CS202 – System Software

Dr. Manish Khare

Lecture 12

Bottom-up Parsing

- Bottom-up parsing starts from the leaf nodes of a tree and works in upward direction till it reaches the root node.

- Here, we start from a sentence and then apply production rules in reverse manner in order to reach the start symbol.

- This corresponds to starting at the leaves of the parse tree, and working back to the root.

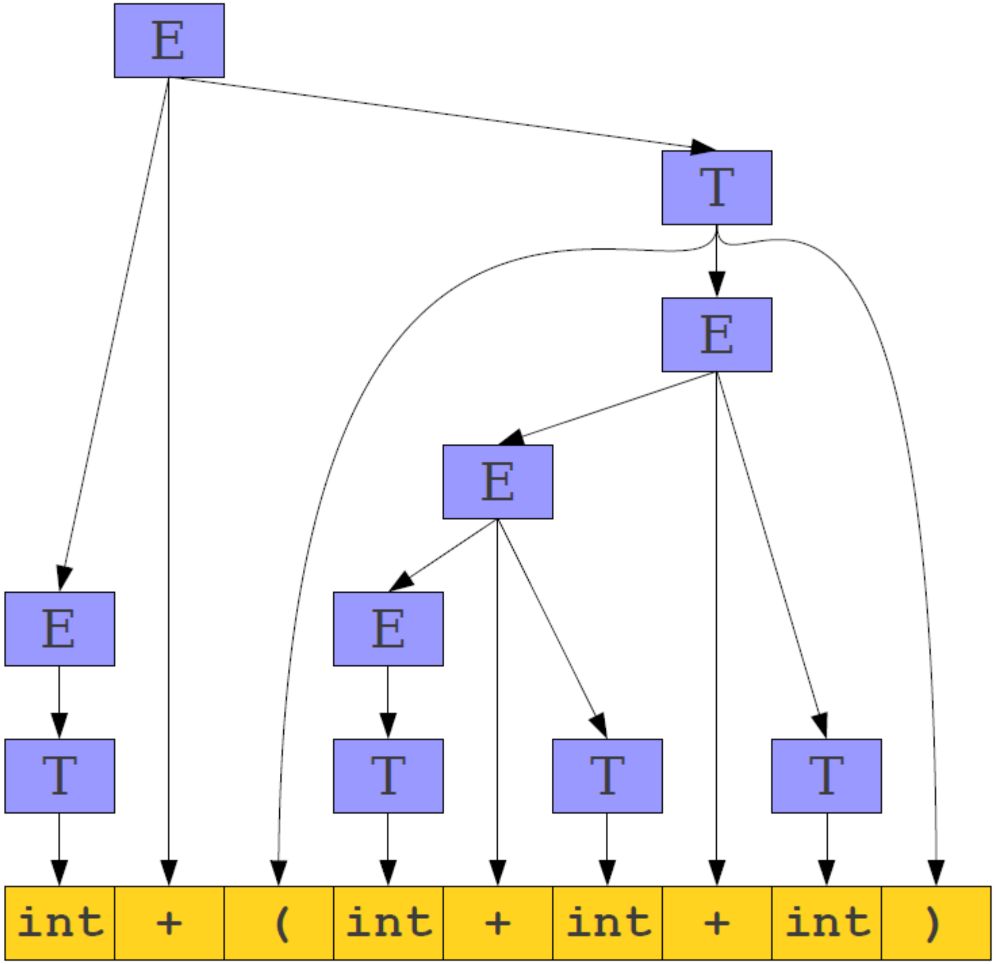
- Bottom-up parsing is also known as shift-reduce parsing

Bottom-up Parsing

- Bottom-up parsing is also known as **shift-reduce parsing** because its two main actions are shift and reduce.
  - At each shift action, the current symbol in the input string is pushed to a stack.
  - At each reduction step, the symbols at the top of the stack (this symbol sequence is the right side of a production) will replaced by the non-terminal at the left side of that production.
  - There are also two more actions: accept and error.

One View of a Bottom-Up Parse

$E \rightarrow T$
$E \rightarrow E + T$
$T \rightarrow$ **int**
$T \rightarrow$ **(E)**



A Second View of a Bottom-Up Parse

$E \rightarrow T$
$E \rightarrow E + T$
$T \rightarrow$ int
$T \rightarrow$ (E)

int + (int + int + int)
⇒ T + (int + int + int)
⇒ E + (int + int + int)
⇒ E + (T + int + int)
⇒ E + (E + int + int)
⇒ E + (E + T + int)
⇒ E + (E + int)
⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E

A Second View of a Bottom-Up Parse

$E \rightarrow T$
$E \rightarrow E + T$
$T \rightarrow \texttt{int}$
$T \rightarrow (E)$

$\texttt{int + (int + int + int)}$
$\Rightarrow T\ \texttt{+ (int + int + int)}$
$\Rightarrow E\ \texttt{+ (int + int + int)}$
$\Rightarrow E + (T\ \texttt{+ int + int)}$
$\Rightarrow E + (E\ \texttt{+ int + int)}$
$\Rightarrow E + (E + T\ \texttt{+ int)}$
$\Rightarrow E + (E\ \texttt{+ int)}$
$\Rightarrow E + (E + T)$
$\Rightarrow E + (E)$
$\Rightarrow E + T$
$\Rightarrow E$

- A left-to-right, bottom-up parse is a rightmost derivation traced in reverse.

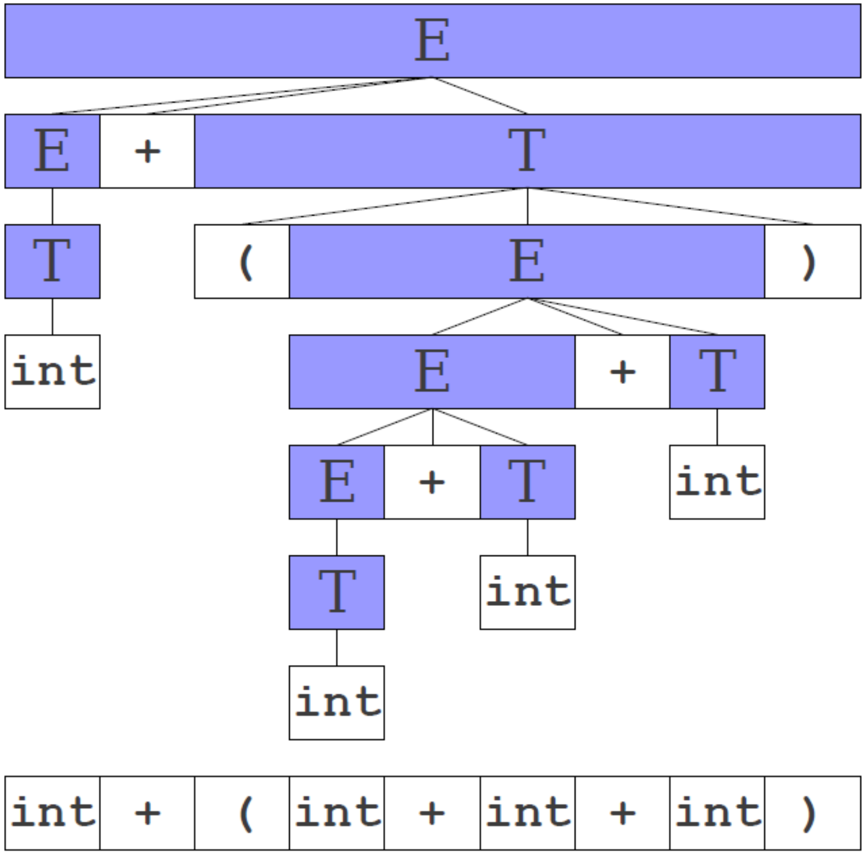
A Third View of a Bottom-Up Parse

```
  int + (int + int + int)
⇒ T + (int + int + int)
⇒ E + (int + int + int)
⇒ E + (T + int + int)
⇒ E + (E + int + int)
⇒ E + (E + T + int)
⇒ E + (E + int)
⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E
```

Each step in this bottom-up parse is called a **reduction**. We **reduce** a substring of the sentential form back to a nonterminal.

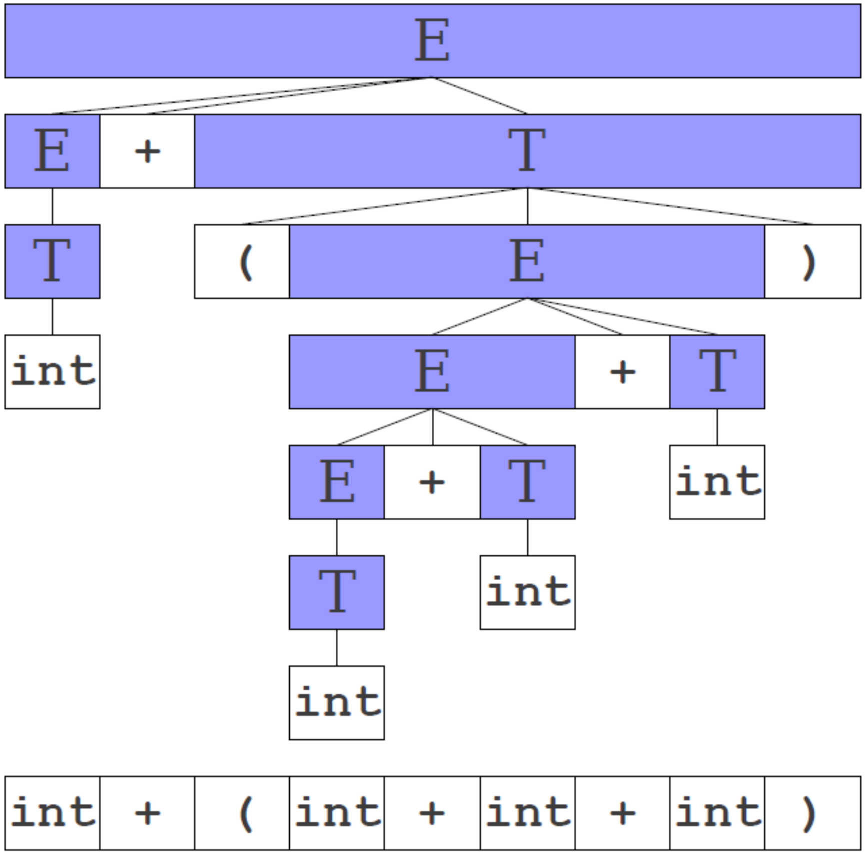
A Third View of a Bottom-Up Parse

int + (int + int + int)
⇒ T + (int + int + int)
⇒ E + (int + int + int)
⇒ E + (T + int + int)
⇒ E + (E + int + int)
⇒ E + (E + T + int)
⇒ E + (E + int)
⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E



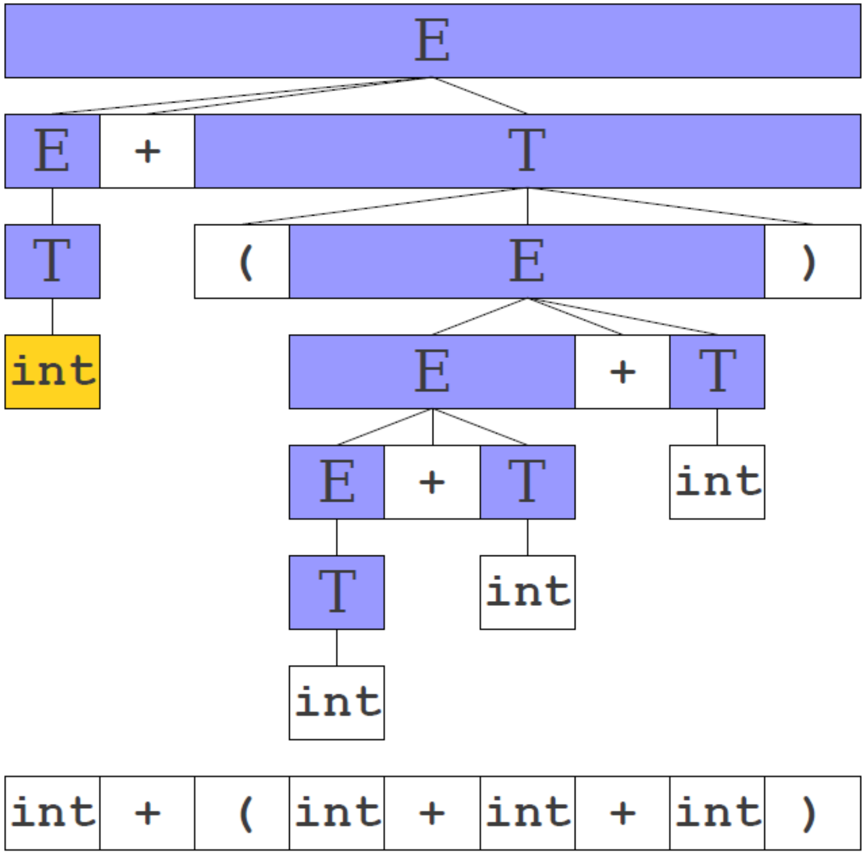
A Third View of a Bottom-Up Parse

int + (int + int + int)
⇒ T + (int + int + int)
⇒ E + (int + int + int)
⇒ E + (T + int + int)
⇒ E + (E + int + int)
⇒ E + (E + T + int)
⇒ E + (E + int)
⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E



