40-414 Compiler Design

Top-Down Parsing

Lecture 5

LL(1) Predictive Parsers

- Parser can "predict" which production to use
  - By looking at the next few tokens
  - No backtracking
- Predictive parsers accept LL(k) grammars
  - L means "left-to-right" scan of input
  - L means "leftmost derivation"
  - k means "predict based on k tokens of lookahead"
  - In practice, LL(1) is used

LL(1) Parsing Table Example

- Left-factored grammar

  $E \rightarrow T\ X$ $\qquad$ $X \rightarrow +\ E \mid \varepsilon$

  $T \rightarrow (\ E\ ) \mid \text{int}\ Y$ $\qquad$ $Y \rightarrow *\ T \mid \varepsilon$

- The LL(1) parsing table:

next input token

	int	*	+	(	)	$
E	T X			T X		
X			+ E		ε	ε
T	int Y			(E)		
Y		* T	ε		ε	ε

leftmost non-terminal

rhs of production to use

LL(1) Parsing Table Example (Cont.)

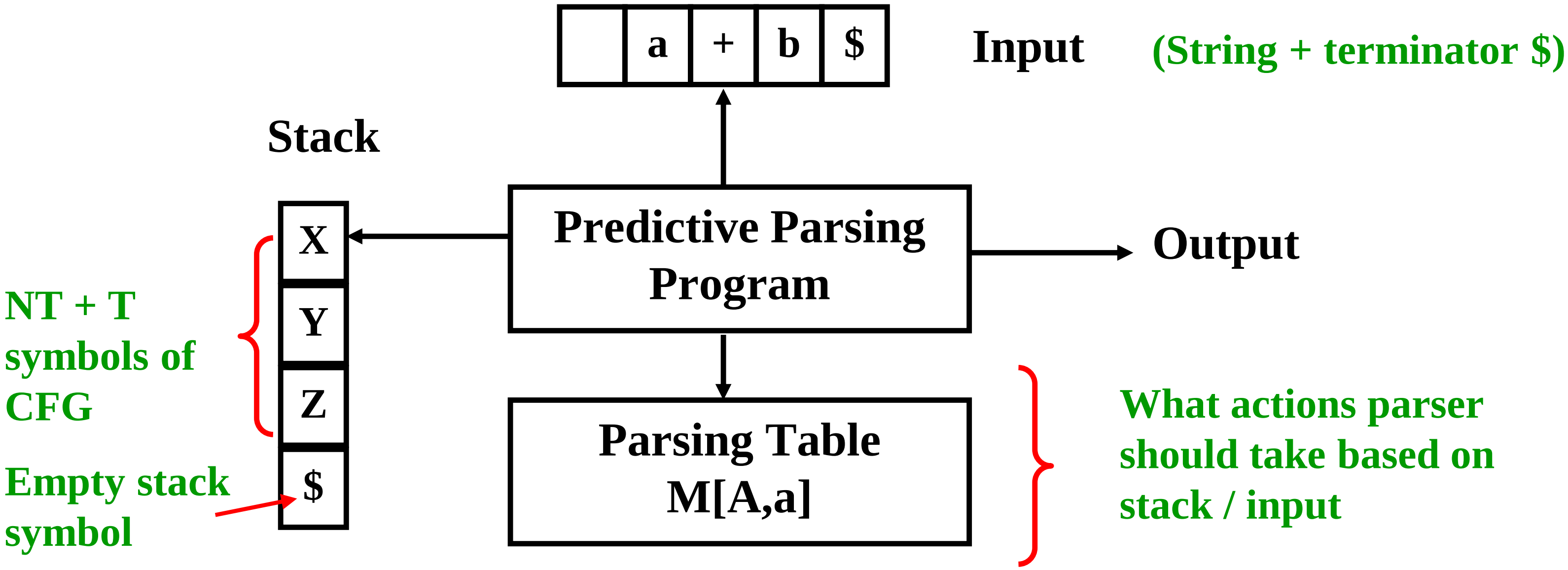
- Consider the [E, int] entry
  - "When current non-terminal is E and next input is int, use production $E \rightarrow T X$"
  - This can generate an int in the first position
- Consider the [Y,+] entry
  - "When current non-terminal is Y and current token is +, get rid of Y"
  - Y can be followed by + only if $Y \rightarrow \varepsilon$

LL(1) Parsing Tables. Errors

- Blank entries indicate error situations

- Consider the [E,*] entry
  - "There is no way to derive a string starting with * from non-terminal E"

