	·>	·>	≺·	<∙	<.	≺·	<.	·>	·>
*	·>	·>	·>	·>	<.	<.	<.	·>	·>
/	·>	·≻	·>	·>	<.	≺·	<.	·>	·>
۸	·>	·>	·>	·>	<.	≺·	<.	·>	·>
id	·>	·>	·>	·>	·≻			·>	·>
(≺·	≺·	≺·	≺·	<∙	≺٠	<.	≐	
)	·>	·>	·>	·>	·≻			·>	·>
\$	<∙	<∙	<∙	<∙	<∙	<∙	<∙		

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

* (c + d) - a

* (c + d) - a

(c + d) - a

c + d) - a

Action
shift
reduce E -> ID
shift
reduce E -> ID
shift
shift

	+	ı	*	/	٨	id	()	\$
+	Ý	Ý	∀ ·	÷	÷	÷	÷	·>	·>
-	·>	·>	<∙	<∙	<∙	<∙	<∙	·>	·>
*				·>	<.	<.	<.	·>	·>
/	·>	·>	·>	·>	<.	<.	<.	·>	·>
^	·>	·>	·>	·>	<.	≺·	≺·	·>	·>
id	·≻	·≻	·>	·>	·			·>	·>
(<.	<.	≺·	≺·	≺·	≺·	≺·	≐	
)	·>	·>	·	·>				·>	·>
\$	<.	<.	<∙	<.	<.	≺·	<.		

\$ tack
\$ \$ a
\$ tack
\$ t

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

* (c + d) - a

* (c + d) - a

(c + d) - a

c + d) - a

Action
shift
reduce E -> ID
shift
reduce E -> ID
shift
shift
shift
shift

	+	-	*	/	۸	id	()	\$
+	·>	·	<.	<.	<.	<.	<.	·>	·>
-	·>	·>	<.	<∙	<.	<.	<.	·>	·>
*	·>	·>	·>	·>	≺·	≺·	<.	·>	·>
/	·>	·>	·≻	·>	≺·	≺·	<.	·>	·>
^	·>	·>	·>	·>	≺·	≺·	<.	·>	·>
id	·>	·>	·>	·>	·>			·>	·>
(≺·	≺·	≺·	≺·	≺·	≺·	<.	÷	
)	·>	·>	·	·>	·>			·>	·>
\$	<∙	<∙	<∙	<∙	<∙	<∙	<.		

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+ d) - a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

	+	-	*	/	٨	id	()	\$
+	Ÿ	Ÿ	∀ ·	÷	÷	∀ ·	∀ ·	·>	·>
-	·≻	·>	<.	≺·	<.	<.	<.	·>	·>
*	·≻	·>	·>	·>	≺·	≺·	≺·	·>	·>
/	·≻	·≻	·>	·>	≺·	<.	<.	·>	·>
^	·≻	·>	·>	·>	≺·	<∙	<∙	·>	·>
id	·≻	·>	·>	·>	·>			·>	·>
(<∙	<∙	<∙	<.	<∙	<∙	<∙	÷	
)	·>	·>	·>	·>	·>			·>	·>
\$	<.	<.	<.	<.	<.	<.	<.		

Stack
\$
\$a
\$E+
\$E+b
\$E+E
\$E+E*
\$E+E*(
\$E+E*(c

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

* (c + d) - a

* (c + d) - a

(c + d) - a

c + d) - a

+ d) - a

Action
shift
reduce E -> ID
shift
reduce E -> ID
shift
shift
shift
shift

	+	-	*	/	<	id	()	\$
+	·>	·>	√ ·	√ ·	÷	<.	<.	·>	·>
-	·>	·>	<∙	<.	<∙	<.	<.	·>	·>
*	·>	·>	·>	·>	≺·	<.	<.	·>	·>
1	·>	·>	· >	·>	≺·	<.	<.	·>	·>
٨	·>	·>	·>	·>	≺·	<.	<.	·>	·>
id	·>	·>	·>	·>	·>			·>	·>
(<.	<.	≺·	<.	≺·	<.	<.	≐	
)	·>	·>	·>	·>	·>			·>	·>
\$	<∙	<∙	<∙	<∙	<∙	<.	<.		

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

* (c + d) - a

* (c + d) - a

(c + d) - a

c + d) - a

+ d) - a

Action
shift
reduce E -> ID
shift
reduce E -> ID
shift
shift
shift
shift
reduce E -> ID

	+	•	*	/	۸	id	()	\$
+	·>	·≻	<.	<.	<.	<.	<.	·>	·>
-	·>	·>	<∙	<∙	<∙	<∙	<∙	·>	·>
*	·>	·>	·>	·>	<∙	<∙	<.	·>	·>
/	·>	·≻	·>	·>	≺·	≺·	≺·	·>	·>
^	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
id	·≻	·≻	·≻	·>	·>			·>	·>
(≺·	≺·	≺·	≺·	≺·	≺·	≺·	≐	
)	·>	·≻	·	·>	·>			·>	·>
\$	≺·	<.	<∙	≺·	<.	≺·	<.		

