

Kap. 4 – del I
Top Down Parsing
INF5110 – v2006

Stein Krogdahl
Ifi, UiO



Innhold

- First og Follow-mengder
 - Boka ser på én parseringsmetode først, uten å se på First/Follow-mengder.
 - Vi tar "teorien" først
- To greie transformasjoner på grammatikker
 - Fjerning av venstre-rekursjon
 - Venstre-faktorisering
- Parsering ved *rekursiv nedstigning* ("recursive descent")
 - Med bygging av tre-struktur
- LL(1) grammatikker

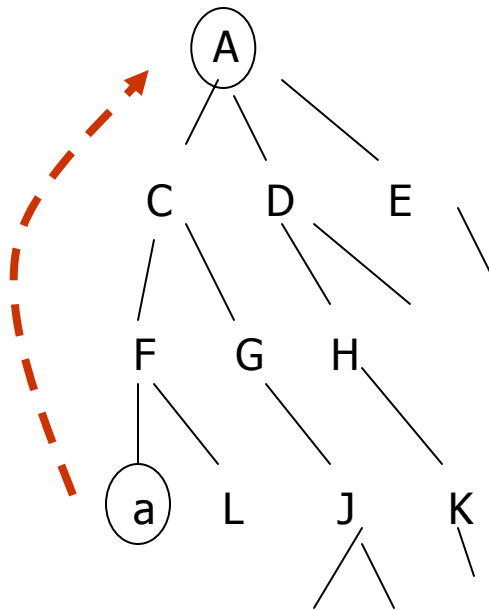


First og Follow-mengder, hvorfor?

- For visse typer syntaksanalyse er det nødvendig å vite hvilke terminalsymboler som kan komme først i strenger som er avledet av en ikke-terminal A . Denne mengden kalles $First(A)$.
 - F.eks. vil $First(\text{if-setning})$ i de fleste språk være bare nøkkelordet "if", mens $First(\text{tilordning})$ ofte vil være mengden $\{\text{navn}, "("\}$.
- Tilsvarende vil vi få behov for å vite hvilke terminaler som kan følge etter en gitt ikke-terminal A i *en eller annen* setningsform (= "halvutviklet setning") i språket. Denne mengden kalles $Follow(A)$
 - F.eks vil $Follow(\text{statement})$ i Tiny-språket (eller snarere i den gitte grammatikken for Tiny-språket) være mengden $\{\text{"end"}, \text{"else"}, \text{"until"}, \text{";"}\}$
- Vi skal se på generelle algoritmer for å finne First- og Follow-mengdene for alle ikke-terminaler i en gitt grammatikk
 - Det som kompliserer disse algoritmene noe er produksjoner av typen $A \rightarrow \varepsilon$, som også gjør at vi kan få $B \Rightarrow^* \varepsilon$, selv om ikke $B \rightarrow \varepsilon$. Slike ikke-terminaler A eller B sies å være *utnullbare* ("nullable")

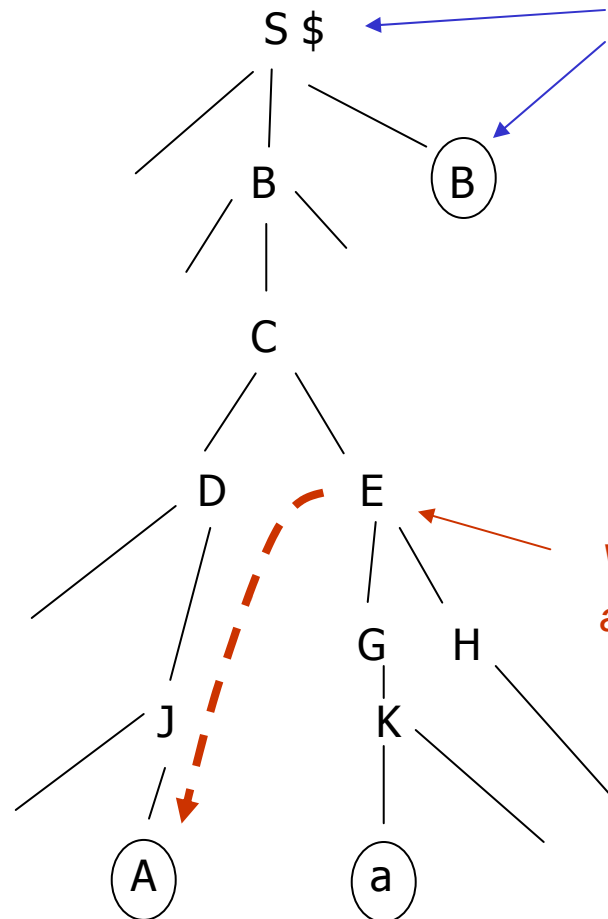
Typiske situasjoner: First- og Follow-mengder