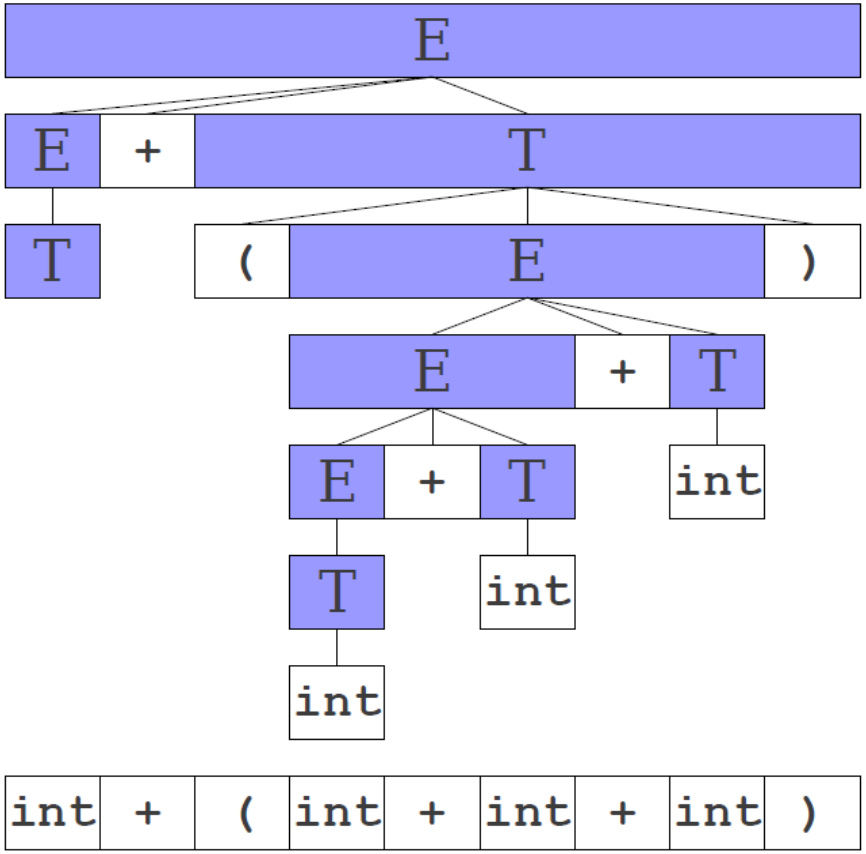
A Third View of a Bottom-Up Parse

int + (int + int + int)
⇒ T + (int + int + int)
⇒ E + (int + int + int)
⇒ E + (T + int + int)
⇒ E + (E + int + int)
⇒ E + (E + T + int)
⇒ E + (E + int)
⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E



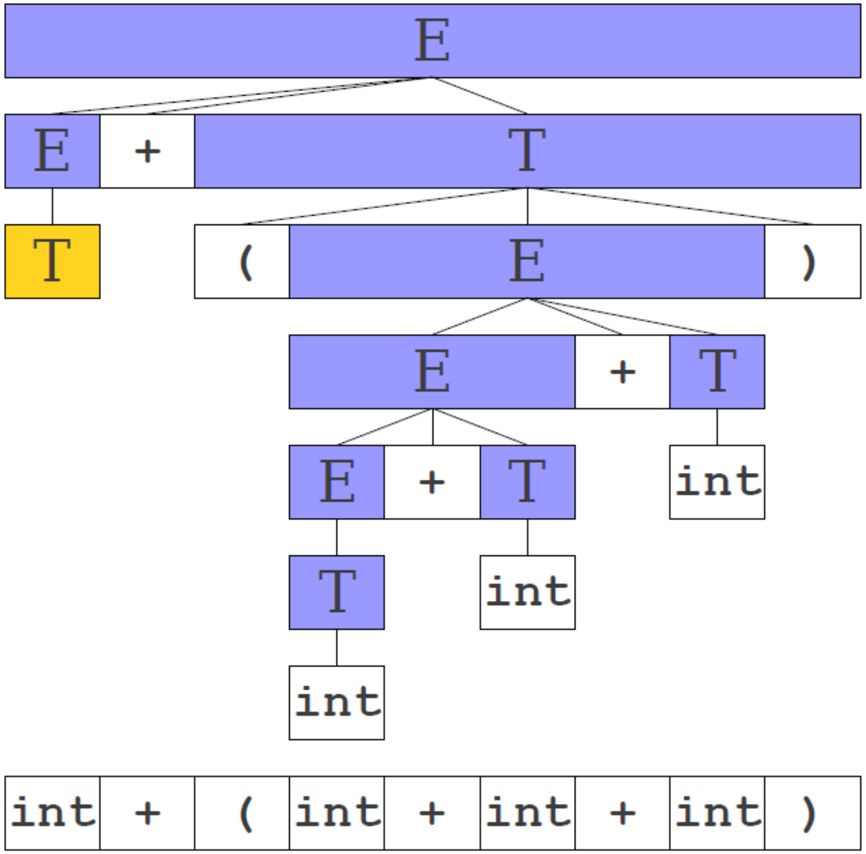
A Third View of a Bottom-Up Parse

⇒ T + (int + int + int)
⇒ E + (int + int + int)
⇒ E + (T + int + int)
⇒ E + (E + int + int)
⇒ E + (E + T + int)
⇒ E + (E + int)
⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E



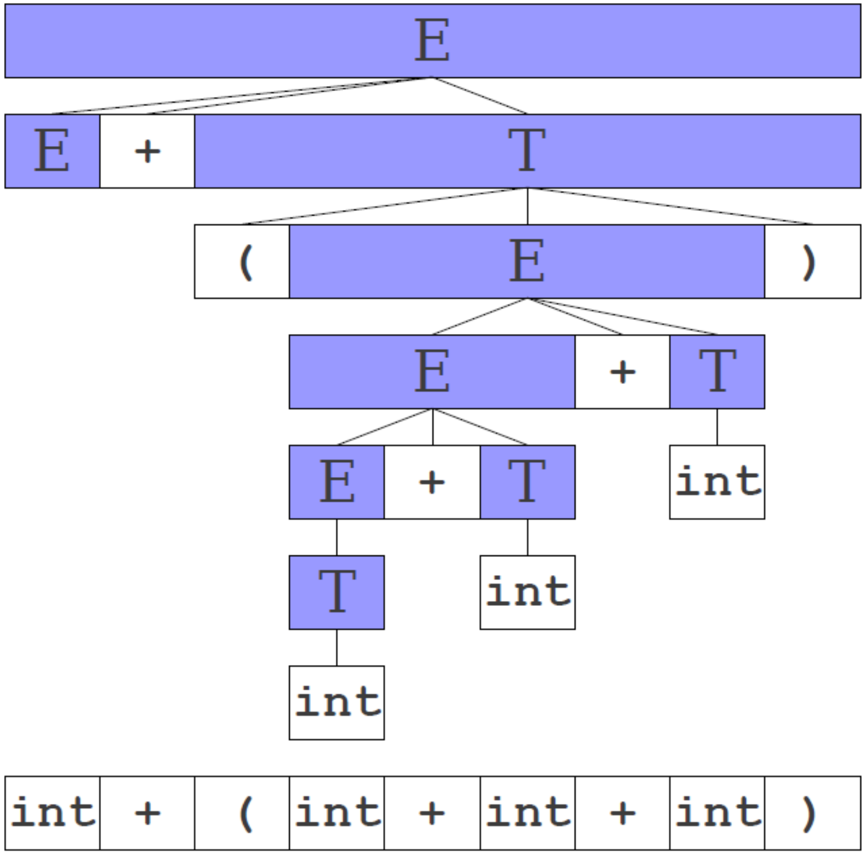
A Third View of a Bottom-Up Parse

⇒ T + (int + int + int)
⇒ E + (int + int + int)
⇒ E + (T + int + int)
⇒ E + (E + int + int)
⇒ E + (E + T + int)
⇒ E + (E + int)
⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E



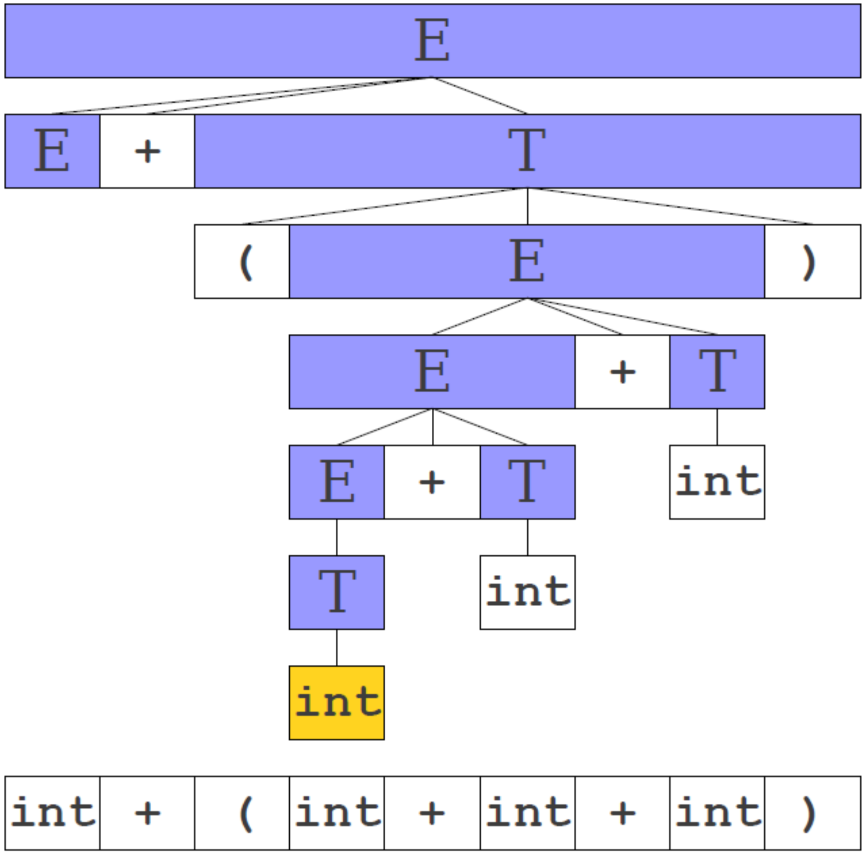
A Third View of a Bottom-Up Parse

⇒ E + (int + int + int)
⇒ E + (T + int + int)
⇒ E + (E + int + int)
⇒ E + (E + T + int)
⇒ E + (E + int)
⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E



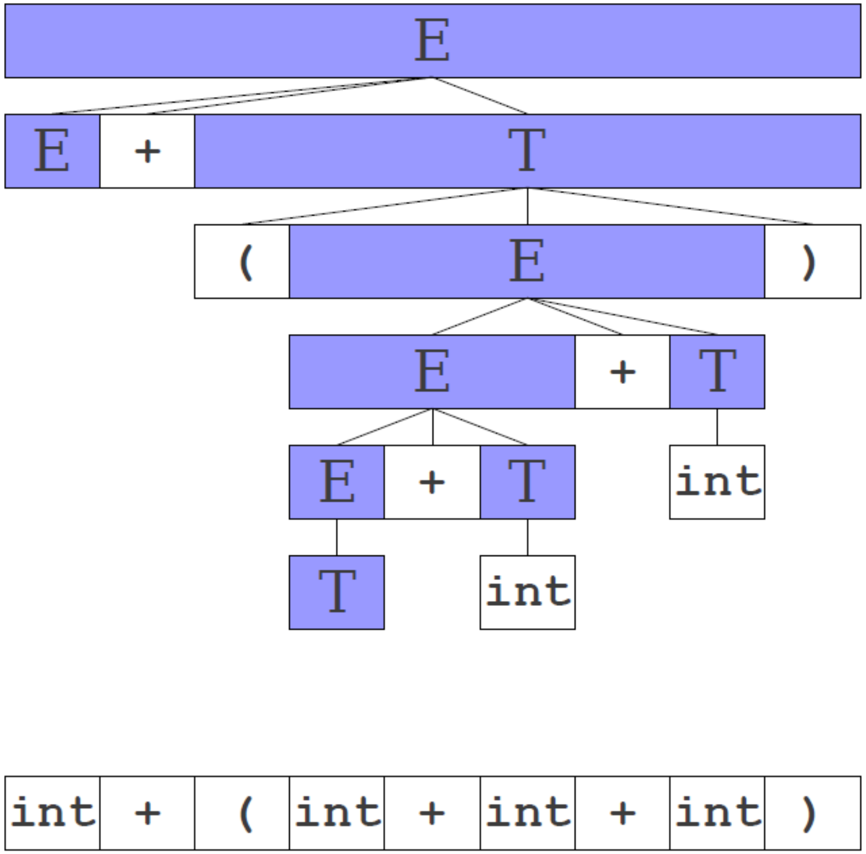
A Third View of a Bottom-Up Parse

$\Rightarrow$ E + (int + int + int)
$\Rightarrow$ E + (T + int + int)
$\Rightarrow$ E + (E + int + int)
$\Rightarrow$ E + (E + T + int)
$\Rightarrow$ E + (E + int)
$\Rightarrow$ E + (E + T)
$\Rightarrow$ E + (E)
$\Rightarrow$ E + T
$\Rightarrow$ E



