URBAN EDGE 21/1/20

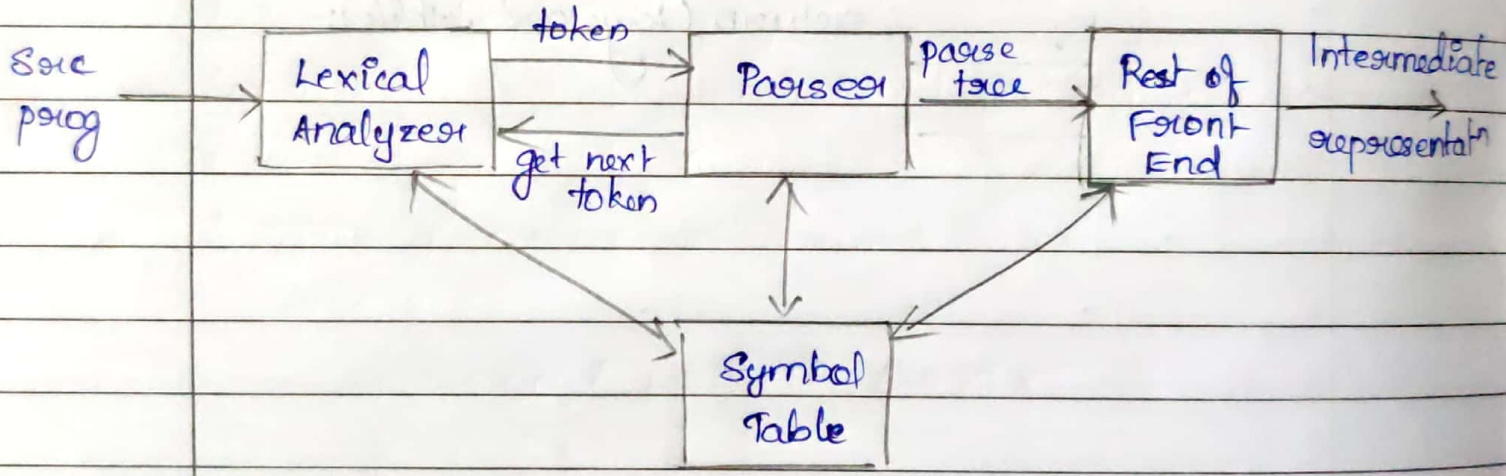
Module 2

Syntax Analyzer

- It creates syntactic structure of the given source prog
- Syntactic structure is mostly a parse tree
- It is also known as parser
- It checks whether a given src prog satisfies the rules implied by a CFG or not.

→ If it satisfies, parser creates parse tree of the prog

The Role of Parser



Error Handling

- Common programming errors

→ Lexical errors

→ Syntactic errors

→ Semantic errors

- Error handler goals

→ Report the presence of errors clearly & accurately

→ Recover from each error quickly enough to detect subsequent errors

URBAN EDGE 21/02/20

→ Add minimal overhead to the processing of correct progs

Context- Free Grammars

- Terminals
- Non-terminals
- Start symbol
- Product's
- In order to design a parser, the grammar must be unambiguous

eg: $E \to E+E \mid E*E \mid id$

⇒ removing ambiguity

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to id$

⇒ Removing left recursion ⇒ $E \to TE'$, $E' \to +TE' \mid \varepsilon$, $T \to FT'$, $T' \to *FT' \mid \varepsilon$

$E \to EA \mid id$

$A \to +E \mid *E$

Parsers

(i) Top-Down Parser

→ Parse tree is created top to bottom, starting from the root.

(ii) Bottom-Up Parser

→ Parse is created bottom to top, starting from the leaves

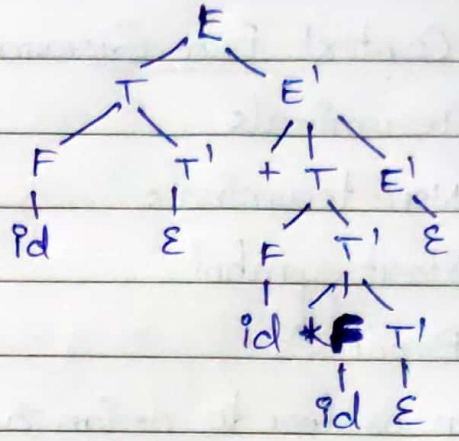
- Both top-down & bottom-up parsers scan the input

- LL → for top-down parser
- LR → " bottom-up "

URBAN EDGE 21/1/20

Top Down Parsing

$E \to TE'$	id+id*id
$E' \to +TE' \mid \varepsilon$	
$T \to FT'$	
$T' \to *FT' \mid \varepsilon$	
$F \to id$	



- 2 methods:

(i) With Backtracking → Recursive Descent parsing

(ii) Without Backtracking → Predictive parsing

- Recursive Descent Parsing

→ Consists of a set of procedures, 1 for each non-terminal.

