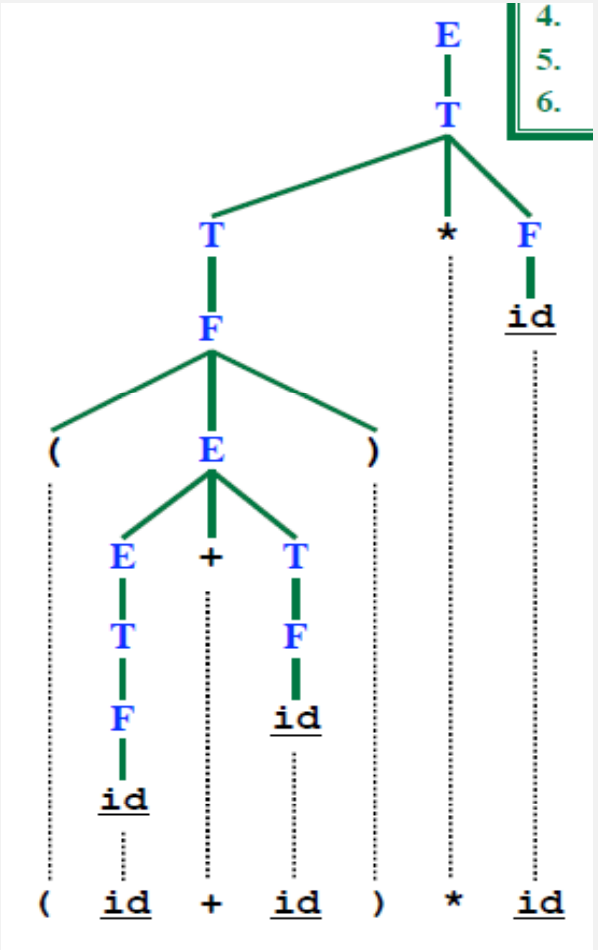
# Bottom-Up Parsing

Lecture 10

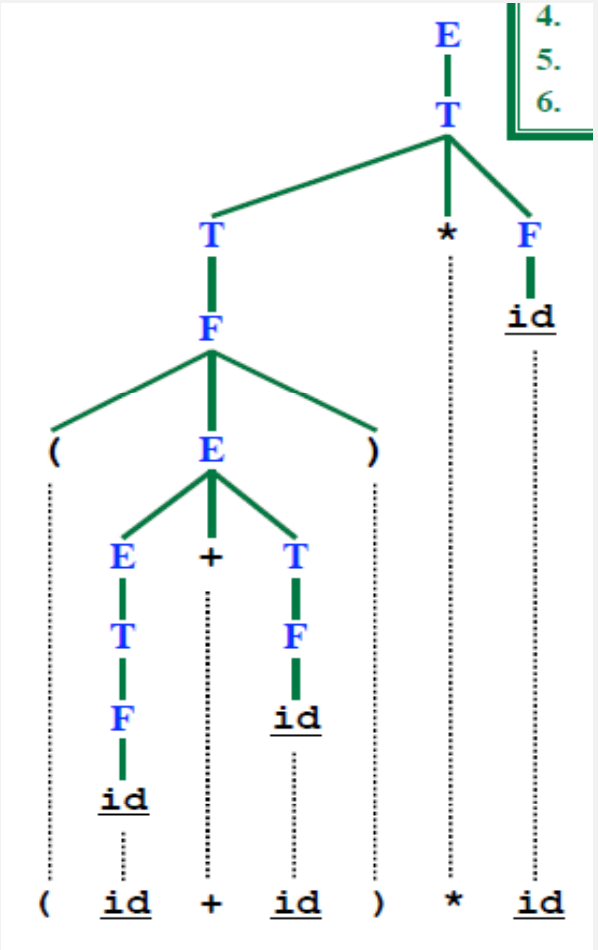
# A Right Most Derivation

| Rule | Right Sentinel Form |
|---|---|
| | E |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |



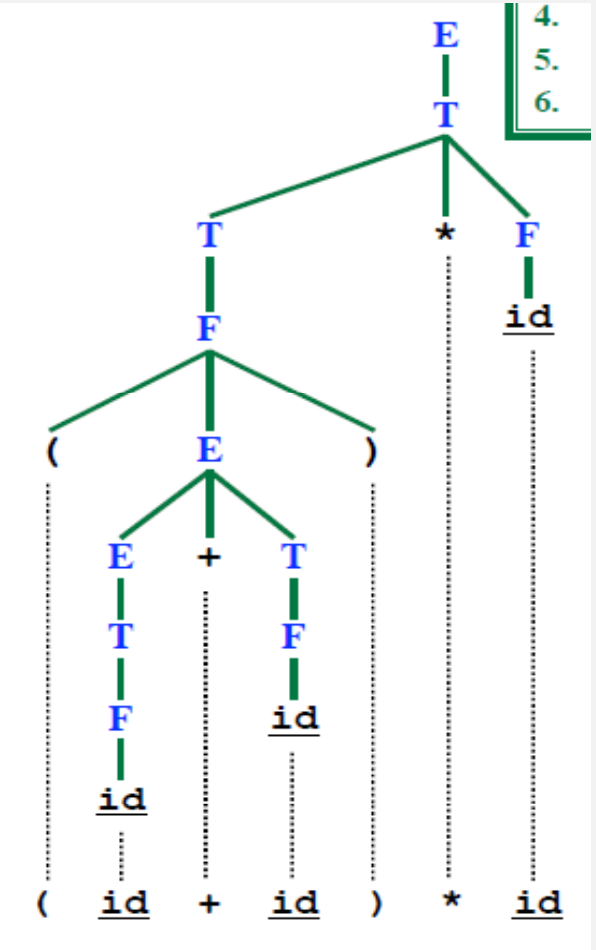
# A Right Most Derivation

| Rule | Right Sentinel Form |
|---|---|
| | E |
| E ⟶ T | T |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |



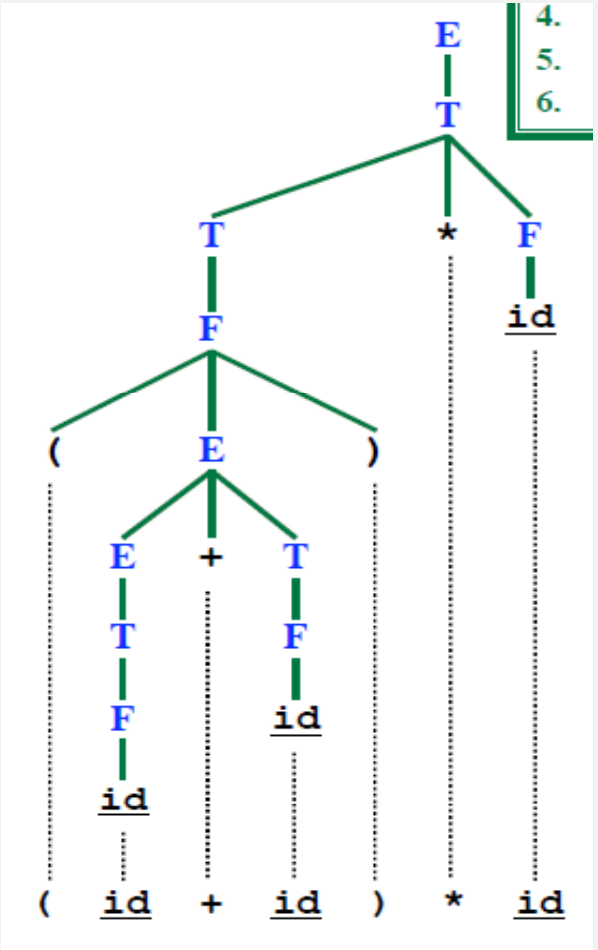
# A Right Most Derivation

| Rule | Right Sentinel Form |
|---|---|
| | E |
| E ⟶ T | T |
| T ⟶ T * F | T * F |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |



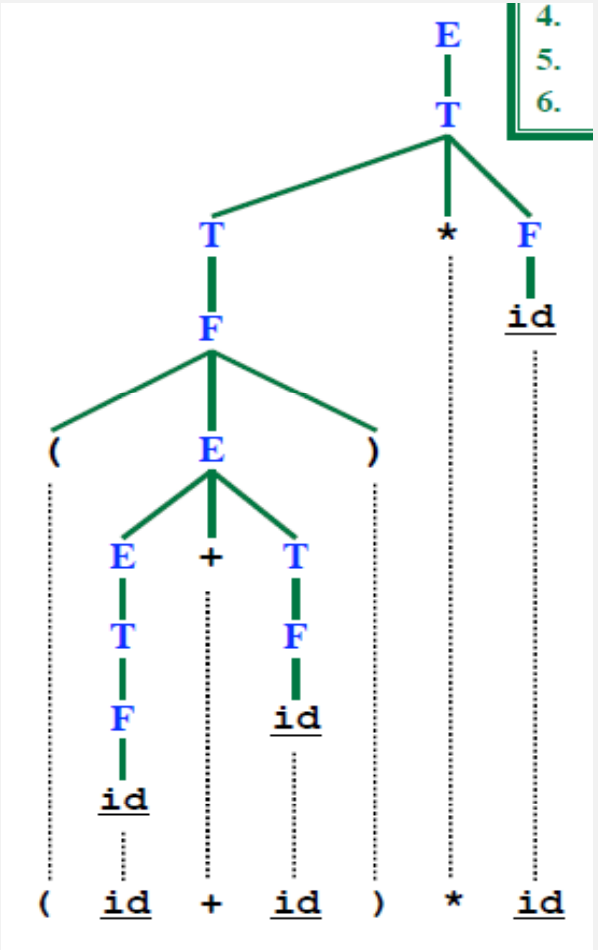
# A Right Most Derivation

| Rule | Right Sentinel Form |
|---|---|
| | E |
| E ⟶ T | T |
| T ⟶ T * F | T * F |
| F ⟶ id | T * id |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |



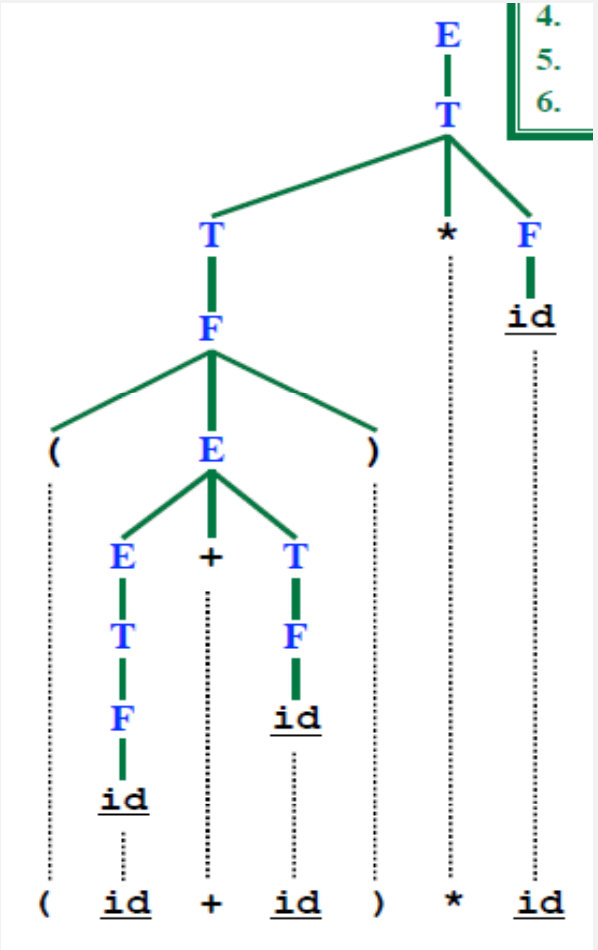
# A Right Most Derivation

| Rule | Right Sentinel Form |
|---|---|
| | E |
| E ⟶ T | T |
| T ⟶ T * F | T * F |
| F ⟶ id | T * id |
| T ⟶ F | F * id |
| | |
| | |
| | |
| | |
| | |
| | |
| | |



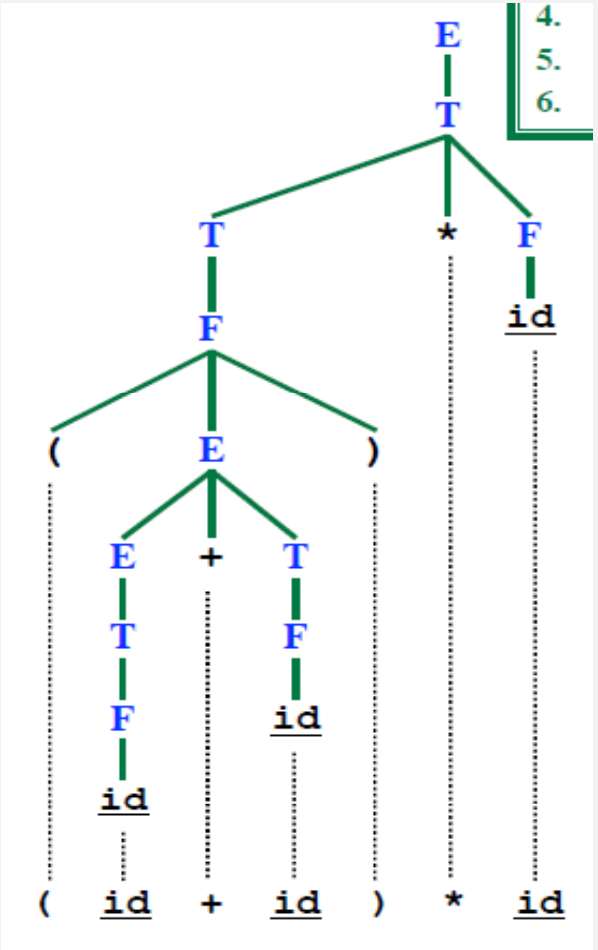
# A Right Most Derivation

| Rule | Right Sentinel Form |
|---|---|
| | E |
| E ⟶ T | T |
| T ⟶ T * F | T * F |
| F ⟶ id | T * id |
| T ⟶ F | F * id |
| F ⟶ ( E ) | (E) * id |
| | |
| | |
| | |
| | |
| | |
| | |



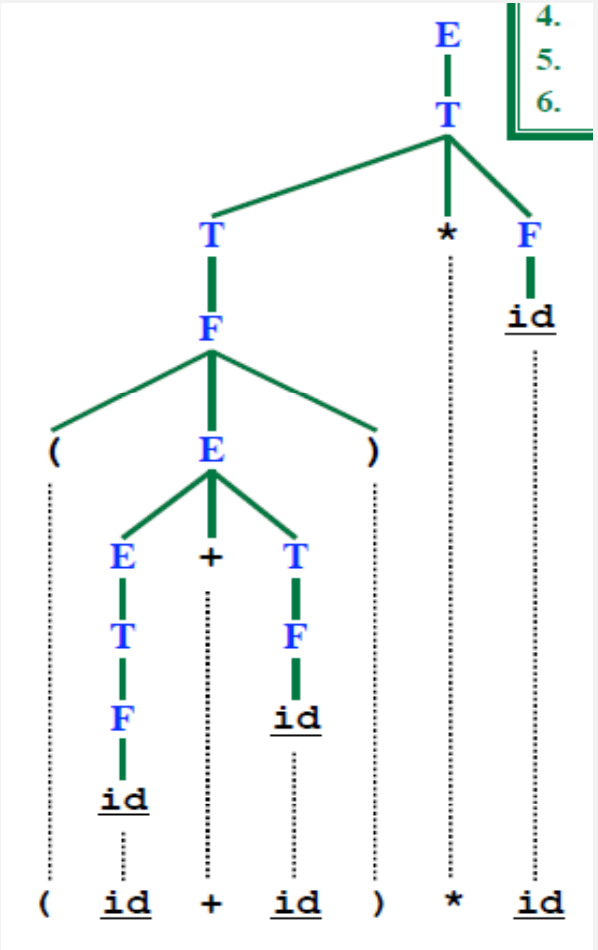
# A Right Most Derivation

| Rule | Right Sentinel Form |
|---|---|
| | E |
| E ⟶ T | T |
| T ⟶ T * F | T * F |
| F ⟶ id | T * id |
| T ⟶ F | F * id |
| F ⟶ ( E ) | (E) * id |
| E ⟶ E + T | (E + T) * id |
| | |
| | |
| | |
| | |
| | |



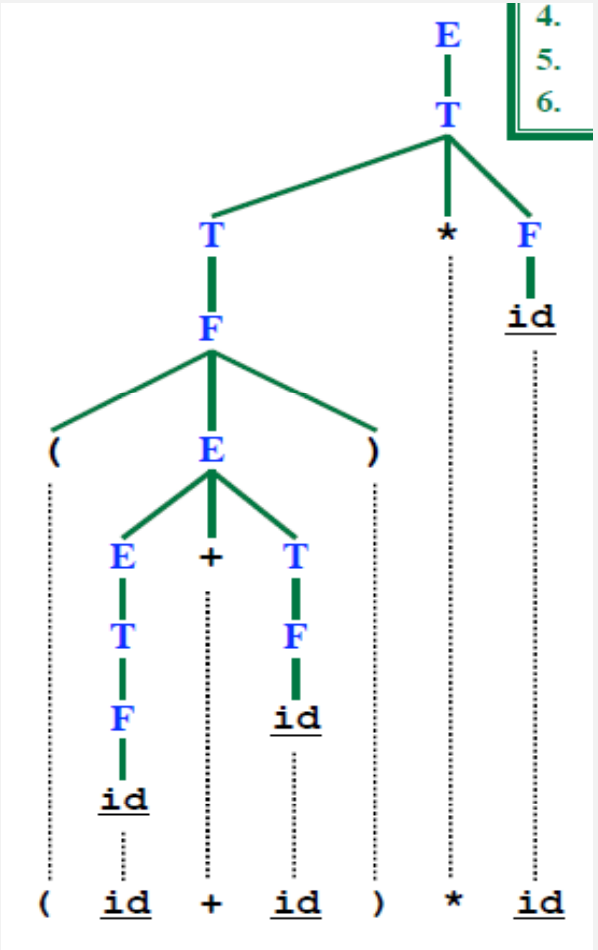
# A Right Most Derivation

| Rule | Right Sentinel Form |
|---|---|
| | E |
| E ⟶ T | T |
| T ⟶ T * F | T * F |
| F ⟶ id | T * id |
| T ⟶ F | F * id |
| F ⟶ ( E ) | (E) * id |
| E ⟶ E + T | (E + T) * id |
| T ⟶ F | (E + F) * id |
| | |
| | |
| | |
| | |