$a \in \text{First}(A)$



— — — — —
 Angir informasjonsflyt
 under algoritmene

$a \in \text{Follow}(A)$



$\$ \in \text{Follow}(B)$

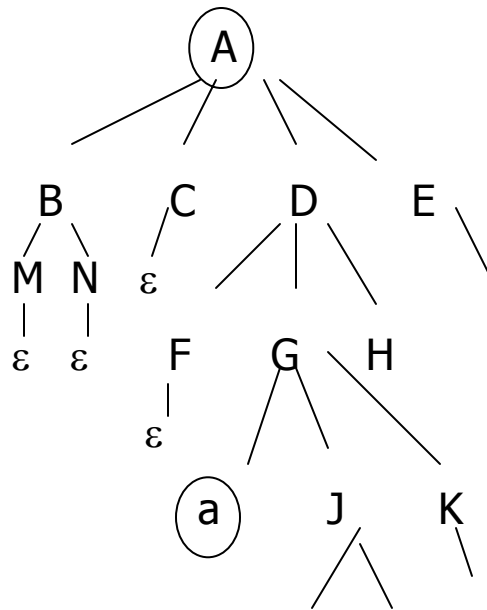
Det at B kan stå på slutten av en setnings-form markeres ved at Follow-mengden til B inneholder \$.

Terminal-symbolet \$ tenkes altså å stå på slutten av enhver setning.

Vet fra før at
 $a \in \text{First}(E)$

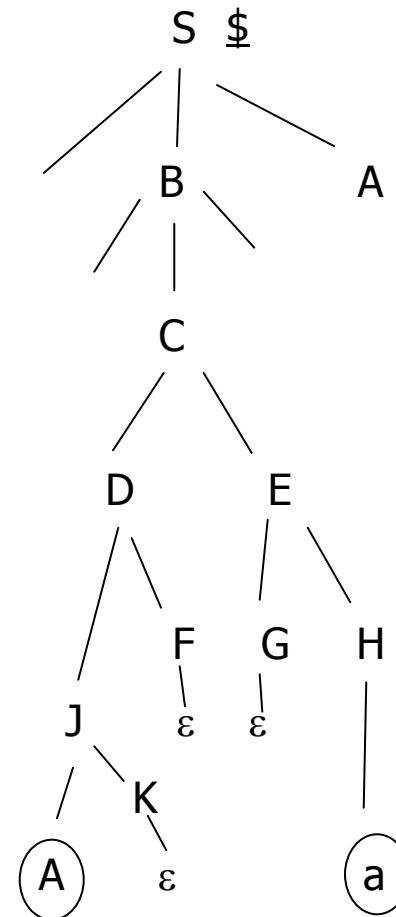
Kompliserte situasjoner: First- og Follow-mengder

$a \in \text{First}(A)$



Her er både B, M, N, C og F utnullbare. Vi markerer det ved å la deres First-mengde inneholde ϵ

$a \in \text{Follow}(A)$



First-mengder

Def



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

terminal

Pr. def. For terminal-symboler: $\text{First}(a) = \{ a \}$

Def.: "A er utnullbar" hvis og bare hvis $A \Rightarrow^* \varepsilon$

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

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

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

Vi snakker
altså også om
First av en hel
streng α av
terminaler og
ikke-
terminaler

```
for all nonterminals A do First(A) := {};  
while there are changes to any First(A) do  
  for each production choice  $A \rightarrow X_1X_2 \dots X_n$  do  
    k := 1 ; Continue := true ;  
    while Continue = true and k <= n do  
      add First( $X_k$ ) - { $\varepsilon$ } to First(A);  
      if  $\varepsilon$  is not in First( $X_k$ ) then Continue := false ;  
      k := k + 1 ;  
    if Continue = true then add  $\varepsilon$  to First(A);
```

Algoritme:

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

Eks. 4.9 Beregning av First-mengde

- (1) $exp \rightarrow exp \text{ addop } term$
- (2) $exp \rightarrow term$
- (3) $addop \rightarrow +$
- (4) $addop \rightarrow -$
- (5) $term \rightarrow term \text{ mulop } factor$
- (6) $term \rightarrow factor$
- (7) $mulop \rightarrow *$
- (8) $factor \rightarrow (exp)$
- (9) $factor \rightarrow \mathbf{number}$

