# Kap.4 – del I
# Top Down Parsering
# INF5110 – v2006

Arne Maus
Ifi, UiO

## Innhold

- Motivering
  - Boka gir først parsering uten First/Follow-mengder og så innfører dem.
  - Vi tar ”teorien” først
- First og Follow-mengder
- Fjerning av venstre-rekursjon
- Venstre-faktorisering
- Rekursive nedstigning
- LL(1) grammatikker

## First og Follow-mengder, hvorfor

- Når vi leser token og skal analysere en ikke-terminal, ønsker vi:
  - finne ut hvilken produksjon (høyreside-alternativ) som kommer nå - ser på:
    - First-mengdene
      Det vi leste må da være starten på ....
  - I kap 4: Sjekke at det som etterfølger det vi nå har lest og tolket som en ikke-terminal, er lovlig etterfølger (feil-sjekking)
    I kap. 5 (bottom up) finne riktig reduksjon av det vi hittil har lest på input .
    - Follow-mengdene

## FirstMengder

*terminal*

Def { First (A) = { a | finnes avledning A $\Rightarrow^*$ a $\alpha$ }
Dessuten: Om A ”er utnullbar”, så er $\varepsilon \in$ First(A)

Per def er da: First(a) = { a }
Def ”A er utnullbar” $\Leftrightarrow$ A $\Rightarrow^* \varepsilon$

Let $X$ be a grammar symbol (a terminal or nonterminal) or $\varepsilon$. Then the set **First($X$)** consisting of terminals, and possibly $\varepsilon$, is defined as follows.

1. If $X$ is a terminal or $\varepsilon$, then First($X$) = {$X$}.
2. If $X$ is a nonterminal, then for each production choice $X \rightarrow X_1X_2 \ldots X_n$, First($X$) contains First($X_1$) − {$\varepsilon$}. If also for some $i < n$, all the sets First($X_1$), . . . , First($X_i$) contain $\varepsilon$, then First($X$) contains First($X_{i+1}$) − {$\varepsilon$}. If all the sets First($X_1$), . . . , First($X_n$) contain $\varepsilon$, then First($X$) also contains $\varepsilon$.

Now define **First($\alpha$)** for any string $\alpha = X_1X_2 \ldots X_n$ (a string of terminals and nonterminals), as follows. First($\alpha$) contains First($X_1$) − {$\varepsilon$}. For each $i = 2, \ldots, n$, if First($X_k$) contains $\varepsilon$ for all $k = 1, \ldots, i-1$, then First($\alpha$) contains First($X_i$) − {$\varepsilon$}. Finally, if for all $i = 1, \ldots, n$, First($X_i$) contains $\varepsilon$, then First($\alpha$) contains $\varepsilon$.

```
for all nonterminals A do First(A) := {};
while there are changes to any First(A) do
  for each production choice A → X1X2 ... Xn do
    k := 1; Continue := true;
    while Continue = true and k <= n do
      add First(Xk) - {ε} to First(A);
      if ε is not in First(Xk) then Continue := false;
      k := k + 1;
    if Continue = true then add ε to First(A);
```

Algoritme:

- Gjør steg 1.
- Gjenta steg 2 til alle Firstmengdene har stabilisert seg.

## Eks. 4.9 Beregning av First-mengde

(1) *exp* → *exp addop term*
(2) *exp* → *term*
(3) *addop* → **+**
(4) *addop* → **-**
(5) *term* → *term mulop factor*
(6) *term* → *factor*
(7) *mulop* → *****
(8) *factor* → **(** *exp* **)**
(9) *factor* → **number**

| Grammar rule | Pass 1 | Pass 2 | Pass 3 |
|---|---|---|---|
| *exp* → *exp addop term* | | | |
| *exp* → *term* | | | First(*exp*) = {(, **number**} |
| *addop* → **+** | First(*addop*) = {**+**} | | |
| *addop* → **-** | First(*addop*) = {**+**, **-**} | | |
| *term* → *term mulop factor* | | | |
| *term* → *factor* | | First(*term*) = {(, **number**} | |
| *mulop* → ***** | First(*mulop*) = {*****} | | |
| *factor* → **(** *exp* **)** | First(*factor*) = {**(**} | | |
| *factor* → **number** | First(*factor*) = {(, **number**} | | |

Kan bare fylle ut en tabell

| | First |
|---|---|
| exp | |
| addop | |
| term | |
| multop | |
| factor | |

## Beregning av Followmengder

β γ *virkårlige streneger*

Def Follow (A) = { a | finnes avledning S $ ⇒* β A a γ }

Dvs. Det finnes en utledning fra startsymbolet S
hvor A kommer rett før a (a er en terminal)

Given a nonterminal *A*, the set **Follow(*A*)**, consisting of terminals, and possibly $, is defined as follows.