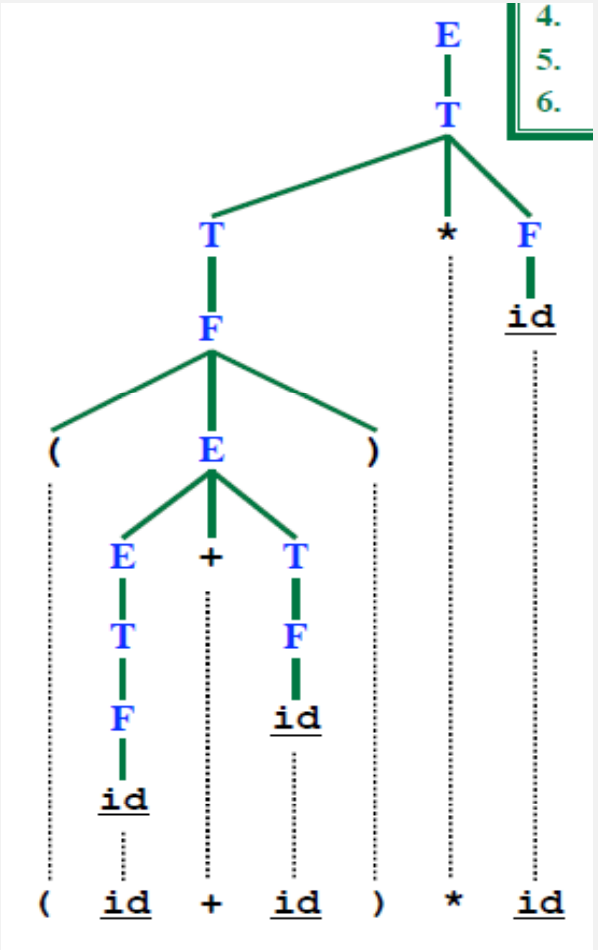
# A Right Most Derivation

| Rule | Right Sentinel Form |
|---|---|
| | E |
| E ⟶ T | T |
| T ⟶ T * F | T * F |
| F ⟶ id | T * id |
| T ⟶ F | F * id |
| F ⟶ ( E ) | (E) * id |
| E ⟶ E + T | (E + T) * id |
| T ⟶ F | (E + F) * id |
| F ⟶ id | (E + id) * id |
| | |
| | |
| | |



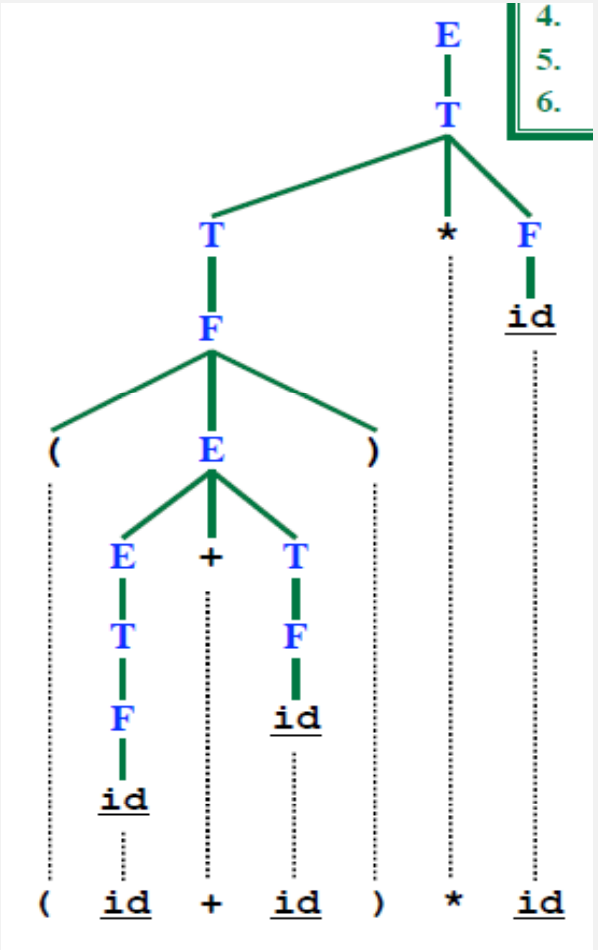
# A Right Most Derivation

| Rule | Right Sentinel Form |
|---|---|
| | E |
| E ⟶ T | T |
| T ⟶ T * F | T * F |
| F ⟶ id | T * id |
| T ⟶ F | F * id |
| F ⟶ ( E ) | (E) * id |
| E ⟶ E + T | (E + T) * id |
| T ⟶ F | (E + F) * id |
| F ⟶ id | (E + id) * id |
| E ⟶ T | (T + id) * id |
| | |
| | |



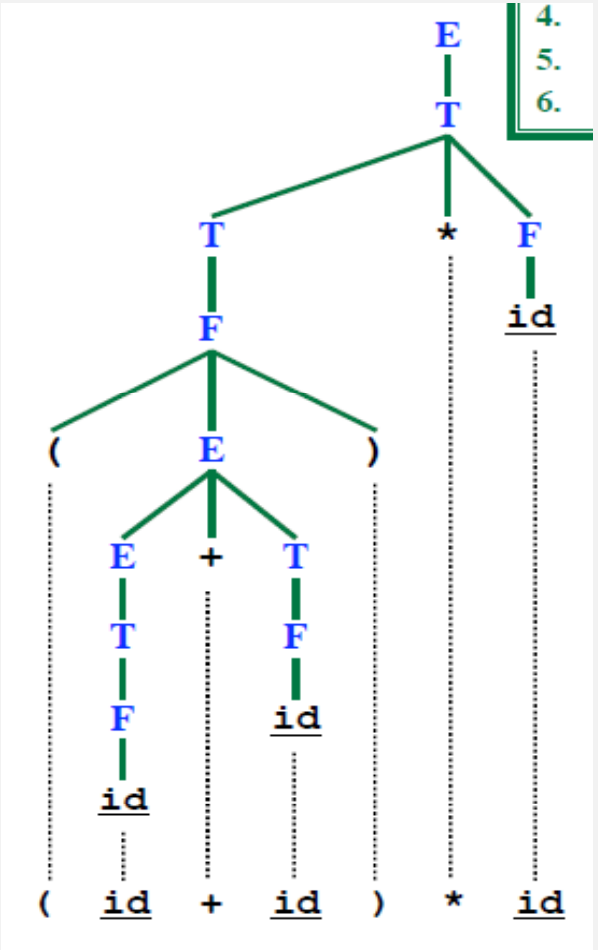
# A Right Most Derivation

| Rule | Right Sentinel Form |
|---|---|
| | E |
| E ⟶ T | T |
| T ⟶ T * F | T * F |
| F ⟶ id | T * id |
| T ⟶ F | F * id |
| F ⟶ ( E ) | (E) * id |
| E ⟶ E + T | (E + T) * id |
| T ⟶ F | (E + F) * id |
| F ⟶ id | (E + id) * id |
| E ⟶ T | (T + id) * id |
| T ⟶ F | (F + id) * id |
| | |