LL(1) Parsing Algorithm



General parser behavior: X : top of stack a : current token

1. When X=a = $ halt, accept, success
2. When X=a ≠ $, POP X off stack, advance input, go to 1.
3. When X is a non-terminal, examine M[X, a], if it is an error, call recovery routine if M[X, a] = {UVW}, POP X, PUSH U,V,W, and DO NOT advance input

LL(1) Parsing Example

Stack	Input	Action
E $	int * int $	T X
T X $	int * int $	int Y
int Y X $	int * int $	terminal
Y X $	* int $	* T
* T X $	* int $	terminal
T X $	int $	int Y
int Y X $	int $	terminal
Y X $	$	ε
X $	$	ε
$	$	ACCEPT

Constructing Parsing Tables: The Intuition

- Consider non-terminal A, production $A \rightarrow \alpha$, & token t
- $T[A,t] = \alpha$ in two cases:

- If $\alpha \rightarrow^* t\ \beta$
  - α can derive a t in the first position
  - We say that $t \in First(\alpha)$

- If $A \rightarrow \alpha$ and $\alpha \rightarrow^* \varepsilon$ and $S \rightarrow^* \beta\ A\ t\ \delta$
  - Useful if stack has A, input is t, and A cannot derive t
  - In this case only option is to get rid of A (by deriving ε)
    - Can work only if t can follow A in at least one derivation
  - We say $t \in Follow(A)$

Constructing LL(1) Parsing Tables

- Construct a parsing table T for CFG G

- For each production $A \to \alpha$ in G do:
  - For each terminal $t \in First(\alpha)$ do
    - $T[A, t] = \alpha$
  - If $\varepsilon \in First(\alpha)$, for each $t \in Follow(A)$ do
    - $T[A, t] = \alpha$
  - If $\varepsilon \in First(\alpha)$ and $\$ \in Follow(A)$ do
    - $T[A, \$] = \alpha$

Example 1

$E \to T X \qquad\qquad X \to + E \mid \varepsilon$

$T \to (E) \mid \text{int } Y \qquad Y \to * \ T \mid \varepsilon$

	int	*	+	(	)	$
E	T X			T X		
X			+ E		ε	ε
T	int Y			(E)		
Y		* T	ε		ε	ε

Example 2

$S \rightarrow Sa \mid b$

$First(S) = \{b\}$

$Follow(S) = \{\$, a\}$

	a	b	$
S		b, Sa	

Notes on LL(1) Parsing Tables

- If any entry is multiply defined then G is not LL(1)
  - If G is ambiguous
  - If G is left recursive
  - If G is not left-factored
  - <u>And in other cases as well</u>

- Most programming language CFGs are not LL(1)

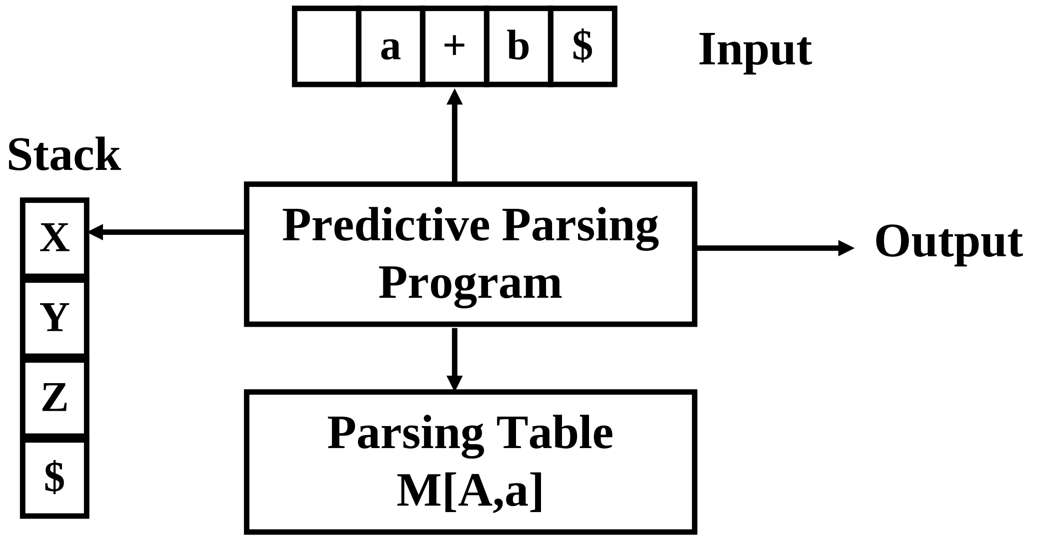
Notes on LL(1) Grammars

Grammar is LL(1) $\Leftrightarrow$ when for all $A \rightarrow \alpha \mid \beta$

1. First(α) $\cap$ First(β) = $\varnothing$; besides, only one of α or β can derive $\in$
2. if α derives $\in$, then Follow(A) $\cap$ First(β) = $\varnothing$

It may not be possible for a grammar to be manipulated into an LL(1) grammar

Implementing Panic Mode in LL(1)



Error situations include:

1. If X is a terminal and it doesn't match current token.
2. If M[X, Input] is empty – No allowable actions

Panic-Mode Recovery

- Assume in a syntax error, non-terminal *A* is on the top of the stack.