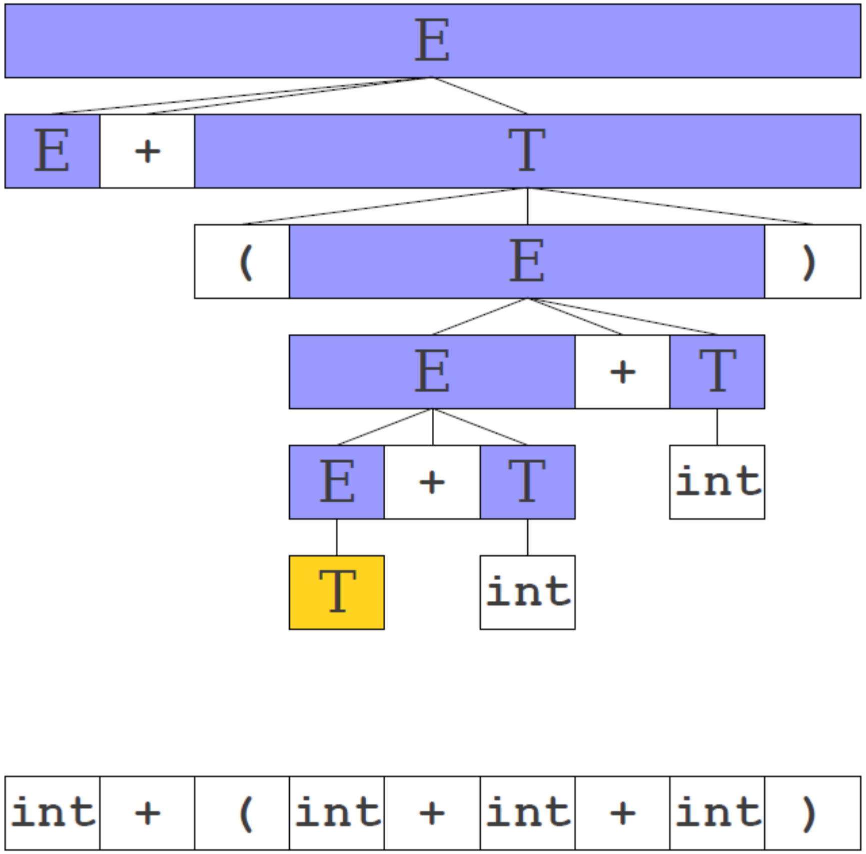
A Third View of a Bottom-Up Parse

$\Rightarrow$ E + (T + int + int)
$\Rightarrow$ E + (E + int + int)
$\Rightarrow$ E + (E + T + int)
$\Rightarrow$ E + (E + int)
$\Rightarrow$ E + (E + T)
$\Rightarrow$ E + (E)
$\Rightarrow$ E + T
$\Rightarrow$ E

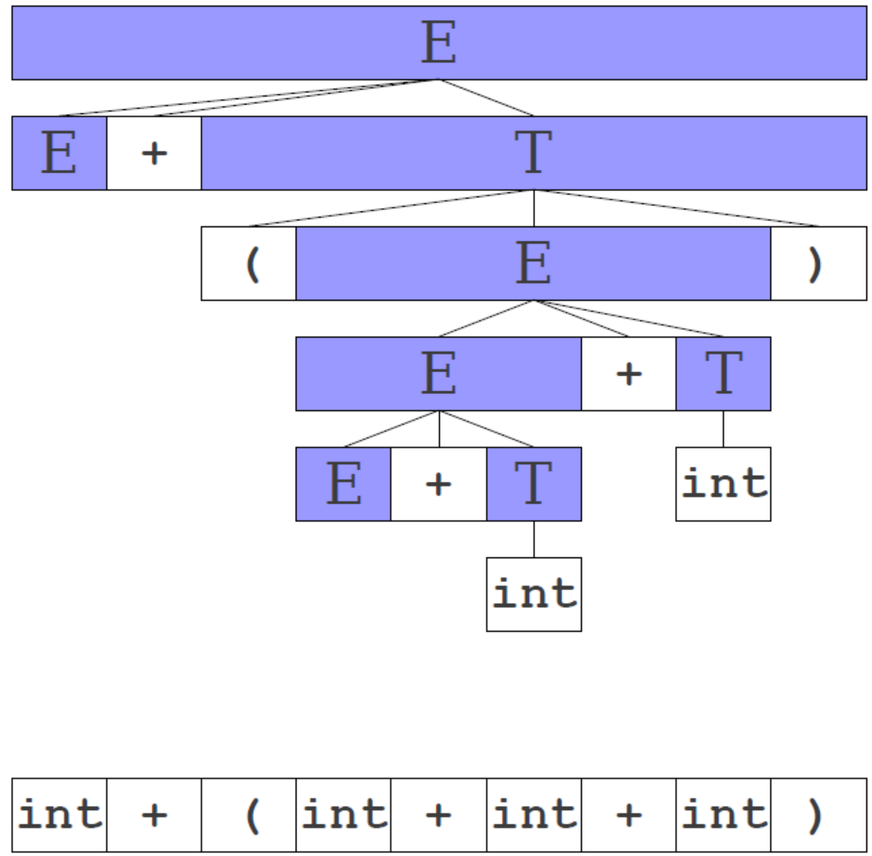


A Third View of a Bottom-Up Parse

$\Rightarrow$ E + (T + int + int)
$\Rightarrow$ E + (E + int + int)
$\Rightarrow$ E + (E + T + int)
$\Rightarrow$ E + (E + int)
$\Rightarrow$ E + (E + T)
$\Rightarrow$ E + (E)
$\Rightarrow$ E + T
$\Rightarrow$ E



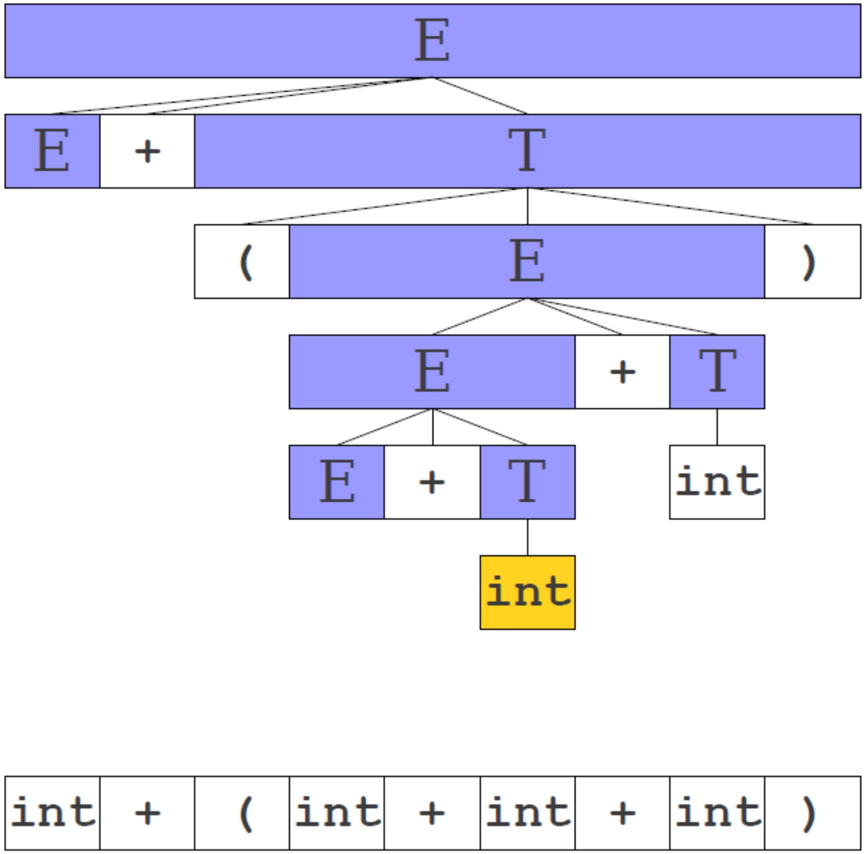
A Third View of a Bottom-Up Parse



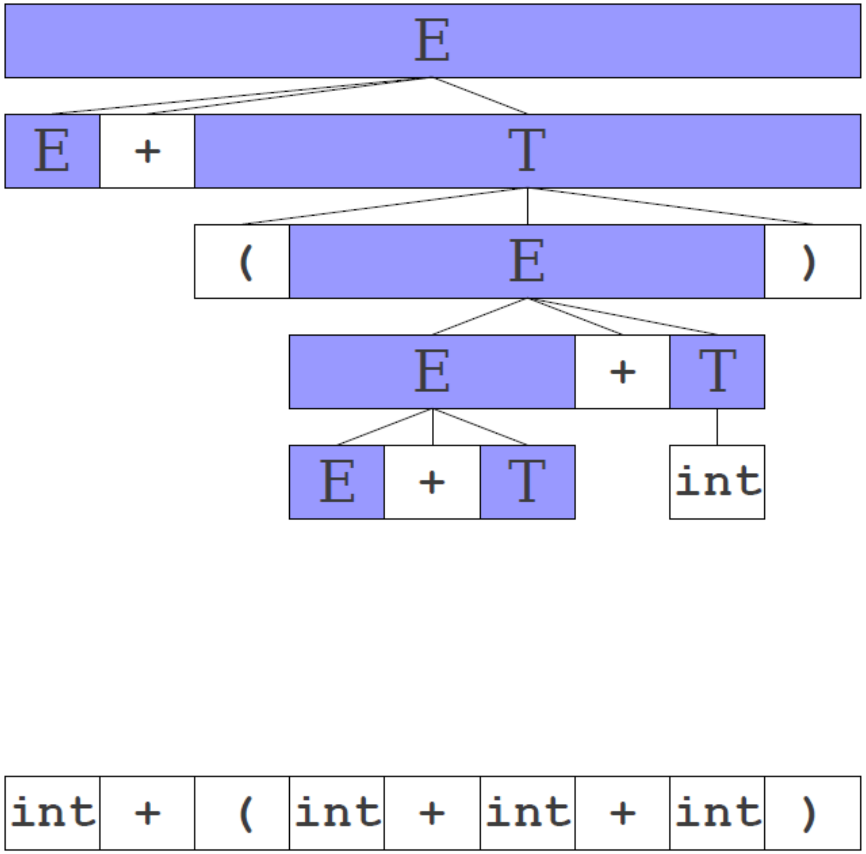
⇒ E + (E + int + int)
⇒ E + (E + T + int)
⇒ E + (E + int)
⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E

A Third View of a Bottom-Up Parse

$\Rightarrow$ E + (E + int + int)
$\Rightarrow$ E + (E + T + int)
$\Rightarrow$ E + (E + int)
$\Rightarrow$ E + (E + T)
$\Rightarrow$ E + (E)
$\Rightarrow$ E + T
$\Rightarrow$ E

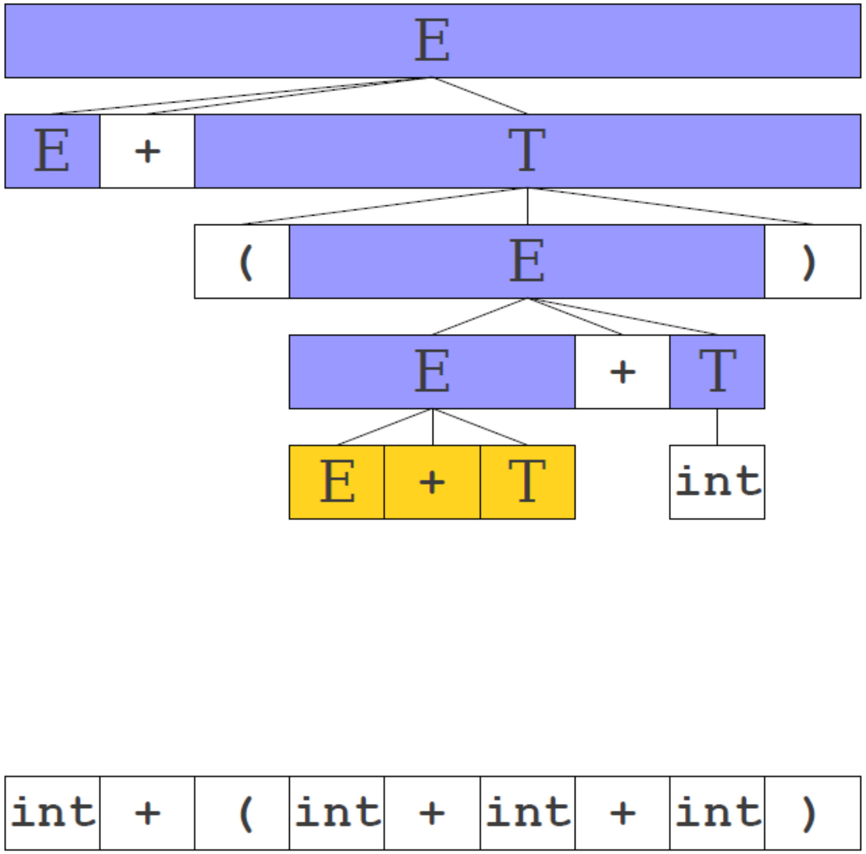


A Third View of a Bottom-Up Parse



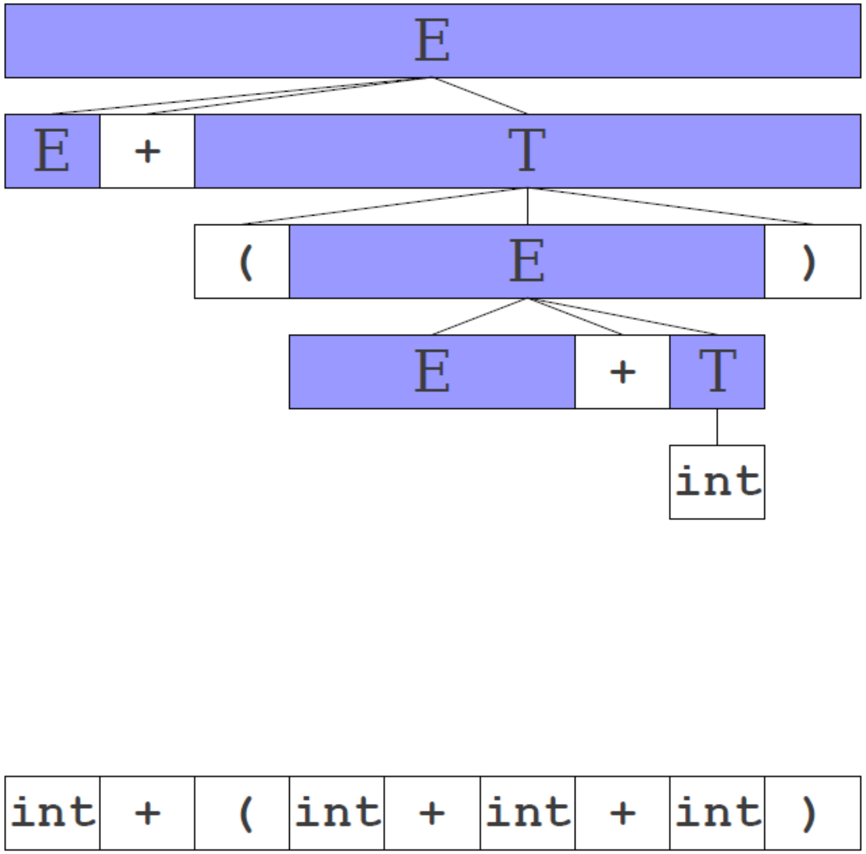
⇒ E + (E + T + int)
⇒ E + (E + int)
⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E

A Third View of a Bottom-Up Parse



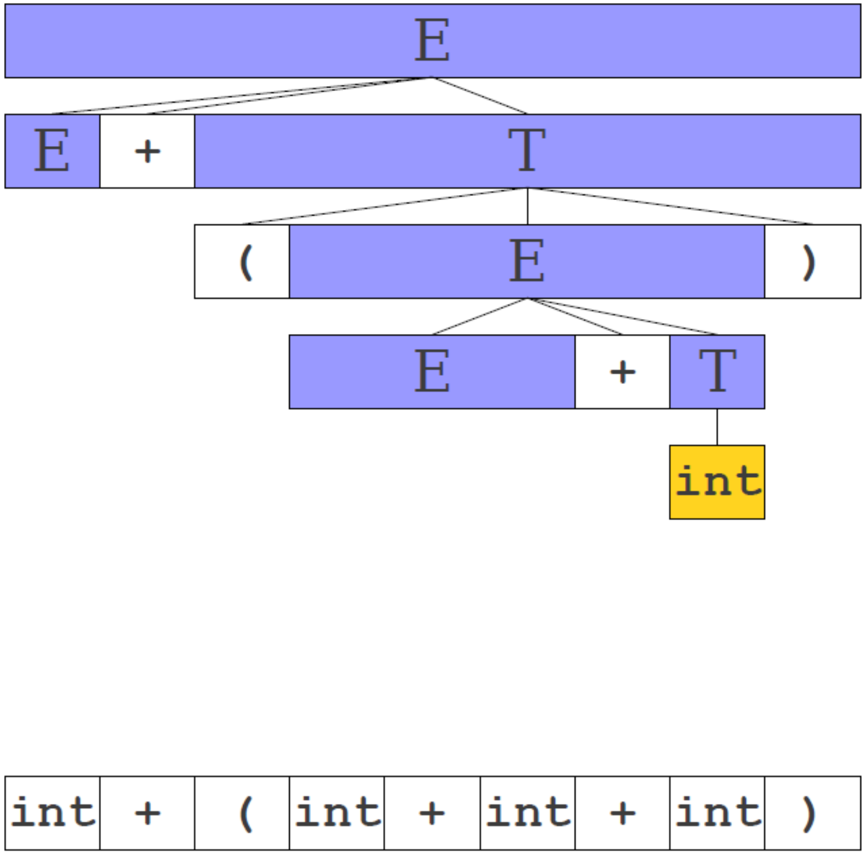
⇒ E + (**E + T** + int)
⇒ E + (E + int)
⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E

A Third View of a Bottom-Up Parse



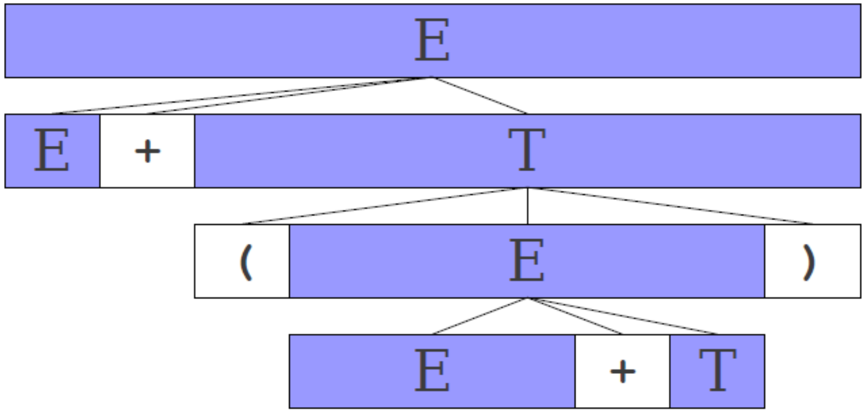
$\Rightarrow$ E + (E + int)
$\Rightarrow$ E + (E + T)
$\Rightarrow$ E + (E)
$\Rightarrow$ E + T
$\Rightarrow$ E

A Third View of a Bottom-Up Parse

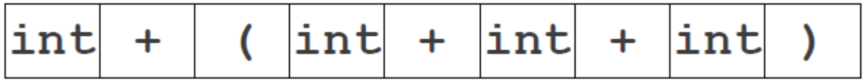


⇒ E + (E + int)
⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E

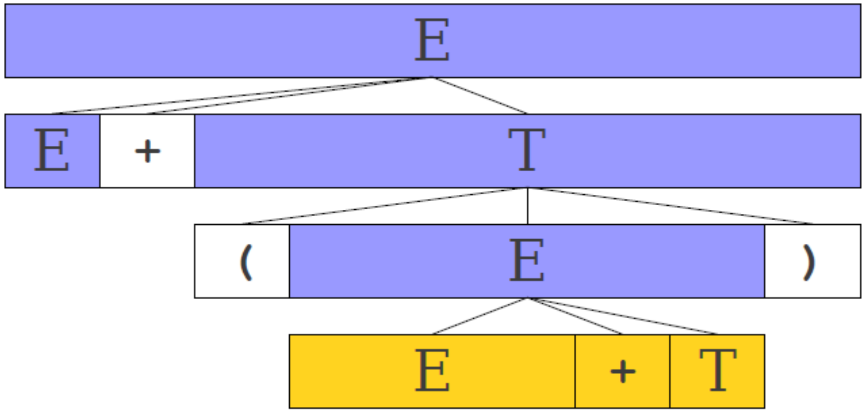
A Third View of a Bottom-Up Parse



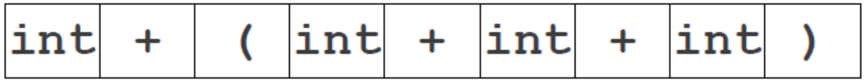
⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E



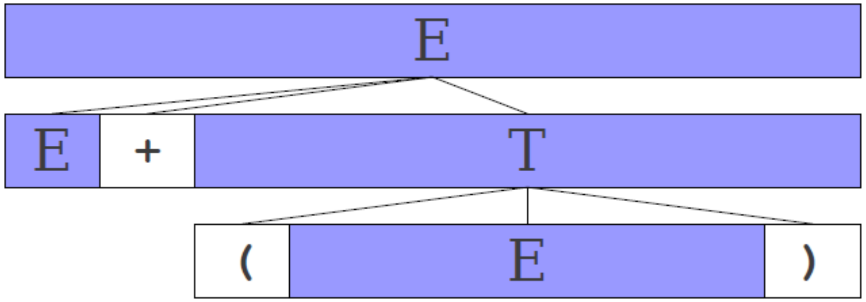
A Third View of a Bottom-Up Parse



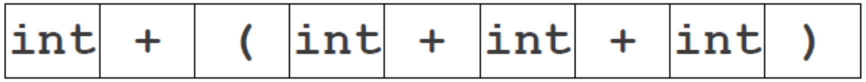
⇒ E + (**E + T**)
⇒ E + (E)
⇒ E + T
⇒ E



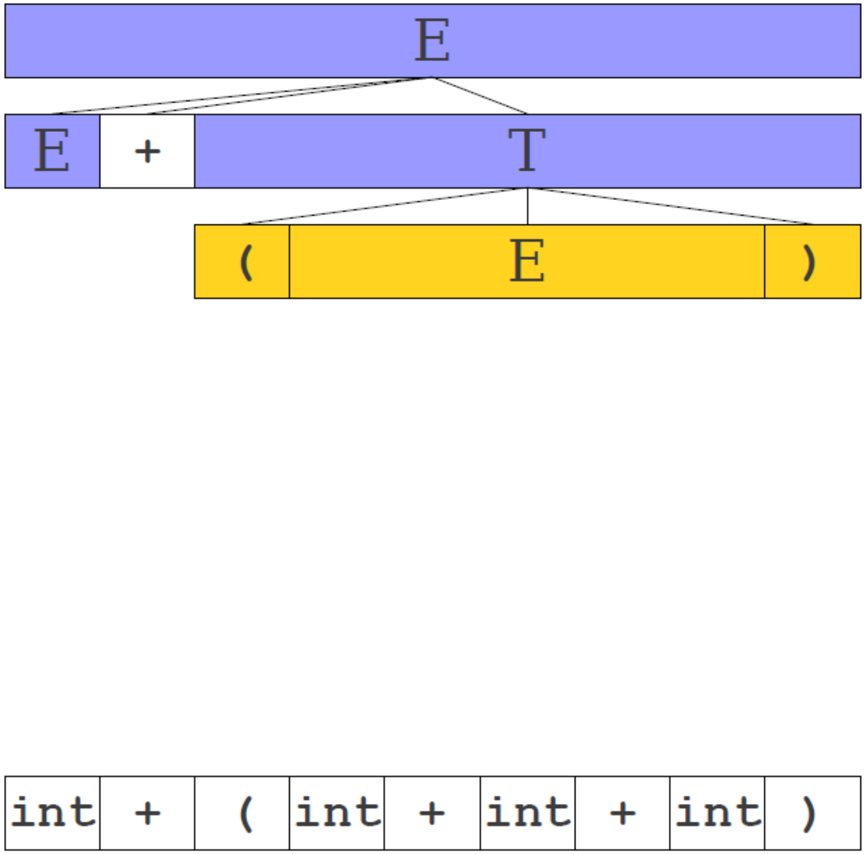
A Third View of a Bottom-Up Parse



$\Rightarrow$ **E + (E)**
$\Rightarrow$ **E + T**
$\Rightarrow$ **E**

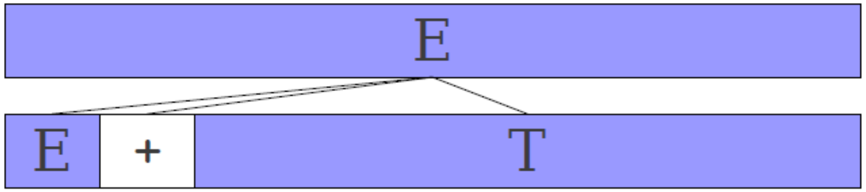


A Third View of a Bottom-Up Parse



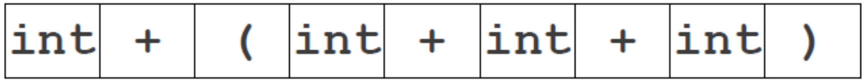
⇒ E + **(E)**
⇒ E + T
⇒ E

A Third View of a Bottom-Up Parse

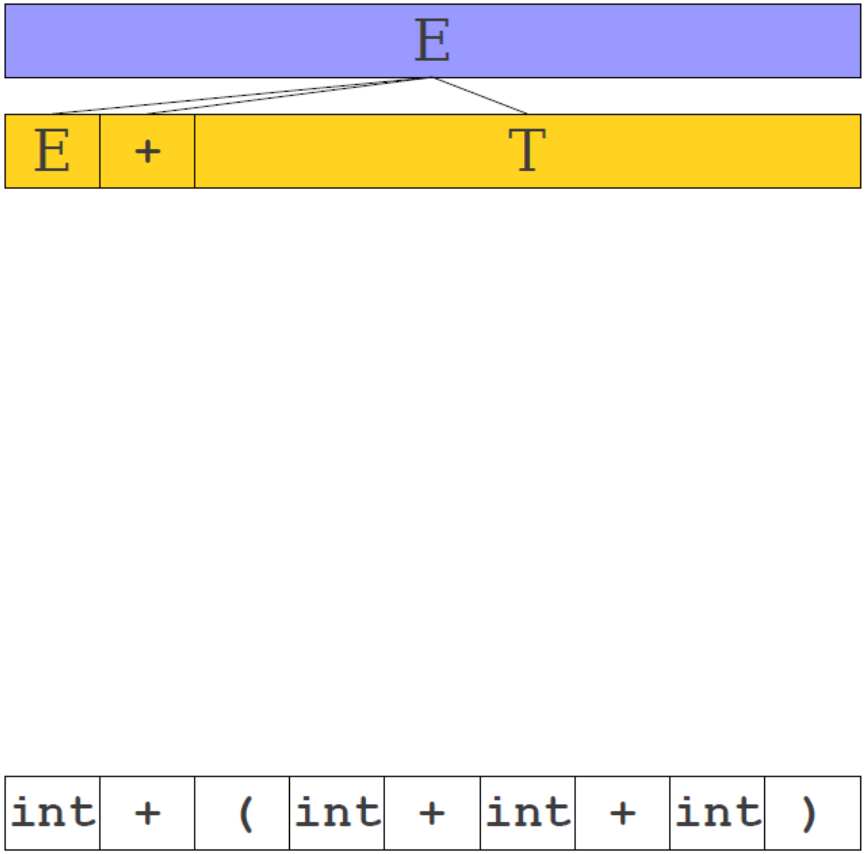


$\Rightarrow$ **E + T**

$\Rightarrow$ **E**



A Third View of a Bottom-Up Parse



$\Rightarrow$ **E + T**

$\Rightarrow$ **E**

A Third View of a Bottom-Up Parse

E

$\Rightarrow$ **E**

int	+	(	int	+	int	+	int	)

Bottom-up Parsing

- "Shift-Reduce" Parsing
- Reduce a string to the start symbol of the grammar.
- At every step a particular sub-string is matched (in left-to-right fashion) to the right side of some production and then it is substituted by the non-terminal in the left hand side of the production.

Consider:

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

abbcde
aAbcde
aAde
aABe
S

Reverse order

Rightmost Derivation:

$S \Rightarrow aABe \Rightarrow aAde \Rightarrow aAbcde \Rightarrow abbcde$

Bottom-up Parsing

- Thus this process of bottom-up parsing is like tracing out the rightmost derivations in reverse.
- The bottom-up parsing as the process of “reducing” a token string to the start symbol of the grammar.
- At each *reduction*, the token string matching the RHS of a production is replaced by the LHS non-terminal of that production.
- The key decisions during bottom-up parsing are about when to reduce and about what production to apply.