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

id id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+ d) - a

+ d) - a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

	+	-	*	/	٨	id	()	\$
+	Ý	Ý	∀ ·	÷	÷	÷	<.	·>	·>
-	·>	·>	<∙	<.	<.	<∙	<.	·>	·>
*	·≻	·>	·>	·>	≺·	<.	<.	·>	·>
/	·≻	·≻	·≻	·≻	<.	≺·	<.	·>	·>
^	·≻	·>	·>	·>	≺·	≺·	<.	·>	·>
id	·≻	·≻	·≻	·≻	·≻			·>	·>
(<.	<.	≺·	≺·	≺·	≺·	<.	≐	
)	·≻	·≻	·≻	·≻	·≻			·>	·>
\$	<∙	<∙	<∙	<∙	<∙	<∙	<∙		

\$\frack
\$
\$a
\$E+
\$E+b
\$E+E
\$E+E*
\$E+E*
\$E+E*(
\$E+E*(c
\$E+E*(E
|
id id id id

Input Action a + b * (c + d) - ashift + b * (c + d) - areduce E -> ID b * (c + d) - ashift *(c + d) - areduce E -> ID *(c + d) - ashift (c + d) - ashift c + d) - ashift + d) - areduce E -> ID + d) - a

	+	-	*	/	٨	id	()	\$
+	·>	·≻	<.	∀	<.	<.	<.	·>	·>
-	·>	·>	<∙	<.	<∙	<∙	<.	·>	·>
*	·≻	·>	·>	·≻	<∙	<∙	<.	·>	·>
/	·>	·>	·>	·>	<.	<.	<.	·>	·>
٨	·>	·>	·>	·>	≺·	≺·	<.	·>	·>
id	·>	·≻	·>	·≻	·>			·>	·>
(≺·	≺·	≺·	≺·	≺·	≺·	<.	≐	
)	·>	·≻	·>	·≻	·>			·>	·>
\$	≺·	<.	<.	<.	<.	≺·	<.		

\$tack \$ \$a \$E+ \$E+b \$E+E* \$E+E* \$E+E*(\$E+E*(c \$E+E*(E | | id id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

* (c + d) - a

* (c + d) - a

(c + d) - a

c + d) - a

+ d) - a

Action
shift
reduce E -> ID
shift
reduce E -> ID
shift
shift
shift
reduce E -> ID
shift

	+	1	*	/	<	id	()	\$
+	·≻	Ý	∀ ·	÷	÷	÷	√·	·>	·>
-	·>	·>	<∙	<∙	<∙	<∙	<∙	·>	·>
*	·>	·>	·>	·>	<∙	<.	≺·	·>	·>
/	·>	·≻	·>	·>	≺·	≺·	≺·	·>	·>
^	·>	·>	·>		≺·	≺·	≺·	·>	·>
id	·>	·≻	·	·>	·>			·>	·>
(∀ ·	≺·	≺·	≺·	≺·	≺·	≺·	≐	
)	·>	·≻	·	·>	·>			·>	·>
\$	≺·	<.	<.	<∙	<.	<∙	<∙		

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

id id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+ d) - a

+ d) - a

d) - a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

	+	-	*	/	۸	id	()	\$
+	Ý	÷	∀ ·	÷	÷	÷	√·	·>	·>
-		·>	<.	<.	<.	<.	<∙	·>	·>
*	·>	·>	·>	·>	<.	≺·	≺·	·>	·>
/	·≻	·≻	·>	·>	<.	≺·	≺·	·>	·>
^	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
id	·≻	·≻	·>	·>	·≻			·>	·>
(≺·	<.	≺·	≺·	≺·	≺·	≺·	≐	
)	·≻	·≻	·>	·>	·≻			·>	·>
\$	<∙	<∙	<∙	<∙	<∙	<∙	<∙		

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

id id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+ d) - a

+ d) - a

d) - a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

	+	•	*	/	۸	id	()	\$
+	Ý	Ÿ	÷	÷	÷	÷	<.	·>	·>
-	·>	·>	<.	≺·	<.	≺·	<.	·>	·>
*	·>	·>	·>	·>	≺·	≺·	<.	·>	·>
/	·>	·>	·>	·>	<.	<.	<.	·>	·>
٨	·>	·>	·>	·>	≺·	≺·	<.	·>	·>
id	·>	·≻	·>	·>	·>			·>	· >
(≺·	<.	≺·	≺·	≺·	≺·	<.	≐	
)	·>	·≻	·>	·>	·≻			·>	·>
\$	<∙	<∙	<∙	<∙	<∙	<∙	<.		