Kan bare fylle ut en tabell

| | First |
|--------|-------|
| exp | |
| addop | |
| term | |
| mulop | |
| factor | |

| Grammar rule | Pass 1 | Pass 2 | Pass 3 |
|--|---|---|--|
| $exp \rightarrow exp$ $addop \text{ } term$ | | | |
| $exp \rightarrow term$ | | | First(exp) = { (, number } |
| $addop \rightarrow +$ | First($addop$) = { + } | | |
| $addop \rightarrow -$ | First($addop$) = { +, - } | | |
| $term \rightarrow term$ $mulop \text{ } factor$ | | | |
| $term \rightarrow factor$ | | First($term$) = { (, number } | |
| $mulop \rightarrow *$ | First($mulop$) = { * } | | |
| $factor \rightarrow (exp)$ | First($factor$) = { (} | | |
| $factor \rightarrow \mathbf{number}$ | First($factor$) = { (, number } | | |

Beregning av Follow-mengder

$\beta \gamma$ virkårlige strenger

Def: $\text{Follow}(A) = \{ a \mid \text{finnes avledning } S \$ \Rightarrow^* \beta A a \gamma \}$

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 $\text{Follow}(A)$.
2. If there is a production $B \rightarrow \alpha A \gamma$, then $\text{First}(\gamma) - \{\epsilon\}$ is in $\text{Follow}(A)$.
3. If there is a production $B \rightarrow \alpha A \gamma$ such that ϵ is in $\text{First}(\gamma)$, then $\text{Follow}(A)$ contains $\text{Follow}(B)$.