1. If *A* is the start symbol, then $ is in Follow(*A*).
2. If there is a production $B \rightarrow \alpha A \gamma$, then First($\gamma$) − {$\varepsilon$} is in Follow(*A*).
3. If there is a production $B \rightarrow \alpha A \gamma$ such that $\varepsilon$ is in First ($\gamma$), then Follow(*A*) contains Follow(*B*).

```
Follow(start-symbol) := {$} ;
for all nonterminals A ≠ start-symbol do Follow(A) := {} ;
while there are changes to any Follow sets do
  for each production A → X1X2 . . . Xn do
    for each Xi that is a nonterminal do
      add First(Xi+1Xi+2 . . . Xn) − {ε} to Follow(Xi)
      (* Note: if i=n, then Xi+1Xi+2 . . . Xn = ε *)
      if ε is in First(Xi+1Xi+2 . . . Xn) then
        add Follow(A) to Follow(Xi)
```

**Algoritme:**

- **Gjør steg 1.**
- **Gjenta steg 2 og 3 til alle Follow-mengdene har stabilisert seg.**

## Beregning av Followmengder 4.12

First(*exp*) = {(, **number**}
First(*term*) = {(, **number**}
First(*factor*) = {(, **number**}
First(*addop*) = {**+**, **-**}
First(*mulop*) = {*****}

Follow(*exp*) = {$, **+**, **-**, **)**}
Follow(*addop*) = {(, **number**}
Follow(*term*) = {$, **+**, **-**, *****, **)**}
Follow(*mulop*) = {(, **number**}
Follow(*factor*) = {$, **+**, **-**, *****, **)**}

(1) *exp* → *exp addop term*
(2) *exp* → *term*
(3) *addop* → **+**
(4) *addop* → **-**
(5) *term* → *term mulop factor*
(6) *term* → *factor*
(7) *mulop* → *****
(8) *factor* → **(** *exp* **)**
(9) *factor* → **number**

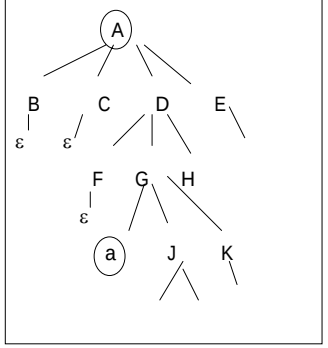
| Grammar rule | Pass 1 | Pass 2 |
|---|---|---|
| *exp* → *exp addop term* | Follow(*exp*) = {$, **+**, **-**}<br>Follow(*addop*) = {(, **number**}<br>Follow(*term*) = {$, **+**, **-**} | Follow(*term*) = {$, **+**, **-**, *****, **)**} |
| *exp* → *term* | | |
| *term* → *term mulop factor* | Follow(*term*) = {$, **+**, **-**, *****}<br>Follow(*mulop*) = {(, **number**}<br>Follow(*factor*) = {$, **+**, **-**, *****} | Follow(*factor*) = {$, **+**, **-**, *****, **)**} |
| *term* → *factor* | | |
| *factor* → **(** *exp* **)** | Follow(*exp*) = {$, **+**, **-**, **)**} | |

Kan bare fylle ut en tabell

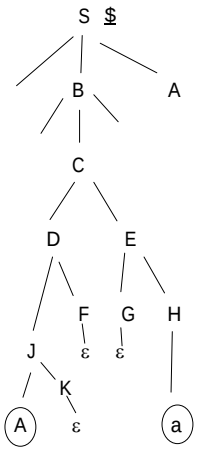
| | Follow |
|---|---|
| exp | |
| addop | |
| term | |
| multop | |
| factor | |

