

INF5110 – V2009

Kap. 3/4: Stoff som i boka står i kap 4, men som er generelt stoff om grammatikker

Stein Krogdahl

Ifi, UiO



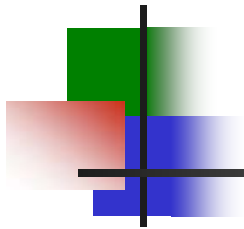
Dagens temaer

- Noe avsluttende fra sist
- First og Follow-mengder
 - Boka ser på en av parseringsmetodene først, uten å se på First/Follow-mengder.
 - Vi tar dette først (hører naturlig til i kap 3, Louden)
- To greie transformasjoner på grammatikker
 - Også naturlig i kap 3
 - Fjerning av venstre-rekursjon
 - Venstre-faktorisering



BNF-grammatikk for TINY

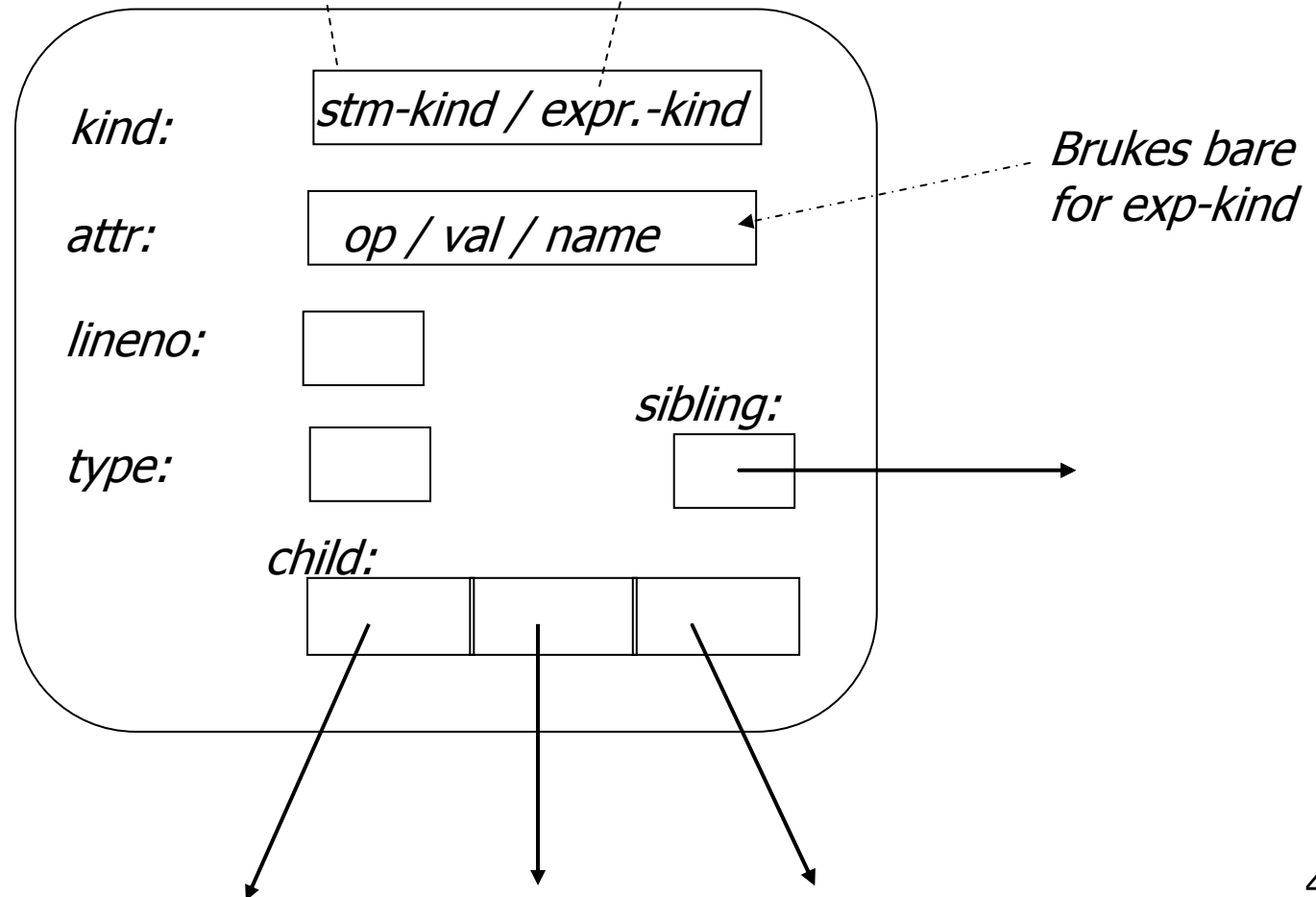
program → *stmt-sequence*
stmt-sequence → *stmt-sequence* ; *statement* | *statement*
statement → *if-stmt* | *repeat-stmt* | *assign-stmt* | *read-stmt* | *write-stmt*
if-stmt → **if** *exp* **then** *stmt-sequence* **end**
 | **if** *exp* **then** *stmt-sequence* **else** *stmt-sequence* **end**
repeat-stmt → **repeat** *stmt-sequence* **until** *exp*
assign-stmt → **identifier** := *exp*
read-stmt → **read** **identifier**
write-stmt → **write** *exp*
exp → *simple-exp* *comparison-op* *simple-exp* | *simple-exp*
comparison-op → < | =
simple-exp → *simple-exp* *addop* *term* | *term*
addop → + | -
term → *term* *mulop* *factor* | *factor*
mulop → * | /
factor → (*exp*) | **number** | **identifier**