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

id id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+ d) - a

+ d) - a

d) - a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

	+	-	*	/	^	id	()	\$
+	·>	·	<.	<.	<.	<.	<.	·>	·>
-	·>	·>	<.	<.	<.	<.	<.	·>	·>
*	·>	·>	·>	·>	<.	≺·	<.	·>	·>
/	·>	·>	·>	·>	<.	<.	<.	·>	·>
٨	·>	·>	·>	·>	<∙	<∙	<.	·>	·>
id	·>	·>	·>	·>	·>			·>	·>
(<.	<∙	<∙	<.	<.	<∙	<.	÷	
)	·>	·≻	·>	·>	·>			·>	·>
\$	<∙	<∙	<∙	<∙	<∙	<∙	<.		

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

\$E+E*(E+d

id id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+d) - a

+ d) - a

d) - a

) - a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

	+	-	*	/	۸	id	()	\$
+	Ý	÷	∀ ·	÷	÷	÷	√·	·>	·>
-	·>	·>	<.	<.	<.	≺·	≺·	·>	·>
*	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
/	·≻	·≻	·>	·≻	≺·	≺·	≺·	·>	·>
^	·>	·>	·>	·>	≺·	≺·	≺٠	·>	·>
id	·>	·>	·>	·>	·>			·>	·>
(<∙	<.	<∙	≺·	≺·	≺·	≺٠	≐	
)	·>	·>	·>	·>	·>			·>	·>
\$	<.	≺·	≺·	<.	≺·	<.	≺·		

Stack \$ \$a \$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c \$E+E*(E

\$E+E*(E+

\$E+E*(E+d

id id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+d) - a

+ d) - a

d) - a

) - a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

	+	-	*	/	۸	id	()	\$
+	Ý	÷	∀ ·	÷	÷	÷	√·	·>	·>
-	·>	·>	<.	<.	<.	≺·	≺·	·>	·>
*	·>	·>	·>	·>	≺·	≺·	≺·	·>	·>
/	·≻	·≻	·>	·≻	≺·	≺·	≺·	·>	·>
^	·>	·>	·>	·>	≺·	≺·	≺٠	·>	·>
id	·≻	·≻	·>	·≻	·>			·>	·>
(<∙	<.	<∙	≺·	≺·	≺·	≺٠	≐	
)	·>	·>	·>	·>	·>			·>	·>
\$	≺·	≺·	≺·	≺·	≺·	<.	≺·		

Stack
\$
\$a
\$E+
\$E+b
\$E+E
\$E+E*
\$E+E*(
\$E+E*(c
\$E+E*(E
\$E+E*(E+
\$E+E*(E+d

id id

id

	+	1	*	/	<	id	()	\$
+	·	÷	∀ ·	÷	÷	÷	÷	·>	·>
-	·>	·>	<∙	<∙	<∙	<∙	<∙	·>	·>
*	·>	·>	·>	·>	<∙	≺·	≺·	·>	·>
/	·>	·≻	·	·>	≺·	≺·	≺·	·>	·>
^	·>	·>	·>	·>	≺·	<.	≺·	·>	·>
id	·>	·≻	·	·>	·>			·>	·>
(≺·	≺·	≺·	≺·	≺·	<.	≺·	≐	
)	·>	·≻	·	·>	·>			·>	·>
\$	≺·	<.	<∙	<∙	<∙	<∙	<∙		

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

\$E+E*(E+d

\$E+E*(E+E

id id id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+ d) - a

+ d) - a

d) - a

) - a

) - a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

shift

reduce E -> ID

		•							
+	Ý	·>							
-	·>							·>	
*	$\cdot \succ$	·>							·>
/	·>	·>	·>	·>	<.	<.	<.	·>	·>
^	·>	·>					<.		
id	·>	·>						·>	·>
(·		<.		<.	<.	≐	
)	·>	·>	·>	·>	·>			·>	·>
\$	<.	<∙	<.	<.	<.	<∙	<.		

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

\$E+E*(E+d

\$E+E*(E+E

id id id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+d) - a

+ d) - a

d) - a

) - a

) – a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

shift

reduce E -> ID

		-							
		·>							
		·>							
	•	·>		•	,	,		•	
/		·>							
٨		·>							
id		·>							
(-	<.	-				<∙		
)		·>						·>	·>
\$	<.	<.	<.	<.	<.	<.	<.		