Shift-reduce parsing

- A shift-reduce parser tries to reduce the given input string into the starting symbol.

a string $\rightarrow$ the starting symbol
reduced to

- At each reduction step, a substring of the input matching to the right side of a production rule is replaced by the non-terminal at the left side of that production rule.

- If the substring is chosen correctly, the right most derivation of that string is created in the reverse order.

Rightmost Derivation: $S \Rightarrow \omega$

Shift-Reduce Parser finds: $\omega \Leftarrow \ldots \Leftarrow S$

Handle

- Handle of a string: Substring that matches the RHS of some production AND whose reduction to the non-terminal on the LHS is a step along the reverse of some rightmost derivation.

- A **handle** of a right sentential form γ ($\equiv \alpha\beta\omega$) is a production rule $A \rightarrow \beta$ and a position of γ

  where the string β may be found and replaced by A to produce the previous right-sentential form in a rightmost derivation of γ.

  $S \Rightarrow \alpha A\omega \Rightarrow \alpha\beta\omega$

i.e. $A \rightarrow \beta$ is a handle of $\alpha\beta\gamma$ at the location immediately after the end of α,

- If the grammar is unambiguous, then every right-sentential form of the grammar has exactly one handle.
- ω is a string of terminals

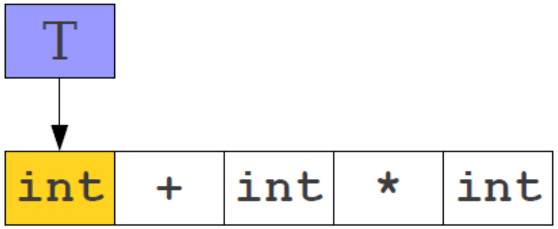
A Detail about Handles

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow \texttt{int}$
$T \rightarrow (E)$

int	+	int	*	int

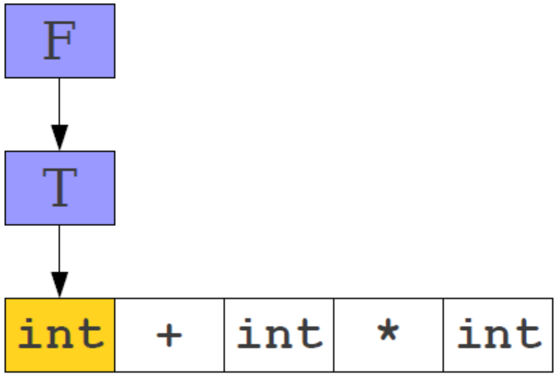
A Detail about Handles

E → **F**
E → **E** `+` **F**
F → **F** `*` **T**
F → **T**
T → `int`
T → `(`**E**`)`



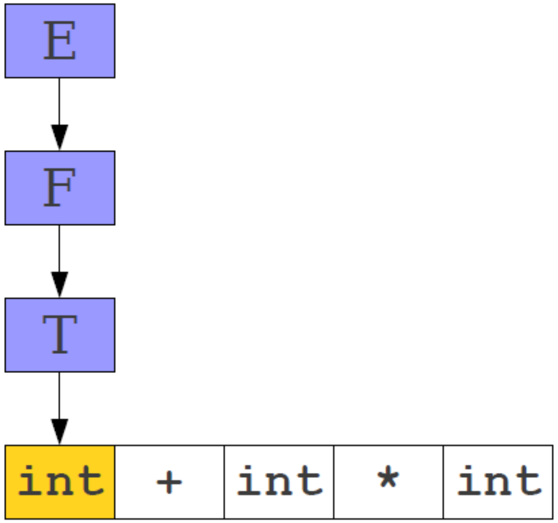
A Detail about Handles

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow$ int
$T \rightarrow$ (E)



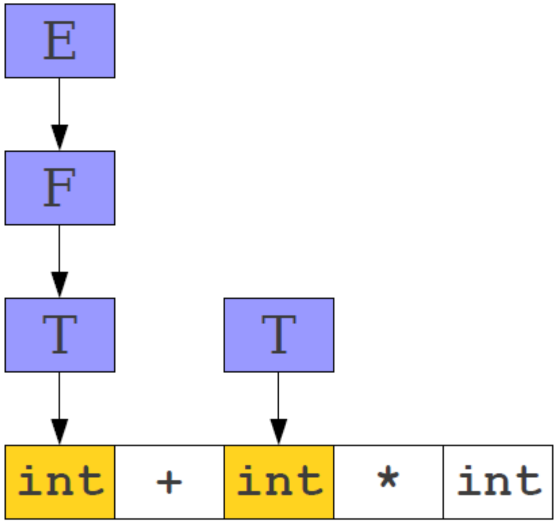
A Detail about Handles

$E \to F$
$E \to E + F$
$F \to F * T$
$F \to T$
$T \to$ int
$T \to$ (E)



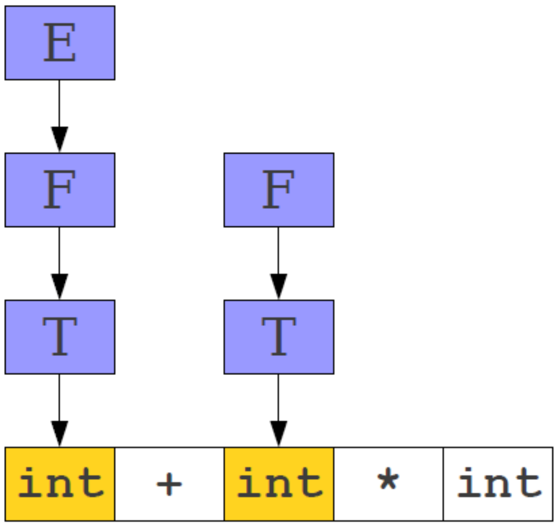
A Detail about Handles

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow$ int
$T \rightarrow (E)$



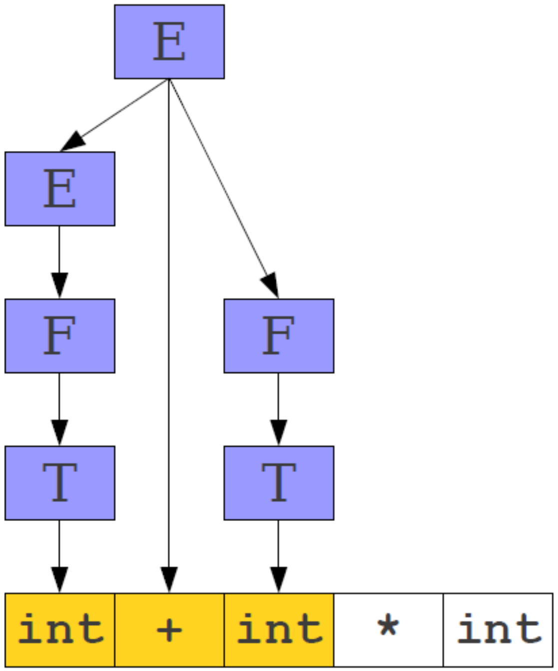
A Detail about Handles

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow$ int
$T \rightarrow (E)$



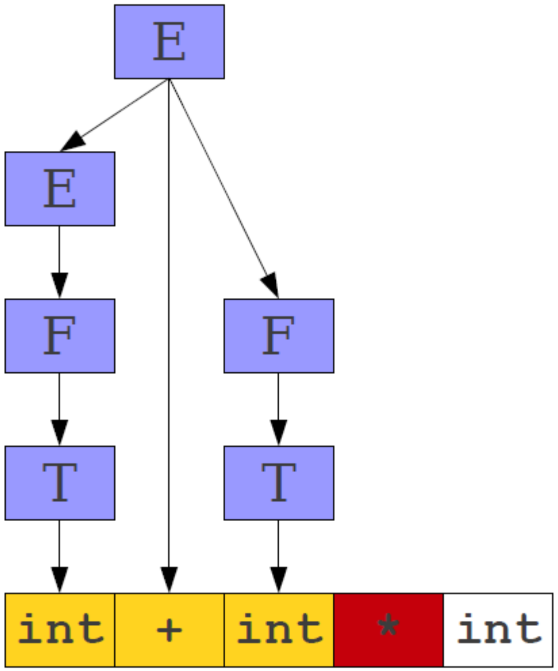
A Detail about Handles

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow$ `int`
$T \rightarrow$ `(E)`



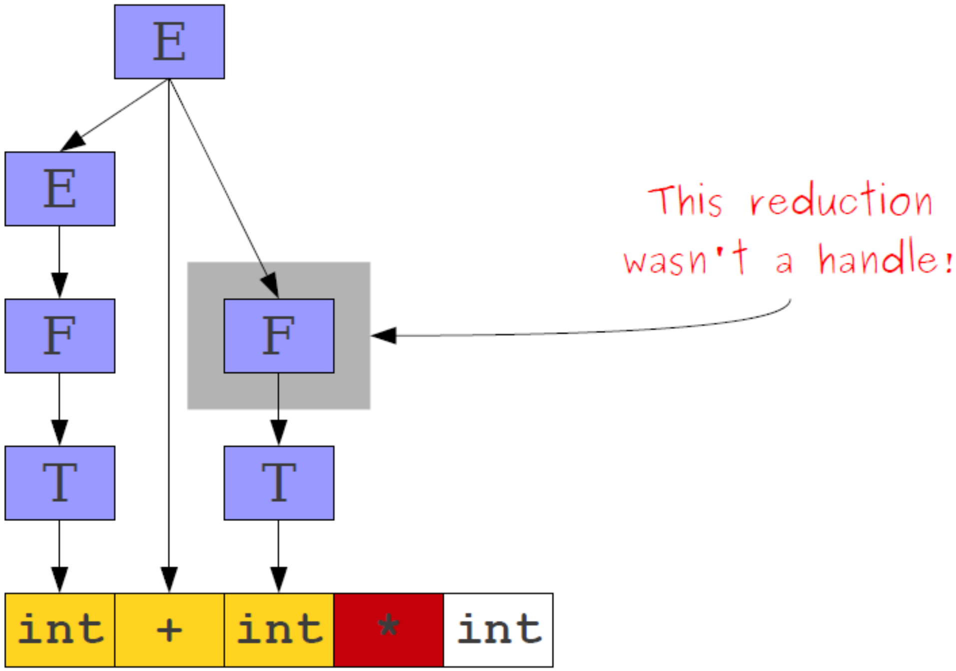
A Detail about Handles

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow$ int
$T \rightarrow (E)$



A Detail about Handles

E → F
E → E + F
F → F * T
F → T
T → int
T → (E)



- The leftmost reduction isn't always the handle.

Finding Handles

- Where do we look for handles?
  - Where in the string might the handle be?
- How do we search for possible handles?
  - Once we know where to search, how do we identify candidate handles?
- How do we recognize handles?
  - Once we've found a candidate handle, how do we check that it really is the handle?

- Question One:
  - Where are handles?

Where are Handles?

- Recall: A left-to-right, bottom-up parse traces a rightmost derivation in reverse.
- Each time we do a reduction, we are reversing a production applied to the *rightmost* nonterminal symbol.
- Suppose that our current sentential form is $\alpha\gamma\omega$, where γ is the handle and $A \rightarrow \gamma$ is a production rule.
- After reducing γ back to **A**, we have the string $\alpha A \omega$.
- Thus ω must consist purely of terminals, since otherwise the reduction we just did was not for the rightmost terminal.

Why This Matters

- Suppose we want to parse the string γ.
- We will break γ into two parts, α and ω, where
  - α consists of both terminals and nonterminals, and
  - ω consists purely of terminals.
- Our search for handles will concentrate purely in α.
- As necessary, we will start moving terminals from ω over into α.