Nodestruktur i C for Tiny

If, Repeat, Assign, Read, Write - tegnes:

Op, Const, id - tegnes:

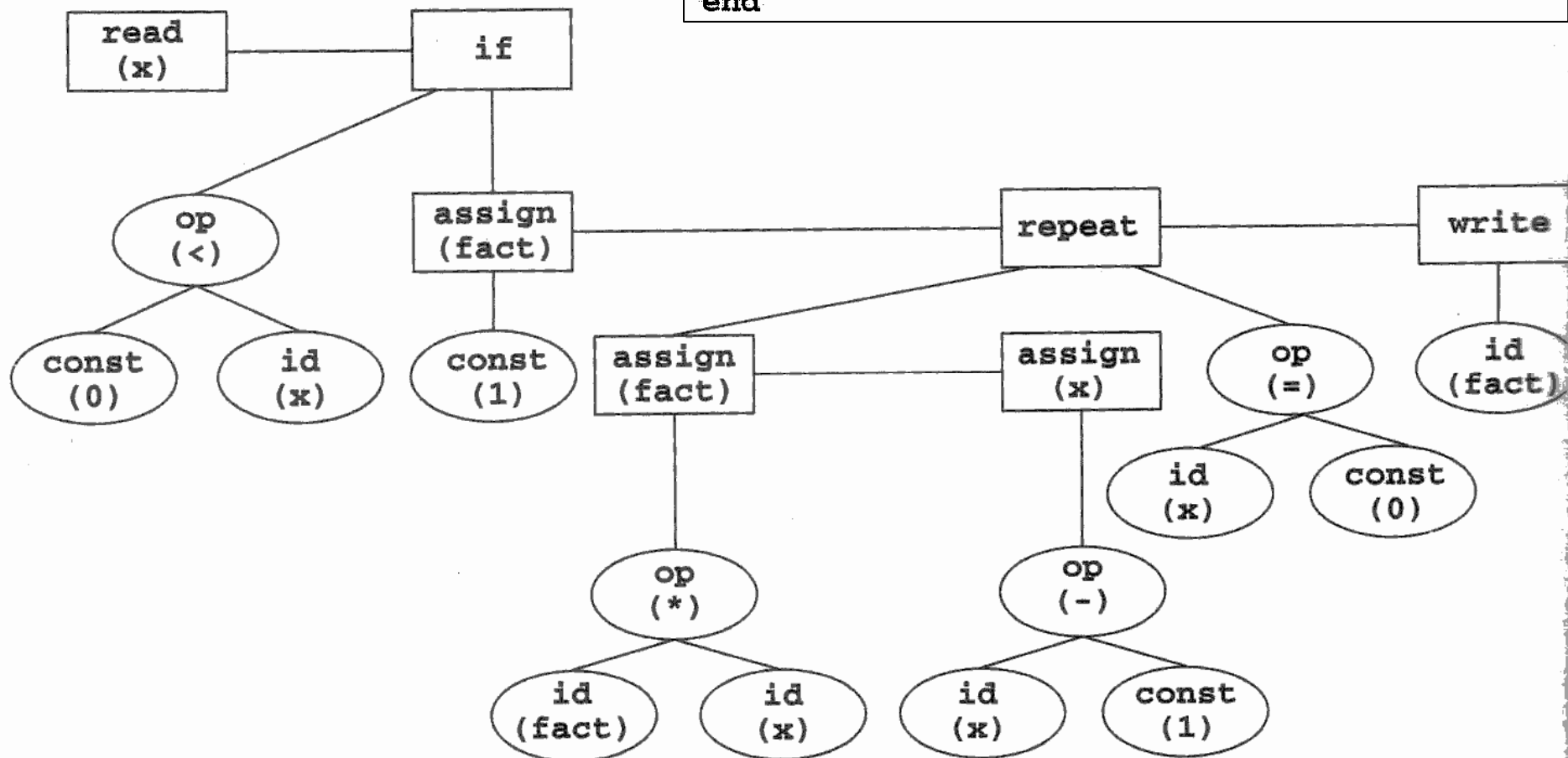


Brukes bare for exp-kind

Denne nodestrukturen passer enda bedre med et OO-språk med klasser /subklasser som implementasjonsspråk.

