# Kap.4 del 2
# Top Down Parsering
# INF5110 – v2006

Stein Krogdahl

Ifi, UiO

# LL(1) –tabell for uttrykks-grammatikk

Har fjernet venstre-rekursjon:

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

Vi får da følgende First- og Follow-mengder:

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

| $M[N, T]$ | ( | **number** | ) | + | - | * | $ |
|---|---|---|---|---|---|---|---|
| *exp* | *exp* → *term exp'* | *exp* → *term exp'* | | | | | |
| *exp'* | | | *exp'* → ε | *exp'* → *addop term exp'* | *exp'* → *addop term exp'* | | *exp'* → ε |
| *addop* | | | | *addop* → + | *addop* → - | | |
| *term* | *term* → *factor term'* | *term* → *factor term'* | | | | | |
| *term'* | | | *term'* → ε | *term'* → ε | *term'* → ε | *term'* → *mulop factor term'* | *term'* → ε |
| *mulop* | | | | | | *mulop* → * | |
| *factor* | *factor* → ( *exp* ) | *factor* → **number** | | | | | |

# Alternativ def. av LL(1) grammatikker

Sier at alle alternativer for A skal ha disjunkte startmengder (og dermed høyst ett utnullbart alternativ, ellers ville $\varepsilon$ være med i startmengden for flere alternativer)

A grammar in BNF is **LL(1)** if the following conditions are satisfied.

1. For every production $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, First($\alpha_i$) $\cap$ First($\alpha_j$) is empty for all $i$ and $j$, $1 \leq i, j \leq n$, $i \neq j$.
2. For every nonterminal $A$ such that First($A$) contains $\varepsilon$, First($A$) $\cap$ Follow($A$) is empty.

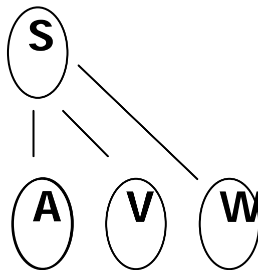
Dersom A er utnullbar, så må First(A) og Follow(A) være disjunkte.

At 1. og 2. til sammen er ekvivalent med det opprinnelige LL(1)-kravet krever et lite bevis.

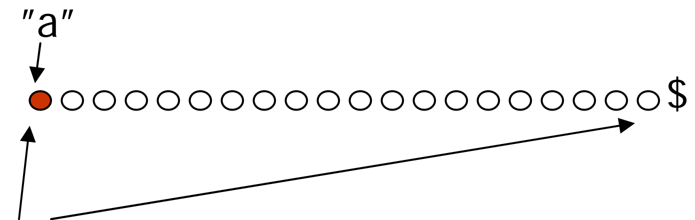
# Kap 4.2: LL(1) –parsering med eksplisitt stakk

- Dette kalles i boka bare ″LL(1)-parsering″
- Gjør logisk sett nøyaktig det samme som rek. desc. parsering for ren BNF.
- Bruker LL(1)-tabellen M[T,N] til å styre parseringen

Start-situasjonen: ( S er start-symbolet)




input:




hele input, med ″a″ som første token

Anta at a ∈ First(S) og at M[S,a] = ″S → AVW″, da bruker vi denne produksjonen ved å erstatte S på stakken med AVW.

Se det generelle steget på neste foil

*Stakk etter init.:*

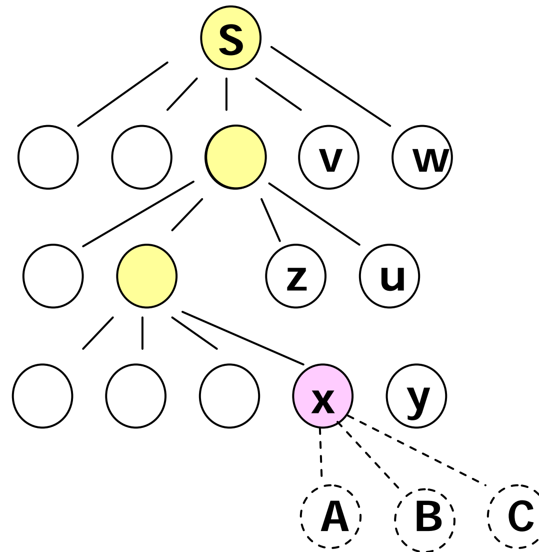
*Stakk etter ett steg:*

Starter med S og slutt-symbolet på stakken (End-Of-File EOF).