Shift/Reduce Parsing

- The bottom-up parsers we will consider are called **shift/reduce** parsers.
  - Contrast with the LL(1) **predict/match** parser.
- Idea: Split the input into two parts:
  - Left substring is our work area; all handles must be here.
  - Right substring is input we have not yet processed; consists purely of terminals.
- At each point, decide whether to:
  - Move a terminal across the split (**shift**)
  - Reduce a handle (**reduce**)

A Sample Shift/Reduce Parse

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow \texttt{int}$
$T \rightarrow (E)$

int	+	int	*	int	+	int

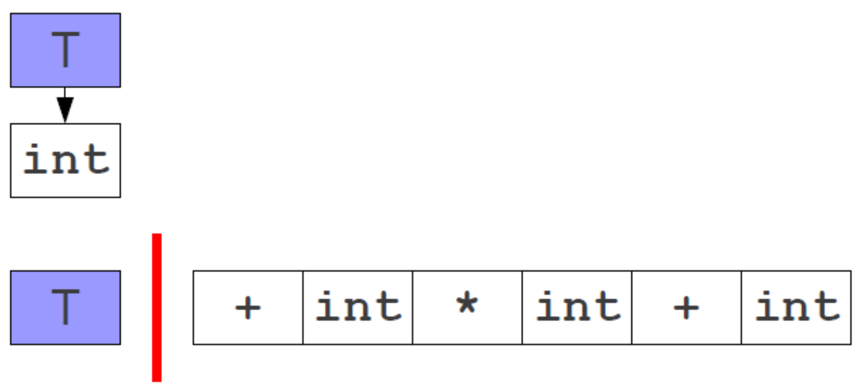
A Sample Shift/Reduce Parse

$\mathbf{E} \rightarrow \mathbf{F}$
$\mathbf{E} \rightarrow \mathbf{E} + \mathbf{F}$
$\mathbf{F} \rightarrow \mathbf{F} * \mathbf{T}$
$\mathbf{F} \rightarrow \mathbf{T}$
$\mathbf{T} \rightarrow$ `int`
$\mathbf{T} \rightarrow$ `(E)`

int	|	+	int	*	int	+	int

A Sample Shift/Reduce Parse

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow$ int
$T \rightarrow$ (E)



A Sample Shift/Reduce Parse

$\mathbf{E} \rightarrow \mathbf{F}$

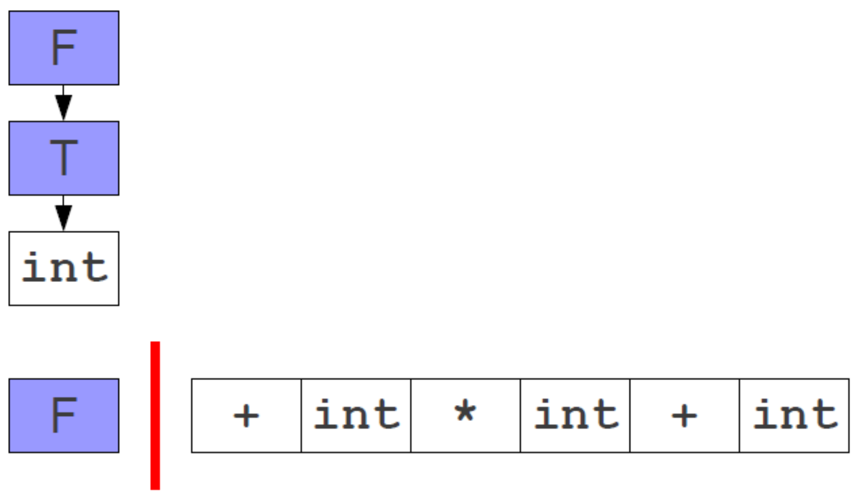
$\mathbf{E} \rightarrow \mathbf{E} + \mathbf{F}$

$\mathbf{F} \rightarrow \mathbf{F} * \mathbf{T}$

$\mathbf{F} \rightarrow \mathbf{T}$

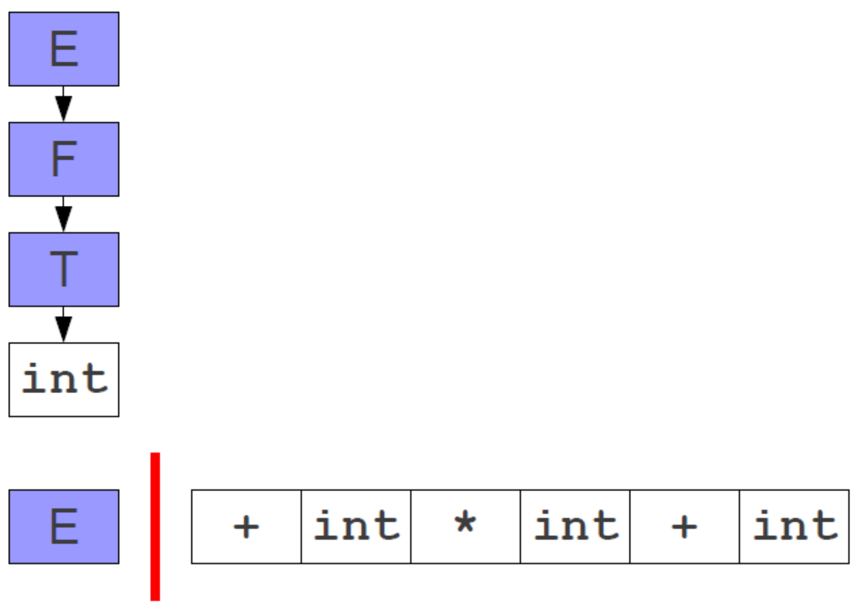
$\mathbf{T} \rightarrow$ `int`

$\mathbf{T} \rightarrow$ `(`$\mathbf{E}$`)`



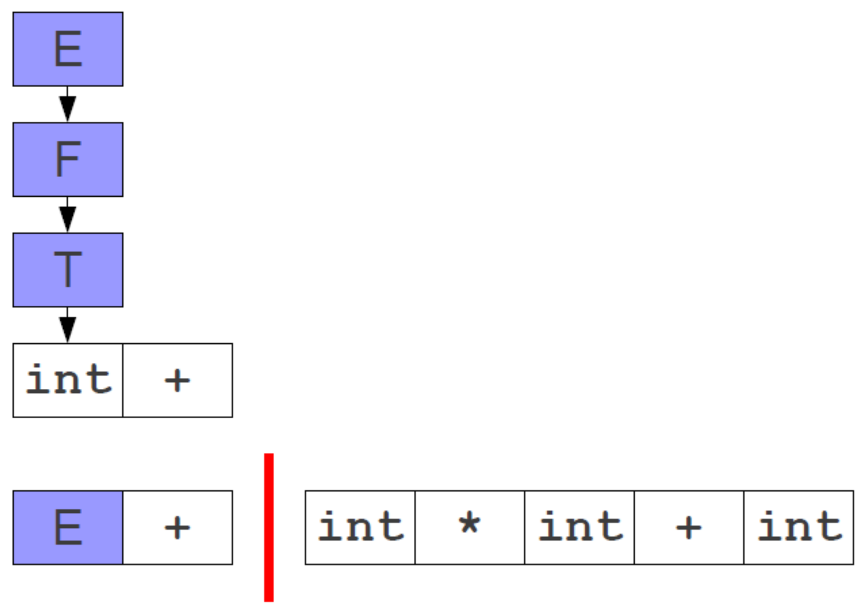
A Sample Shift/Reduce Parse

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow \texttt{int}$
$T \rightarrow (E)$



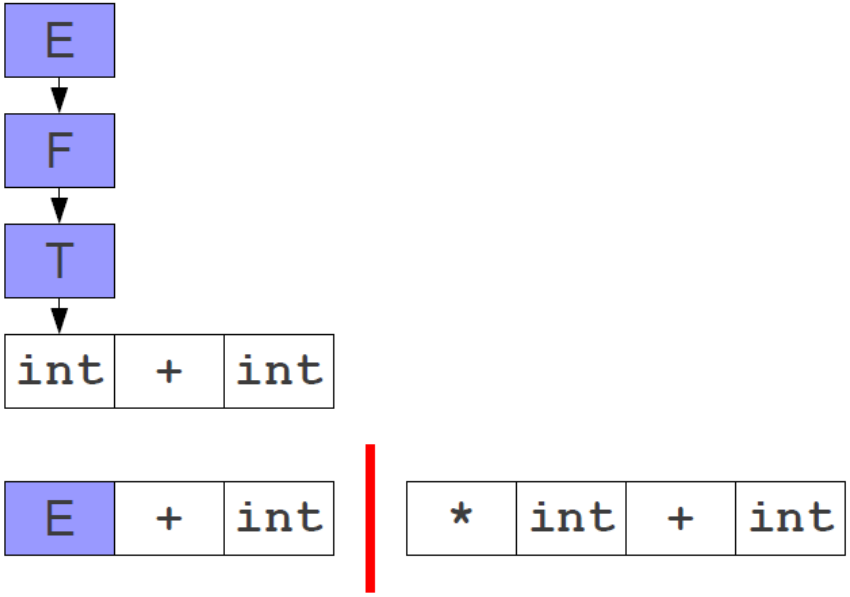
A Sample Shift/Reduce Parse

$E \to F$
$E \to E + F$
$F \to F * T$
$F \to T$
$T \to \texttt{int}$
$T \to (E)$



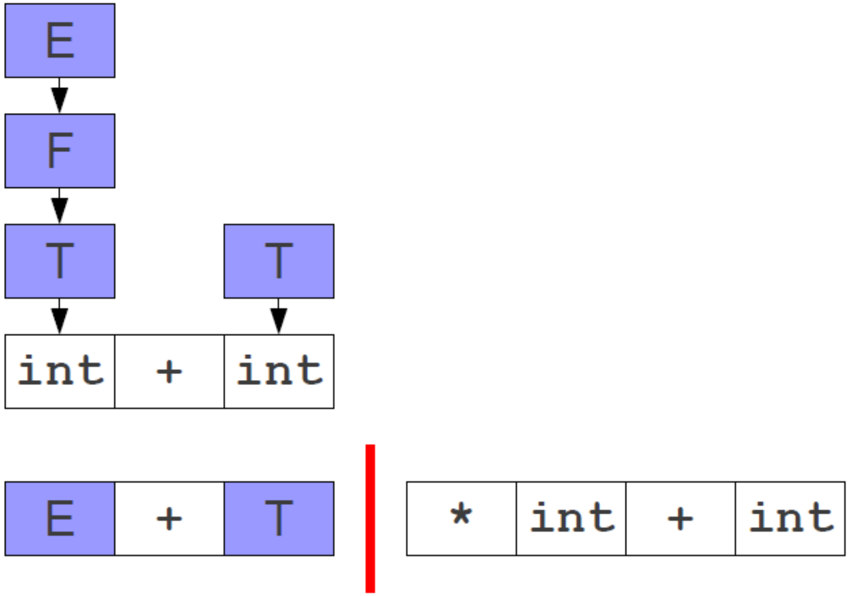
A Sample Shift/Reduce Parse

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow$ int
$T \rightarrow$ (E)



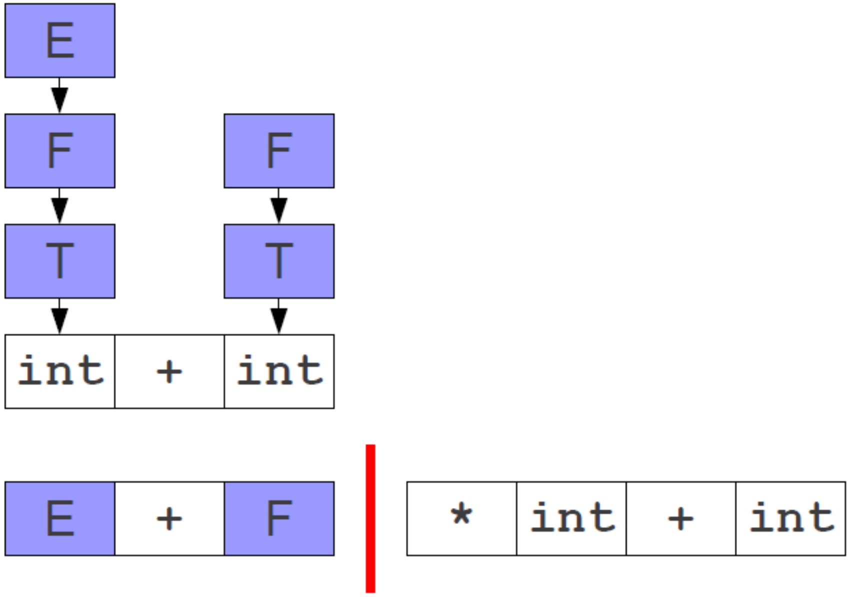
A Sample Shift/Reduce Parse

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow$ int
$T \rightarrow (E)$



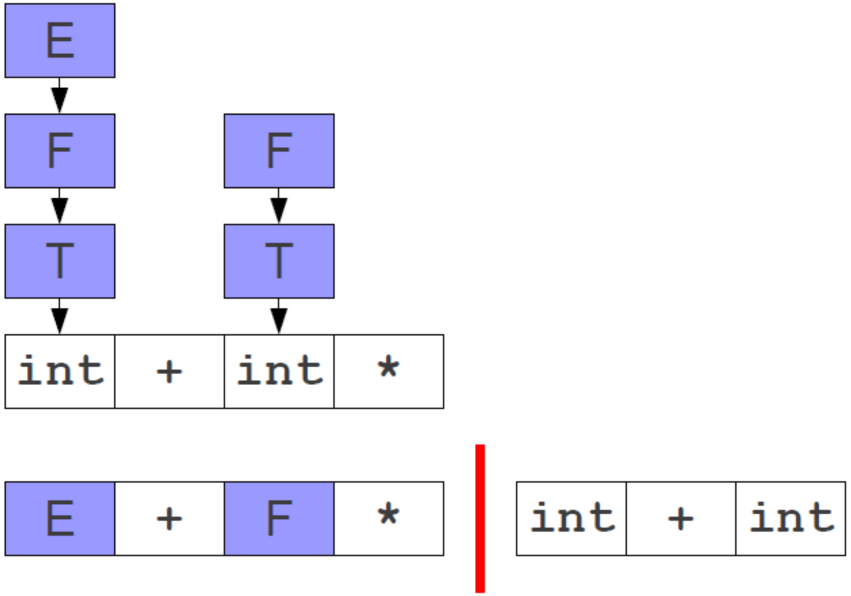
A Sample Shift/Reduce Parse

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow$ `int`
$T \rightarrow (E)$