- The choice for a synchronizing set is important.
  - define the synchronizing set of *A* to be Follow(*A*). Then skip input until a token in Follow(*A*) appears and then pop *A* from the stack. Resume parsing...

  - add symbols of FIRST(*A*) to the synchronizing set. In this case, we skip input and once we find a token in FIRST(*A*), we resume parsing from *A*.

Panic-Mode Recovery (Cont.)

Modify the empty cells of the Parsing Table.

1. if M[A, a] = {empty} and a belongs to Follow(A) then we set M[A, a] = "synch"

Error-recovery Strategy :

If A=top-of-the-stack and a=current-token,

1. If A is NT and M[A, a] = {empty} then skip a from the input.
2. If A is NT and M[A, a] = {synch} then pop A.
3. If A is a terminal and A!=a then pop A (This is essentially inserting A before a).

Parse Table / Example

	id	+	*	(	)	$
E	T E′			T E′	synch	synch
E′		+ T E′			∈	∈
T	F T′	synch		F T′	synch	synch
T′		∈	* F T′		∈	∈
F	id	synch	synch	(E)	synch	synch

Pop top of stack NT for "synch" cells

Skip current-token for empty cells

$E \rightarrow T E'$
$E' \rightarrow + T E' \mid \varepsilon$
$T \rightarrow F T'$
$T' \rightarrow * F T' \mid \varepsilon$
$F \rightarrow (E) \mid id$

Parsing Example

	id	+	*	(	)	$
E	T E'			T E'	synch	synch
E'		+ T E'			∈	∈
T	F T'	synch		F T'	synch	synch
T'		∈	* F T'		∈	∈
F	id	synch	synch	(E)	synch	synch

STACK	INPUT	Remark
E $	+ id * + id $	error, skip +
E $	id * + id $	
T E' $	id * + id $	
F T' E' $	id * + id $	
id T' E' $	id * + id $	
T' E' $	* + id $	
* F T' E' $	* + id $	
F T' E' $	+ id $	

Possible Error Msg:
"Misplaced +
I am skipping it"

$E \rightarrow T E'$
$E' \rightarrow + T E' \mid \varepsilon$
$T \rightarrow F T'$
$T' \rightarrow * F T' \mid \varepsilon$
$F \rightarrow (E) \mid id$

Parsing Example (Cont.)

	id	+	*	(	)	$
E	T E'			T E'	synch	synch
E'		+ T E'			∈	∈
T	F T'	synch		F T'	synch	synch
T'		∈	* F T'		∈	∈
F	id	synch	synch	(E)	synch	synch

STACK	INPUT	Remark
F T' E' $	+ id $	error, M[F,+] = synch, F is popped
T' E' $	+ id $	
E' $	+ id $	
+ T E' $	+ id $	
T E' $	id $	
F T' E' $	id $	
id T' E' $	id $	
T' E' $	$	
E' $	$	
$	$	

Possible Error Msg: "Missing Term"

$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid id$

Other Parsing Methods

Top-Down Parsing Methods (Cont.)