\$tack \$ \$a \$E+ \$E+b \$E+E \$E+E* \$E+E* \$E+E*(c

\$E+E*(E \$E+E*(E+ \$E+E*(E+d \$E+E*(E+E

Input

a + b * (c + d) - a + b * (c + d) - a b * (c + d) - a * (c + d) - a * (c + d) - a

(c + d) - ac + d) - a

+ d) - a

+ d) - a

d) - a) - a

) – a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift shift shift

reduce E -> ID

shift shift

reduce E -> ID

reduce E -> E+E

	+	-	*	/	٨	id	()	\$
+	·>	·>			<.	<.	<.		·>
-	·>	·>	<.		<.	<.	,	·>	·>
*	·>				<.	<.	<.	·>	·>
/	·>	·		·>		·	<∙	·>	·>
٨	·>			·>		<∙		·>	·>
id	·>	·>	·>	·>	·>			·>	·>
(<.	<∙	<.	<.	<.	<∙	<∙	÷	
)	·>	·>	·>	·>	·>			·>	·>
\$	<.	<.	<.	<.	<.	<.	<.		

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

\$E+E*(E+d

\$E+E*(E+E

\$E+E*(E

id id E+E

id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+d)-a

+ d) - a

d) - a

) - a

) - a

) - a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

shift

reduce E -> ID

reduce E -> E+E

When do we stop popping the stack?

when the E + E is popped, the following hold

- 1. the top of the stack is a terminal which is (
- 2. the last popped terminal is +
- 3. (≺· +

so we stop

what is popped is between a pair $\prec \cdot$ and $\cdot \succ : \quad \prec \cdot$ RHS of reduction $\cdot \succ$

	+	•	*	/	٨	id	()	\$
+	Ý	Ý	<.	÷	÷	∀ ·	∀ ·	·	·>
-	·>	·>	<∙	<.	<∙	<∙	<.	·>	·>
*	·>			·≻	<∙	<∙	<∙	·≻	·>
/	·>		·>	·>	<.	<.	<.	·>	·>
٨	·≻			·≻	<.	<.	<∙	·≻	·>
id	·>		·>	·>				·>	·>
(<∙	<∙	<∙	<.	<∙	<∙	<∙	÷	
)	·≻		·>	·>				·>	·>
\$	≺٠	<∙	<∙	<∙	<∙	<∙	≺٠		

S	tac	:k
\$		

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

\$E+E*(E+d

\$E+E*(E+E

\$E+E*(E | | | id id E+E

id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+d)-a

+ d) - a

d) - a

) - a

) - a

) – a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

shift

reduce E -> ID

reduce E -> E+E

Stack
\$
\$ a
\$E+
\$E+b
\$E+E
\$E+E*
\$E+E*(
\$E+E*(c
\$E+E*(E
\$E+E*(E+
\$E+E*(E+d
\$E+E*(E+E
\$E+E*(E = =

id id

Input Action a + b * (c + d) - ashift + b * (c + d) - areduce E -> ID b * (c + d) - ashift *(c + d) - areduce E -> ID *(c + d) - ashift (c + d) - ashift c + d) - ashift + d) - areduce E -> ID + d) - ashift d) - a shift) - a reduce E -> ID) - a reduce E -> E+E shift

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

\$E+E*(E+d

\$E+E*(E+E

\$E+E*(E

\$E+E*(E)

ΦE+E"(E)

id id E+E

id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+d)-a

+d) - a

d) - a

) - a

) - a

) - a

- а

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

shift

reduce E -> ID

reduce E -> E+E

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

\$E+E*(E+d

\$E+E*(E+E

\$E+E*(E

\$E+E*(E)

id id E+E

id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+ d) - a

+ d) - a

d) - a

) - a

) - a

) - a

- a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

shift

reduce E -> ID

reduce E -> E+E

Stack
\$
\$ a
\$E+
\$E+b
\$E+E
\$E+E*
\$E+E*(
\$E+E*(c
\$E+E*(E
\$E+E*(E+
\$E+E*(E+d
\$E+E*(E+E
\$E+E*(E
\$E+E*(E)

id id

Input	Action
a + b * (c + d) - a	shift
+ b * (c + d) - a	reduce E -> ID
b * (c + d) - a	shift
* (c + d) - a	reduce E -> ID
* (c + d) - a	shift
(c + d) - a	shift
c + d) - a	shift
+ d) - a	reduce E -> ID
+ d) - a	shift
d) - a	shift
) - a	reduce E -> ID
) - a	reduce E -> E+E
) - a	shift
y - a	reduce E -> (E)

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

\$E+E*(E+d

\$E+E*(E+E

\$E+E*(E

\$E+E*(E)

\$E+E*E

id id (E)

E+E

id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+ d) - a

+d) - a

d) - a

) - a

) - a

) - a

- a

- a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

shift

reduce E -> ID

reduce E -> E+E

shift

reduce E -> (E)