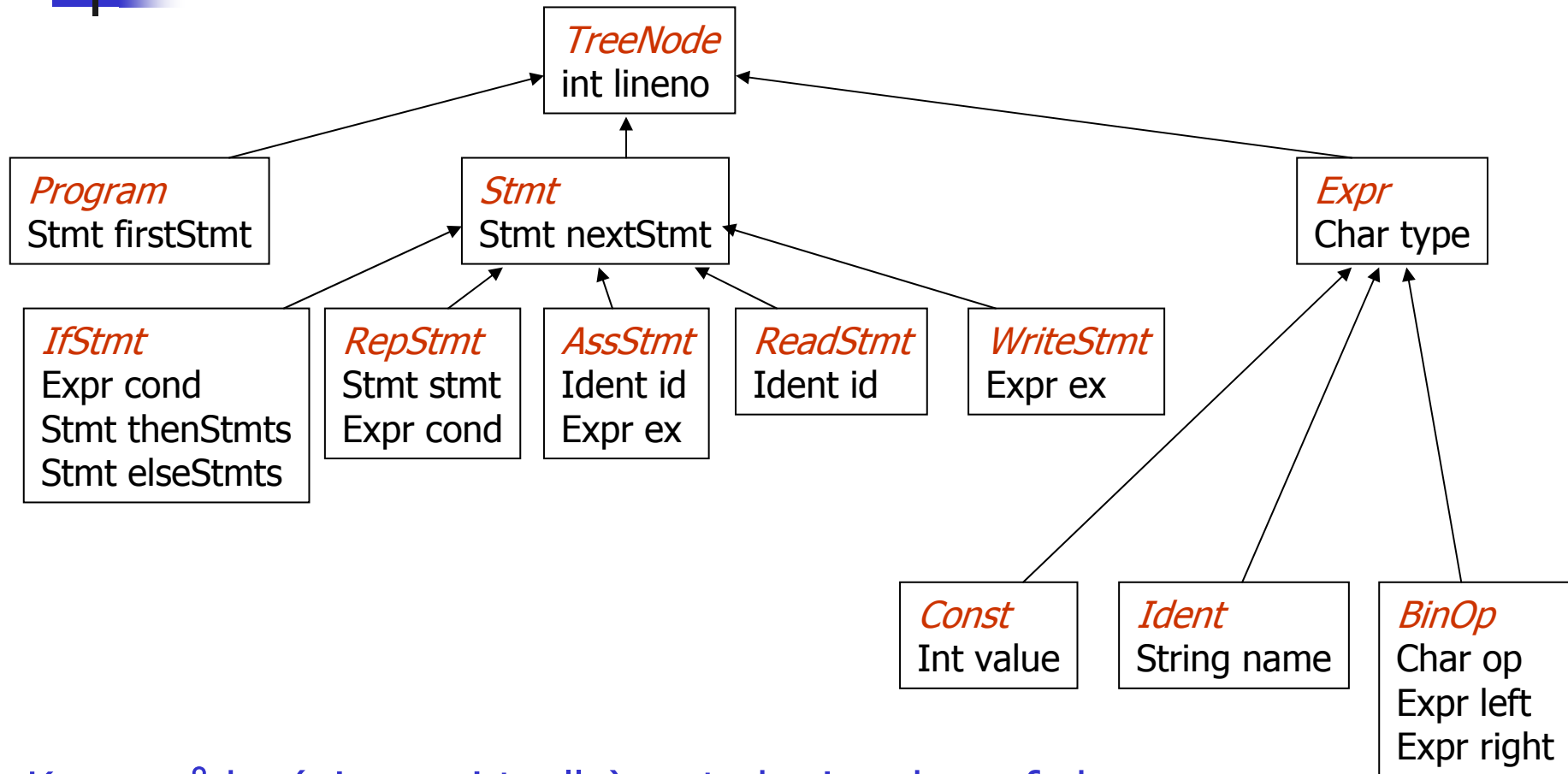
Abstrakt syntakstre for Tiny-programmet:

```
read x; { input an integer }  
if 0 < x then { don't compute if x <= 0 }  
  fact := 1;  
  repeat  
    fact := fact * x;  
    x := x - 1  
  until x = 0;  
  write fact { output factorial of x }  
end
```



Nodeklasser for OO-utgave av abst.-synt.-tre for Tiny-språket

Merk: Dette er altså en fast subklasse-struktur i kompilatoren, og må ikke forveksles med det abstrakte syntaks-treet for et gitt program



Kan også ha (gjerne virtuelle) metoder i nodene, f.eks:

- *doSemAnalyses()*; Gjør semantisk analyse av noden, og subtreet det er rot i
- *generateCode()*; Genererer kode for noden, og for subtreet det er rot i



Noen spørsmål om Tiny-grammatikken

program → *stmt-sequence*
stmt-sequence → *stmt-sequence* ; *statement* | *statement*
statement → *if-stmt* | *repeat-stmt* | *assign-stmt* | *read-stmt* | *write-stmt*
if-stmt → **if** *exp* **then** *stmt-sequence* **end**
 | **if** *exp* **then** *stmt-sequence* **else** *stmt-sequence* **end**
repeat-stmt → **repeat** *stmt-sequence* **until** *exp*
assign-stmt → **identifier** := *exp*
read-stmt → **read identifier**
write-stmt → **write** *exp*
exp → *simple-exp* *comparison-op* *simple-exp* | *simple-exp*
comparison-op → < | =
simple-exp → *simple-exp* *addop* *term* | *term*
addop → + | -
term → *term* *mulop* *factor* | *factor*
mulop → * | /
factor → (*exp*) | **number** | **identifier**

- Er grammatikken entydig?
- Hva om vi vil tillate tomme setninger
- Hva om vi vill ha semikolon etter og ikke mellom setningene?
- Hva slags assosiativitet og presedens er det for operatorene?