→ Execution begins with the procedure for start symbol.

→ A typical procedure for a non-terminal.

```
void A() {
    choose an A-production, A → X1 X2 ... Xk
    for (i=1 to k) {
        if (Xi is a nonterminal)
            call procedure Xi();
        else if (Xi equals the current input symbol a)
            advance the i/p to the next symbol;
        else /* an error has occurred */
        }
    }
```

eg: $E \to TE'$ $T \to FT'$ $F \to id$

$E' \to +TE' \mid \varepsilon$ $T' \to *FT' \mid \varepsilon$

URBAN EDGE 21/1/20

```
E()
{ T()
  E'()
}

E'()
{ if (ip='+')
      increment ip;
      T();
      E'();
  else if (" ")
      ε
  else
      error msg
}

T()
{ F()
  T'()
}
```

S → cAd

A → ab | a

```
Procedure S()
Begin
    if ip='c'
        Advance();
    endif
    A();
    if ip='d'
        Advance();
        return true;
    endif
end
```

```
Procedure A()
Begin
    iptr = ip;
    if ip='a'
        Advance();
        if ip='b'
            Advance();
            return true;
        endif
    endif
    ip = iptr;
    if ip='a'
        Advance();
        return true;
    endif
end
```

→ Recursive descent parsing supports backtracking.

28/1/20

Predictive Parsers

- A CFG may be parsed by recursive-descent parser that needs no backtracking. if left recursions eliminated : left factoring

transformations applied.

- Building a predictive parser using recursive procedures by:

→ Creating transition diagrams

→ Match terminals, & making procedure call for non-terminals

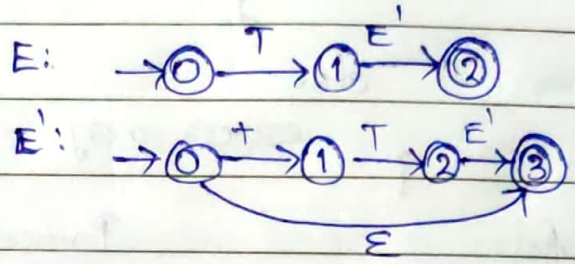
eg. $E \to TE'$

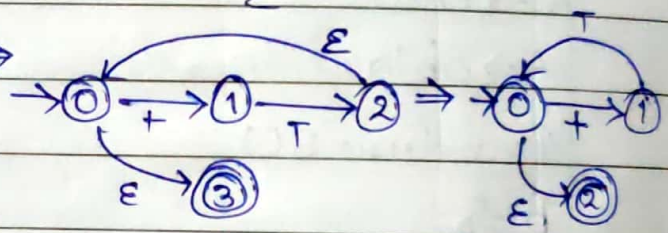
$E' \to +TE' | \varepsilon$

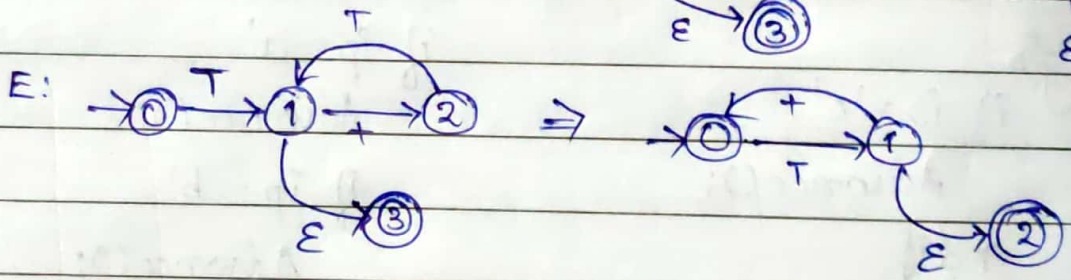
$T \to FT'$

$T' \to *FT' | \varepsilon$

$F \to id$

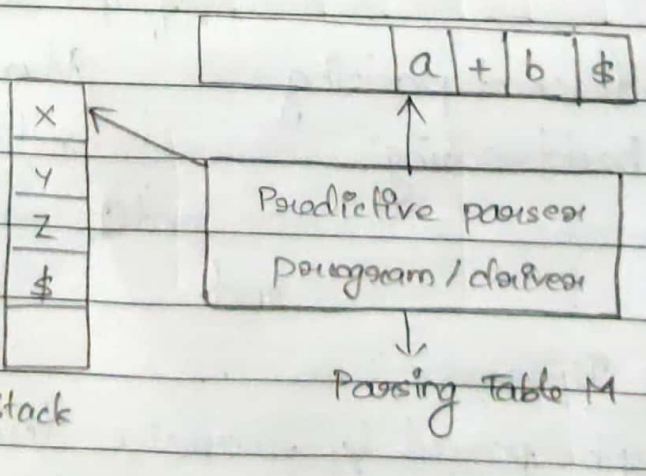






Non-Recursive Predictive Parsing

- A non-recursive predictive parser can be built by maintaining a stack explicitly.



- The parser mimics a leftmost derivation.

URBAN EDGE 23/1/20

The non-recursive predictive parser has an i/p buffer, a stack, a parsing table, & an o/p stream. The i/p buffer contains the string to be parsed, followed by the end marker $, stack contains a sequence of grammar symbols. The parsing table is constructed by an algorithm.

X → Top of stack $ → End of string

a → Current i/p symbol

(i) If X=a=$
    accept

(ii) else error

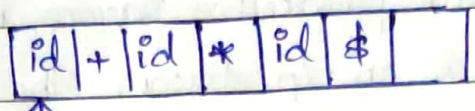
(iii) If X=a≠$
    POP();

(iv) If X is a non-terminal
    POP();
    sequence of PUSH();

eg:

Non-Terminal	Input Symbol					
	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→id			F→(E)		

URBAN EDGE 23/1/20



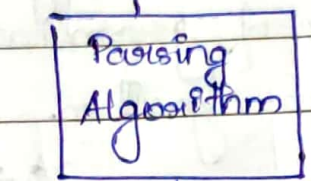
X = E
a = id

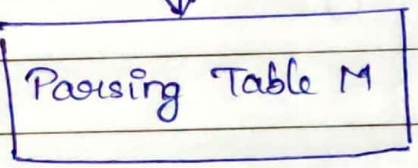
M[X, a]
M[E, id]

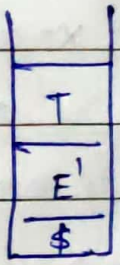
~~E → TE'~~ E → TE'

Top of stack = E (Non-terminal)

⇒ Pop E & push TE'

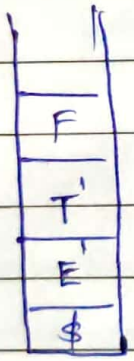






⇒ X = T

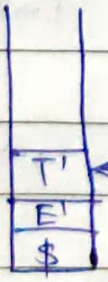
Top of stack = T ⇒ Pop T & push FT'

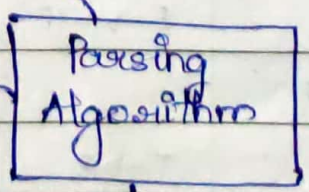


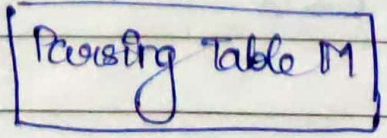
X = F
a = id

Top of stack = F ⇒ Pop F & push id

X = a = id ⇒ Pop id & advance i/p pointer to next symbol



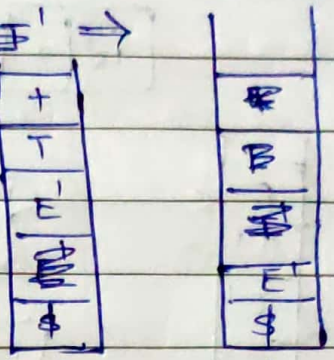




URBAN EDGE 23/1/20

$X = T'$ ⇒ Pop T' & push ε ⇒

$a = +$

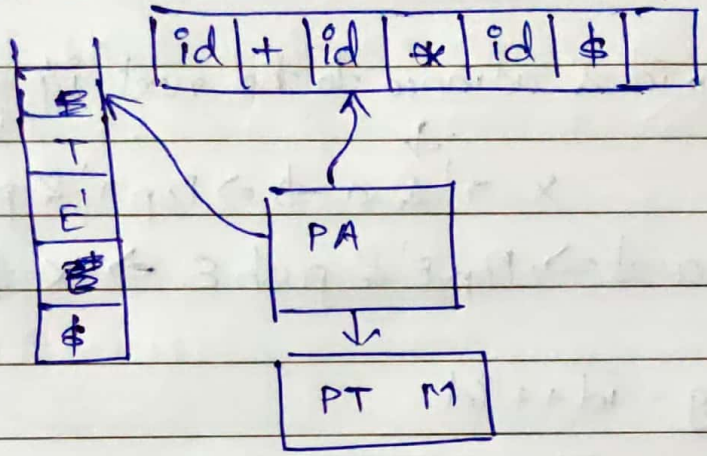


$X = E'$ ⇒ Pop E' & push $+TE'$ ⇒

$a = +$

$X = +$

$a = +$ ⇒ Pop + & advance to next i/p symbol



$X = E$ T

$a = id$

ε ⇓

Pop T & push FT'

⇓

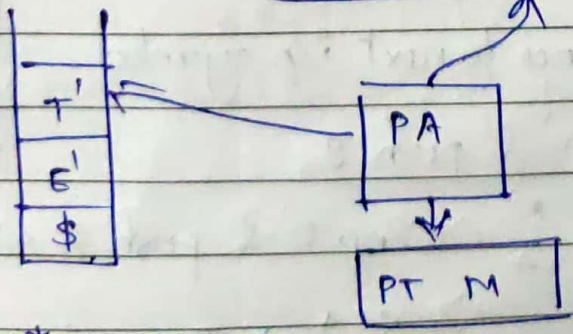
$X = F$ ⇒ Pop F & push id ⇒

$a = id$

$X = id$

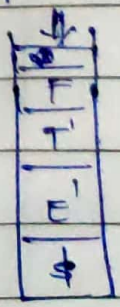
$a = id$ ⇒ Pop id & advance to next i/p symbol

⇓



$X = T'$ ⇒ Pop T' &

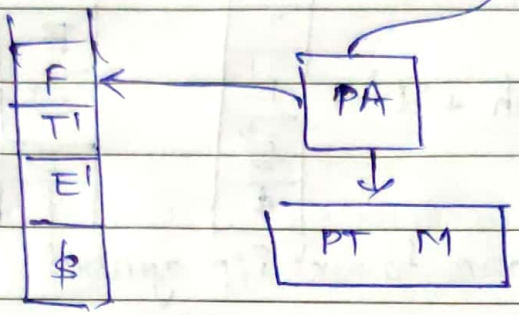
$a = *$ push ε FT'



$X = \varepsilon$ ⇒ Pop ε & ~~push ε~~ ⇒

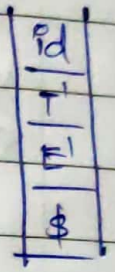
$a = *$ advance to next i/p symbol

URBAN EDGE 23/1/20



X = F ⇒ Pop F & push id

a = id



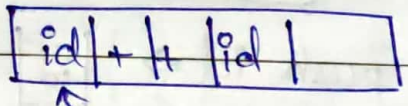
X = id
a = id ⇒ Pop id & advance to the next i/p symbol

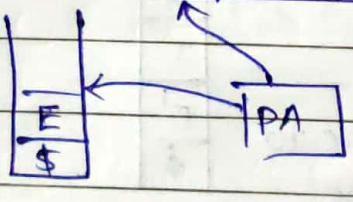
⇓

X = T' & a = $ ⇒ Pop T' & push ε

⇒ X = E' & a = $ ⇒ Pop E' & push ε ⇒ X = $ & a = $ ⇒ accept

i/p string = id + + id

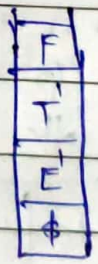




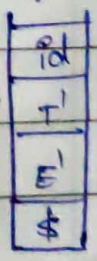
X = E ⇒ Pop E & push TE'

a = id

X = T
a = id ⇒ Pop T & push FT' ⇒



X = F
a = id ⇒ Pop F & push id



X = id
a = id ⇒ Pop id & advance to next i/p symbol

⇒ X = T'
a = + ⇒ Pop T' & push ε

⇒ X = E'
a = + ⇒ Pop E' & push ε + TE'

⇒ X = +
a = + ⇒ Pop + & advance to next i/p symbol

⇒ X = T, a = + ⇒ ~~Pop T & push~~ Error

URBAN EDGE 24/1/20

Construct^{n} of Predictive Parsing Table

Input: Grammar G

Output: Parsing Table

Method:-

(i) For each product^{n} $A \to \alpha$ of grammar, do steps 2 & 3.

(ii) For " terminal a in FIRST(α), add $A \to \alpha$ to M[A, a]

(iii) If ε is in FIRST(α), add $A \to \alpha$ to M[A, b] for each terminal b in FOLLOW(A). If ε is in FIRST(α) & \$ is in FOLLOW(A), add $A \to \alpha$ to M[A, \$]

(iv) Make each undefined entry of M be error

FIRST & FOLLOW

- FIRST(X) ~~= set of terminals~~

→ If X is a terminal, FIRST(X) = {X}

→ If $X \to \varepsilon$ is a product^{n}, then add ε to FIRST(X)

→ If $X \to Y_1 Y_2 \cdots Y_k$ is a product^{n},

★ Place a in FIRST(X) if a in FIRST(Y_i) & ε is in all of FIRST(Y_1) ... FIRST(Y_{i-1})

★ Add ε to FIRST(X) if ε is in all FIRST(Y_i).

eg: $X \to aB \mid +A \mid bC$

FIRST(X) = {a, +, b}

$X \to \varepsilon$

FIRST(X) = {ε, ...}

$X \to Y_1 Y_2 \cdots Y_k$

FIRST(X) = FIRST(Y_1)

$X \to \varepsilon \Rightarrow$ FIRST(X) = FIRST(Y_1) ∪ FIRST(Y_2)

$X \to Y_2 Y_3 \cdots Y_k$

URBAN EDGE 24/1/20

$X \to Y_1 Y_2 \ldots Y_k \Rightarrow X \to \varepsilon$

if $Y_1 \to \varepsilon, Y_2 \to \varepsilon, \ldots Y_k \to \varepsilon$

FIRST(X) = FIRST(Y_1) ∪ FIRST(Y_2) ... FIRST(Y_k) ∪ {ε}

- FOLLOW

→ Place $ in FOLLOW(S), S is the start symbol, & $ is the end-marker.

If there is a production $A \to \alpha B \beta$, then

★ Everything in FIRST(β) except for ε, is placed in FOLLOW(B).

If there is a production $A \to \alpha B$, or a production $A \to \alpha B \beta$, & FIRST(β) contains ε, then

★ Everything in FOLLOW(A) is in FOLLOW(B).

$A \to \alpha B \beta$

FOLLOW(B) = FIRST(β)

eg: $A \to \underset{B}{B} \underset{\beta}{+C}$ ~~FIRST(B) =~~ FOLLOW(B) = FIRST(β) = FIRST(+C) = FIRST(+)

$A \to \underset{B}{B} \underset{\beta}{CDE}$ FOLLOW(B) = FIRST(β) = FIRST(CDE) = FIRST(C)

eg: $E \to TE'$ FIRST(E) = {id} = FIRST(T)

$E' \to +TE' \mid \varepsilon$ FIRST(E') = {+, ε}

$T \to FT'$ FIRST(T) = {id} = FIRST(F)

$T' \to *FT' \mid \varepsilon$ FIRST(T') = {*, ε}

$F \to id$ FIRST(F) = {id}

Adding 2 more options to production of F

$F \to id \mid (E) \mid \varepsilon \Rightarrow$ FIRST(F) = {id, (, ε}

(→ Terminal symbol. FIRST(T) = {id, (, ε, *}.

URBAN EDGE 24/1/20

FIRST (E) = {id, (, ε, *, +}

E → TE' FOLLOW(E) = {$}

E' → +TE' | ε FOLLOW(E') = {$}

T → FT' FOLLOW(T) = {+, $}

T' → *FT' | ε FOLLOW(T') = {+, $}

F → id FOLLOW(F) = {*, +, $}

FOLLOW(X) ⇒ Take those productions which contain X on RHS

FOLLOW(E') ⇒ E → TE' } A → αBβ
E' → +TE' } A → αB
→ Both are in A → αB form
B = E'

FOLLOW(T) ⇒ E → TE' }
E' → +TE' } Both are in A → αBβ form
B = T

FOLLOW(T) = FIRST(E') U FOLLOW(E) U FOLLOW(E') − {ε}

since FIRST (E') contains ε

FOLLOW(T') ⇒ T' → *FT' }
T → FT' } Both are in A → αB form
= FOLLOW(T)

FOLLOW(F) ⇒ T → FT' }
T' → *FT' } Both are in A → αB form ⇒ FOLLOW(F) = FOLLOW(T')

① S → iEtSS' | a

S' → eS | ε

E → b

Find FIRST & FOLLOW.

ans FIRST(S) = {i, a} FIRST(E) = {b}

FIRST(S') = {e, ε}

URBAN EDGE 24/1/20

FOLLOW(S) = {$, e}

FOLLOW(S') = {$, e}

FOLLOW(E) = {t}

FOLLOW(S) ⇒ S → iEtSS' ⇒ A → αBβ = FIRST(S') ∈ -ε

S' → eS ⇒ αB

= FOLLOW(S')

FOLLOW(S') ⇒ S → iEtSS' ⇒ A → αBβ ⇒ FOLLOW(S)

FOLLOW(E) ⇒ S → iEtSS' ⇒ A → αBβ ⇒ FIRST(t) = t

E → TE'	FIRST(E) = FIRST(T) = FIRST(F) = {id, (}	FOLLOW(E) = {), $}
E' → +TE' | ε		FOLLOW(E') = {), $}
T → FT'		FOLLOW(T) = {+,), $}
T' → *FT' | ε	FIRST(E') = {+, ε}	FOLLOW(T') = {+,), $}
F → id | (E)	FIRST(T') = {*, ε}	FOLLOW(F) = {+, *,), $}

	id	+	*	(	)	$
E	E → TE'			E → TE'		
E'		E' → +TE'			E' → ε	E' → ε
T	T → FT'			T → FT'		
T'		T' → ε	T' → *FT'		T' → ε	T' → ε
F	F → id			F → (E)		

E → TE' FIRST(T) = {id, (} = FIRST(E)

~~A → α~~ ↳ A → α form. Find FIRST(α) ↑

E' → +TE' FIRST(α) = {+} E' → ε ⇒ FIRST(α) = ε

A α

If FIRST(α) = ε, We need to consider FOLLOW(A) i.e. FOLLOW(E') ⇒ {), $}

URBAN EDGE 24/1/20

$T \to FT'$ $\quad FIRST(\alpha) = FIRST(F) = \{id, (\}$

$T' \to *FT'$ $\quad FIRST(\alpha) = \{*\}$ $\quad T' \to \varepsilon \Rightarrow FOLLOW(T') = \{+,), \$\}$

$F \to id$ $\quad FIRST(\alpha) = \{id\}$ $\quad F \to (E) \Rightarrow FIRST(\alpha) = \{(\}$

① $S \to iEtSS' \mid a$

$S' \to eS \mid \varepsilon$

$E \to b$

If parsing table doesn't contain multiple entries, it is known as LL(1) grammar.

Find parsing table

ans. $FIRST(S) = \{i, a\}$ $\quad FOLLOW(S) = \{\$, e\}$

$FIRST(S') = \{e, \varepsilon\}$ $\quad FOLLOW(S') = \{\$, e\}$

$FIRST(E) = \{b\}$ $\quad FOLLOW(E) = \{t\}$

	a	b	e	i	t	$
S	$S \to a$			$S \to iEtSS'$		
S'			$S' \to eS$			
			$S' \to \varepsilon$			$S' \to \varepsilon$
E		$E \to b$				

$S \to iEtSS' \Rightarrow FIRST(\alpha) = i$ $\quad S \to a \Rightarrow FIRST(\alpha) = a$

$S' \to eS \Rightarrow FIRST(\alpha) = e$ $\quad S' \to \varepsilon \Rightarrow FOLLOW(S) = \{\$, e\}$

$E \to b \Rightarrow FIRST(\alpha) = b$