## Generelt om situasjonen for neste symbol i First- eller Follow-mengden

**a ∈ First (A)**




**a ∈ Follow(A)**

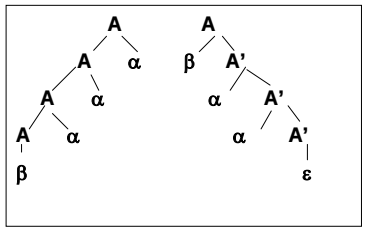


# Fjerning av venstre-rekursjon - case 1

Kan skrives på EBNF

$A \rightarrow A\alpha \mid \beta$

$A \rightarrow \beta \ \{\alpha\}$




$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \varepsilon$

Eksempel: $exp \rightarrow exp\ addop\ term \mid term$

This is of the form $A \rightarrow A\ \alpha \mid \beta$, with $A = exp$, $\alpha = addop\ term$, and $\beta = term$. Rewriting this rule to remove left recursion, we obtain

$$exp \rightarrow term\ exp'$$
$$exp' \rightarrow addop\ term\ exp' \mid \varepsilon$$

# Fjerning av venstre-rekursjon

**Case 2:**

$$A \rightarrow A\,\alpha_1 \mid A\,\alpha_2 \mid \ldots \mid A\,\alpha_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$$

$$\Downarrow$$

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A'$$
$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A' \mid \varepsilon$$

# Case 3 (indirekte venstre-rekursjon)

**A → Ba | Aa | c**
**B → Bb | Ab | d**

- **Her er det feil i boka i algoritmen s.159**
- **Ikke med som pensum**

$$A \rightarrow B\,a\,A' \mid c\,A'$$
$$A' \rightarrow a\,A' \mid \varepsilon$$
$$B \rightarrow B\,b \mid A\,b \mid d$$

$$A \rightarrow B\,a\,A' \mid c\,A'$$
$$A' \rightarrow a\,A' \mid \varepsilon$$
$$B \rightarrow B\,b \mid B\,a\,A'\,b \mid c\,A'\,b \mid d$$

$$A \rightarrow B\,a\,A' \mid c\,A'$$
$$A' \rightarrow a\,A' \mid \varepsilon$$
$$B \rightarrow c\,A'\,b\,B' \mid d\,B'$$
$$B' \rightarrow b\,B' \mid a\,A'\,b\,B' \mid \varepsilon$$

# Eks side 160 – Fjerning av venstre-rekursjon

*Den tradisjonelle entydige grammatikken*

$$exp \rightarrow exp\ addop\ term \mid term$$
$$addop \rightarrow \mathbf{+} \mid \mathbf{-}$$
$$term \rightarrow term\ mulop\ factor \mid factor$$
$$mulop \rightarrow \mathbf{*}$$
$$factor \rightarrow \mathbf{(}\ exp\ \mathbf{)} \mid \mathbf{number}$$

*Med fjernet venstre-rekursjon*

$$exp \rightarrow term\ exp'$$
$$exp' \rightarrow addop\ term\ exp' \mid \varepsilon$$
$$addop \rightarrow \mathbf{+} \mid \mathbf{-}$$
$$term \rightarrow factor\ term'$$
$$term' \rightarrow mulop\ factor\ term' \mid \varepsilon$$
$$mulop \rightarrow \mathbf{*}$$
$$factor \rightarrow \mathbf{(}\ exp\ \mathbf{)} \mid \mathbf{number}$$

# Venstre-faktorisering

Problem Ønsker ikke alternativer som starter likt:
A → abc B | ab C | DE

Løsning: A → ab A' | DE
A' → c B | C

Gjenta så lenge det er muligheter

Velg lengst mulig prefix

Gjør som over. Pass på å få med alle alternativer med dette prefixet

**while** *there are changes to the grammar* **do**
  **for** *each nonterminal A* **do**
    *let α be a prefix of maximal length that is shared by two or more production choices for A*
    **if** α ≠ ε **then**
      *let* $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ *be all the production choices for A and suppose that* $\alpha_1, \ldots, \alpha_k$ *share α, so that* $A \to \alpha\beta_1 \mid \ldots \mid \alpha\beta_k \mid \alpha_{k+1} \mid \ldots \mid \alpha_n$, *the* $\beta_j$*'s share no common prefix, and the* $\alpha_{k+1}, \ldots, \alpha_n$ *do not share α*
      *replace the rule* $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ *by the rules*
      $A \to \alpha A' \mid \alpha_{k+1} \mid \ldots \mid \alpha_n$
      $A' \to \beta_1 \mid \ldots \mid \beta_k$