Svar på spørsmålene på forrige foil

- Er grammatikken entydig?
 - Dette er generelt uavgjørbar for generelle BNF-grammatikker
 - Vi skal se på metode for å avgjøre det for mange praktiske grammatikker
 - Denne er ihvertfall delt opp i presedens-nivåer, og har assosiativites-angivelse, og er nok derved entydig
- Hva om vi vil tillate tomme setninger
 - Det er bare å sette til et tomt alternativ for *statement*
 - Den ser ut til fremdeles å være entydig
- Hva om vi vil ha semikolon etter og ikke mellom setningene?
 - Bytt ut reglen for *stmt-sequence* med:
stmt-sequence - \rightarrow *statement* ; | *stmt-sequence* *statement* ;
- Hva slags assosiativitet og presedens er det for operatorene?
 - Høyest presedens * / Venstre-assosiativ
 - Midlere presedens + - Venstre-assosiativ
 - Lavest presedens < = Ikke-assosiativitet (bare to operander)
 - Kunne altså brukt flertydig grammatikk, med disse tilleggs-reglene

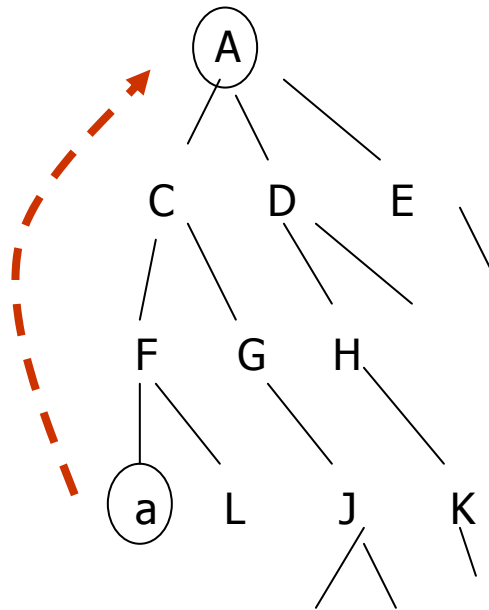


First og Follow-mengder, hvorfor?

- For visse typer syntaksanalyse er det nødvendig å vite hvilke terminalsymboler som kan komme først i strenger som er avledet fra en ikketerminal A . Denne mengden kalles $First(A)$.
 - F.eks. vil $First(\textit{if-setning})$ i de fleste språk være bare nøkkelordet *"if"*, mens $First(\textit{tilordning})$ ofte vil være mengden $\{\textit{navn}, \textit{"("}\}$.
- Tilsvarende vil vi få behov for å vite hvilke terminaler som kan følge etter en gitt ikketerminal A i *en eller annen* setningsform (= "halvutviklet setning") i språket. Denne mengden kalles $Follow(A)$
 - F.eks vil $Follow(\textit{statement})$ i Tiny-språket (eller snarere i den gitte grammatikken for Tiny-språket) være mengden $\{\textit{";"}, \textit{"end"}, \textit{"else"}, \textit{"until"}\}$
- 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 $B \rightarrow \varepsilon$ direkte. Slike ikketerminaler A eller B sies å være *utnullbare* ("nullable")

