SYNTAX ANALYSIS

BOTTOM-UP PARSING

Operator Precedence Parser

E → E + T | T

T → T * V | V

V → a | b | c | d

Input: a + b * c * d

	a	b	c	d	+	*	$
a					>	>	>
b					>	>	>
c					>	>	>
d					>	>	>
+	<	<	<	<	>	<	>
*	<	<	<	<	>	>	>
$	<	<	<	<	<	<	A

Operator Precedence Parser

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a

Operator Precedence Parser

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a
$ a	>	+ b * c * d $	Pop a

Operator Precedence Parser

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a
$ a	>	+ b * c * d $	Pop a
$	<	+ b * c * d $	Push +

Operator Precedence Parser

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a
$ a	>	+ b * c * d $	Pop a
$	<	+ b * c * d $	Push +
$ +	<	b * c * d $	Push b

Operator Precedence Parser

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a
$ a	>	+ b * c * d $	Pop a
$	<	+ b * c * d $	Push +
$ +	<	b * c * d $	Push b
$ + b	>	* c * d $	Pop b

Operator Precedence Parser

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a
$ a	>	+ b * c * d $	Pop a
$	<	+ b * c * d $	Push +
$ +	<	b * c * d $	Push b
$ + b	>	* c * d $	Pop b
$ +	<	* c * d $	Push *

Operator Precedence Parser

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a
$ a	>	+ b * c * d $	Pop a
$	<	+ b * c * d $	Push +
$ +	<	b * c * d $	Push b
$ + b	>	* c * d $	Pop b
$ +	<	* c * d $	Push *
$ + *	<	c * d $	Push c

Operator Precedence Parser

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a
$ a	>	+ b * c * d $	Pop a
$	<	+ b * c * d $	Push +
$ +	<	b * c * d $	Push b
$ + b	>	* c * d $	Pop b
$ +	<	* c * d $	Push *
$ + *	<	c * d $	Push c
$ + * c	>	* d $	Pop c

Operator Precedence Parser

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a
$ a	>	+ b * c * d $	Pop a
$	<	+ b * c * d $	Push +
$ +	<	b * c * d $	Push b
$ + b	>	* c * d $	Pop b
$ +	<	* c * d $	Push *
$ + *	<	c * d $	Push c
$ + * c	>	* d $	Pop c
$ + *	>	* d $	Pop *

Operator Precedence Parser

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a
$ a	>	+ b * c * d $	Pop a
$	<	+ b * c * d $	Push +
$ +	<	b * c * d $	Push b
$ + b	>	* c * d $	Pop b
$ +	<	* c * d $	Push *
$ + *	<	c * d $	Push c
$ + * c	>	* d $	Pop c
$ + *	>	* d $	Pop *
$ +	<	* d $	Push *

Operator Precedence Parser

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a
$ a	>	+ b * c * d $	Pop a
$	<	+ b * c * d $	Push +
$ +	<	b * c * d $	Push b
$ + b	>	* c * d $	Pop b
$ +	<	* c * d $	Push *
$ + *	<	c * d $	Push c
$ + * c	>	* d $	Pop c
$ + *	>	* d $	Pop *
$ +	<	* d $	Push *
$ + *	<	d $	Push d

Operator Precedence Parser

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a
$ a	>	+ b * c * d $	Pop a
$	<	+ b * c * d $	Push +
$ +	<	b * c * d $	Push b
$ + b	>	* c * d $	Pop b
$ +	<	* c * d $	Push *
$ + *	<	c * d $	Push c
$ + * c	>	* d $	Pop c
$ + *	>	* d $	Pop *
$ +	<	* d $	Push *
$ + *	<	d $	Push d
$ + * d	>	$	Pop d

Operator Precedence Parser

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a
$ a	>	+ b * c * d $	Pop a
$	<	+ b * c * d $	Push +
$ +	<	b * c * d $	Push b
$ + b	>	* c * d $	Pop b
$ +	<	* c * d $	Push *
$ + *	<	c * d $	Push c
$ + * c	>	* d $	Pop c
$ + *	>	* d $	Pop *
$ +	<	* d $	Push *
$ + *	<	d $	Push d
$ + * d	>	$	Pop d
$ + *	>	$	Pop *