# eks1:

*stmt-sequence* → *stmt* **;** *stmt-sequence* | *stmt*
*stmt* → **s**

*stmt-sequence* → *stmt stmt-seq'*
*stmt-seq'* → **;** *stmt-sequence* | ε

# Litt om stoffet i kap. 4

- First og Follow-mengder
  - Boka tar det sist, vi først
- Begrepet LL(1) – parsering
  - Boka reserverer dette begrepet for den metoden (kap 4.2 s. 152) der man bruker en eksplisitt stakk ( isteden for rekursive metoder, som i "recursive decent" – framgangsmåten)
  - Det vanlige er å bruke betegnelsen LL(1) –parsering også når rec.decent metoden brukes slavisk på en ren BNF-grammatikk
  - Kravet til en LL(1) – grammatikk kommer like tydelig fram ut fra begge disse metodene. Vi skal tenke like mye rec.decent som eksplisitt stakk. (boka gjør bare det siste).
  - Ofte brukes betegnelsen LL(1) – parsering også om rec.decent metoden brukt ut fra syntaksdiagrammer eller EBNF, men da er det uklart hva LL(1) – kravet til en grammatikk er.

# "Rekursiv nedstigning " – parsering

*exp* → *exp addop term* | *term*
*addop* → **+** | **-**
*term* → *term mulop factor* | *factor*
*mulop* → *****
*factor* → **(** *exp* **)** | **number**

- Skriv en funksjon /prosedyre/metode for hver ikke-terminal
- La denne lese/sjekke høyreside-alternativene

**token :** **if** ← *Global variabel*

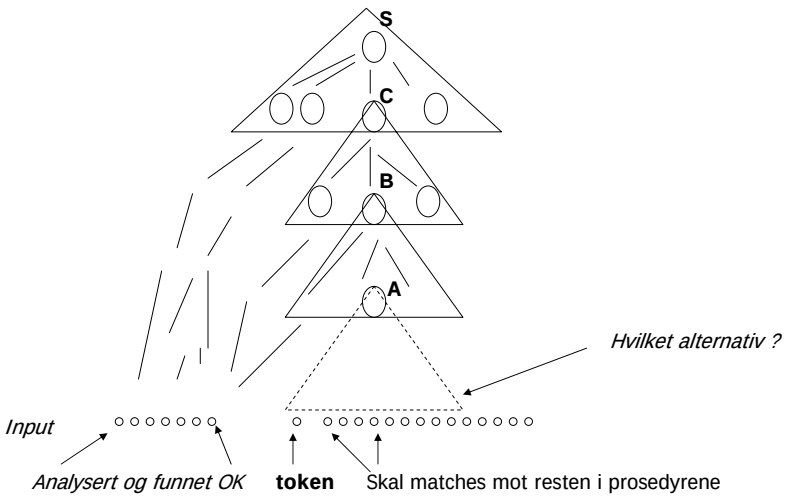
"Typisk" rec.dec. prosedyre for det enkle' tilfellet.

```
procedure factor ;
begin
  case token of
  ( :    match( ( ) ;
         exp ;
         match( ) ) ;
  number :
         match(number) ;
  else error ;
  end case ;
end factor ;
```

Sjekker at angitt terminal kommer, og "leser til neste". Brukes ofte bare for å lese (sjekken må slå til).

```
procedure match ( expectedToken ) ;
begin
  if token = expectedToken then
    getToken ;
  else
    error ;
  end if ;
end match ;
```

# Situasjonen under rekursiv parsering




# Litt mer kompliserte tilfeller kan løses med EBNF

*Opprinnelig*

*if-stmt* → **if** ( *exp* ) *statement* [ **else** *statement* ]

*Skrives ut som:*

*if-stmt* → **if** ( *exp* ) *statement*
| **if** ( *exp* ) *statement* **else** *statement*

*R-D-prosedyre:*

```
procedure ifStmt ;
begin
   match (if) ;
   match ( ( ) ;
   exp ;
   match ( ) ) ;
   statement ;
   if token = else then
      match (else) ;
      statement ;
   end if ;
end ifStmt ;
```

N.B: Kunne også bruke venstre- faktorisering. Da ville dette bli en egen prosedyre "elsePart":

**ifStmt** → **if (exp) stmt elsePart**
**elsePart** → **ε | else stmt**

# Venstre-rekursjon gir problemer

***Gir uendelig mange rekursive kall***

*exp* → *exp addop term* | *term*

N.B. Kan også fjernevenstre-rekursjon på trad måte, se senere foil.