γ er utnullbar

```
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 Follow-mengder: 4.12

$$\begin{aligned} \text{Follow}(exp) &= \{ \$, +, -,) \} \\ \text{Follow}(addop) &= \{ (, \mathbf{number} \} \\ \text{Follow}(term) &= \{ \$, +, -, *,) \} \\ \text{Follow}(mulop) &= \{ (, \mathbf{number} \} \\ \text{Follow}(factor) &= \{ \$, +, -, *,) \} \end{aligned}$$

$$\begin{aligned} \text{First}(exp) &= \{ (, \mathbf{number} \} \\ \text{First}(term) &= \{ (, \mathbf{number} \} \\ \text{First}(factor) &= \{ (, \mathbf{number} \} \\ \text{First}(addop) &= \{ +, - \} \\ \text{First}(mulop) &= \{ * \} \end{aligned}$$

- (1) $exp \rightarrow exp\ addop\ term$
- (2) $exp \rightarrow term$
- (3) $addop \rightarrow +$
- (4) $addop \rightarrow -$
- (5) $term \rightarrow term\ mulop\ factor$
- (6) $term \rightarrow factor$
- (7) $mulop \rightarrow *$
- (8) $factor \rightarrow (exp)$
- (9) $factor \rightarrow \mathbf{number}$

| Grammar rule | Pass 1 | Pass 2 |
|--|---|--|
| $exp \rightarrow exp\ addop\ term$ | $\text{Follow}(exp) = \{ \$, +, - \}$ $\text{Follow}(addop) = \{ (, \mathbf{number} \}$ $\text{Follow}(term) = \{ \$, +, - \}$ | $\text{Follow}(term) = \{ \$, +, -, *,) \}$ |
| $exp \rightarrow term$ | | |
| $term \rightarrow term\ mulop\ factor$ | $\text{Follow}(term) = \{ \$, +, -, * \}$ $\text{Follow}(mulop) = \{ (, \mathbf{number} \}$ $\text{Follow}(factor) = \{ \$, +, -, * \}$ | $\text{Follow}(factor) = \{ \$, +, -, *,) \}$ |
| $term \rightarrow factor$ | | |
| $factor \rightarrow (exp)$ | $\text{Follow}(exp) = \{ \$, +, -,) \}$ | |

Kan bare fylle ut en tabell

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

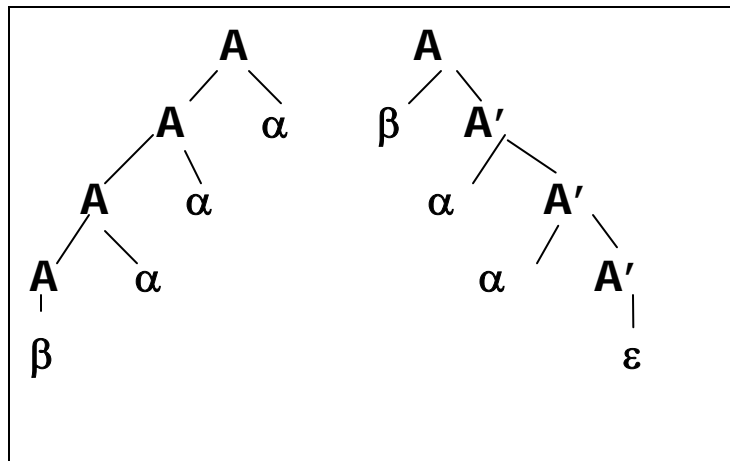


Et par greie transformasjoner

- For noen typer syntaksanalyse er det visse krav til grammatikken for at metoden skal fungere. To vanlige krav er:
 - Grammatikken må ikke ha venstrerekursjon
 - Altså, det må ikke finnes produksjoner av typen: $A \rightarrow A \alpha \mid \dots$
 - F.eks. går da ikke: $\text{expr} \rightarrow \text{expr} + \text{term} \mid \text{term}$
 - Grammatikken må ikke ha noen B (ikke-terminal), slik at to alternativer for B starter på samme måte.
 - Altså, det må ikke finnes slike: $B \rightarrow \alpha \beta \mid \dots \mid \alpha \gamma \mid \dots \quad \alpha \neq \varepsilon$
 - F.eks. går da ikke:
 $\text{if-setn} \rightarrow \text{if (bet) setn-sekv end} \mid \text{if (bet) setn-sekv else setn-sekv end}$

Fjerning av venstre-rekursjon - case 1

Kan skrives på EBNF

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

$$A \rightarrow \beta A'$$
$$A' \rightarrow \alpha A' \mid \epsilon$$

Eksempel:

$$exp \rightarrow exp \text{ addop term} \mid term$$

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

$$exp \rightarrow term exp'$$
$$exp' \rightarrow \text{addop term } exp' \mid \epsilon$$



Fjerning av venstre-rekursjon

Case 2:

$$A \rightarrow A \alpha_1 \mid A \alpha_2 \mid \dots \mid A \alpha_n \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$$



$$\begin{aligned} A &\rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_m A' \\ A' &\rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_n A' \mid \varepsilon \end{aligned}$$



Case 3 (indirekte venstre-rekursjon)

$$\begin{aligned} A &\rightarrow Ba \mid Aa \mid c \\ B &\rightarrow Bb \mid Ab \mid d \end{aligned}$$

- Her er det feil i boka i algoritmen s.159
- Ikke med som pensum