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a
$ a	>	+ b * c * d $	Pop a
$	<	+ b * c * d $	Push +
$ +	<	b * c * d $	Push b
$ + b	>	* c * d $	Pop b
$ +	<	* c * d $	Push *
$ + *	<	c * d $	Push c
$ + * c	>	* d $	Pop c
$ + *	>	* d $	Pop *
$ +	<	* d $	Push *
$ + *	<	d $	Push d
$ + * d	>	$	Pop d
$ + *	>	$	Pop *
$ +	>	$	Pop +

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Stack	Relation	Input	Comment
$	<	a + b * c * d $	Push a
$ a	>	+ b * c * d $	Pop a
$	<	+ b * c * d $	Push +
$ +	<	b * c * d $	Push b
$ + b	>	* c * d $	Pop b
$ +	<	* c * d $	Push *
$ + *	<	c * d $	Push c
$ + * c	>	* d $	Pop c
$ + *	>	* d $	Pop *
$ +	<	* d $	Push *
$ + *	<	d $	Push d
$ + * d	>	$	Pop d
$ + *	>	$	Pop *
$ +	>	$	Pop +
$	Accept	$	Accept

$E \rightarrow E + T \mid T$ $T \rightarrow T * V \mid V$ $V \rightarrow a \mid b \mid c \mid d$ Input: a + b * c * d

Comment
Push a
Pop a
Push +
Push b
Pop b
Push *
Push c
Pop c
Pop *
Push *
Push d
Pop d
Pop *
Pop +
Accept

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Comment
Pop a (1)
Pop b (2)
Pop c (3)
Pop * (4)
Pop d (5)
Pop * (6)
Pop + (7)
Accept

Sequence to be processed:

a , b , c , * , d , * , +

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Sequence to be processed:

a , b , c , * , d , * , +

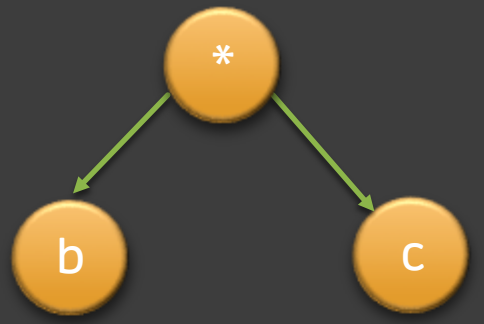
Stack
c
b
a
$

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Sequence to be processed:

a , b , c , * , d , * , +

Stack
a
$



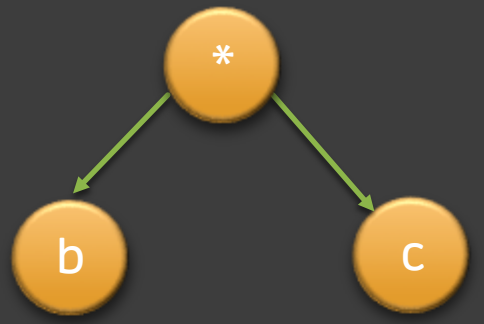
Sub Tree 1

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Sequence to be processed:

a , b , c , * , d , * , +

Stack
Sub Tree 1
a
$



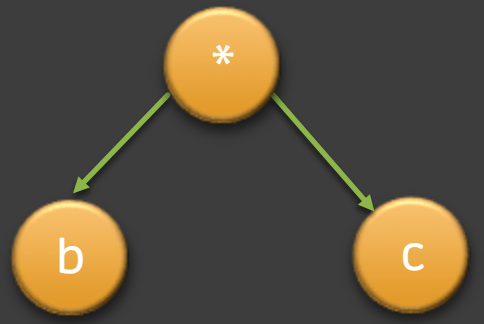
Sub Tree 1

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Sequence to be processed:

a , b , c , * , d , * , +

Stack
d
Sub Tree 1
a
$



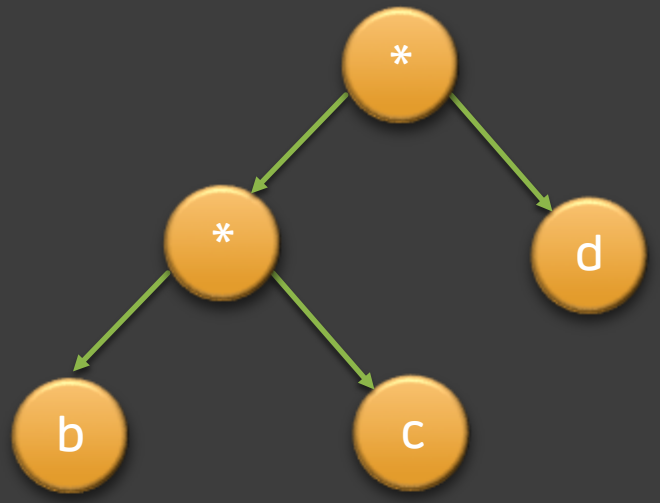
Sub Tree 1

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Sequence to be processed:

a , b , c , * , d , * , +

Stack
a
$



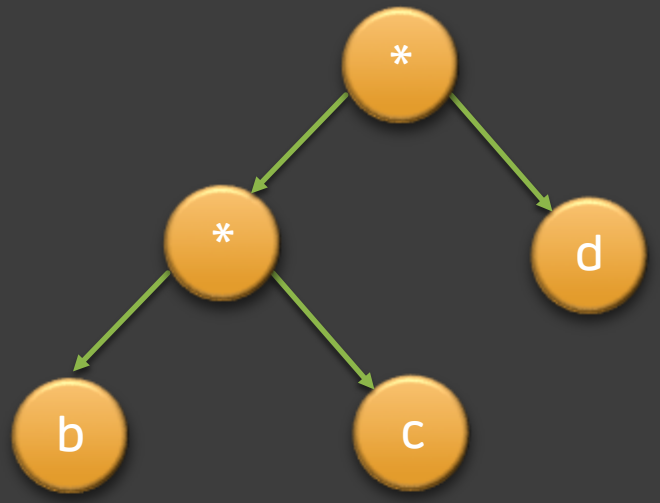
Sub Tree 2

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Sequence to be processed:

a , b , c , * , d , * , +

Stack
Sub Tree 2
a
$



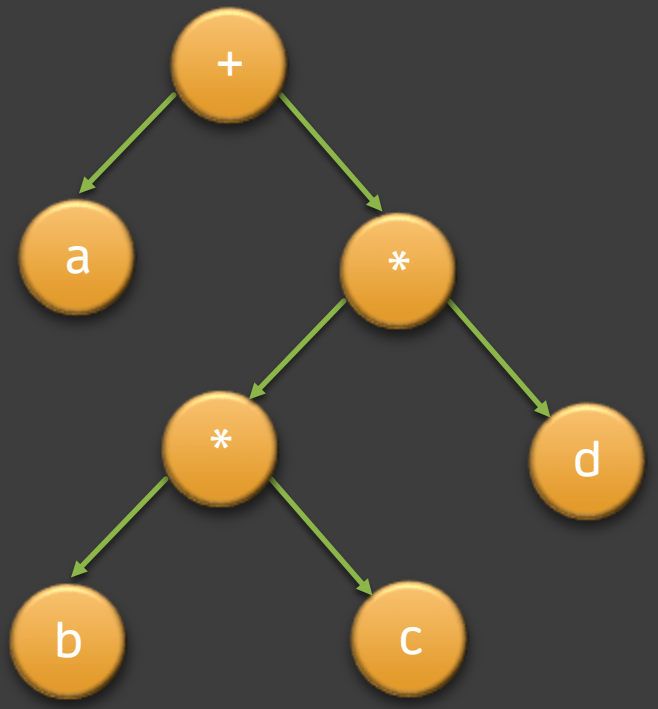
Sub Tree 2

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Sequence to be processed:

a , b , c , * , d , * , +

Stack
$



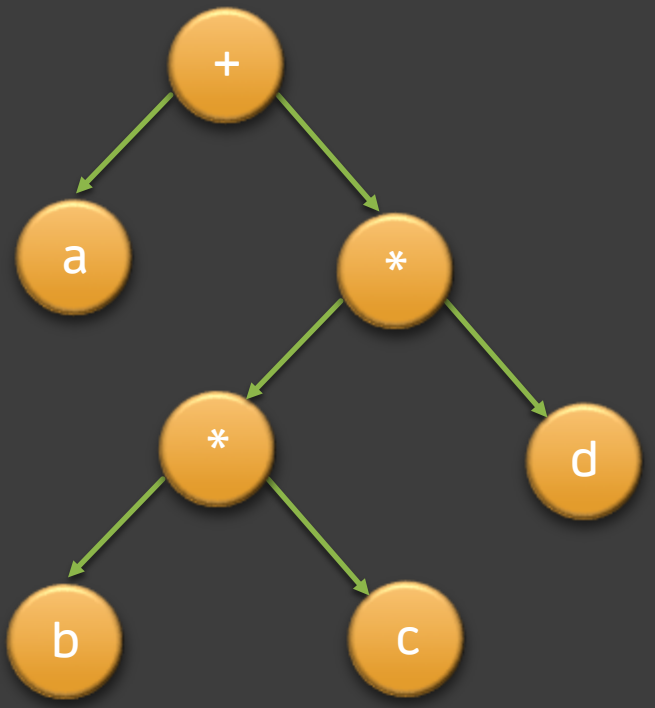
Sub Tree 3

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Sequence to be processed:

a , b , c , * , d , * , +

Stack
Sub Tree 3
$



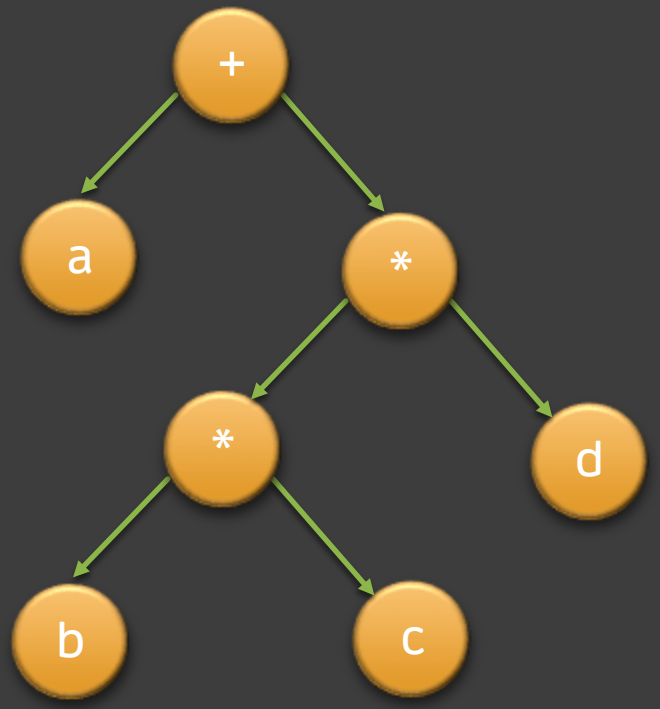
Sub Tree 3

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Sequence to be processed:

a , b , c , * , d , * , +

Stack
$

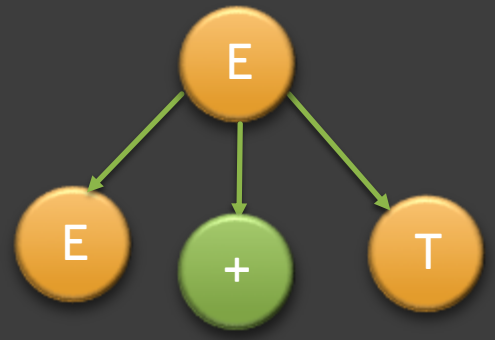


Final Tree

E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Comment
Pop a (7)
Pop b (6)
Pop c (5)
Pop * (4)
Pop d (3)
Pop * (2)
Pop + (1) : E → E + T
Accept