Stack	Inp
\$	a +
\$ a	+
\$E+	b *
\$E+b	* (0
\$E+E	* (0
\$E+E*	(c +
\$E+E*(C +
\$E+E*(c	+ d
\$E+E*(E	+ d
\$E+E*(E+	d) -
\$E+E*(E+d) -
\$E+E*(E+E) -
\$E+E*(E) -
\$E+E*(E)	- a
\$E+E*E	y - 0
id id (E)·>	
/\\ E+E	

id id

Input	Action
a + b * (c + d) - a	shift
+ b * (c + d) - a	reduce E -> ID
b * (c + d) - a	shift
* (c + d) - a	reduce E -> ID
* (c + d) - a	shift
(c + d) - a	shift
c + d) - a	shift
+ d) - a	reduce E -> ID
+ d) - a	shift
d) - a	shift
) - a	reduce E -> ID
) - a	reduce E -> E+E
) - a	shift
- a	reduce E -> (E)
√ - a	

Stack	Inp
\$	a +
\$ a	+
\$E+	b *
\$E+b	* (
\$E+E	* (
\$E+E*	(c -
\$E+E*(C +
\$E+E*(c	+ d
\$E+E*(E	+ d
\$E+E*(E+	d)
\$E+E*(E+d) -
\$E+E*(E+E) -
\$E+E*(E) -
\$E+E*(E)	- 0
\$E+E*E \ id id (E) \>	y- 0

E+E

id id

but Action + b * (c + d) - a shift b * (c + d) - areduce E -> ID * (c + d) - ashift (c + d) - areduce E -> ID (c + d) - ashift + d) - ashift + d) - a shift d) - a reduce E -> ID d) - a shift shift - a reduce E -> ID а reduce E -> E+E а shift а reduce E -> (E) a reduce E -> E*E a

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

\$E+E*(E+d

\$E+E*(E+E

\$E+E*(E

\$E+E*(E)

\$E+E*E

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+ d) - a

+ d) - a

d) - a

) - a

) - a

) – a

- a

- a

- a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

shift

reduce E -> ID

reduce E -> E+E

shift

reduce E -> (E)

reduce E -> E*E

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

\$E+E*(E+d

\$E+E*(E+E

\$E+E*(E

\$E+E*(E)

\$E+E*E

id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+d)-a

+d) - a

d) - a

) - a

) - a

) – a

- a

- а

а

ıt Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

shift

reduce E -> ID

reduce E -> E+E

shift

reduce E -> (E)

reduce E -> E*E

C	4ء	_	ı
2	ια	C	ĸ

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

\$E+E*(E+d

\$E+E*(E+E

\$E+E*(E

\$E+E*(E)

\$E+E*E

\$E+E id E*E id (E) E+E

id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+d)-a

+d) - a

d) - a

) - a

) - a

) - a

- a

- a

- a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

shift

reduce E -> ID

reduce E -> E+E

shift

reduce E -> (E)

reduce E -> E*E

reduce E -> E+E

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

\$E+E*(E+d

\$E+E*(E+E

\$E+E*(E

\$E+E*(E)

\$E+E*E

\$E+E

\$E

id E*E
id (E)
E+E

id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+ d) - a

+ d) - a

d) - a

) - a

) - a

) – a

- a

- a

- a

- a

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

shift

reduce E -> ID

reduce E -> E+E

shift

reduce E -> (E)

reduce E -> E*E

reduce E -> E+E

Stack	Input
\$	a + b * (c + d) - a
\$a	+ b * (c + d) - a
\$E+	b * (c + d) - a
\$E+b	* (c + d) - a
\$E+E	* (c + d) - a
\$E+E*	(c + d) - a
\$E+E*(c + d) - a
\$E+E*(c	+ d) - a
\$E+E*(E	+ d) - a
\$E+E*(E+	d) - a
\$E+E*(E+d) - a
\$E+E*(E+E) - a
\$E+E*(E) - a
\$E+E*(E)	- a
\$E+E*E	- a
\$E+E	- a
\$E	- a
E+E <	
id E*E	
id (E)	
/\\ E+E	
id id	

Action shift reduce E -> ID shift reduce E -> ID shift shift shift reduce E -> ID shift shift reduce E -> ID reduce E -> E+E shift reduce E -> (E) reduce E -> E*E reduce E -> E+E

Stack	Input
\$	a + b * (c + d) - a
\$ a	+ b * (c + d) - a
\$E+	b * (c + d) - a
\$E+b	* (c + d) - a
\$E+E	* (c + d) - a
\$E+E*	(c + d) - a
\$E+E*(c + d) - a
\$E+E*(c	+ d) - a
\$E+E*(E	+ d) - a
\$E+E*(E+	d) - a
\$E+E*(E+d) - a
\$E+E*(E+E) - a
\$E+E*(E) - a
\$E+E*(E)	- a
\$E+E*E	- a
\$E+E	- a
\$E	- a
E+E <	
id E*E	
id (E)	
/\\ r . r	
E+E 	

id id

Action shift reduce E -> ID shift reduce E -> ID shift shift shift reduce E -> ID shift shift reduce E -> ID reduce E -> E+E shift reduce E -> (E) reduce E -> E*E reduce E -> E+E shift