- Egner seg bare til maskinell behandling

$$\begin{aligned} A &\rightarrow B a A' \mid c A' \\ A' &\rightarrow a A' \mid \varepsilon \\ B &\rightarrow B b \mid A b \mid d \end{aligned}$$

$$\begin{aligned} A &\rightarrow B a A' \mid c A' \\ A' &\rightarrow a A' \mid \varepsilon \\ B &\rightarrow B b \mid B a A' b \mid c A' b \mid d \end{aligned}$$

$$\begin{aligned} A &\rightarrow B a A' \mid c A' \\ A' &\rightarrow a A' \mid \varepsilon \\ B &\rightarrow c A' b B' \mid d B' \\ B' &\rightarrow b B' \mid a A' b B' \mid \varepsilon \end{aligned}$$



Eks side 160 – Fjerning av venstre-rekursjon

Den tradisjonelle entydige grammatikken

$$\begin{aligned} \text{exp} &\rightarrow \text{exp addop term} \mid \text{term} \\ \text{addop} &\rightarrow + \mid - \\ \text{term} &\rightarrow \text{term mulop factor} \mid \text{factor} \\ \text{mulop} &\rightarrow * \\ \text{factor} &\rightarrow (\text{exp}) \mid \mathbf{number} \end{aligned}$$

Med fjernet venstre-rekursjon (beholder entydigheten)

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



Venstrefaktorisering, problemet "B \rightarrow $\alpha \beta \mid \alpha \gamma \mid \dots$ "

$stmt\text{-}sequence \rightarrow stmt \ ; \ stmt\text{-}sequence \mid stmt$
 $stmt \rightarrow \mathbf{s}$

$stmt\text{-}sequence \rightarrow stmt \ stmt\text{-}seq'$
 $stmt\text{-}seq' \rightarrow \ ; \ stmt\text{-}sequence \mid \epsilon$

$if\text{-}setn \rightarrow \mathbf{if} (bet) \ setn\text{-}sekv \ \mathbf{end} \mid \mathbf{if} (bet) \ setn\text{-}sekv \ \mathbf{else} \ setn\text{-}sekv \ \mathbf{end}$

$if\text{-}setn \rightarrow \mathbf{if} (bet) \ setn\text{-}sekv \ \mathbf{else\text{-}eller\text{-}end}$
 $\mathbf{else\text{-}eller\text{-}end} \rightarrow \mathbf{else} \ setn\text{-}sekv \ \mathbf{end} \mid \mathbf{end}$

Avansert venstre-faktorisering

Et litt mer komplisert tilfelle:

$$A \rightarrow abc B \mid abC \mid aE$$

Etter steg 1: $A \rightarrow ab A' \mid aE$
 $A' \rightarrow c B \mid C$

Etter steg 2 $A \rightarrow a A''$
(og ferdig): $A'' \rightarrow b A' \mid E$
 $A' \rightarrow c B \mid C$

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 $\alpha \neq \varepsilon$ **then**

let $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$ be all the production choices for A
and suppose that $\alpha_1, \dots, \alpha_k$ share α , so that
 $A \rightarrow \alpha \beta_1 \mid \dots \mid \alpha \beta_k \mid \alpha_{k+1} \mid \dots \mid \alpha_n$, the β_j 's share
no common prefix, and the $\alpha_{k+1}, \dots, \alpha_n$ do not share α

replace the rule $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$ by the rules

$$A \rightarrow \alpha A' \mid \alpha_{k+1} \mid \dots \mid \alpha_n$$
$$A' \rightarrow \beta_1 \mid \dots \mid \beta_k$$

} Gjenta så lenge det er muligheter

} Velg lengst mulig prefiks

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



Litt om stoffet i kap. 4:

Dette bør leses om igjen etter kapitlet:

- First og Follow-mengder
 - Boka tar det et stykke uti, vi tok det først
- Begrepet LL(1)-parsering (kommer)
 - 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.

Parseringsmetode: "Rekursiv nedstigning"

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

Hoved-idé:

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

Inneværende token: if ← *Global variabel*

"Typisk"

rec.dec.
prosedyre

for det
enkle'

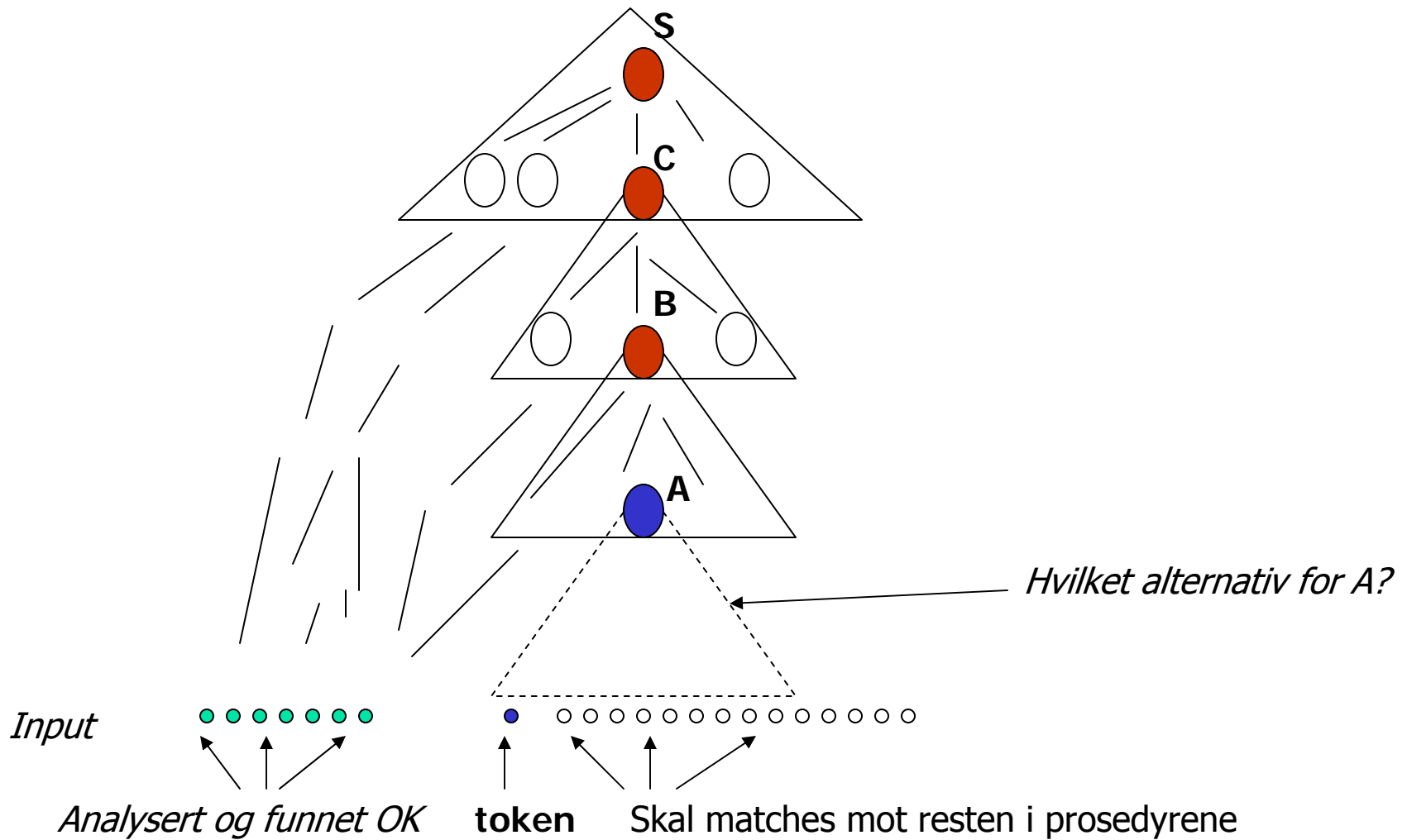
tilfellet.

```
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 parsing



Litt mer kompliserte tilfeller kan løses med EBNF

Opprinnelig

$if\text{-stmt} \rightarrow \mathbf{if} (exp) statement [\mathbf{else} statement]$

Skrives ut som:

$if\text{-stmt} \rightarrow \mathbf{if} (exp) statement$
 $\quad | \mathbf{if} (exp) statement \mathbf{else} statement$

R-D-prosedyre:

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

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

$ifStmt \rightarrow \underline{\mathbf{if}} (exp) stmt \text{ elsePart}$
 $\text{elsePart} \rightarrow \varepsilon | \underline{\mathbf{else}} stmt$

Venstre-rekursjon gir problemer

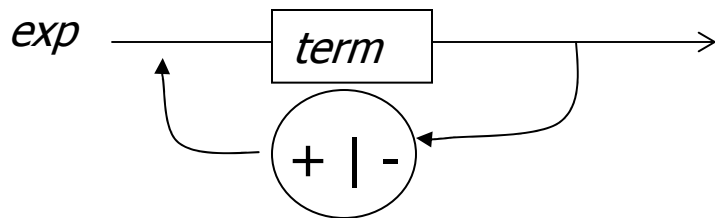
Gir uendelig mange rekursive kall

$exp \rightarrow exp \text{ addop } term \mid term$

Bruker EBNF:

$exp \rightarrow term \{ \text{addop } term \}$

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

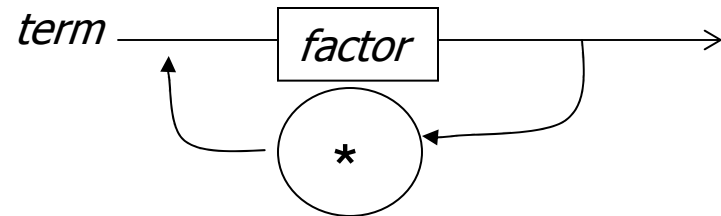


NB: Kan også fjerne venstre-rekursjon på trad måte, se senere foil.

$term \rightarrow term \text{ multop } factor \mid factor$

$term \rightarrow factor \{ \text{mulop } factor \}$

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



Hvordan "lage noe" under rec.-decent parsing?

- Mål: Ønsker å bygge abstrakt syntaks-tre
- Men foreløpig (som kan være forvirrende!):
 - 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"

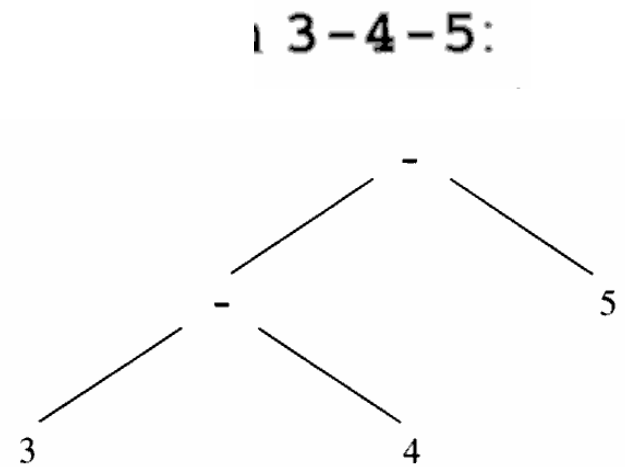
3 + 4 + 5

Bygging av syntaks-treet

```
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 ;
```