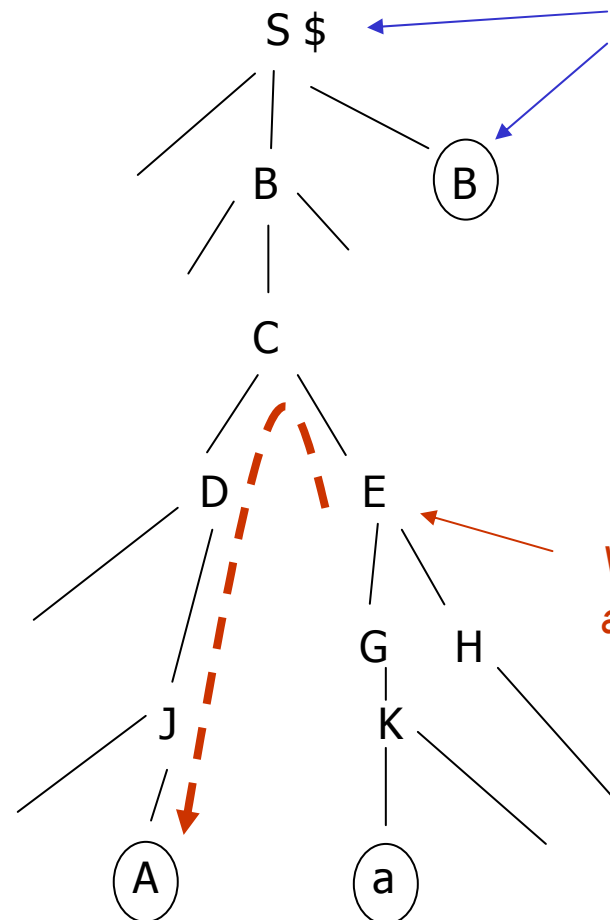
Typiske situasjoner: First- og Follow-mengder

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



----->
Angir informasjonsflyt
under algoritmene

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



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

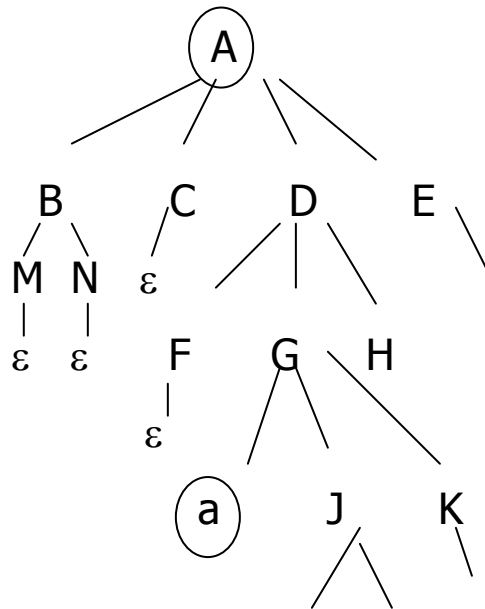
Det at B kan stå på slutten av en setnings-form markeres ved å la Follow-mengden til B inneholde \$.

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

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

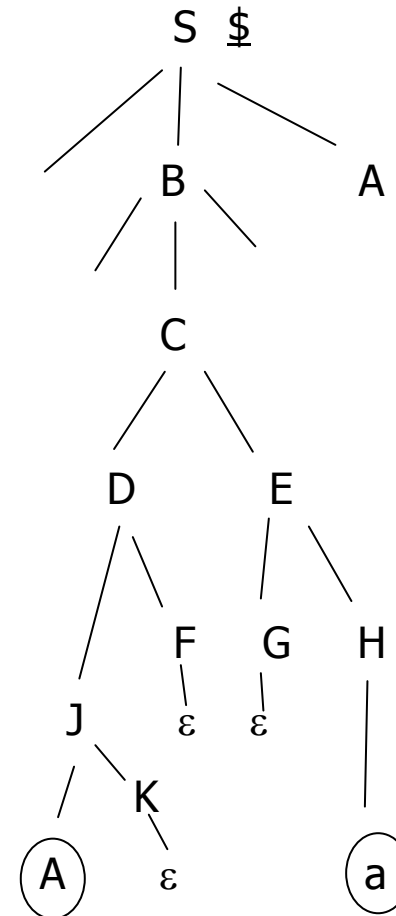
Kompliserte situasjoner: First- og Follow-mengder

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



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.: $\left\{ \begin{array}{l} \text{First}(A) = \{ a \mid \text{finnes avledning } A \Rightarrow^* a \alpha \} \\ \text{Dessuten: Om } A \text{ "er utnullbar", s\aa er } \varepsilon \in \text{First}(A) \\ \text{Pr. def. For terminal-symboler: } \text{First}(a) = \{ a \} \end{array} \right.$

terminal

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 nonterminals), 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 .

← Tar vi som *algoritme* for å finne *First*()

← 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  $k \leq n$  do  
      add  $\text{First}(X_k) - \{\varepsilon\}$  to  $\text{First}(A)$ ;  
      if  $\varepsilon$  is not in  $\text{First}(X_k)$  then  $Continue := false$ ;  
       $k := k + 1$ ;  
    if  $Continue = true$  then add  $\varepsilon$  to  $\text{First}(A)$ ;
```

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) $factor \rightarrow \mathbf{number}$

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 }		

Kan enklere fylle ut en tabell:

	First
exp	
addop	
term	
multop	
factor	

NB: Kan ta reglene i vilkårlig rekkefølge, frem og tilbake

Beregning av Follow-mengder

$\beta \gamma$ virkårlige strenger

Def: $\text{Follow}(A) = \{ a \mid \text{finnes avledning } S \Rightarrow^* \beta A a \gamma \}$
For stratsymbolet S er $\$$ med i followmenden

Dvs.: Det finnes en avledning fra setningsformen " $S \$$ " til en setningsform 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

- (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) $factor \rightarrow \mathbf{number}$

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

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

Kan bare fylle ut en tabell:

	Follow
exp	
addop	
term	
multop	
factor	

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



Et par greie transformasjoner

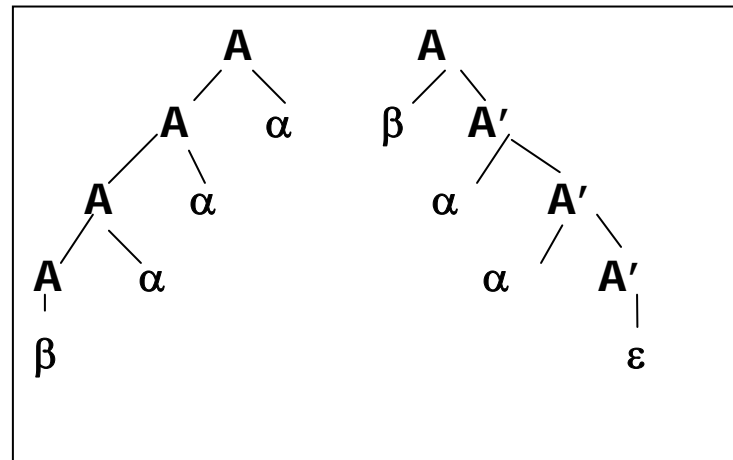
- For noen typer syntaksanalyse er det visse krav til grammatikken for at metoden skal fungere. To vanlige krav er:
 - Grammatikken må ikke ha venstrerekursjon
 - Altså, det må ikke finnes produksjoner av typen: $A \rightarrow A \alpha \mid \dots$
 - F.eks. går da ikke: $\text{expr} \rightarrow \text{expr} + \text{term} \mid \text{term}$
 - 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

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

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



$$A \rightarrow \beta A'$$

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

NB: Litt dumt, siden den opprinnelige venstreassosiative strukturen forsvinner. Må ev. korrigeres for.

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

Case 2:

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

Kan skrives: $A \rightarrow (\beta_1 \mid \beta_2 \mid \dots \mid \beta_m) (\alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n)^*$



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



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, men ikke assosiativiteten):

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



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

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

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

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

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

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

Blir til: $if\text{-}setn \rightarrow \mathbf{if} (bet) \ setn \ \mathbf{else\text{-}eller\text{-}tom}$
 $\mathbf{else\text{-}eller\text{-}tom} \rightarrow \mathbf{else} \ setn \mid \epsilon$

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