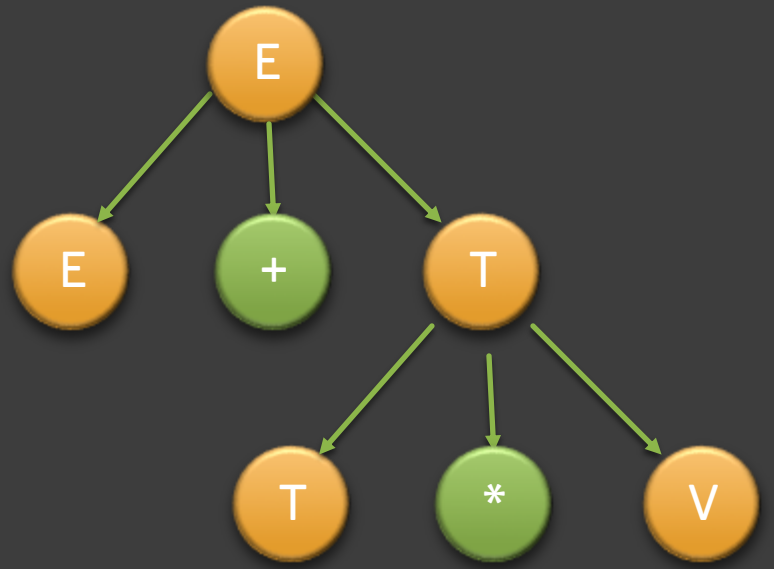
Construction of Entire Tree:



E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Comment
Pop a (7)
Pop b (6)
Pop c (5)
Pop * (4)
Pop d (3)
Pop * (2) : T → T * V
Pop + (1) : E → E + T
Accept

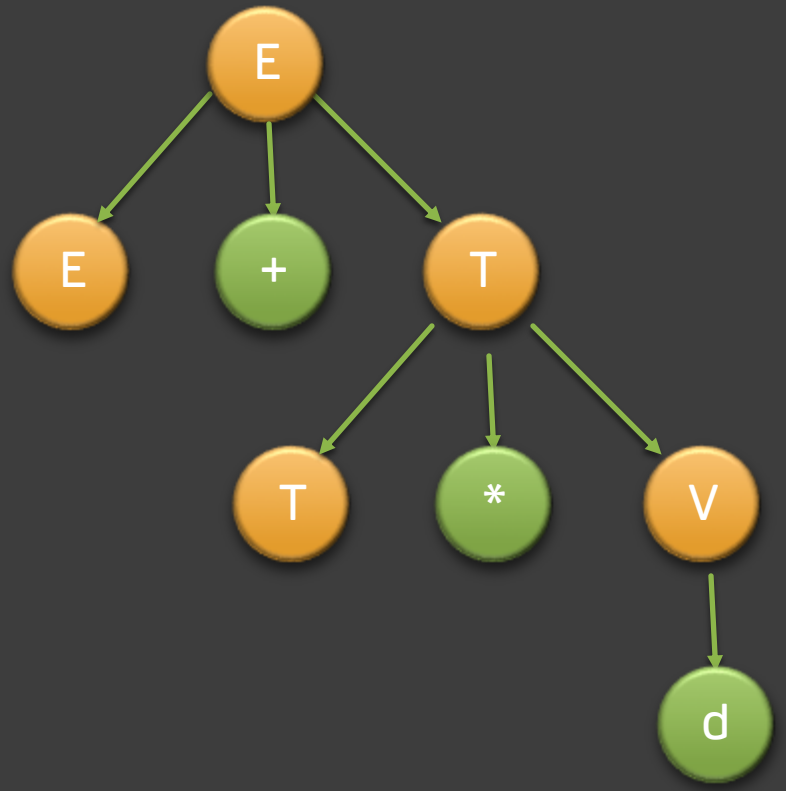
Construction of Entire Tree:



E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Comment
Pop a (7)
Pop b (6)
Pop c (5)
Pop * (4)
Pop d (3) : V → d
Pop * (2) : T → T * V
Pop + (1) : E → E + T
Accept