På stakken har vi det vi kke er ″ferdig med″

# Steget i LL(1)-parsering med stakk

La X være på toppen av stakken og "a" er nåværende token (neste på input)



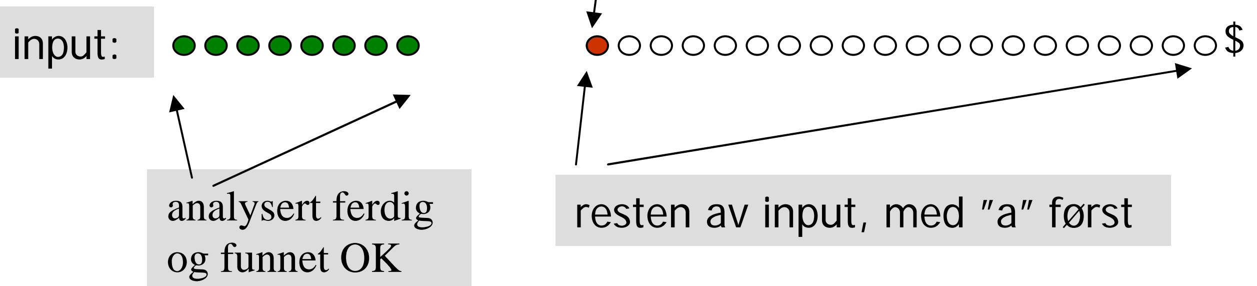

<u>STEGET:</u>

<u>Om X er en terminal:</u>

Da må X == a. Fjern X (pop ) og les a (ikke vist i figur)

<u>Om X ikke-terminal:</u>

Velg aksjon M[X,a]. Om ikke "error", så fjern X (pop X). Anta M[X,a] er "X-> ABC" og legg da (Push) ABC på stakken (N.B. vi har <u>ikke</u> lest "a" nå)




*Stakk før:*

*Stakk etter:*

Dette er en setningsform som oppstår under <u>venstre-avledning</u> fra startesymbolet til input-setningen.

# Utførelse av LL(1)-persering med tabell M[N,T] og eksplisitt stakk . Merk: noen feil må rettes.

(* *assumes* $ *marks the bottom of the stack and the end of the input* *)<br>
*push the start symbol onto the top of the parsing stack* ;<br>
**while** *the top of the parsing stack* ≠ $ ~~**and** *the next input token* ≠ $~~ **do**<br>
  **if** *the top of the parsing stack is terminal a*<br>
      **and** *the next input token* = *a*<br>
  **then** (* *match* *)<br>
    *pop the parsing stack* ;<br>
    *advance the input* ;<br>
  **else if** *the top of the parsing is nonterminal A*<br>
      **and** *the next input token is terminal a*<br>
      **and** *parsing table entry M[A, a] contains*<br>
          *production* $A \rightarrow X_1 X_2 \ldots X_n$<br>
  **then** (* *generate* *)<br>
    *pop the parsing stack* ;<br>
    **for** $i$ := $n$ **downto** 1 **do**<br>
      *push* $X_i$ *onto the parsing stack* ;<br>
  **else** *error* ;<br>
**if** ~~*the top of the parsing stack* = $~~<br>
      ~~**and**~~ *the next input token* = $<br>
**then** *accept*<br>
**else** *error* ;

Det ville stoppe for tidlig (se eksempler)

"top" er terminal

"top" er ikke-terminal

Pusk baklengs på stakken (slik at første symbol kommer på toppen)

Dette vet vi holder

Aksept hvis vi greide å tolke hele input som S

Hit hvis input er "for lang"

/
i
g

| 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* | | | 1 *else-part* → else *statement*<br>2 *else-part* → ε | | | *else-part* → ε |
| *exp* | | | | *exp* → 0 | *exp* → 1 | |