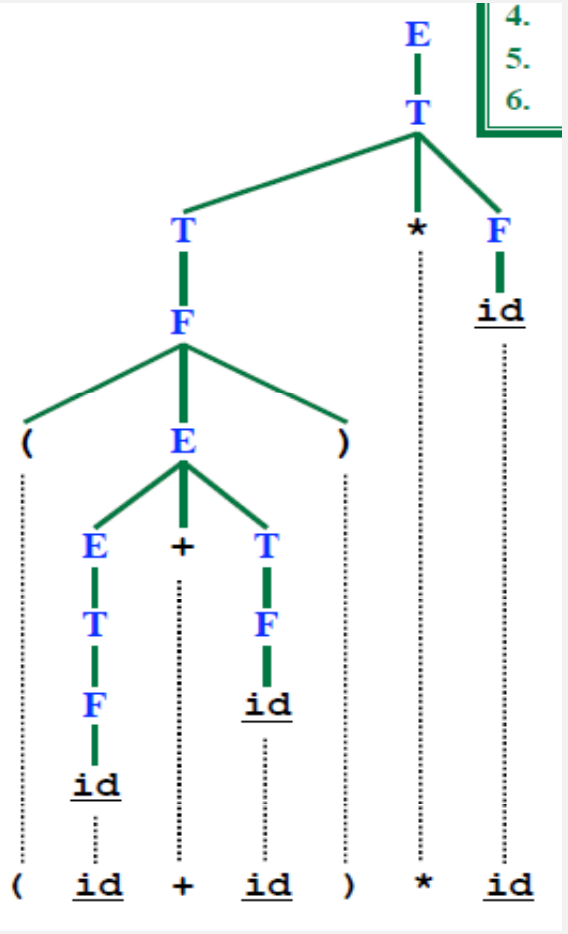
# A Right Most Derivation

| Rule | Right Sentinel Form |
|---|---|
| | E |
| E ⟶ T | T |
| T ⟶ T * F | T * F |
| F ⟶ id | T * id |
| T ⟶ F | F * id |
| F ⟶ ( E ) | (E) * id |
| E ⟶ E + T | (E + T) * id |
| T ⟶ F | (E + F) * id |
| F ⟶ id | (E + id) * id |
| E ⟶ T | (T + id) * id |
| T ⟶ F | (F + id) * id |
| F ⟶ id | (id + id) * id |



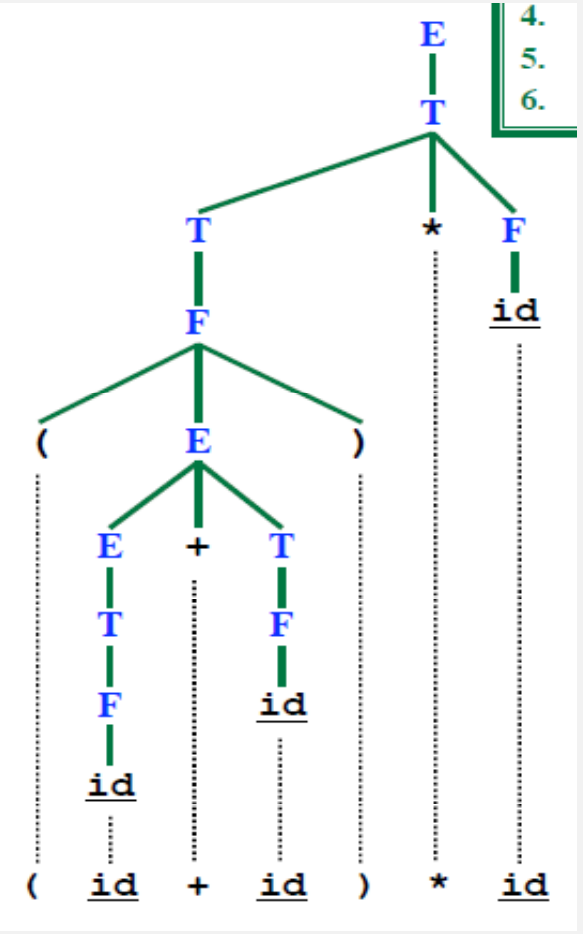
# A Right Most Derivation in Reverse

| Rule | Right Sentinel Form |
|---|---|
| F ⟶ id | (id + id) * id |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |



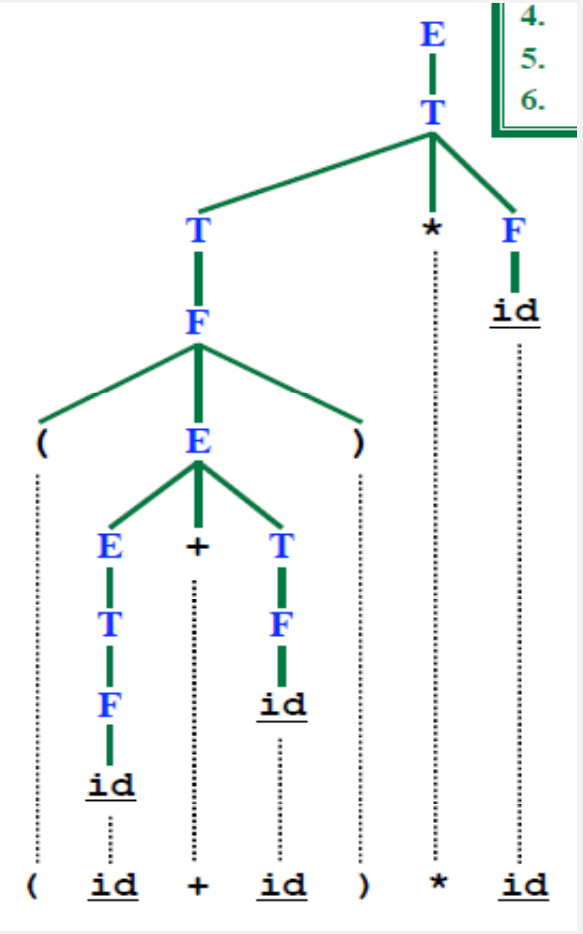
# A Right Most Derivation in Reverse

| Rule | Right Sentinel Form |
|---|---|
| F ⟶ id | (id + id) * id |
| T ⟶ F | (F + id) * id |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |



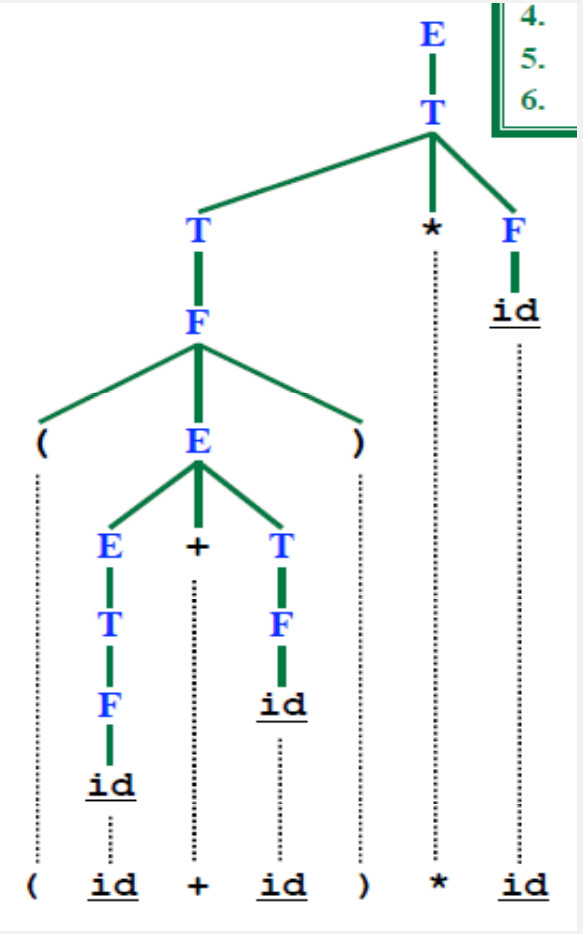
# A Right Most Derivation in Reverse

| Rule | Right Sentinel Form |
|---|---|
| F ⟶ id | (id + id) * id |
| T ⟶ F | (F + id) * id |
| E ⟶ T | (T + id) * id |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |



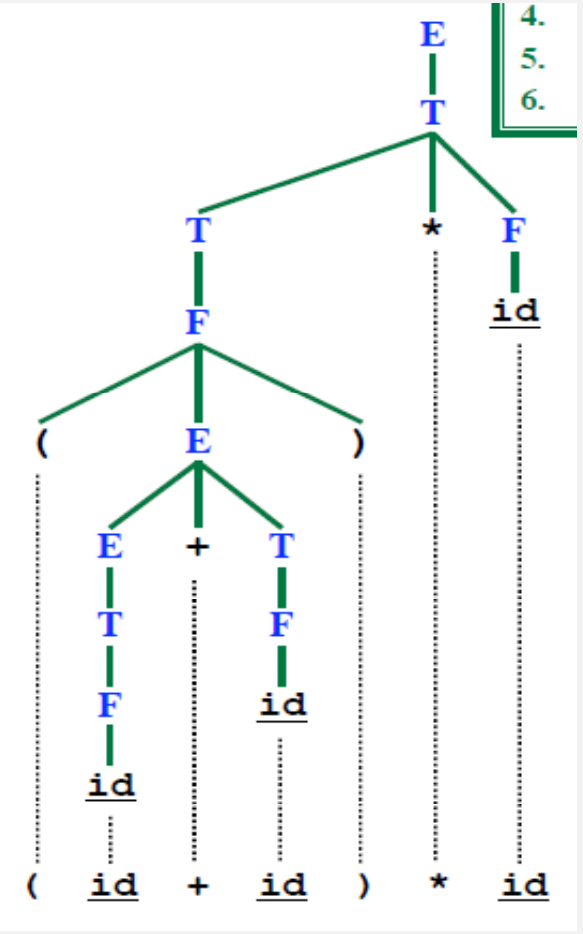
# A Right Most Derivation in Reverse

| Rule | Right Sentinel Form |
|---|---|
| F ⟶ id | (id + id) * id |
| T ⟶ F | (F + id) * id |
| E ⟶ T | (T + id) * id |
| F ⟶ id | (E + id) * id |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |



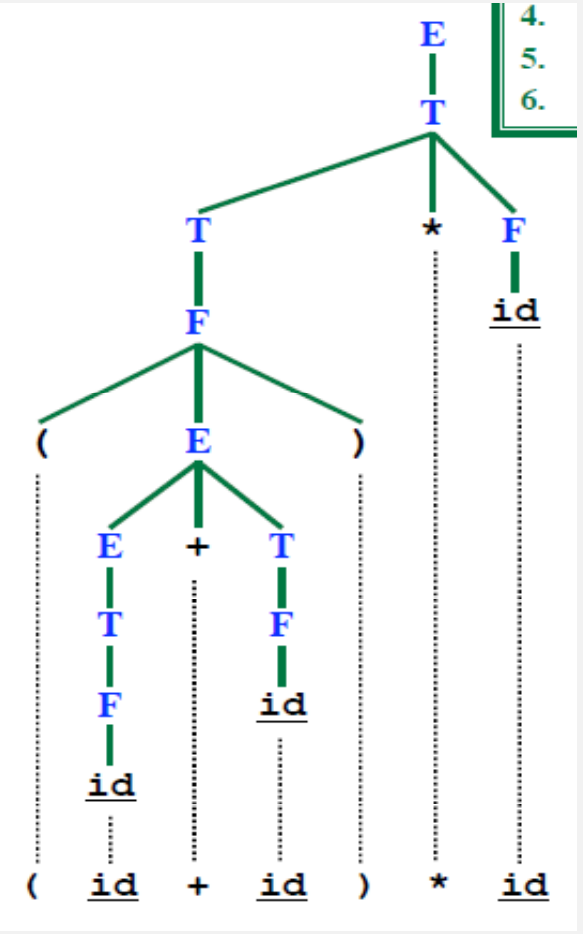
# A Right Most Derivation in Reverse

| Rule | Right Sentinel Form |
|---|---|
| F ⟶ id | (id + id) * id |
| T ⟶ F | (F + id) * id |
| E ⟶ T | (T + id) * id |
| F ⟶ id | (E + id) * id |
| T ⟶ F | (E + F) * id |
| | |
| | |
| | |
| | |
| | |
| | |
| | |



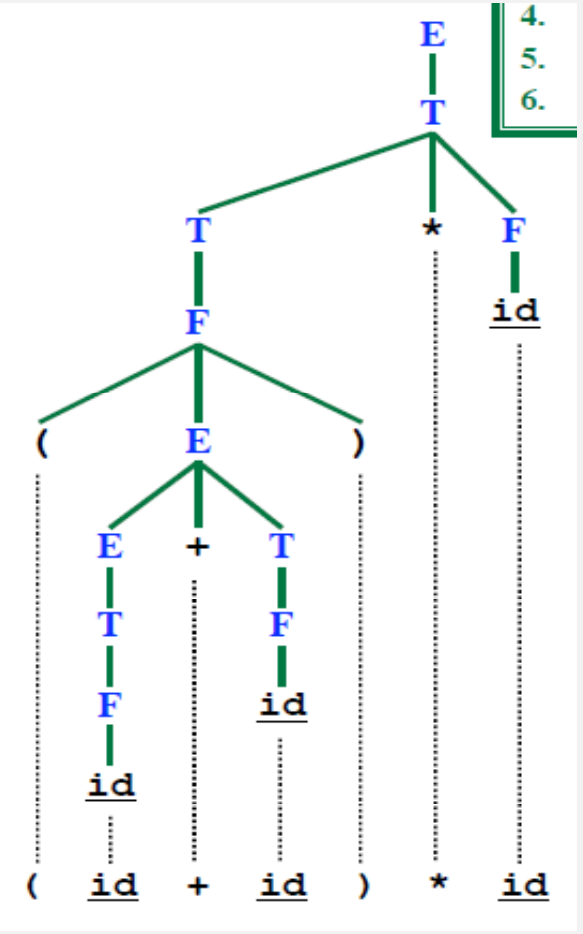
# A Right Most Derivation in Reverse

| Rule | Right Sentinel Form |
|---|---|
| F ⟶ id | (id + id) * id |
| T ⟶ F | (F + id) * id |
| E ⟶ T | (T + id) * id |
| F ⟶ id | (E + id) * id |
| T ⟶ F | (E + F) * id |
| E ⟶ E + T | (E + T) * id |
| F ⟶ ( E ) | (E) * id |
| | |
| | |
| | |
| | |
| | |



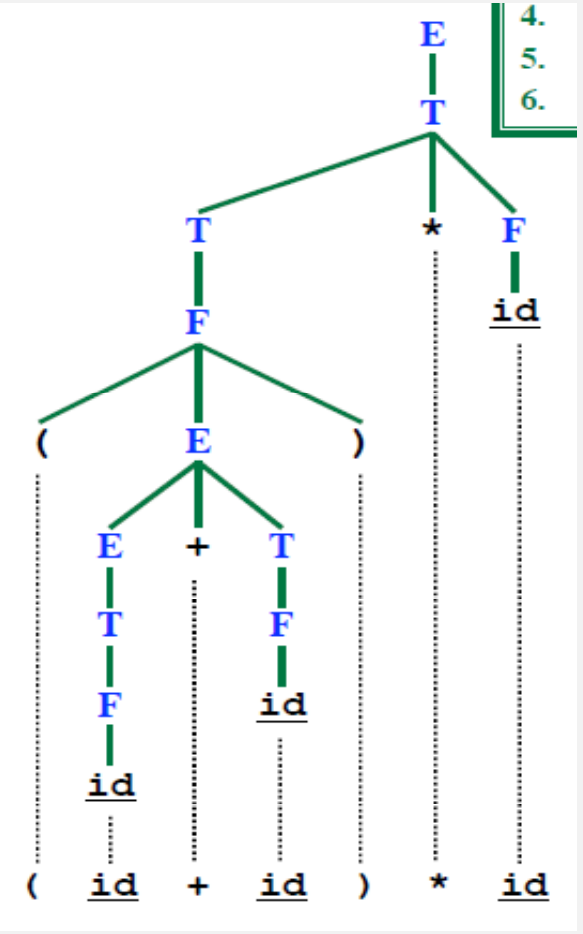
# A Right Most Derivation in Reverse

| Rule | Right Sentinel Form |
|---|---|
| F ⟶ id | (id + id) * id |
| T ⟶ F | (F + id) * id |
| E ⟶ T | (T + id) * id |
| F ⟶ id | (E + id) * id |
| T ⟶ F | (E + F) * id |
| E ⟶ E + T | (E + T) * id |
| F ⟶ ( E ) | (E) * id |
| T ⟶ F | F * id |
| | |
| | |
| | |
| | |



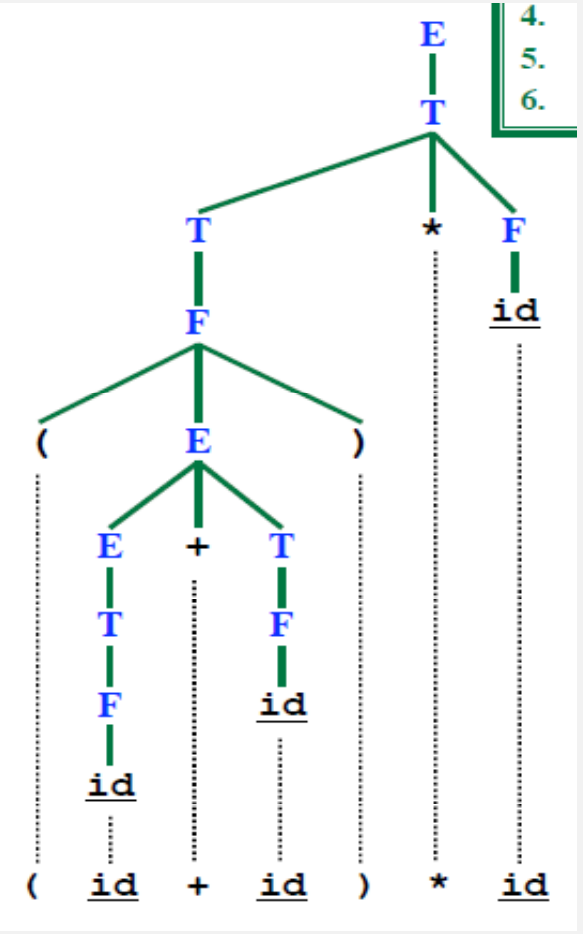
# A Right Most Derivation in Reverse

| Rule | Right Sentinel Form |
|---|---|
| F ⟶ id | (id + id) * id |
| T ⟶ F | (F + id) * id |
| E ⟶ T | (T + id) * id |
| F ⟶ id | (E + id) * id |
| T ⟶ F | (E + F) * id |
| E ⟶ E + T | (E + T) * id |
| F ⟶ ( E ) | (E) * id |
| T ⟶ F | F * id |
| F ⟶ id | T * id |
| | |
| | |
| | |



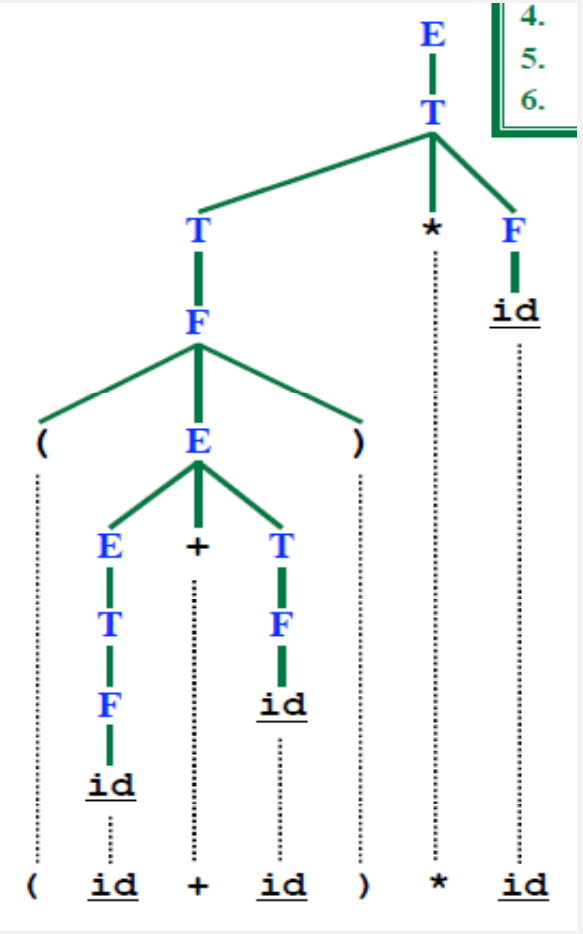
# A Right Most Derivation in Reverse

| Rule | Right Sentinel Form |
|---|---|
| F ⟶ id | (id + id) * id |
| T ⟶ F | (F + id) * id |
| E ⟶ T | (T + id) * id |
| F ⟶ id | (E + id) * id |
| T ⟶ F | (E + F) * id |
| E ⟶ E + T | (E + T) * id |
| F ⟶ ( E ) | (E) * id |
| T ⟶ F | F * id |
| F ⟶ id | T * id |
| T ⟶ T * F | T * F |
| | |
| | |



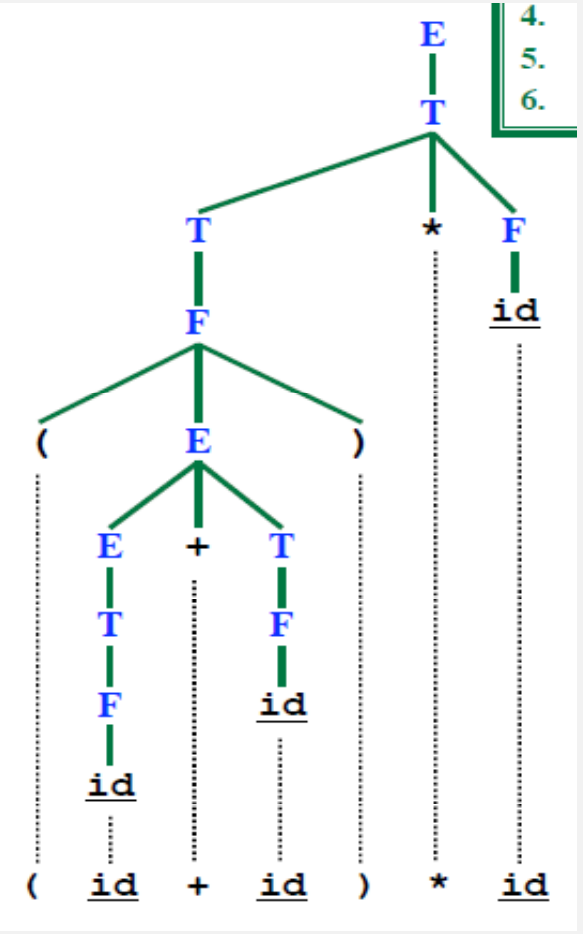
# A Right Most Derivation in Reverse

| Rule | Right Sentinel Form |
|---|---|
| F ⟶ id | (id + id) * id |
| T ⟶ F | (F + id) * id |
| E ⟶ T | (T + id) * id |
| F ⟶ id | (E + id) * id |
| T ⟶ F | (E + F) * id |
| E ⟶ E + T | (E + T) * id |
| F ⟶ ( E ) | (E) * id |
| T ⟶ F | F * id |
| F ⟶ id | T * id |
| T ⟶ T * F | T * F |
| E ⟶ T | T |
| | |



# A Right Most Derivation in Reverse

| Rule | Right Sentinel Form |
|---|---|
| F ⟶ id | (id + id) * id |
| T ⟶ F | (F + id) * id |
| E ⟶ T | (T + id) * id |
| F ⟶ id | (E + id) * id |
| T ⟶ F | (E + F) * id |
| E ⟶ E + T | (E + T) * id |
| F ⟶ ( E ) | (E) * id |
| T ⟶ F | F * id |
| F ⟶ id | T * id |
| T ⟶ T * F | T * F |
| E ⟶ T | T |
| | E |



# Handle

A *handle* is a substring of grammar symbols in a *right-sentential form* that matches a right-hand side of a production

Given a right-sentential form γ,
A handle is

- A position in γ
- A rule $A \rightarrow \beta$

Such that if you do a reduction by $A \rightarrow \beta$ at that point,
it is a valid step in a rightmost derivation.
In other words...
Let $\gamma = \alpha\beta w$
then
$S \rightarrow^{*} \alpha A w \rightarrow_{rm} \alpha\beta w$

| Rule | Right Sentinel Form |
|---|---|
| F ⟶ id | (id + id) * id |
| | (F + id) * id |

# Handle

1. S → f A B e
2. A → A g c
3. A → g
4. B → d

***A rightmost derivation, in reverse:***

Input String:

f **g** g c d e

# Handle

1. S → f A B e
2. A → A g c
3. A → g
4. B → d

***A rightmost derivation, in reverse:***

Input String:

f **g** g c d e

Reduce by **A → g**

f **A g c** d e

# Handle

1. S → f A B e
2. A → A g c
3. A → g
4. B → d

***A rightmost derivation, in reverse:***

Input String:

f **g** g c d e

Reduce by **A → g**

f **A g c** d e

Reduce by **A → A g c**

f A **d** e

# Handle

1. S → f A B e
2. A → A g c
3. A → g
4. B → d

***A rightmost derivation, in reverse:***

Input String:

f **g** g c d e

Reduce by **A → g**

f **A g c** d e

Reduce by **A → A g c**

f **A** d e

Reduce by **B → d**

f **A B** e

# Handle

1. S → f A B e
2. A → A g c
3. A → g
4. B → d

***A rightmost derivation, in reverse:***

Input String:

f **g** g c d e

Reduce by **A → g**

f **A g c** d e

Reduce by **A → A g c**

f A **d** e

Reduce by **B → d**

f **A B e**

Reduce by **S → f A B e**

S

# Handle

1. S → f A B e
2. A → A g c
3. A → g
4. B → d

***A rightmost derivation, in reverse:***

Input String:

f **g** g c d e

Reduce by **A → g**

f **A g c** d e

Reduce by **A → A g c**

f **A d** e

Reduce by **B → d**

f **A B e**

Reduce by **S → f A B e**

S

**Success: The handles are in red**

# Handle

1. S → f A B e
2. A → A g c
3. A → g
4. B → d

***A rightmost derivation, in reverse:***

Input String:

f **g** g c d e

Reduce by **A → g**

f A **g** c d e

Reduce by **A → A g c**

f A A c d e

Now we are stuck

No way to continue reducing

Must be careful in deciding when to reduce, or else we may get stuck

# Shift Reduce Parsing

***Goal:***

Find handles and perform reductions.

**Is there a handle on the top of the stack?**

**Yes: Do a reduction**

**No: Shift another input symbol onto the stack**

***Possible Actions:***

***Shift***

Push current input symbol onto stack

Advance input to next symbol

***Reduce***

A handle is on the top of the stack

Pop the handle

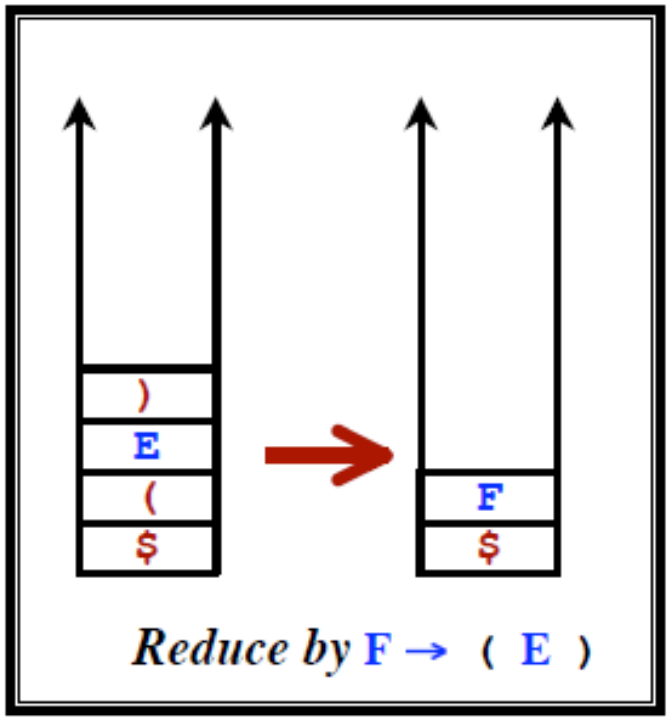
Push the lefthand side of the rule

***Accept***

Report success and terminate

***Error***

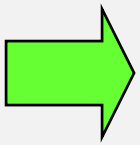
Report error and terminate



# Notation for Shift-Reduce Parsing

Grammar

$E \rightarrow E + E \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

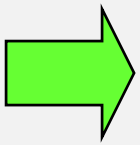


| Stack | input | Action |
|---|---|---|
| $ | id * id $ | Shift |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Notation for Shift-Reduce Parsing

Grammar

$E \rightarrow E + E \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

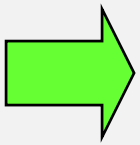


| Stack | input | Action |
|---|---|---|
| $ | id * id $ | Shift |
| $id | * id $ | reduce by F → id |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Notation for Shift-Reduce Parsing

Grammar

$E \rightarrow E + E \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

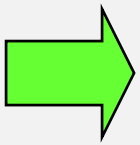


| Stack | input | Action |
|---|---|---|
| $ | id * id $ | Shift |
| $id | * id $ | reduce by F → id |
| $F | * id $ | reduce by T → F |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Notation for Shift-Reduce Parsing

Grammar

$E \rightarrow E + E \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

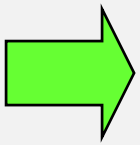


| Stack | input | Action |
|---|---|---|
| $ | id * id $ | Shift |
| $id | * id $ | reduce by F → id |
| $F | * id $ | reduce by T → F |
| $T | * id $ | Shift |
| | | |
| | | |
| | | |
| | | |
| | | |

# Notation for Shift-Reduce Parsing

Grammar

$E \rightarrow E + E \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

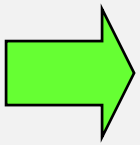


| Stack | input | Action |
|---|---|---|
| $ | id * id $ | Shift |
| $id | * id $ | reduce by F → id |
| $F | * id $ | reduce by T → F |
| $T | * id $ | Shift |
| $T * | id $ | shift |
| | | |
| | | |
| | | |
| | | |

# Notation for Shift-Reduce Parsing

Grammar

$E \rightarrow E + E \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

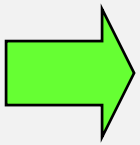


| Stack | input | Action |
|---|---|---|
| $ | id * id $ | Shift |
| $id | * id $ | reduce by F → id |
| $F | * id $ | reduce by T → F |
| $T | * id $ | Shift |
| $T * | id $ | shift |
| $T * id | $ | reduce by F → id |
| | | |
| | | |
| | | |

# Notation for Shift-Reduce Parsing

Grammar

$E \rightarrow E + E \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

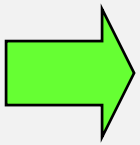


| Stack | input | Action |
|---|---|---|
| $ | id * id $ | Shift |
| $id | * id $ | reduce by F → id |
| $F | * id $ | reduce by T → F |
| $T | * id $ | Shift |
| $T * | id $ | shift |
| $T * id | $ | reduce by F → id |
| $T*F | $ | reduce by T → T * F |
| | | |
| | | |

# Notation for Shift-Reduce Parsing

Grammar

$E \rightarrow E + E \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

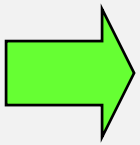


| Stack | input | Action |
|---|---|---|
| $ | id * id $ | Shift |
| $id | * id $ | reduce by F → id |
| $F | * id $ | reduce by T → F |
| $T | * id $ | Shift |
| $T * | id $ | shift |
| $T * id | $ | reduce by F → id |
| $T*F | $ | reduce by T → T * F |
| $T | $ | reduce by E → T |
| | | |

# Notation for Shift-Reduce Parsing

Grammar

$E \rightarrow E + E \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$



| Stack | input | Action |
|---|---|---|
| $ | id * id $ | Shift |
| $id | * id $ | reduce by F → id |
| $F | * id $ | reduce by T → F |
| $T | * id $ | Shift |
| $T * | id $ | shift |
| $T * id | $ | reduce by F → id |
| $T*F | $ | reduce by T → T * F |
| $T | $ | reduce by E → T |
| $E | $ | accept |

# How do we know what to do at each step?

***Given:***

- The stack and the current input symbol
- The tables (ACTION and GOTO)

Should be deterministic

**Reduce-Reduce Conflict**

Can reduce by 2 different rules... Which to use???

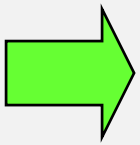
**Shift-Reduce Conflict**

Can either shift or reduce... Which to do???

# How do we know what to do at each step?

Grammar

$E \rightarrow E + E \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$



| Stack | input | Action |
|---|---|---|
| $ | id * id $ | Shift |
| $id | * id $ | reduce by F → id |
| $F | * id $ | reduce by T → F |
| $T | * id $ | Shift (or reduce) |
| $T * | id $ | Shift |
| $T * id | $ | reduce by F → id |
| $T*F | $ | reduce by T → T * F |
| $T | $ | reduce by E → T |
| $E | $ | Accept |

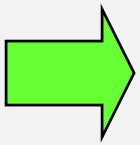
# How do we know what to do at each step?

**Grammar**

$C \rightarrow A\ B$

$A \rightarrow \mathbf{a}$

$B \rightarrow \mathbf{a}$



| Stack | input | Action |
|---|---|---|
| $ | aa$ | Shift |
| $a | a$ | Reduce $A \rightarrow \mathbf{a}$ or $B \rightarrow \mathbf{a}$ ? |
| $A | a$ | Shift |
| $Aa | $ | Reduce $A \rightarrow \mathbf{a}$ or $B \rightarrow \mathbf{a}$ ? |
| $AB | $ | Reduce |
| $C | $ | Accept |

# How do we know what to do at each step?

***LR Parsing Approach:***

Build Tables

Each table entry will have one action (SHIFT, REDUCE, ACCEPT, or ERROR)

- Failure when building the tables?
- Some entry has multiple actions

The grammar is not LR

- LR Grammars are unambiguous
- Only one rightmost derivation

There is only one handle at each step

# LR(0) Items

- An *LR(0) item* of a grammar *G* is a production of *G* with a • at some position of the right-hand side
- Thus, a production

  $A \rightarrow X\ Y\ Z$

  has four items:

  $[A \rightarrow \bullet\ X\ Y\ Z]$
  $[A \rightarrow X \bullet\ Y\ Z]$
  $[A \rightarrow X\ Y \bullet\ Z]$
  $[A \rightarrow X\ Y\ Z \bullet]$
- Note that production $A \rightarrow \varepsilon$ has one item $[A \rightarrow \bullet]$

# LR(0) Items

## Grammar

1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → ( E )
6. F → id

## Item Set

$E \longrightarrow \bullet\ E + T$
$E \longrightarrow E\ \bullet + T$
$E \longrightarrow E + \bullet\ T$
$E \longrightarrow E + T\ \bullet$

$E \longrightarrow \bullet\ T$
$E \longrightarrow T\ \bullet$

$T \longrightarrow \bullet\ T * F$
$T \longrightarrow T\ \bullet * F$
$T \longrightarrow T * \bullet\ F$
$T \longrightarrow T * F\ \bullet$

$T \longrightarrow \bullet\ F$
$T \longrightarrow F\ \bullet$

$F \longrightarrow \bullet\ (\ E\ )$
$F \longrightarrow (\ \bullet\ E\ )$
$F \longrightarrow (\ E\ \bullet\ )$
$F \longrightarrow (\ E\ )\ \bullet$

$F \longrightarrow \bullet\ id$
$F \longrightarrow id\ \bullet$

# The Closure Operation for LR(0) Items

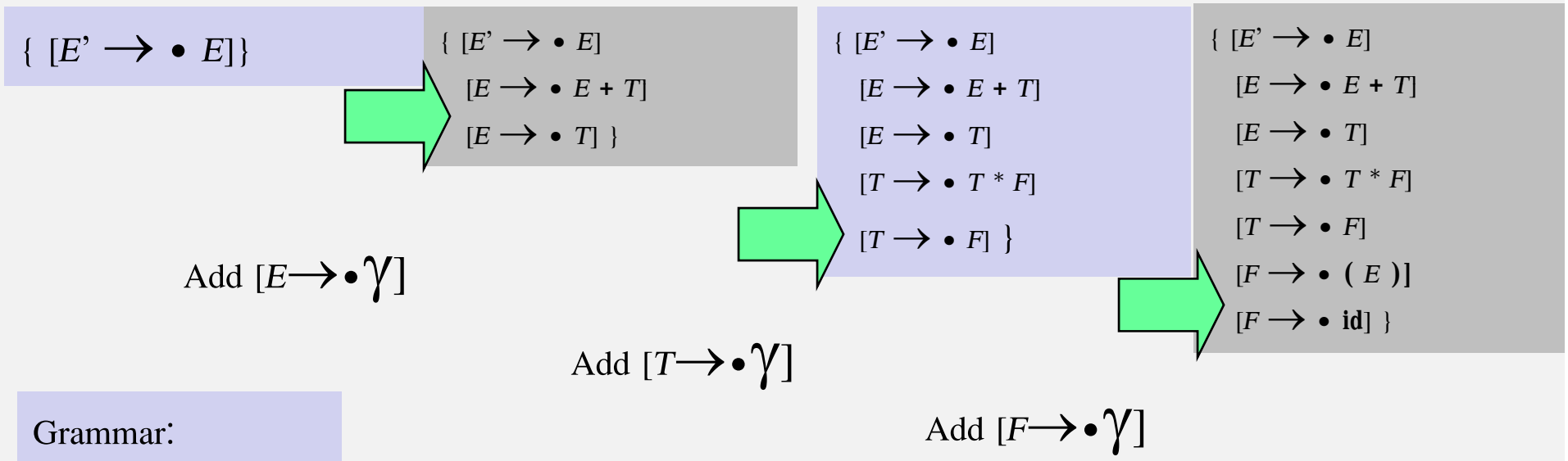
1. Start with *closure*(*I*) = *I*
2. If [$A \rightarrow \alpha \bullet B \beta$] $\in$ *closure*(*I*) then for each production $B \rightarrow \gamma$ in the grammar, add the item [$B \rightarrow \bullet \gamma$] to *I* if not already in *I*
3. Repeat 2 until no new items can be added

# Constructing the set of LR(0) Items of a Grammar

1. The grammar is augmented with a new start symbol $S'$ and production $S' \rightarrow S$
2. Initially, set $C = closure(\{[S' \rightarrow \bullet S]\})$ (this is the start state of the DFA)
3. For each set of items $I \in C$ and each grammar symbol $X \in (N \cup T)$ such that $goto(I,X) \notin C$ and $goto(I,X) \neq \emptyset$, add the set of items $goto(I,X)$ to $C$
4. Repeat 3 until no more sets can be added to $C$

# The Closure Operation (Example)

*closure*({[*E*' → • *E*]}) =




Grammar:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow ( E )$

$F \rightarrow \mathbf{id}$

# The Closure Operation (Example)

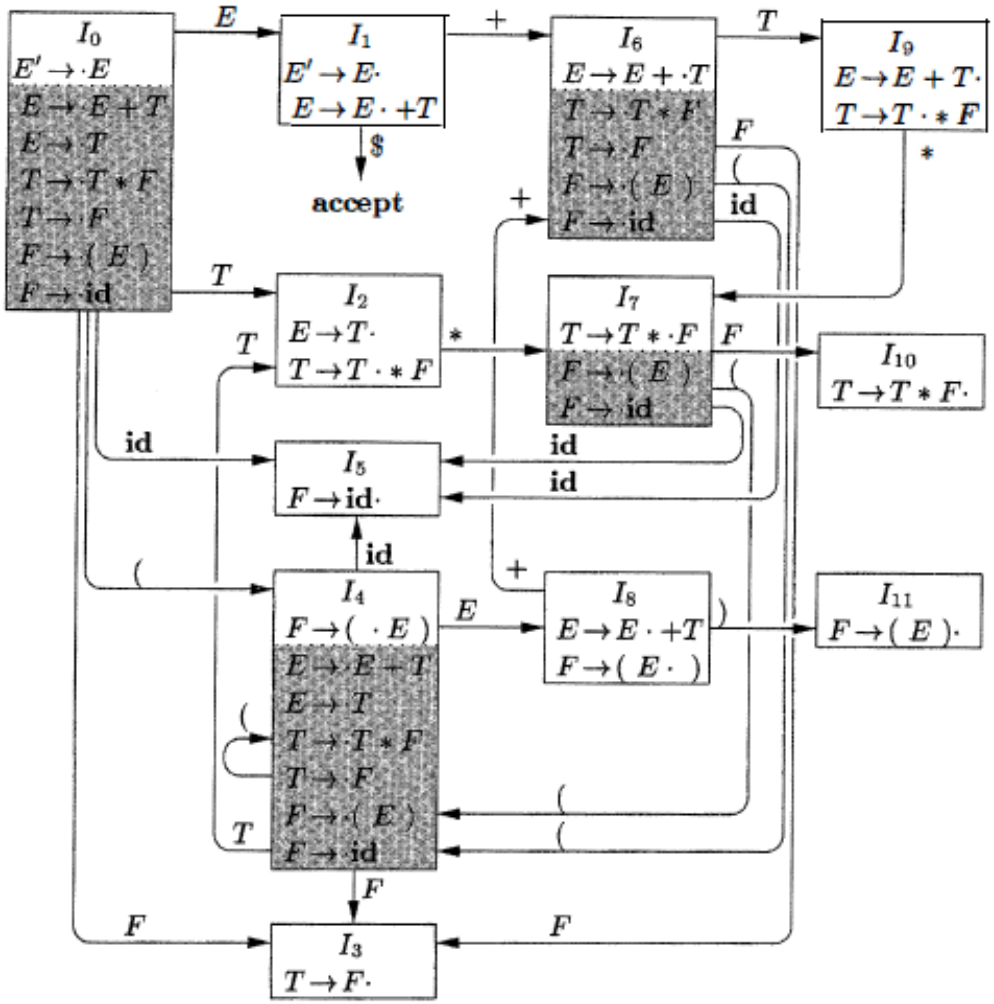



Figure 4.31: LR(0) automaton for the expression grammar (4.1)

# The Goto Operation for LR(0) Items

1. For each item $[A \rightarrow \alpha \bullet X \beta] \in I$, add the set of items *closure*($\{[A \rightarrow \alpha X \bullet \beta]\}$) to *goto*($I$,$X$) if not already there
2. Repeat step 1 until no more items can be added to *goto*($I$,$X$)
3. Intuitively, *goto*($I$,$X$) is the set of items that are valid for the viable prefix $\gamma X$ when $I$ is the set of items that are valid for $\gamma$

# The Goto Operation (Example 1)

Suppose $I$ =

$\{ [E' \rightarrow \bullet\ E]$
$[E \rightarrow \bullet\ E + T]$
$[E \rightarrow \bullet\ T]$
$[T \rightarrow \bullet\ T * F]$
$[T \rightarrow \bullet\ F]$
$[F \rightarrow \bullet\ (\ E\ )]$
$[F \rightarrow \bullet\ \mathbf{id}]\ \}$

Then $goto(I,E)$

$= closure(\{[E' \rightarrow E \bullet, E \rightarrow E \bullet + T]\})$

$= \{ [E' \rightarrow E \bullet]\quad [E \rightarrow E \bullet + T]\ \}$

Grammar:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (\ E\ )$

$F \rightarrow \mathbf{id}$

# The Goto Operation (Example 2)

Suppose $I = \{ [E' \rightarrow E \bullet], [E \rightarrow E \bullet + T] \}$

Then $goto(I,+) = closure(\{[E \rightarrow E + \bullet T]\}) =$

$\{ [E \rightarrow E + \bullet T]$
$[T \rightarrow \bullet T * F]$
$[T \rightarrow \bullet F]$
$[F \rightarrow \bullet ( E )]$
$[F \rightarrow \bullet \mathbf{id}] \}$

Grammar:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow ( E )$

$F \rightarrow \mathbf{id}$

Thank You