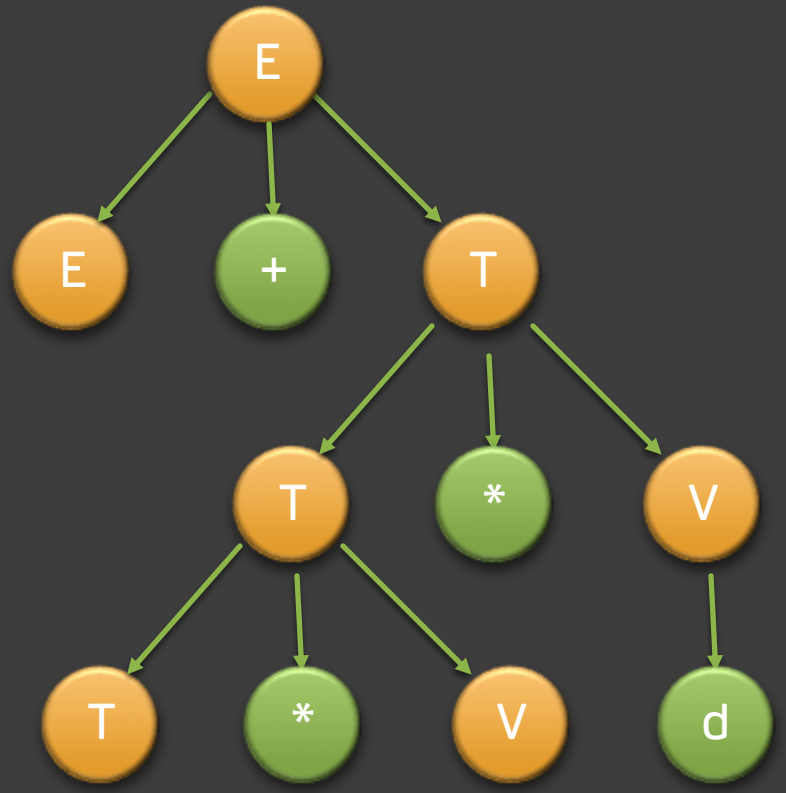
Construction of Entire Tree:



E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Comment
Pop a (7)
Pop b (6)
Pop c (5)
Pop * (4) : T → T * V
Pop d (3) : V → d
Pop * (2) : T → T * V
Pop + (1) : E → E + T
Accept

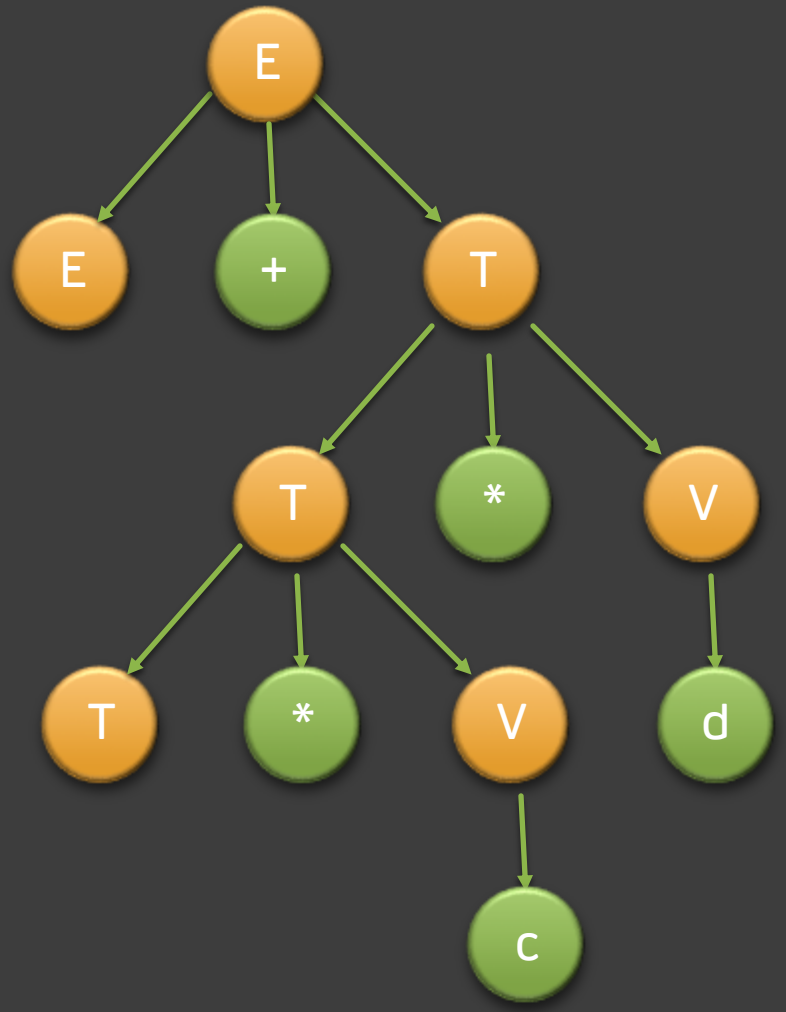
Construction of Entire Tree:



E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Comment
Pop a (7)
Pop b (6)
Pop c (5) : V → c
Pop * (4) : T → T * V
Pop d (3) : V → d
Pop * (2) : T → T * V
Pop + (1) : E → E + T
Accept

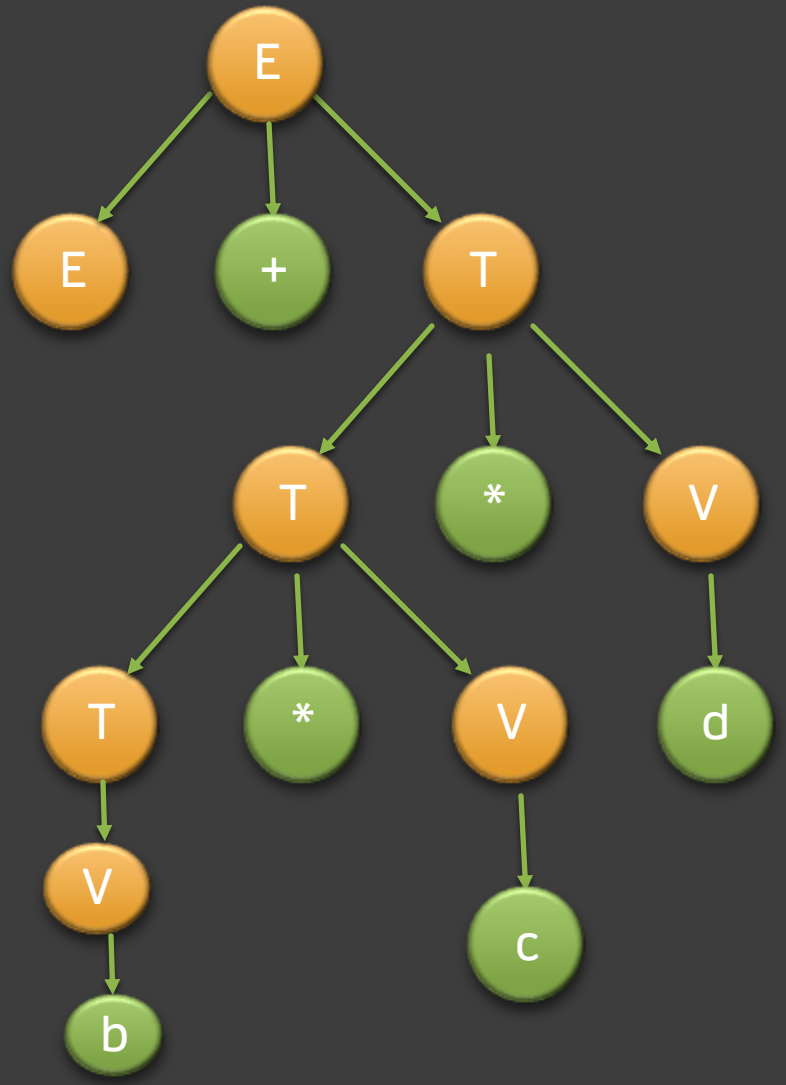
Construction of Entire Tree:



E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Comment
Pop a (7)
Pop b (6) : T → V, V → b
Pop c (5) : V → c
Pop * (4) : T → T * V
Pop d (3) : V → d
Pop * (2) : T→ T * V
Pop + (1) : E → E + T
Accept

Construction of Entire Tree:



E → E + T | T T → T * V | V V → a | b | c | d Input: a + b * c * d

Comment
Pop a (7) : E → T, T→ V, V → a
Pop b (6) : T → V, V → b
Pop c (5) : V → c
Pop * (4) : T → T * V
Pop d (3) : V → d
Pop * (2) : T → T * V
Pop + (1) : E → E + T
Accept

Construction of Entire Tree: