COMPILER DESIGN

Syntactic analysis – part II

First and follow sets

Recursive descent and table-driven predictive parsing

Generating FIRST and FOLLOW sets

Generating FIRST sets

- If $\alpha \overset{*}{\Rightarrow} \beta$, where β begins with a terminal symbol **x**, then $\mathtt{x} \in \mathtt{FIRST}(\alpha)$.

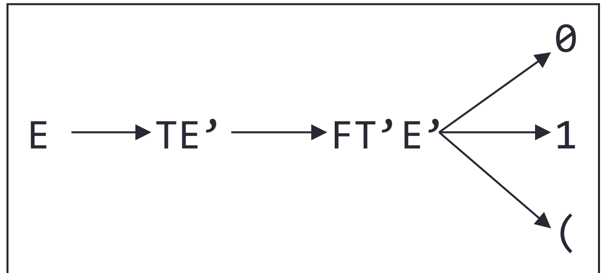
- Algorithmic definition:

```
FIRST(A) =
1. if ( (A∈T) ∨ (A is ε) )
   then FIRST(A) = {A}
2. if ( (A∈N) ∧ (A→S1S2…Sk ∈ R) | Si∈(N∪T) )
   then 2.1. FIRST(A) ⊇ (FIRST(S1) - {ε})
        2.2. if ∃i<k (ε ∈ FIRST(S1),…, FIRST(Si) )
             then FIRST(A) ⊇ FIRST(Si+1)
        2.3. if (ε ∈ FIRST(S1),…, FIRST(Sk) )
             then FIRST(A) ⊇ {ε}
```

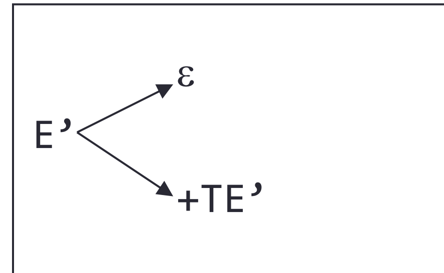
- Or, generate the **lookahead tree** for **A**

Example: lookahead trees

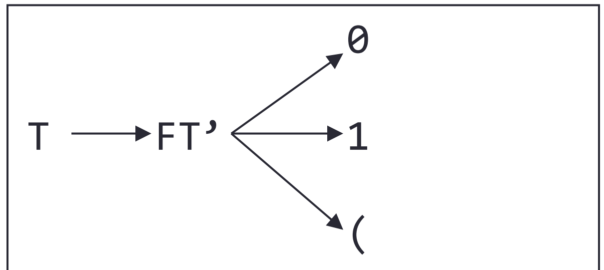
$$
\begin{array}{lcl}
E & \rightarrow & TE' \\
E' & \rightarrow & \varepsilon \mid +TE' \\
T & \rightarrow & FT' \\
T' & \rightarrow & \varepsilon \mid *FT' \\
F & \rightarrow & (\ E\ ) \mid 0 \mid 1
\end{array}
$$



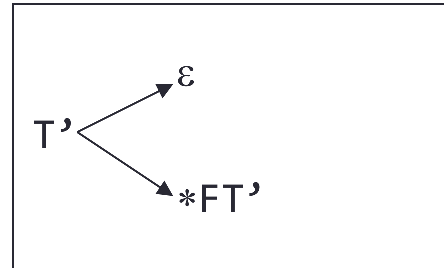
FIRST(E) = {0,1,(}



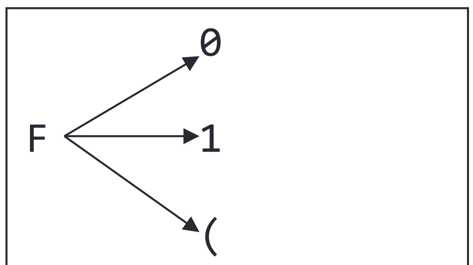
FIRST(E’) = {ε,+}



FIRST(T) = {0,1,(}



FIRST(T’) = {ε,*}



FIRST(F) = {0,1,(}

Generating the FOLLOW sets

- FOLLOW(A) is the set of terminals that can come right after an **A** in any sentential form derivable from the grammar of the language.

- Algorithmic definition:

```
FOLLOW( A | A ∈ N ) =
1. if ( A == S )
   then ( FOLLOW(A) ⊇ {$})
2. if ( B→αAβ ∈ R )
   then ( FOLLOW(A) ⊇ (FIRST(β) - {ε}) )
3. if ( (B→αAβ ∈ R) ∧ (β⇒*ε) )
   then ( FOLLOW(A) ⊇ FOLLOW(B) )
```

Example

```
E  → TE'
E' → ε | +TE'
T  → FT'
T' → ε | *FT'
F  → ( E ) | 0 | 1
```

```
FST(E)  : { 0,1,( }
FST(E') : { ε,+ }
FST(T)  : { 0,1,( }
FST(T') : { ε,* }
FST(F)  : { 0,1,( }
```

```
FLW(E)  : { $^1, )^4 }                  : { $,) }
FLW(E') : { {$,)}^6 }                   : { $,) }
FLW(T)  : { {+}^2, {$,)}^5,7 }          : { +,$,) }
FLW(T') : { {+,$,)}^9 }                 : { +,$,) }
FLW(F)  : { {*}^3, {+,$,)}^8,10 }       : { *,+,$,) }
```

```
1. (1) :          : (A == S) ⇒ ( FLW(A) ⊇ '$')                      : FLW(E) ⊇ {$}
2. (2) : E→TE'    : (B→αAβ) ⇒ ( FLW(A) ⊇ (FST(β) - {ε}) )           : FLW(T) ⊇ (FST(E') – {ε})
   (2) : E'→+TE'  : (B→αAβ) ⇒ ( FLW(A) ⊇ (FST(β) - {ε}) )           : FLW(T) ⊇ (FST(E') – {ε})
3. (2) : T→FT'    : (B→αAβ) ⇒ ( FLW(A) ⊇ (FST(β) - {ε}) )           : FLW(F) ⊇ (FST(T') – {ε})
   (2) : T'→*FT'  : (B→αAβ) ⇒ ( FLW(A) ⊇ (FST(β) - {ε}) )           : FLW(F) ⊇ (FST(T') – {ε})
4. (2) : F →(E)   : (B→αAβ) ⇒ ( FLW(A) ⊇ (FST(β) - {ε}) )           : FLW(E) ⊇ (FST( ) ) – {ε})
5. (3) : E→TE'    : ( (B→αAβ) ∧ (ε ∈ FST(β) ) ⇒ ( FLW(A) ⊇ FLW(B) )  : FLW(T) ⊇ FLW(E)
6. (3) : E→TE'    : ( (B→αAβ) ∧ (ε ∈ FST(β) ) ⇒ ( FLW(A) ⊇ FLW(B) )  : FLW(E')⊇ FLW(E)
7. (3) : E'→+TE'  : ( (B→αAβ) ∧ (ε ∈ FST(β) ) ⇒ ( FLW(A) ⊇ FLW(B) )  : FLW(T) ⊇ FLW(E')
   (3) : E'→+TE'  : ( (B→αAβ) ∧ (ε ∈ FST(β) ) ⇒ ( FLW(A) ⊇ FLW(B) )  : FLW(E')⊇ FLW(E')
8. (3) : T→FT'    : ( (B→αAβ) ∧ (ε ∈ FST(β) ) ⇒ ( FLW(A) ⊇ FLW(B) )  : FLW(F) ⊇ FLW(T)
9. (3) : T→FT'    : ( (B→αAβ) ∧ (ε ∈ FST(β) ) ⇒ ( FLW(A) ⊇ FLW(B) )  : FLW(T')⊇ FLW(T)
10.(3) : T'→*FT'  : ( (B→αAβ) ∧ (ε ∈ FST(β) ) ⇒ ( FLW(A) ⊇ FLW(B) )  : FLW(F) ⊇ FLW(T')
   (3) : T'→*FT'  : ( (B→αAβ) ∧ (ε ∈ FST(β) ) ⇒ ( FLW(A) ⊇ FLW(B) )  : FLW(T')⊇ FLW(T')
```

```
FOLLOW( A | A ∈ N ) =
1. if ( A == S )
   then ( FOLLOW(A) ⊇ {$})
2. if ( B→αAβ ∈ R )
   then ( FOLLOW(A) ⊇ (FIRST(β) - {ε}) )
3. if ( (B→αAβ ∈ R) ∧ (β⇒ε) )
   then ( FOLLOW(A) ⊇ FOLLOW(B) )
```

Recursive descent predictive parsing

Method

- Build FIRST and FOLLOW sets
- For each non-terminal, we have a corresponding function
- In each function, for each possible right-hand-side of the corresponding productions, we have a possible path to follow.
- The choice is made according to the FIRST set of the right hand sides.
- If one of the alternatives is of the form $A \rightarrow \varepsilon$, the path is followed according to the FOLLOW set of the left-hand-side of the production.
- If no valid path is found, the function returns *false.*
- If any of the paths is followed without error, the function returns *true*.

Constructing the parser

- Main parser function is:

```
parse(){
    lookahead = nextToken()
    if (startSymbol() ∧ match(“$”) ) return(true)
    else return(false)}
```

- function to match tokens with the lookahead symbol (next token in input):

```
match(token){
    if (lookahead == token)
        lookahead = nextToken(); return(true)
    else
        lookahead = nextToken(); return(false)}
```

Constructing the parser

- For each non-terminal in the grammar, we construct a parsing function. All parsing functions have the same form:

```
LHS(){                                                  // LHS→RHS1 | RHS2 | … | ε
  if (lookahead ∈ FIRST(RHS1) )                         // LHS→RHS1
    if (non-terminals() ∧ match(terminals) )
      write(“LHS→RHS1”) ; return(true)
    else return(false)
  else if (lookahead ∈ FIRST(RHS2) )                    // LHS→RHS2
    if (non-terminals() ∧ match(terminals) )
      write(“LHS→RHS2”) ; return(true)
    else return(false)
  else if …                                             // other right hand sides
  else if (lookahead ∈ FOLLOW(LHS) )                    // only if LHS→ε exists
      write(“LHS→ε”) ; return(true)
  else
    return(false)}
```

Example

```
E  → TE′
E′ → ε | +TE′
T  → FT′
T′ → ε | *FT′
F  → ( E ) | 0 | 1
```

```
E(){ // E→TE’
    if (lookahead ∈ FIRST(TE’) )
        if (T() ∧ E’() )
            write(“E→TE’ ”) ; return(true)
        else return(false)
    else
        return(false)}
```

Example

```
E  → TE′
E′ → ε | +TE′
T  → FT′
T′ → ε | *FT′
F  → ( E ) | 0 | 1
```

```
E’(){ // E’→+TE’ | ε
    if (lookahead ∈ FIRST(+TE’) )
        if ( match(‘+’) ∧ T() ∧ E’() )
            write(“E’→+TE’ ”) ; return(true)
        else return(false)
    else if (lookahead ∈ FOLLOW(E’) )
        write(“E’→ε”) ; return(true)
    else
        return(false)}
```

Example

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow \varepsilon \mid +TE' \\
T &\rightarrow FT' \\
T' &\rightarrow \varepsilon \mid *FT' \\
F &\rightarrow (\ E\ ) \mid 0 \mid 1
\end{aligned}
$$

```
T(){ // T→FT’
    if (lookahead ∈ FIRST(FT’) )
        if (F() ∧ T’() )
            write(“T→FT’ ”) ; return(true)
        else return(false)
    else
        return(false)}
```

Example

```
E  → TE′
E′ → ε | +TE′
T  → FT′
T′ → ε | *FT′
F  → ( E ) | 0 | 1
```

```
T’(){ // T’→*FT’ | ε
    if (lookahead ∈ FIRST(*FT’) )
        if ( match(‘*’) ∧ F() ∧ T’() )
            write(“T’→*FT’ ”) ; return(true)
        else return(false)
    else if (lookahead ∈ FOLLOW(T’) )
        write(“T’→ε”) ; return(true)
    else
        return(false)}
```

Example

```
E  → TE′
E′ → ε | +TE′
T  → FT′
T′ → ε | ∗FT′
F  → ( E ) | 0 | 1
```

```
F(){ // F→ 0 | 1 | (E)
     if (lookahead ∈ FIRST(0) )
          if ( match(‘0’) )
               write(“F→0”) ; return(true)
          else return(false)
     else if (lookahead ∈ FIRST(1) )
          if ( match(‘1’) )
               write(“F→1”) ; return(true)
          else return(false)
     else if (lookahead ∈ FIRST((E)) )
          if ( match(‘(’) ∧ E() ∧ match(‘)’) )
               write(“F→(E)”) ; return(true)
          else return(false)
     else
          return(false)}
```

Table-driven predictive parsing

Method

- Build FIRST and FOLLOW sets
- Build the parser table
- Implement the parser algorithm

Building the parsing table

- Algorithm:

```
1. ∀p : ( (p ∈ R) ∧ (p : A→α) )
          do steps 2 and 3
2. ∀t : ( (t ∈ T) ∧ (t ∈ FIRST(α)) )
          add A→α to TT[A, t]
3. if ( ε ∈ FIRST(α) )
          ∀t : ((t ∈ T) ∧ (t ∈ FOLLOW(A)) )
               add A→α to TT[A, t]
4. ∀e : ( (e ∈ TT) ∧ (e == ∅) )
          add "error" to e
```

Building the parsing table

```
E  → TE′
E′ → ε | +TE′
T  → FT′
T′ → ε | *FT′
F  → ( E ) | 0 | 1
```

```
FST(E)  : { 0,1,( }
FST(E’) : { ε,+ }
FST(T)  : { 0,1,( }
FST(T’) : { ε,* }
FST(F)  : { 0,1,( }
```

```
FLW(E)  : { $,) }
FLW(E’) : { $,) }
FLW(T)  : { +,$,) }
FLW(T’) : { +,$,) }
FLW(F)  : { *,+,$,) }
```

```
1. ∀p : ( (p ∈ R) ∧ (p : A→α) )
        do steps 2 and 3
2. ∀t : ( (t ∈ T) ∧ (t ∈ FIRST(α)) )
        add A→α to TT[A, t]
3. if ( ε ∈ FIRST(α) )
        ∀t : ( (t ∈ T) ∧ (t ∈ FOLLOW(A)) )
              add A→α to TT[A, t]
4. ∀e : ( (e ∈ TT) ∧ (e == ∅) )
        add “error” to e
```

```
r1: E → TE’   : FIRST(TE’)  = {0,1,(} : TT[E,0][E,1][E,(]
r2: E’→ +TE’  : FIRST(+TE’)= {+}      : TT[E’,+]
r3: E’→ ε     : FOLLOW(E’) = {$,)}    : TT[E’,$][E’,)]
r4: T → FT’   : FIRST(FT’)  = {0,1,(} : TT[T,0][T,1][T,(]
r5: T’→ *FT’  : FIRST(*FT’)= {*}      : TT[T’,*]
r6: T’→ ε     : FOLLOW(T’) = {+,$,)}  : TT[T’,+][T’,$][T’,)]
r7: F → 0     : FIRST(0)    = {0}     : TT[F,0]
r8: F → 1     : FIRST(1)    = {1}     : TT[F,1]
r9: F → (E)   : FIRST((E))  = {(}     : TT[F,(]
```

Building the parsing table

```
r1: E  → TE’       r5: T’→ *FT’
r2: E’→ +TE’       r6: T’→ ε
r3: E’→ ε          r7: F → 0
r4: T  → FT’       r8: F → 1
                   r9: F → (E)
```

TT	0	1	(	)	+	*	$
E	r1	r1	r1				
E’				r3	r2		r3
T	r4	r4	r4				
T’				r6	r6	r5	r6
F	r7	r8	r9				

Parsing algorithm

```
parse(){
    push($)
    push(S)
    a = nextToken()
    while ( top() ≠ $ ) do
        x = top()
        if ( x ∈ T )
            if  ( x == a )
                pop() ; a = nextToken()
            else
                skipErrors() ; error = true
        else
            if ( TT[x,a] ≠ 'error' )
                pop() ; inverseRHSMultiplePush(TT[x,a])
            else
                skipErrors() ; error = true
    if ( ( a ≠ $ ) ∨ ( error == true ) )
        return(false)
    else
        return(true)}
```

*: **skipErrors()** will be explained in lecture 5

Table parsing example

	Stack	Input	Production	Derivation
1	$E	(0+1)*0$		E
2	$E	(0+1)*0$	r1:E→TE'	⇒TE'
3	$E'T	(0+1)*0$	r4:T→FT'	⇒FT'E'
4	$E'T'F	(0+1)*0$	r9:F→(E)	⇒(E)T'E'
5	$E'T')E(	(0+1)*0$		
6	$E'T')E	0+1)*0$	r1:E→TE'	⇒(TE')T'E'
7	$E'T')E'T	0+1)*0$	r4:T→FT'	⇒(FT'E')T'E'
8	$E'T')E'T'F	0+1)*0$	r7:F→0	⇒(0T'E')T'E'
9	$E'T')E'T'0	0+1)*0$		
10	$E'T')E'T'	+1)*0$	r6:T'→ε	⇒(0E')T'E'
11	$E'T')E'	+1)*0$	r2:E'→+TE'	⇒(0+TE')T'E'
12	$E'T')E'T+	+1)*0$		

Table parsing example

	Stack	Input	Production	Derivation
13	$E’T’)E’T	1)*0$	r4:T→FT’	⇒(0+FT’E’)T’E’
14	$E’T’)E’T’F	1)*0$	r8:F→1	⇒(0+1T’E’)T’E’
15	$E’T’)E’T’1	1)*0$		
16	$E’T’)E’T’	)*0$	r6:T’→ε	⇒(0+1E’)T’E’
17	$E’T’)E’	)*0$	r3:E’→ε	⇒(0+1)T’E’
18	$E’T’)	)*0$		
19	$E’T’	*0$	r5:T’→*FT’	⇒(0+1)*FT’E’
20	$E’T’F*	*0$		
21	$E’T’F	0$	r7:F→0	⇒(0+1)*0T’E’
22	$E’T’0	0$		
23	$E’T’	$	r6:T’→ε	⇒(0+1)*0E’
24	$E’	$	r3:E’→ε	⇒(0+1)*0
25	$	$		success