Bruker EBNF:

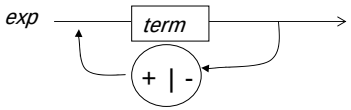
*exp* → *term* { *addop term* }

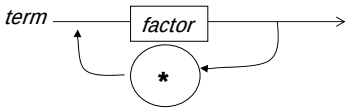
```
procedure exp ;
begin
   term ;
   while token = + or token = - do
      match (token) ;
      term ;
   end while ;
end exp ;
```

*term* → *term multop factor* | *factor*

*term* → *factor* { *mulop factor* }

```
procedure term ;
begin
   factor ;
   while token = * do
      match (token) ;
      factor ;
   end while ;
end term ;
```




# Hvordan "lage noe" under rec.-decent parsering?

- Mål: Ønsker å bygge abstrakt syntaks-tre
- Men foreløpig:
  - beregner verdien av et uttrykk (med venstre-assosiativitet)

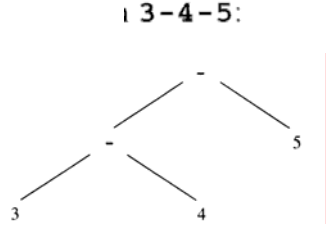
```
function exp : integer ;
var temp : integer ;
begin
   temp := term ;
   while token = + or token = - do
      case token of
         + : match (+) ;
             temp := temp + term ;
         - : match (-) ;
             temp := temp - term ;
      end case ;
   end while ;
   return temp ;
end exp ;
```

Kall

- Kan lett bygges ut til full "kalkulator" 3 + 4 + 5

# Bygging av syntaks-treet

I 3-4-5:




```
function exp : syntaxTree ;
var temp, newtemp : syntaxTree ;
begin
   temp := term ;
   while token = + or token = - do
     case token of
      + : match (+) ;
          newtemp := makeOpNode(+) ;
          leftChild(newtemp) := temp ;
          rightChild(newtemp) := term ;
          temp := newtemp ;
       - : match (-) ;
          newtemp := makeOpNode(-) ;
          leftChild(newtemp) := temp ;
          rightChild(newtemp) := term ;
          temp := newtemp ;
     end case ;
   end while ;
   return temp ;
end exp ;
```

Kall

Alternativt:
newtemp.leftChild

Kall

Merk: Dersom det bare er en "term", så lages ingen ny node. Vi leverer den vi har fått

# Skrive prosedyre for:

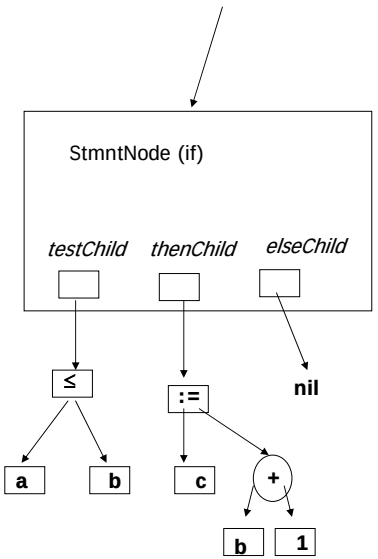
*factor → (exp) | number*

```
proc factor : syntaxTree;
var fact: syntaxTree;
begin
    case token of
    ( :  match ( ( ) ;
         fact = exp ;
         match ( ) ) ;

    number :
         match ( number ) ;
         fact =makeNumberNode( number) ;

    else error(.....) ;
    end case
    return fact;
end factor;
```

# Node for generering for if-setning

if-stmt -> if (exp) stmt [else tstmt]

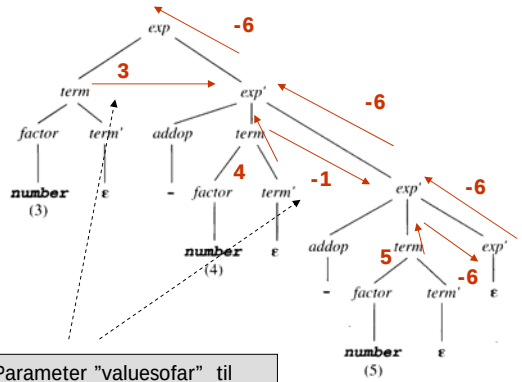
```
function ifStatement : syntaxTree ;
var temp : syntaxTree ;
begin
   match (if) ;
   match ( ( ) ;
   temp := makeStmtNode(if) ;
   testChild(temp) := exp ;
   match ( ) ) ;
   thenChild(temp) := statement ;
   if token = else then
      match (else) ;
      elseChild(temp) := statement ;
   else
      elseChild(temp) := nil ;
   end if ;
end ifStatement ;
```



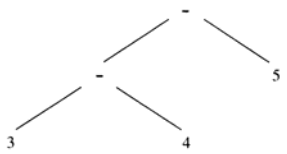

Rec.decent etter tradisjonell fjerning av venstre-rekursjon (treet er nå høyre assosiativt istedenfor venstre). Det må rettes opp.