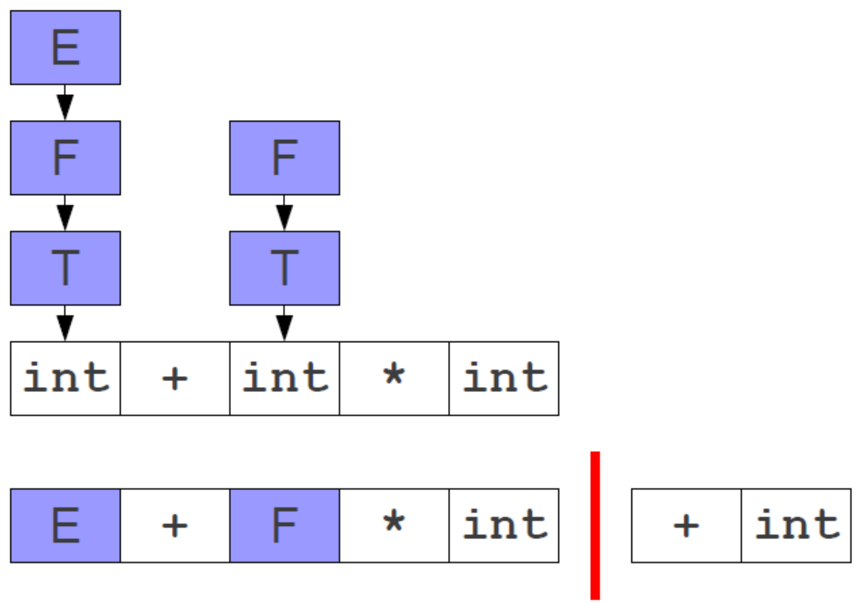
A Sample Shift/Reduce Parse

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow$ int
$T \rightarrow (E)$



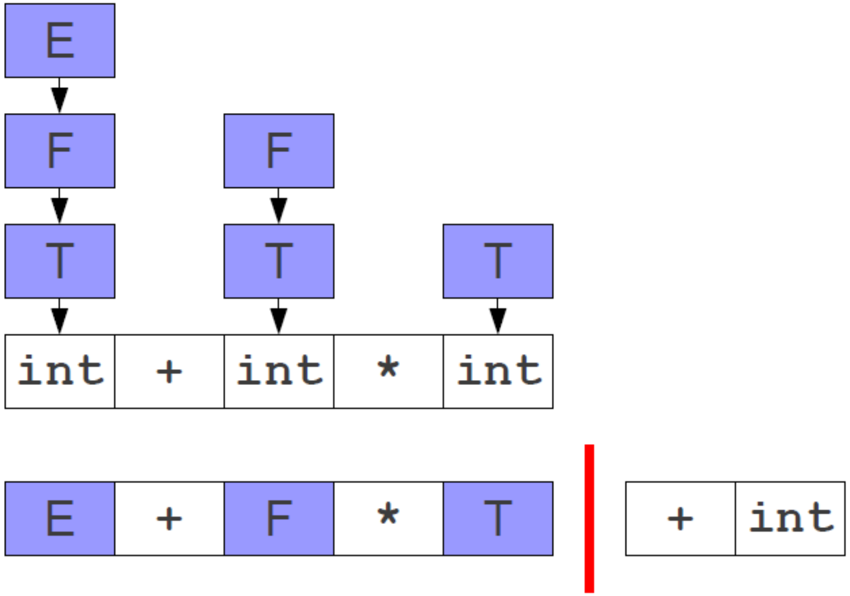
A Sample Shift/Reduce Parse

$E \rightarrow F$
$E \rightarrow E$ + F
$F \rightarrow F$ * T
$F \rightarrow T$
$T \rightarrow$ int
$T \rightarrow$ (E)



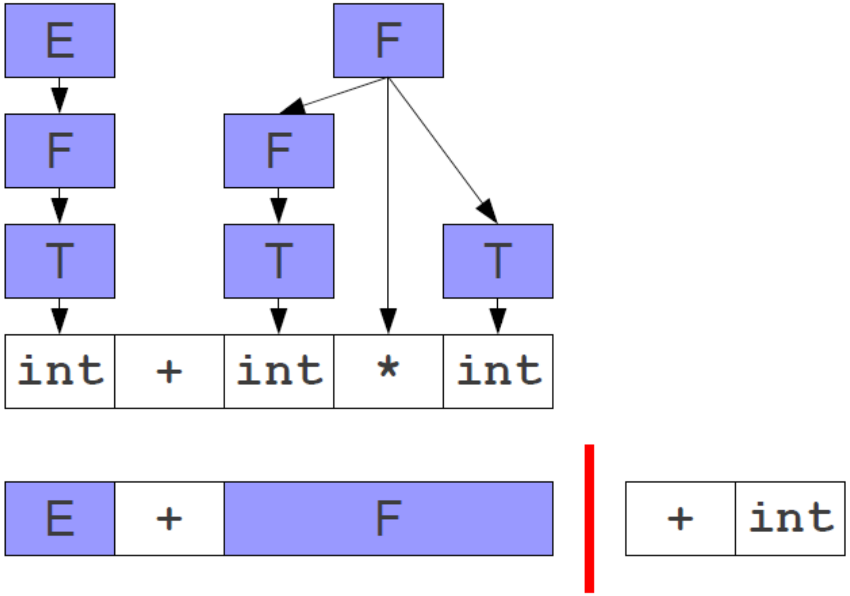
A Sample Shift/Reduce Parse

$E \rightarrow F$
$E \rightarrow E$ + F
$F \rightarrow F$ * T
$F \rightarrow T$
$T \rightarrow$ int
$T \rightarrow$ (E)



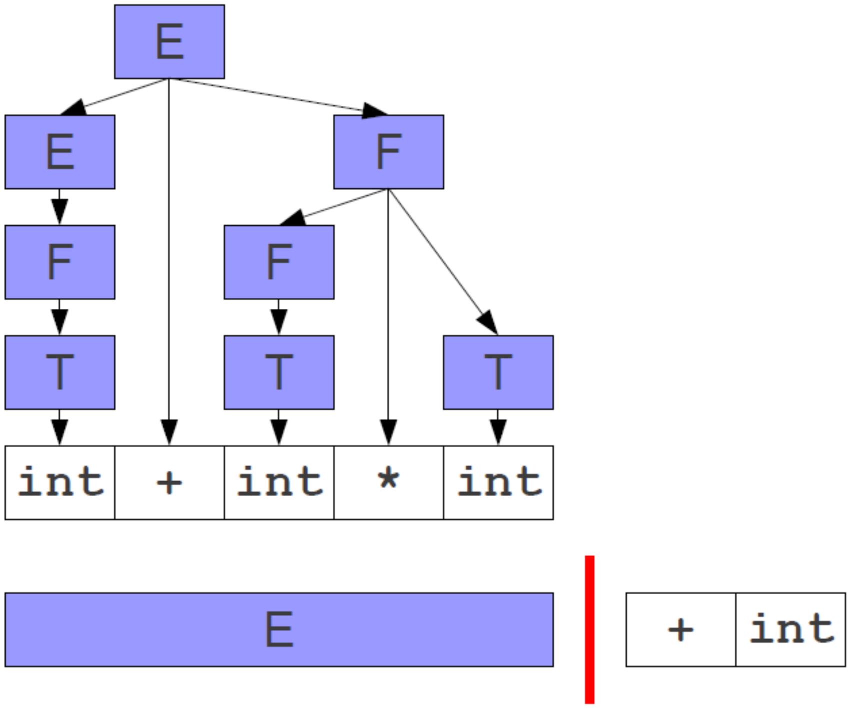
A Sample Shift/Reduce Parse

E → F
E → E + F
F → F * T
F → T
T → int
T → (E)



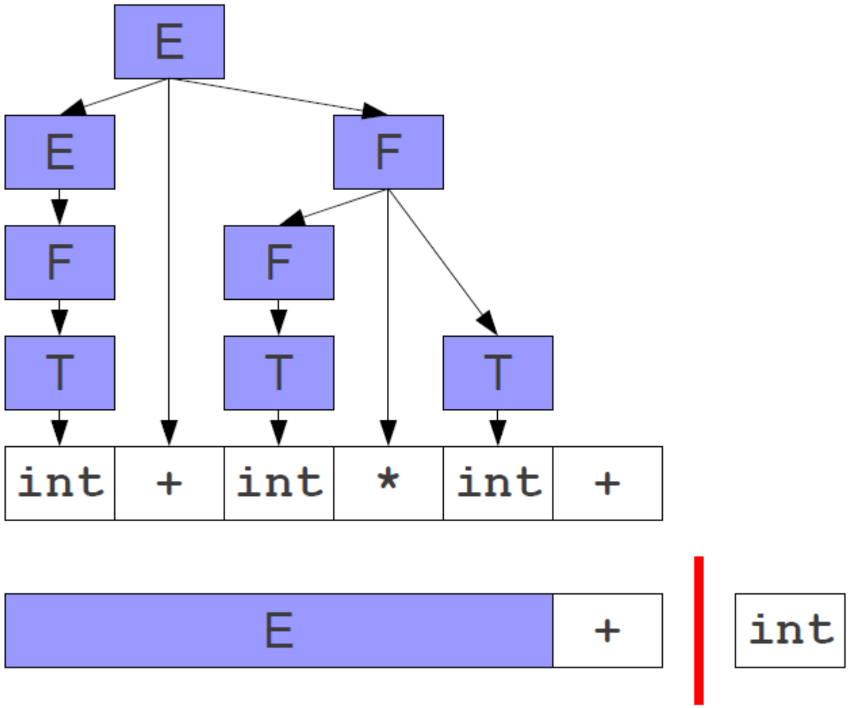
A Sample Shift/Reduce Parse

E → F
E → E + F
F → F * T
F → T
T → int
T → (E)



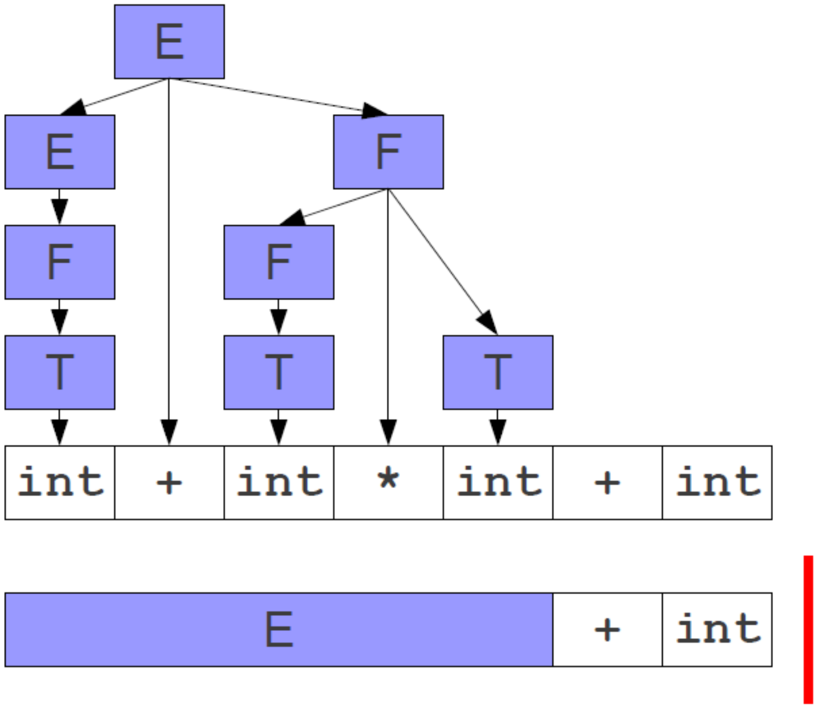
A Sample Shift/Reduce Parse

$E \rightarrow F$
$E \rightarrow E$ + F
$F \rightarrow F$ * T
$F \rightarrow T$
$T \rightarrow$ int
$T \rightarrow$ (E)



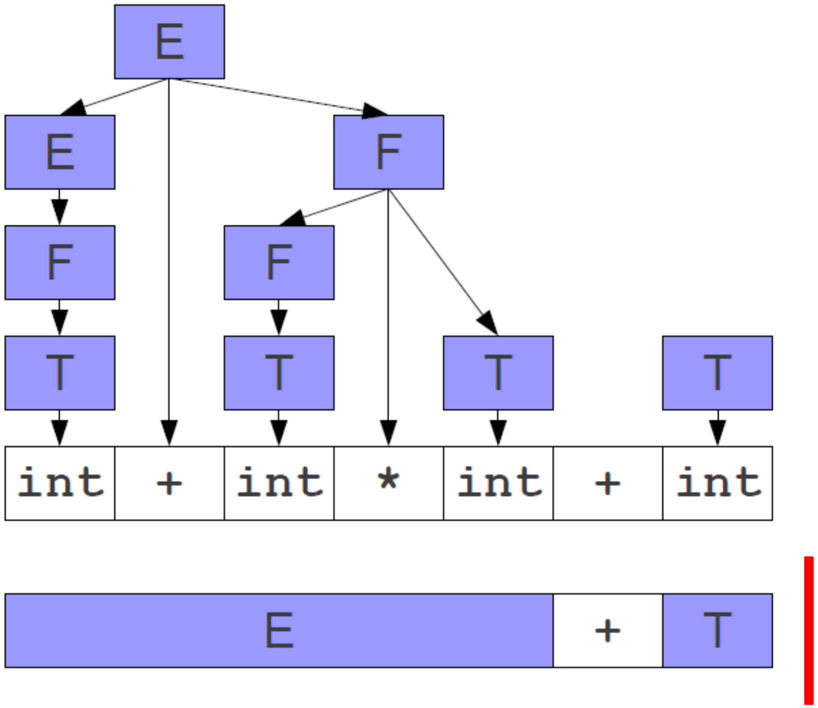
A Sample Shift/Reduce Parse

E → F
E → E + F
F → F * T
F → T
T → int
T → (E)



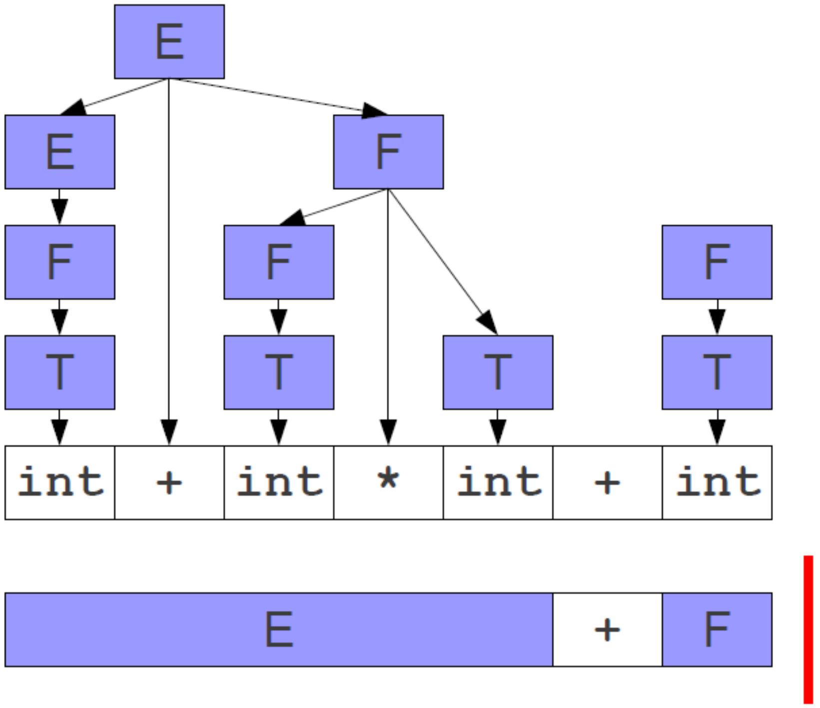
A Sample Shift/Reduce Parse

E → F
E → E + F
F → F * T
F → T
T → int
T → (E)