_			
C	+~	^	v
	1(1	ι.	◣

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

\$E+E*(E+d

\$E+E*(E+E

\$E+E*(E

\$E+E*(E)

\$E+E*E

\$E+E

\$E

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+d)-a

+ d) - a

d) - a

) - a

) - a

) – a

- a

- a

- a

- a

а

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

shift

reduce E -> ID

reduce E -> E+E

shift

reduce E -> (E)

reduce E -> E*E

reduce E -> E+E

Stack	In
\$	a
\$a	+
\$E+	b
\$E+b	*
\$E+E	*
\$E+E*	(c
\$E+E*(c ·
\$E+E*(c	+ (
\$E+E*(E	+ (
\$E+E*(E+	d)
\$E+E*(E+d) -
\$E+E*(E+E) -
\$E+E*(E) -
\$E+E*(E)	_
\$E+E*E	-
\$E+E	-
\$E	-
\$E-	√ a
E+E	

id id

Action nput + b * (c + d) - ashift b * (c + d) - a reduce E -> ID *(c + d) - ashift (c + d) - areduce E -> ID (c + d) - ashift + d) - a shift + d) - ashift d) - a reduce E -> ID d) - a shift shift - a reduce E -> ID - a reduce E -> E+E - a shift - a reduce E -> (E) а reduce E -> E*E а reduce E -> E+E а shift а

Stack	Inp
\$	a +
\$ a	+
\$E+	b *
\$E+b	* (
\$E+E	* (
\$E+E*	(c ·
\$E+E*(C +
\$E+E*(c	+ 0
\$E+E*(E	+ 0
\$E+E*(E+	d)
\$E+E*(E+d) -
\$E+E*(E+E) -
\$E+E*(E) -
\$E+E*(E)	- 6
\$E+E*E	- 6
\$E+E	- 6
\$E	- 6
\$E-	▼ a
E+E	

id id

put + b * (c + d) - a b * (c + d) - a* (c + d) - a(c + d) - a(c + d) - a+ d) - a+ d) - a d) - a d) - a - a - a - a - a a a a a

Action shift reduce E -> ID shift reduce E -> ID shift shift shift reduce E -> ID shift shift reduce E -> ID reduce E -> E+E shift reduce E -> (E) reduce E -> E*E reduce E -> E+E shift shift

Stack

\$

\$a

\$E+

\$E+b

\$E+E

\$E+E*

\$E+E*(

\$E+E*(c

\$E+E*(E

\$E+E*(E+

\$E+E*(E+d

\$E+E*(E+E

\$E+E*(E

\$E+E*(E)

\$E+E*E

\$E+E

\$E

\$E-

\$E-a

E+E id E*E id (E)

id id

Input

a + b * (c + d) - a

+ b * (c + d) - a

b * (c + d) - a

*(c + d) - a

*(c + d) - a

(c + d) - a

c + d) - a

+d) - a

+d)-a

d) - a

) - a

) - a

) – a

- a

- a

- a

– а

_

а

Action

shift

reduce E -> ID

shift

reduce E -> ID

shift

shift

shift

reduce E -> ID

shift

shift

reduce E -> ID

reduce E -> E+E

shift

reduce E -> (E)

reduce E -> E*E

reduce E -> E+E

shift

Stack Input \$ \$a \$E+ \$E+b \$E+E \$E+E* \$E+E*(\$E+E*(c \$E+E*(E \$E+E*(E+ \$E+E*(E+d) - a \$E+E*(E+E) - a \$E+E*(E) - a \$E+E*(E) - a \$E+E*E - a \$E+E - a \$E - a \$Eа \$E-a \$ E+E E*E id id (E) E+E

id id

a + b * (c + d) - ashift + b * (c + d) - areduce E -> ID b * (c + d) - ashift *(c + d) - areduce E -> ID *(c + d) - ashift (c + d) - ashift c + d) - ashift + d) - areduce E -> ID + d) - ashift d) - a shift reduce E -> ID reduce E -> E+E shift reduce E -> (E) reduce E -> E*E reduce E -> E+E shift shift

Action

Stack	Input
\$	a + b
\$ a	+ b *
\$E+	b * (c
\$E+b	* (c +
\$E+E	* (c +
\$E+E*	(c + d)
\$E+E*(c + d)
\$E+E*(c	+ d) -
\$E+E*(E	+ d) -
\$E+E*(E+	d) - a
\$E+E*(E+d) - a
\$E+E*(E+E) - a
\$E+E*(E) - a
\$E+E*(E)	- a
\$E+E*E	- a
\$E+E	- a
\$E	- a
\$E-	a
\$E-a	\$
E+E ·>	
id E*E	
id (E)	
E+E	

id id

b * (c + d) - ashift * (c + d) - a reduce E -> ID (c + d) - ashift reduce E -> ID + d) - a+d)-ashift d) - a shift d) - a shift reduce E -> ID - a shift - a shift а reduce E -> ID a reduce E -> E+E a shift a reduce E -> (E) reduce E -> E*E reduce E -> E+E shift shift reduce E -> ID

Stack

STUCK
\$
\$ a
\$E+
\$E+b
\$E+E
\$E+E*
\$E+E*(
\$E+E*(c
\$E+E*(E
\$E+E*(E+
\$E+E*(E+d
\$E+E*(E+E
\$E+E*(E
\$E+E*(E)
\$E+E*E
\$E+E
\$E
\$E-
\$E-a
\$E-E
E+E id
id E*E
id (Ę)
E+E
id id
14 14

shift reduce E -> ID shift reduce E -> ID shift shift shift reduce E -> ID shift shift reduce E -> ID reduce E -> E+E shift reduce E -> (E) reduce E -> E*E reduce E -> E+E shift shift reduce E -> ID

Stack	Input
\$	a + b
\$ a	+ b *
\$E+	b * (c
\$E+b	* (c +
\$E+E	* (c +
\$E+E*	(c + d
\$E+E*(c + d)
\$E+E*(c	+ d) -
\$E+E*(E	+ d) -
\$E+E*(E+	d) – a
\$E+E*(E+d) - a
\$E+E*(E+E) - a
\$E+E*(E) - a
\$E+E*(E)	- a
\$E+E*E	- a
\$E+E	- a
\$E	- a
\$E-	а
\$E-a	\$
\$Ę-E	\$
E+E id .> id E*E	

E+E

id id

+ b * (c + d) - a shift b * (c + d) - a reduce E -> ID * (c + d) - ashift (c + d) - areduce E -> ID (c + d) - ashift + d) - ashift + d) - a shift d) - a reduce E -> ID d) - a shift shift - a reduce E -> ID а reduce E -> E+E а shift а reduce E -> (E) a reduce E -> E*E a reduce E -> E+E a shift a shift \$ reduce E -> ID \$

Stack	Input
\$	a + b * (c + d) - a
\$a	+ b * (c + d) - a
\$E+	b * (c + d) - a
\$E+b	* (c + d) - a
\$E+E	* (c + d) - a
\$E+E*	(c + d) - a
\$E+E*(c + d) - a
\$E+E*(c	+ d) - a
\$E+E*(E	+ d) - a
\$E+E*(E+	d) - a
\$E+E*(E+d) - a
\$E+E*(E+E) - a
\$E+E*(E) - a
\$E+E*(E)	- a
\$E+E*E	- a
\$E+E	- a
\$E	- a
\$E-	a
\$E-a	\$
\$E-E	\$
E+E id ·> id E*E	
id (E)	

E+E

id id

shift reduce E -> ID shift reduce E -> ID shift shift shift reduce E -> ID shift shift reduce E -> ID reduce E -> E+E shift reduce E -> (E) reduce E -> E*E reduce E -> E+E shift shift reduce E -> ID reduce E -> E-E

id id

Stack	Input	Action
\$	a + b * (c + d) - a	shift
\$ a	+ b * (c + d) - a	reduce
\$E+	b * (c + d) - a	shift
\$E+b	* (c + d) - a	reduce
\$E+E	* (c + d) - a	shift
\$E+E*	(c + d) - a	shift
\$E+E*(c + d) - a	shift
\$E+E*(c	+ d) - a	reduce
\$E+E*(E	+ d) - a	shift
\$E+E*(E+	d) - a	shift
\$E+E*(E+d) - a	reduce
\$E+E*(E+E) - a	reduce
\$E+E*(E) - a	shift
\$E+E*(E)	- a	reduce
\$E+E*E	- a	reduce
\$E+E	- a	reduce
\$E	- a	shift
\$E-	a	shift
\$E-a	\$	reduce
\$E-E	\$	reduce
\$E E+E id id E*E id (E) E+E	\$	

luce E -> ID ft luce E -> ID ft ft ft luce E -> ID ft ft luce E -> ID luce E -> E+E ft luce E -> (E) luce E -> E*E luce E -> E+E ft ft luce E -> ID luce E -> E-E

id id

Ctank	Tnn+	Action
Stack	Input	Action
\$	a + b * (c + d) - a	shift
\$ a	+ b * (c + d) - a	reduce E -> ID
\$E+	b * (c + d) - a	shift
\$E+b	* (c + d) - a	reduce E -> ID
\$E+E	* (c + d) - a	shift
\$E+E*	(c + d) - a	shift
\$E+E*(c + d) - a	shift
\$E+E*(c	+ d) - a	reduce E -> ID
\$E+E*(E	+ d) - a	shift
\$E+E*(E+	d) - a	shift
\$E+E*(E+d) - a	reduce E -> ID
\$E+E*(E+E) - a	reduce E -> E+E
\$E+E*(E) - a	shift
\$E+E*(E)	- a	reduce E -> (E)
\$E+E*E	- a	reduce E -> E*E
\$E+E	- a	reduce E -> E+E
\$E	- a	shift
\$E-	a	shift
\$E-a	\$	reduce E -> ID
\$E-E	\$	reduce E -> E-E
\$E E-E return E+E id id E*E id (E) E+E	\$	

Dealing with non-terminals

We have the non-terminals on the stack as they are pushed when a reduction occurs

Given that the grammar is an operator grammar, we can have at most one non-terminal on the top of the stack. There is always a terminal on the top of the stack or just below

In the algorithm we assume that stack.terminalpeek() ignores non-terminals and returns the terminal symbol at the top of the stack or just below

HEURISTIC FOR DETERMINING PRECDENCE RELATIONSHIPS

We assume we have a set of operators with

- precedence levels
- associativity (left or right)
- operators at the same level have the same associativity

We assume the input is of the form w \$

We have the following heuristics for determining $\prec \cdot$, $\cdot >$, and \doteq relationships between operators

- 1. if op1 has higher precedence level than op2, then
 - op1 → op2
 - op2 < · op1

Example: $* \rightarrow +$

- + < *
- ^ ·> +
- + < . ^

HEURISTIC FOR DETERMINING PRECDENCE RELATIONSHIPS

- 2. if op1 and op2 are operators of the same operator precedence, possibly the same operator, then
 - If they are left associative:
 - op1 ·> op2
 - op2 ·> op1

Example. + and - are left associative, so we have

- + '> +
- + ·> -
- .> +
- **·≻ -**
- If they are right associative:
 - op1 < · op2
 - op2 < · op1

Example. ^ is right associative, so we have

- ^ ·<· ^
- ^ <- ^

HEURISTIC FOR DETERMINING PRECDENCE RELATIONSHIPS

3. Also, we have the following

```
1.
                          ID
           op
                   \prec·
           ID
2.
                          op
                   ·>
3.
                          (
           ор
                   ≺·
4.
                          op
                   <∙
5.
           op
                          )
                   ·>
6.
                   ·>
                          op
           $
7.
                          op
                   \prec·
                           $
8.
           op
                   ·>
9.
                   ÷
           $
10.
                          id
                   <∙
                          $
           id
11.
                   ·>
           $
12.
                           (
                   \prec·
13.
                   \prec·
14.
                          id
                   \prec·
                          $
15.
                   ·>
16.
                   ·>
17.
           id
                   \cdot >
```

If we have a unary operator uop that is not a binary operator, we can support it as follows

- op < · uop for every other operator op. op can be unary or binary
- uop \prec op if uop has lower operator precedence level than op
- uop ·> op if uop has higher operator precedence level than op

If we have a unary operator that is also a binary operator, like MINUS, we cannot incorporate it in the scheme!

Example id*-id is not easily parsed

One solution is to have the getToken() function make the distinction by looking at the context in which the operator appears.

Example In FORTRAN a minus sign is unary if the previous token is an operator, LPAREN, COMMA, or EQUAL

If we have a unary operator uop that is not a binary operator, we can support it as follows

- op < · uop for every other operator op. op can be unary or binary
- uop \prec op if uop has lower operator precedence level than op
- uop ·> op if uop has higher operator precedence level than op

If we have a unary operator that is also a binary operator, like MINUS, we cannot incorporate it in the scheme!

Example id*-id is not easily parsed

One solution is to have the getToken() function make the distinction by looking at the context in which the operator appears.

Example In FORTRAN a minus sign is unary if the previous token is an operator, LPAREN, COMMA, or EQUAL

If we have a unary operator uop that is not a binary operator, we can support it as follows

- op < · uop for every other operator op. op can be unary or binary
- uop \prec op if uop has lower operator precedence level than op
- uop ·> op if uop has higher operator precedence level than op

If we have a unary operator that is also a binary operator, like MINUS, we cannot incorporate it in the scheme!

Example id*-id is not easily parsed

One solution is to have the getToken() function make the distinction by looking at the context in which the operator appears.

Example In FORTRAN a minus sign is unary if the previous token is an operator, LPAREN, COMMA, or EQUAL

If we have a unary operator uop that is not a binary operator, we can support it as follows

- op \prec · uop for every other operator op. op can be unary or binary
- uop \prec op if uop has lower operator precedence level than op
- uop ·> op if uop has higher operator precedence level than op

If we have a unary operator that is also a binary operator, like MINUS, we cannot incorporate it in the scheme!

Example id*-id is not easily parsed

One solution is to have the getToken() function make the distinction by looking at the context in which the operator appears.

Example In FORTRAN a minus sign is unary if the previous token is an operator, LPAREN, COMMA, or EQUAL