Lage tre eller beregne verdi : 3 – 4 – 5

*exp → term exp'*
*exp' → addop term exp' | ε*
*addop → + | -*
*term → factor term'*
*term' → mulop factor term' | ε*
*mulop → **
*factor → ( exp ) | **number***




Det abstrakte syntakstreet vi eventuelt ønsker å lage:




Parameter "valuesofar" til prosedyren "exp"

(For trebygging ville den være: "rootOfTreeSoFar")

## Prosedyrer for beregning av verdi (tre-bygging tilsvarende)

$exp \rightarrow term\ exp'$
$exp' \rightarrow addop\ term\ exp' \mid \varepsilon$
$addop \rightarrow$ **+** $\mid$ **-**
$term \rightarrow factor\ term'$
$term' \rightarrow mulop\ factor\ term' \mid \varepsilon$
$mulop \rightarrow$ *
$factor \rightarrow$ **(** $exp$ **)** $\mid$ **number**

Bare analyse

```
procedure exp ;
begin
   term ;
   exp' ;
end exp ;
```

```
procedure exp' ;
begin
   case token of
    + : match (+) ;
        term ;
        exp' ;
    - : match (-) ;
        term ;
        exp' ;
   end case ;
end exp' ;
```

Med beregning (trebygging tilsvarende)

```
function exp : integer ;
var temp : integer ;
begin
   temp := term ;
   return exp'(temp) ;
end exp ;
```

NB.: Parameter

```
function exp' ( valsofar : integer ) : integer ;
begin
   if token = + or token = - then
      case token of
       + : match (+) ;
           valsofar := valsofar + term ;
       - : match (-) ;
           valsofar := valsofar - term ;
      end case ;
      return exp'(valsofar) ;
   else return valsofar ;
end exp' ;
```

ε -alternativet

Leverer verdien uendret oppover igjen.

## LL(1) – grammatikk

$A \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid .... \mid \alpha_n$

- LL(1) -kravet for en "ren BNF-grammatikk". Det som kreves for at en Rek. desc.-parsering skal fungere direkte fra grammatikken.
- For å avgjøre om en grammatikk er "LL(1)": Sett opp tabell M[N,T] med mulige aksjoner for alle mulige situasjoner, slik:

1. If $A \rightarrow \alpha$ is a production choice, and there is a derivation $\alpha \Rightarrow^* a\ \beta$, where $a$ is a token, then add $A \rightarrow \alpha$ to the table entry $M[A, a]$.
2. If $A \rightarrow \alpha$ is a production choice, and there are derivations $\alpha \Rightarrow^* \varepsilon$ and $S\ \$ \Rightarrow^* \beta\ A\ a\ \gamma$, where $S$ is the start symbol and $a$ is a token (or \$), then add $A \rightarrow \alpha$ to the table entry $M[A, a]$.

1. Altså, $a \in First(\alpha)$

2. Altså, dersom:
- $\alpha$ er utnullbar, og
- $a \in Follow(A)$

Definisjon:
En grammatikk er LL(1) dersom M[N,T] er entydig for alle situasjoner (eller angir "error")

## Oppsett av LL(1) –tabell

$statement \rightarrow if\text{-}stmt \mid$ **other**
$if\text{-}stmt \rightarrow$ **if (** $exp$ **)** $statement\ else\text{-}part$
$else\text{-}part \rightarrow$ **else** $statement \mid \varepsilon$
$exp \rightarrow$ **0** $\mid$ **1**

- Venstre-faktorisering utført
- Er ikke vestrerekursiv

| | First | Follow |
|---|---|---|
| statement | **other, if** | **$, else** |
| if-stmt | **if** | **$, else** |
| else-part | **else,ε** | **$, else** |
| exp | **0, 1** | **)** |

| M[N, T] | if | other | else | 0 | 1 | $ |
|---|---|---|---|---|---|---|
| statement | statement → if-stmt | statement → other | | | | |
| if-stmt | if-stmt → if ( exp ) statement else-part | | | | | |
| else-part | | | else-part → else statement<br>else-part → ε | | | else-part → ε |
| exp | | | | exp → 0 | exp → 1 | |

**Merk:**

-fjernet venstre-rek.

-- Utført venstre-fakt.

-er **ikke** nok til å garantere LL(1)-grammatikk.

**fordi tabellen er ikke entydig**