Alternativt:
newtemp.leftChild

Kall



Kall

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

Skriv prosedyre med trebygging for:

factor → (*exp*) / *number*

```
proc factor : syntaxTree;  
var fact: syntaxTree;  
begin  
  case token of  
    (:  
  
    number :  
  
    else error(.....) ;  
  end case  
  return fact;  
end factor;
```


Prosedyre med trebygging for:

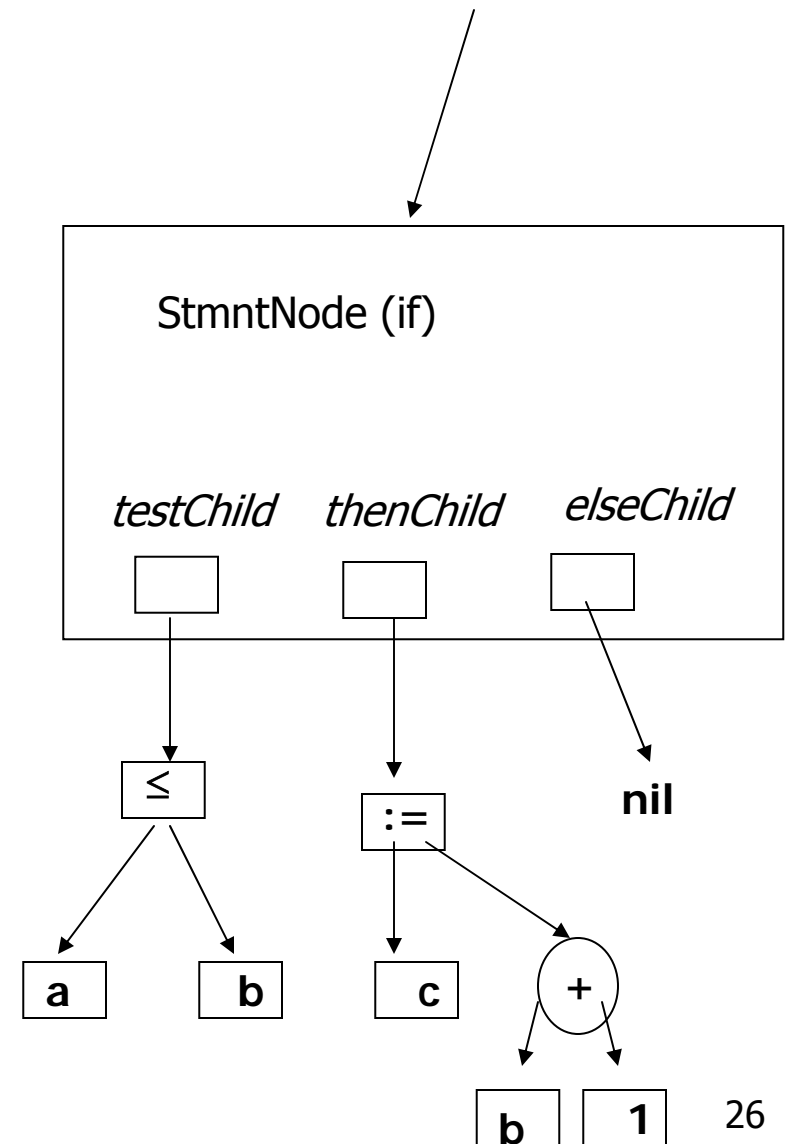
factor → (*exp*) / *number*

```
proc factor : syntaxTree;  
var fact: syntaxTree;  
begin  
  case token of  
  ( :  
    match "(" ;  
    fact = exp ;  
    match ")" ;  
  
    number :  
    fact = makeNumberNode(number) ;  
    match (number) ;  
  
    else error(.....) ;  
  end case  
  return fact;  
end factor;
```