LL(1)-parsering med en stakk ut fra en tabell som ikke er LL(1).
Må gjøre valg i tvetydige situasjoner

| Parsing stack | Input | Action |
|---|---|---|
| $ S | i(0)i(1)oeo$ | S → I |
| $ I | i(0)i(1)oeo$ | I → i ( E ) S L |
| $ L S ) E ( i | i(0)i(1)oeo$ | match |
| $ L S ) E ( | (0)i(1)oeo$ | match |
| $ L S ) E | 0)i(1)oeo$ | E → 0 |
| $ L S ) 0 | 0)i(1)oeo$ | match |
| $ L S ) | )i(1)oeo$ | match |
| $ L S | i(1)oeo$ | S → I |
| $ L I | i(1)oeo$ | I → i ( E ) S L |
| $ L L S ) E ( i | i(1)oeo$ | match |
| $ L L S ) E ( | (1)oeo$ | match |
| $ L L S ) E | 1)oeo$ | E → 1 |
| $ L L S ) 1 | 1)oeo$ | match |
| $ L L S ) | )oeo$ | match |
| $ L L S | oeo$ | S → o |
| $ L L o | oeo$ | match |
| $ L L | eo$ | L → e S |
| $ L S e | eo$ | match |
| $ L S | o$ | S → o |
| $ L o | o$ | match |
| $ L | $ | L → ε |
| $ | $ | accept |

# Når kompilatoren oppdager feil

- minstekrav:
  - Tester løpende at programmet er OK, og gir fornuftig feilmelding ved feil (men stopper)
- Vanlig krav ved feil ("error recovery"):
  - Gir fornuftig feilmelding.
  - "Blar forbi feilen" og fortsetter kompileringen (blar forbi så lite som mulig).
  - Vanligvis vil man slutte å lage maskinkode etter feil (Men noe "feilrettende" kompilatorer forsøker det – lite brukt)
  - Det er for *syntaksfeil* det er vanskeligst å ta opp tråden etter feil.

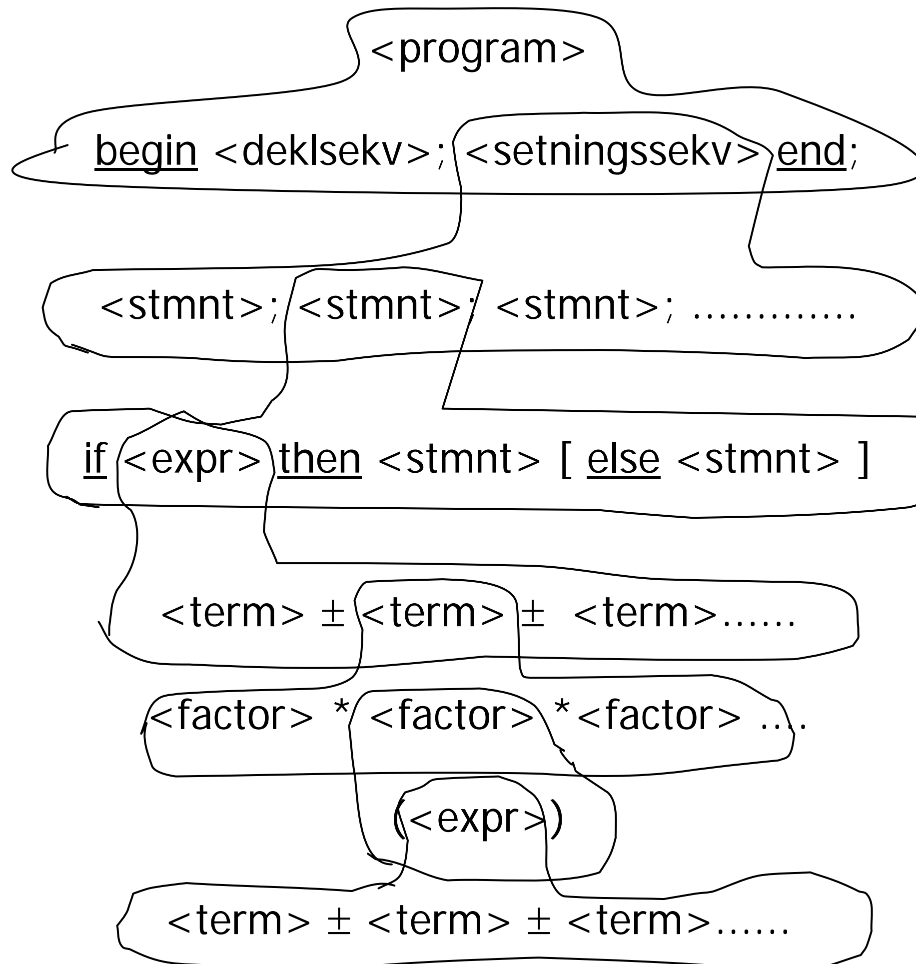
# Spesielt viktig ved syntaksfeil:

- Forsøke å unngå feilmeldinger som bare er følgefeil
- Rapportere feil så tidlig som mulig, helst så snart det man har lest *ikke kan forlenges til et riktig program*
- Man må passe på at man ikke blir gående i løkke rapportere feil *uten å lese noe fra input.*

# Behandling av Syntaksfeil

ved "recursive decent" parsering.

Metode:"Panic mode" og synkroniserings-mengde




**Synchset (stakk eller parameter):**

end

; First(stmt)

navn if while for ...

then First(stmt) else

\+ - First(term)

( tall navn

* First(factor)

)

\+ - ( tall navn

# Syntaksfeil ved "rec. descent" – 2

Ut fra skissen er det greit å finne:

- hvem som skal ta opp tråden
- "hvor" denne skal gjøre av det

Algoritme:

For hvert input-symbol framover:

Let gjennom stakken topp → bunn

Om finnes:

Den" tilsvarende proc" skal ta opp tråden

Den vet selv hvor den skal fortsette, med dette symbolet

Om input-symbol ikke er i stakken:

- gå til neste input-symbol

Det som ikke er greit, er å programmere dette uten at den vakre strukturen ved "rec. descent" blir helt ødelagt.

# Uttrykksprosedyrer ved "error recovery"
# Filosofien her er litt annerledes (og noe uklar?)

```
procedure exp ( synchset ) ;
begin
   checkinput ( { (, number }, synchset ) ;
   if not ( token in synchset ) then
      term ( synchset ) ;
      while token = + or token = - do
        match (token) ;
        term ( synchset ) ;
      end while ;
      checkinput ( synchset, { (, number }) ;
 end if;
end exp ;
```

? (ved "if not ( token in synchset ) then"): **if** token **in** {(,number} **then** ...

Også { +, - } ? (ved "term ( synchset )" i while-løkken)

**Hovedfilosofi:**

"checkinput" kalles to ganger: Først for å sjekke at konstruksjonen starter riktig, etterpå for å sjekke at symbolet etter konstruksjonen er lovlig.

Bruker parameter, ikke stakk

Prosedyrene må selv ta opp tråden riktig når de får igjen kontrollen:

match(t) er som før:

- tester input mot t
- kaller eventuelt "error" (som nå returnerer!)
- kaller <u>ikke</u> "scanto(...)

```
procedure factor ( synchset ) ;
begin
 checkinput ( { (, number }, synchset ) ;
 if not ( token in synchset ) then
  case token of
  ( :  match( ( ) ;
        exp ( { ) } ) ;
        match( ) ) ;
  number :
        match(number) ;
  else error ;
  end case ;
  checkinput ( synchset, { (, number }) ;
 end if ;
end factor ;
```

Hvorfor ikke også "synchset"? (ved "exp ( { ) } )")

```
procedure scanto ( synchset ) ;
begin
  while not ( token in synchset ∪ { $ }) do
     getToken ;
end scanto ;

procedure checkinput ( firstset, followset ) ;
begin
  if not ( token in firstset ) then
     error ;
     scanto ( firstset ∪ followset ) ;
  end if ;
end;
```

# Pensum i kap. 4

Blir lagt ut på hjemmesidens pensumliste!

Hele kapittelet, med følgende modifikasjoner:

- side 159-160 : Ikke detaljene i "Case 3", men man skal vite at en slik algoritme finnes.
- Ikke avsnitt 4.2.4 side 166-167
- Sidene 168 og 173: Vi bruker litt mer intuitive definisjoner på First og Follow (se foilene og trykkfeillisten, men også diskusjonen på punktene 1. og 2. øverst på s. 178 der koblingen til de intuitive definisjonene er gjort)
- Ikke avsnittene 4.5.2 og 4.5.3 side 186 – 189