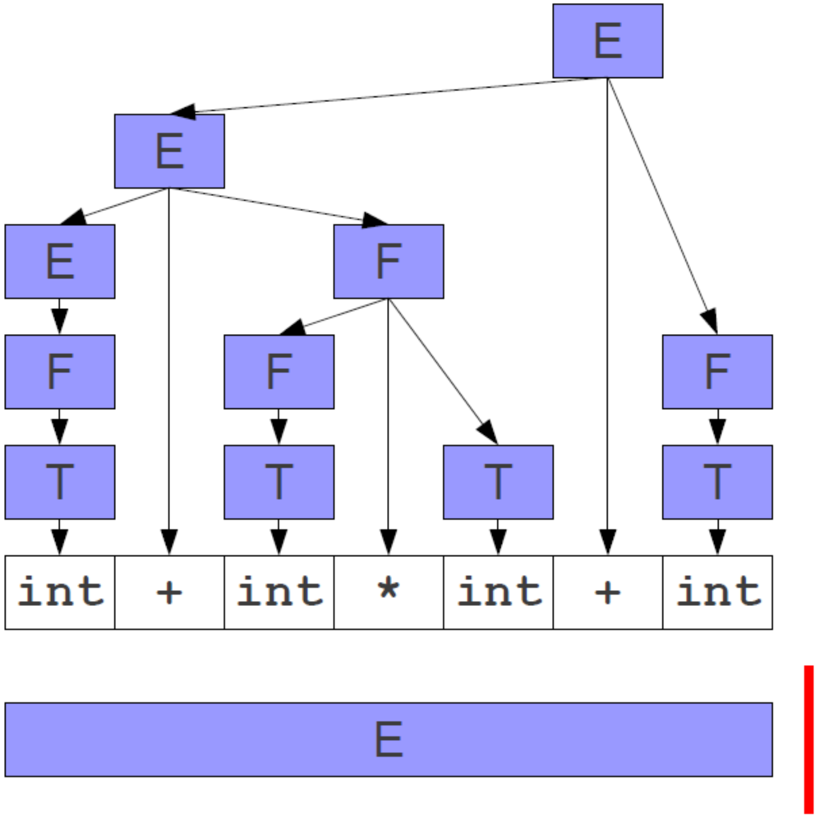
A Sample Shift/Reduce Parse

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow$ `int`
$T \rightarrow (E)$



A Sample Shift/Reduce Parse

$E \rightarrow F$
$E \rightarrow E$ + F
$F \rightarrow F$ * T
$F \rightarrow T$
$T \rightarrow$ int
$T \rightarrow$ (E)



Consider the Grammar

$$E \rightarrow E + E \mid E * E \mid id$$

Right sentential form	Handle	Production Rule
id1 + id2 * id3	id1	$E \rightarrow id$
E + id2 * id3	id2	$E \rightarrow id$
E + E * id3	id3	$E \rightarrow id$
E + E * E	E * E	$E \rightarrow E * E$
E + E	E + E	$E \rightarrow E + E$
E		

Handle

Consider:

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

$$S \Rightarrow \underline{aABe} \Rightarrow aA\underline{d}e \Rightarrow a\underline{Abc}de \Rightarrow a\underline{b}bcde$$

It follows that:

$S \rightarrow aABe$ is a handle of $\underline{aABe}$ in location 1.

$B \rightarrow d$ is a handle of $aA\underline{d}e$ in location 3.

$A \rightarrow Abc$ is a handle of $a\underline{Abc}de$ in location 2.

$A \rightarrow b$ is a handle of $a\underline{b}bcde$ in location 2.

Handle Pruning

- A rightmost derivation in reverse can be obtained by "handle-pruning."
- Apply this to the previous example.

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

abbcde

Find the handle = b at loc. 2

aAbcde

b at loc. 3 is not a handle:

aAAcde

... blocked.

Handle Pruning

- The process of discovering a handle & reducing it to the appropriate left-hand side is called *handle pruning*.

Handle pruning forms the basis for a bottom-up parsing method.

- To construct a rightmost derivation

  $S=\gamma_0 \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \ldots \Rightarrow \gamma_{n-1} \Rightarrow \gamma_n = \omega$

  input string

Apply the following simple algorithm

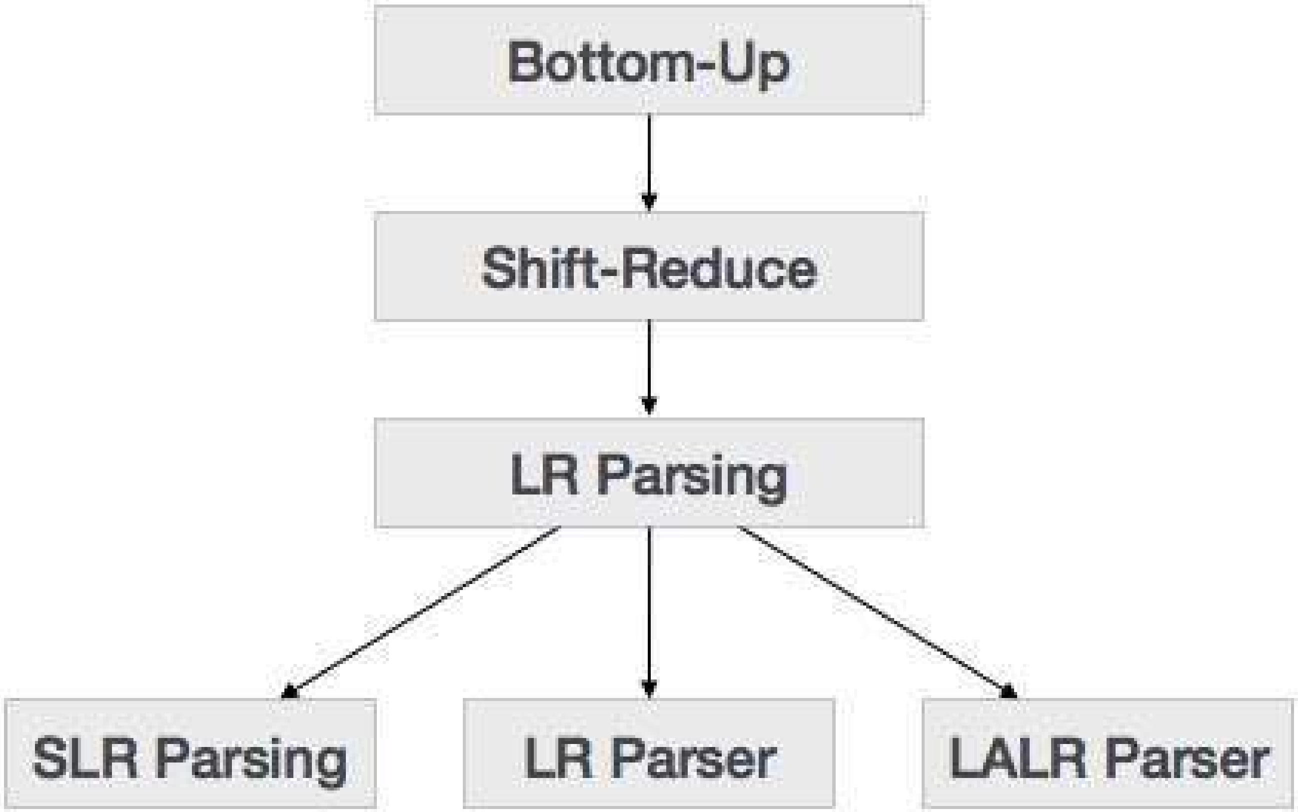
- Start from γ_n, find a handle $A_n \rightarrow \beta_n$ in γ_n, and replace β_n by A_n to get γ_{n-1}.
- Then find a handle $A_{n-1} \rightarrow \beta_{n-1}$ in γ_{n-1}, and replace β_{n-1} by A_{n-1} to get γ_{n-2}.
- Repeat this, until we reach S.

- Here it is apparent that the string appearing to the right of a handle contains only terminal symbols

- Since here the grammar is ambiguous the choices for the handles can be different depending upon right derivations used.

- This process can be made algorithmic using a stack implementation

Types of Bottom up Parsing



Shift –reduce Parsing

- Shift-reduce parsing is a form of bottom-up parsing in which a stack holds grammar symbols and an input buffer holds the rest of the tokens to be parsed.

- We use $ to mark the bottom of the stack and also the end of the input.

- During a left-to-right scan of the input tokens, the parser shifts zero or more input tokens into the stack, until it is ready to reduce a string β of grammar symbols on top of the stack.

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid id$

Right-Most Derivation of **id+id*id**

$E \Rightarrow E+T \Rightarrow E+T*F \Rightarrow E+T*id \Rightarrow E+F*id$

$\Rightarrow E+id*id \Rightarrow T+id*id \Rightarrow F+id*id \Rightarrow id+id*id$

Right-Most Sentential Form	Reducing Production
$\underline{id}+id*id$	$F \rightarrow id$
$\underline{F}+id*id$	$T \rightarrow F$
$\underline{T}+id*id$	$E \rightarrow T$
$E+\underline{id}*id$	$F \rightarrow id$
$E+\underline{F}*id$	$T \rightarrow F$
$E+T*\underline{id}$	$F \rightarrow id$
$E+\underline{T*F}$	$T \rightarrow T*F$
$\underline{E+T}$	$E \rightarrow E+T$
E	

Handles are red and underlined in the right-sentential forms

STACK	INPUT	ACTION
$\$$	$\mathbf{id}_1 * \mathbf{id}_2\,\$$	shift
$\$\,\mathbf{id}_1$	$* \mathbf{id}_2\,\$$	reduce by $F \to \mathbf{id}$
$\$\,F$	$* \mathbf{id}_2\,\$$	reduce by $T \to F$
$\$\,T$	$* \mathbf{id}_2\,\$$	shift
$\$\,T *$	$\mathbf{id}_2\,\$$	shift
$\$\,T * \mathbf{id}_2$	$\$$	reduce by $F \to \mathbf{id}$
$\$\,T * F$	$\$$	reduce by $T \to T * F$
$\$\,T$	$\$$	reduce by $E \to T$
$\$\,E$	$\$$	accept

A Stack Implementation of A Shift-Reduce Parser

- There are four possible actions of a shift-parser action:

  1. **Shift** : The next input symbol is shifted onto the top of the stack.
  2. **Reduce**: Replace the handle on the top of the stack by the non-terminal.
  3. **Accept**: Successful completion of parsing.
  4. **Error**: Parser discovers a syntax error, and calls an error recovery routine.

- Initial stack just contains only the end-marker $.

- The end of the input string is marked by the end-marker $.

A Stack Implementation of A Shift-Reduce Parser

- Two problems:
  - locate a handle and
  - decide which production to use (if there are more than two candidate productions).

- General Construction: using a stack:
  - "shift" input symbols into the stack until a handle is found on top of it.
  - "reduce" the handle to the corresponding non-terminal.

  - other operations:
    - "accept" when the input is consumed and only the start symbol is on the stack, also: "error"

A Stack Implementation of A Shift-Reduce Parser

Stack	Input	Action
$	id+id*id$	shift
$id	+id*id$	reduce by F → id
$F	+id*id$	reduce by T → F
$T	+id*id$	reduce by E → T
$E	+id*id$	shift
$E+	id*id$	shift
$E+id	*id$	reduce by F → id
$E+F	*id$	reduce by T → F
$E+T	*id$	shift
$E+T*	id$	shift
$E+T*id	$	reduce by F → id
$E+T*F	$	reduce by T → T*F
$E+T	$	reduce by E → E+T
$E	$	accept

Exercise

- Consider the following grammar-
  - $S \rightarrow (L) \mid a$
  - $L \rightarrow L , S \mid S$
- Parse the input string (a , (a , a)) using a shift-reduce parser.

Stack	Input Buffer	Parsing Action
$	(a , (a , a)) $	Shift
$ (	a , (a , a)) $	Shift
$ (a	, (a , a)) $	Reduce S → a
$ (S	, (a , a)) $	Reduce L → S
$ (L	, (a , a)) $	Shift
$ (L ,	(a , a)) $	Shift
$ (L , (	a , a)) $	Shift
$ (L , (a	, a)) $	Reduce S → a
$ (L , (S	, a)) $	Reduce L → S
$ (L , (L	, a)) $	Shift
$ (L , (L ,	a)) $	Shift
$ (L , (L , a	)) $	Reduce S → a
$ (L , (L , S)	)) $	Reduce L → L , S

Stack	Input Buffer	Parsing Action
$ (L , (L	)) $	Shift
$ (L , (L)	) $	Reduce S → (L)
$ (L , S	) $	Reduce L → L , S
$ (L	) $	Shift
$ (L)	$	Reduce S → (L)
$ S	$	Accept

Exercise

- Consider the following grammar-
  - $S \rightarrow 0S0 \mid 1S1 \mid 2$
- Perform shift reduce parser for input string ‘102011’.

Stack	Input	Action
$	102011$	Shift
$1	02011$	Shit
$10	2011$	Shift
$102	011$	Shift
$10S	011$	Reduce S → 2
$10S0	11$	Shift
$1S	11$	Reduce S → 0S0
$1S1	1$	Shift
$S	1$	Reduce S → 1S1
$S	1$	Reject