Parsering av if-setning, med tre-generering

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

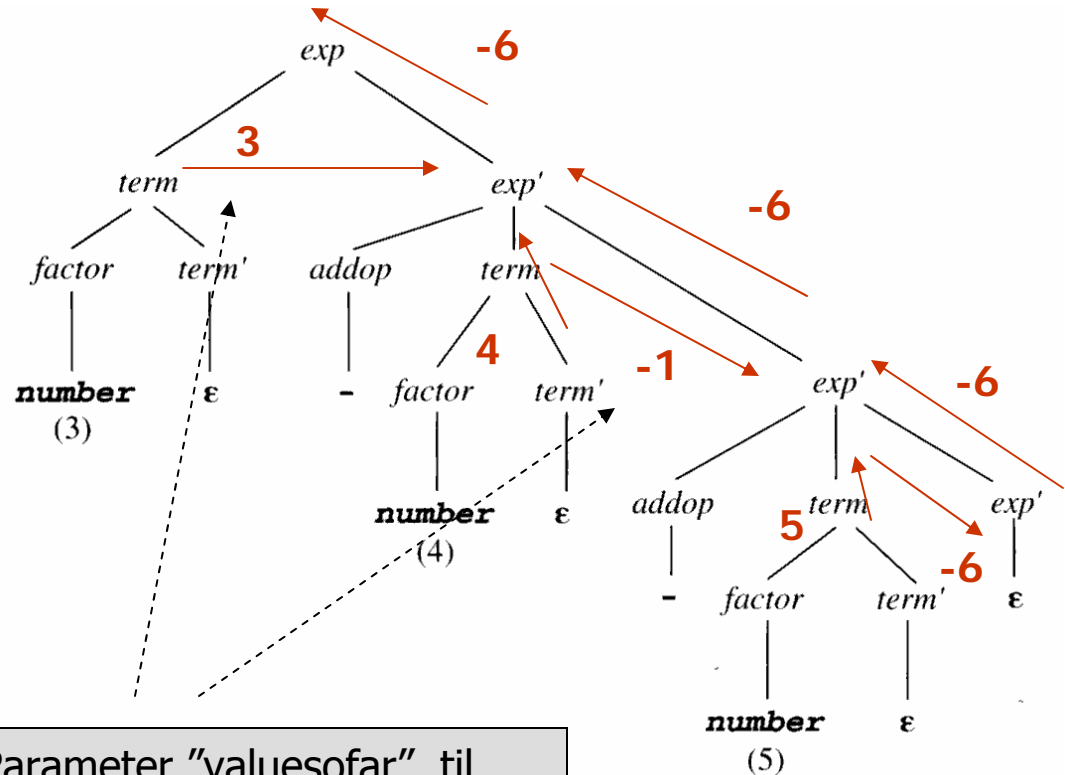
```
function ifStatement : syntaxTree ;  
var temp : syntaxTree ;  
begin  
  match (if) ;  
  match ( ( ) ;  
  temp := makeStmtNode(if) ;  
  testChild(temp) := exp ;  
  match ( ) ;  
  thenChild(temp) := statement ;  
  if token = else then  
    match (else) ;  
    elseChild(temp) := statement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



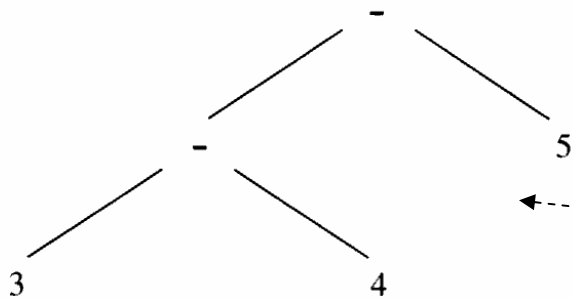
Rec.decent etter tradisjonell fjerning av venstre-rekursjon (treet er nå høyre assosiativt istedenfor venstre). Det må korrigeres for.

Lage tre eller beregne verdi : 3 - 4 - 5

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



Det abstrakte syntakstreet vi ønsker å lage:



Parameter "valuesofar" til prosedyren "exp"
 For trebygging ville den være: "rootOfTreeSoFar"

Proedyrer for beregning av verdi (tre-bygging tilsvarende)

```
exp → term exp'  
exp' → addop term exp' |  $\epsilon$   
addop → + | -  
term → factor term'  
term' → mulop factor term' |  $\epsilon$   
mulop → *  
factor → ( exp ) | 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

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

- LL(1) -kravet for en "ren BNF-grammatikk". Det som kreves for at en Rek. descent-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. Altså,
 $a \in \text{First}(\alpha)$

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 $M[A, a]$.

2. Altså, dersom:

- α er utnullbar, og
- $a \in \text{Follow}(A)$

Definisjon:

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

Oppsett av LL(1) –tabell

$statement \rightarrow if-stmt \mid \mathbf{other}$

$if-stmt \rightarrow \mathbf{if} (exp) statement else-part$

$else-part \rightarrow \mathbf{else} statement \mid \epsilon$

$exp \rightarrow \mathbf{0} \mid \mathbf{1}$

- 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 else-part → ε | | | else-part → ε |
| exp | | | | exp → 0 | exp → 1 | |

Merk:

- fjerne venstre-rek.
- Utføre venstre-fakt.
- er generelt **ikke** nok til å garantere LL(1)-grammatikk.

Fordi tabellen ikke ble entydig