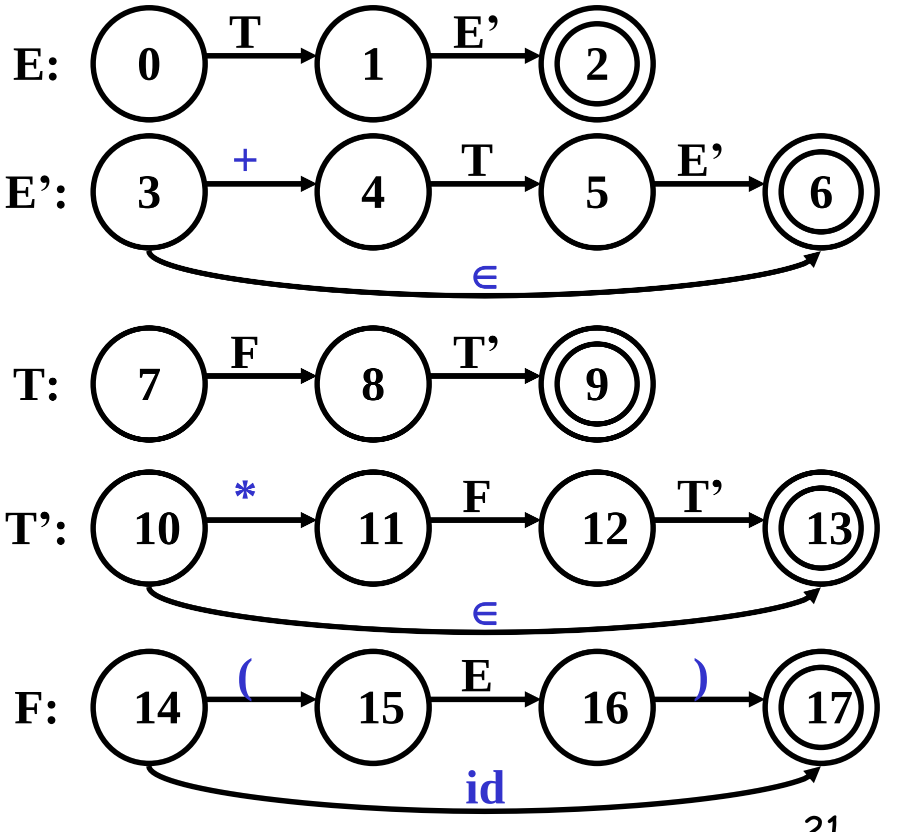
Transition Diagrams

Transition Diagrams

$E \rightarrow TE'$ $T \rightarrow FT'$ $F \rightarrow (E) \mid id$
$E' \rightarrow + TE' \mid \in$ $T' \rightarrow * FT' \mid \in$

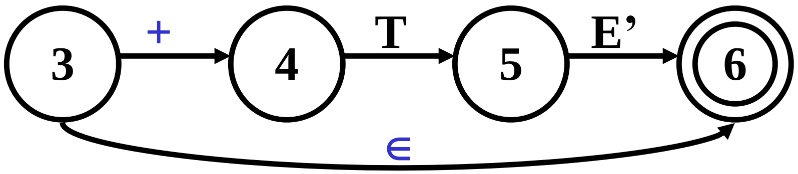
- Unlike lexical equivalents, each edge represents a token

- Transition implies: if token, match input else call proc

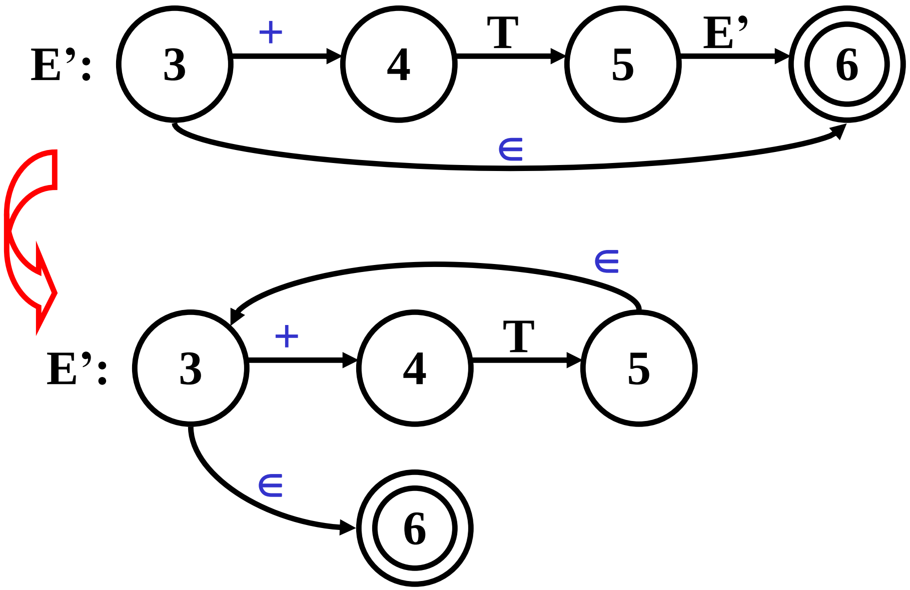


Transition Diagrams can be Simplified

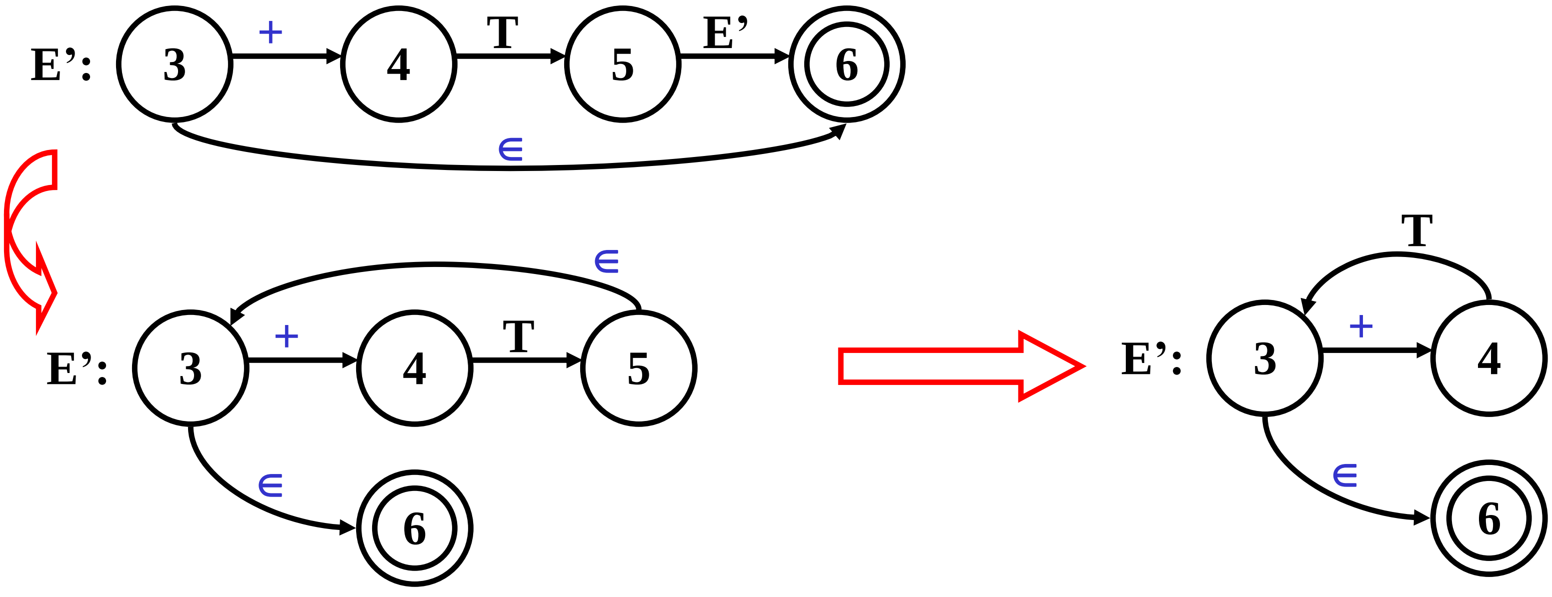
E':



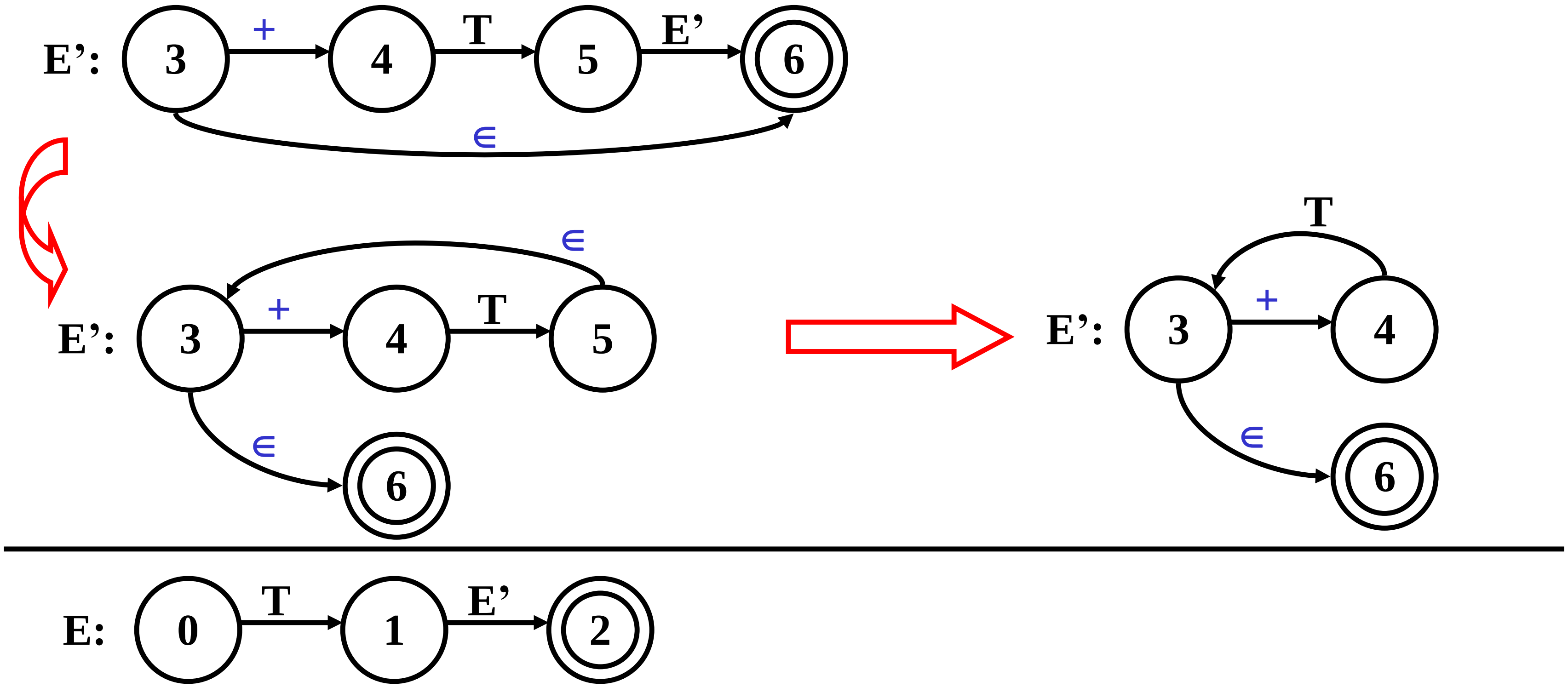
Transition Diagrams can be Simplified (2)



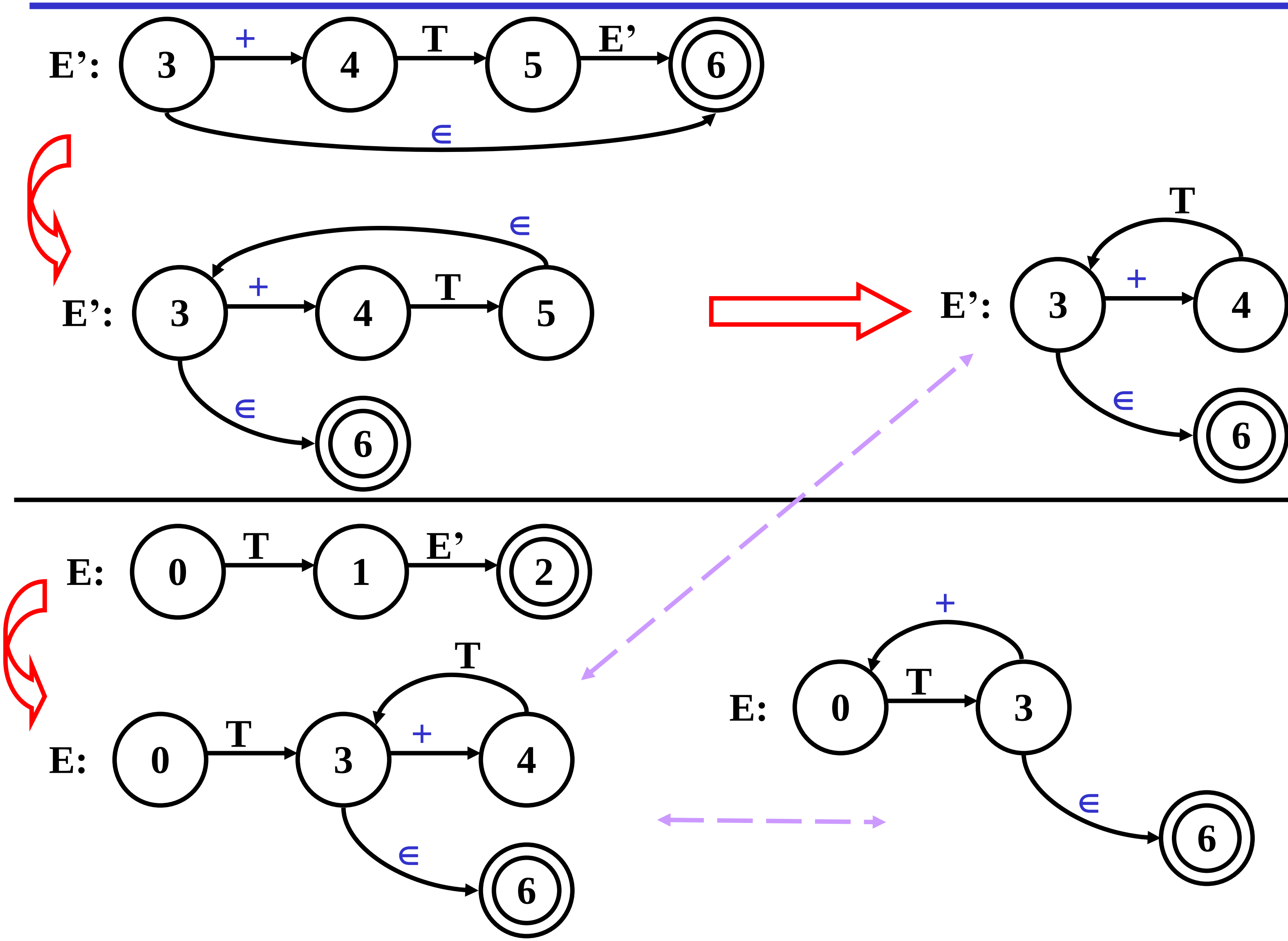
Transition Diagrams can be Simplified (3)



Transition Diagrams can be Simplified (4)

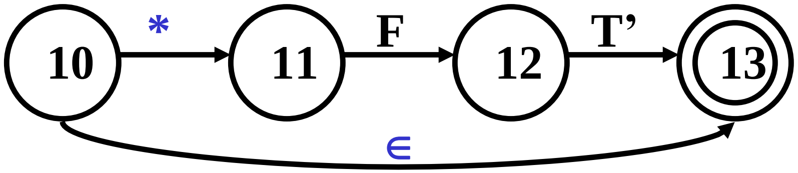


Transition Diagrams can be Simplified (5)

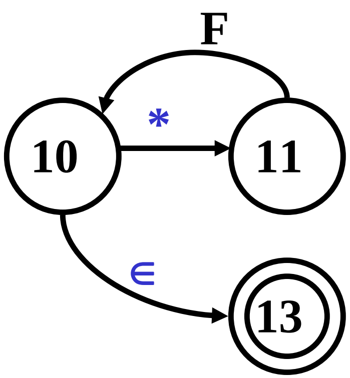


Similar steps for T and T'

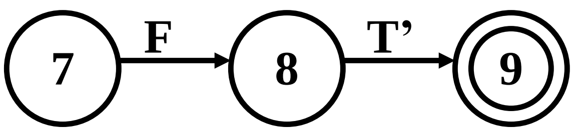
T':



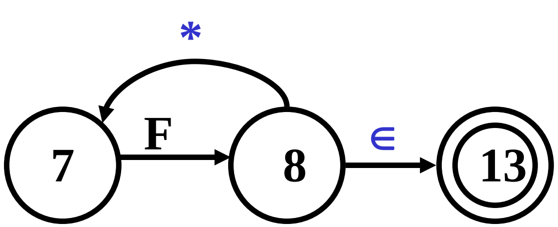
T':



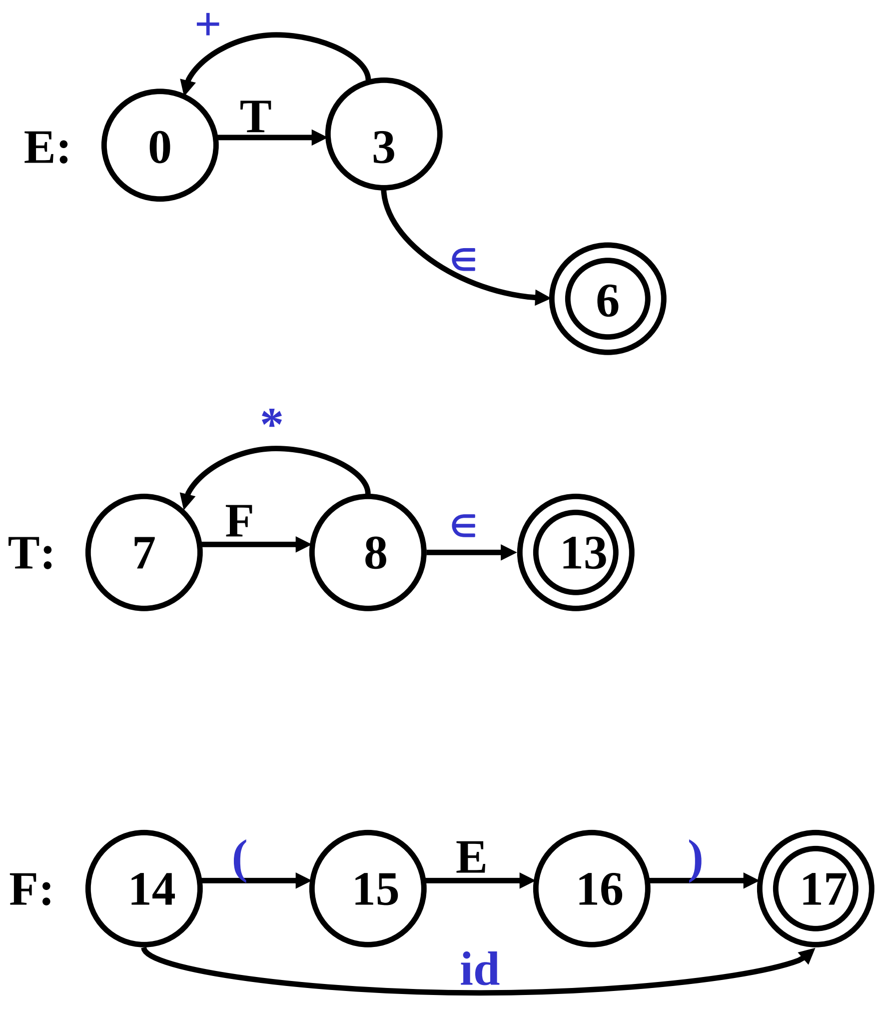
T:



T:



Simplified Transition diagrams



Implementing Panic-Mode Recovery

- The choice for the synchronizing set is important for improving the performance of the panic mode method.

- We define First(A) $\cup$ Follow(A) as the synchronizing set of non-terminal A.

Implementing Panic-Mode Recovery (Cont.)

Suppose the parser is in diagram A, the current token is a, and a syntax error is detected:

1. if a $\notin$ Follow (A),
   Report the error by 'illegal a found on line N',
   where N is the line number of token a,
   then get the next token from the scanner, and then call diagram A .

2. if a $\in$ Follow (A),
   Report the error by: 'missing A[1] on line N', where N is the line number of token a; then resume parsing by exiting from A.

[1] Note that in a real compiler, in the error message, A should be replaced by a simple token that can be derived from A.

Implementing Panic-Mode Recovery (Cont.)

3. Suppose the error has been caused by a mismatch between the current token a and the expected token b on link L in Diagram A:
   Report the error by the message 'missing b on line N, where N is the line number of token a, and continue the parsing in diagram A from the end of link L.

Question?

Choose the next parse state given the grammar, parse table, and current state below. The initial string is:

if true then { true } else { if false then { false } } $

	if	then	else	{	}	true	false	$
E	if B then { E } E'				ε	B	B	ε
E'			else { E }		ε			ε
B						true	false	

	Stack	Input
Current	E' $	else { if false then { false } } $
○	$	$
○	else {E} $	else { if false then { false } } $
○	E} $	if false then { false } } $
○	else {if B then {E} E' } $	else { if false then { false } } $

E → if B then { E } E'
    | B | ε
E' → else { E } | ε
B → true | false

Question?

For the given grammar, find the First and Follow of Non-terminals and the Parse table

$S \rightarrow$ i E t S S′ | a	First(S) =	Follow(S) =
$S' \rightarrow$ e S | $\in$	First(S′) =	Follow(S′) =
$E \rightarrow$ b	First(E) =	Follow(E) =

	a	b	e	i	t	$
S						
S′						
E						

Question?

For the given grammar, find the First and Follow of Non-terminals and the Parse table

$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \epsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \epsilon$
$F \rightarrow (\ E\ ) \mid id$

First(E,T,F) =

First(E') =

First(T') =

Follow(E) =

Follow(E') =

Follow(T) =

Follow(T') =

Follow(F) =

	id	+	*	(	)	$
E						
E'						
T						
T'						
F						

Question?

- Consider the grammar

$E \rightarrow T\,X \qquad\qquad X \rightarrow +\,E \mid \varepsilon$

$T \rightarrow (\,E\,) \mid \text{int}\;Y \qquad\qquad Y \rightarrow *\,T \mid \varepsilon$

- Convert the given grammar to a transition diagram
- Simplify the Diagram (if it is possible)
- Write a step-by-step parsing of input 'int * int'
- Draw the parse tree of the input