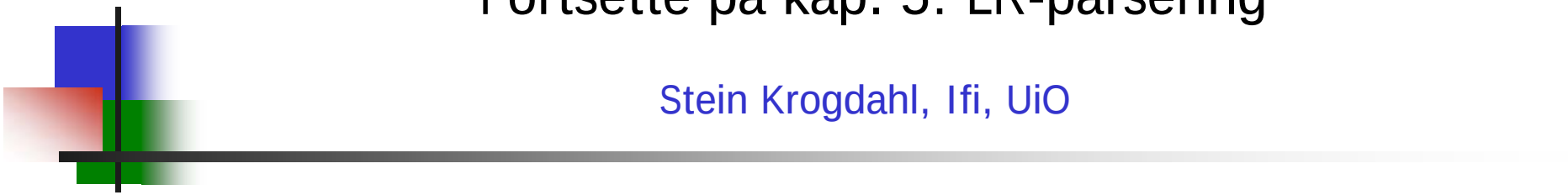


# INF5110, 22/2-2007

Dagens tema:

## Fortsette på kap. 5: LR-parsering

Stein Krogdahl, Ifi, UiO

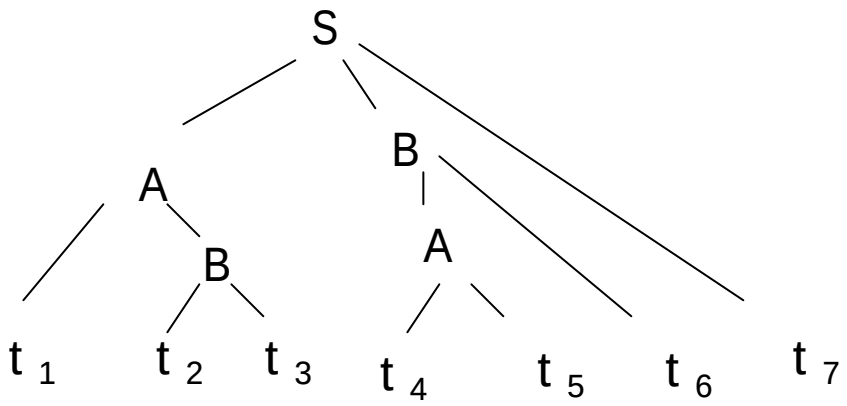
## Videre fremover:

Både tirsdag 27/2 og torsdag 1/3:

Forelesning og orientering om Oblig 1

Start å lese om CUP – et verktøy for å implementere LR-grammatikker. Se forelesningsplanen for CUP sin hjemmeside.

# ”Bottom up” parsering (nedenfra-og-opp)




LR-parsering og grammatikker

- LR(0)
- SLR(1)
- LR(1)
- LALR(1)

-Autimatisert

-YACC, Bison ( LALR(1) )

# Prinsippet og datastrukturen for LR-parsering

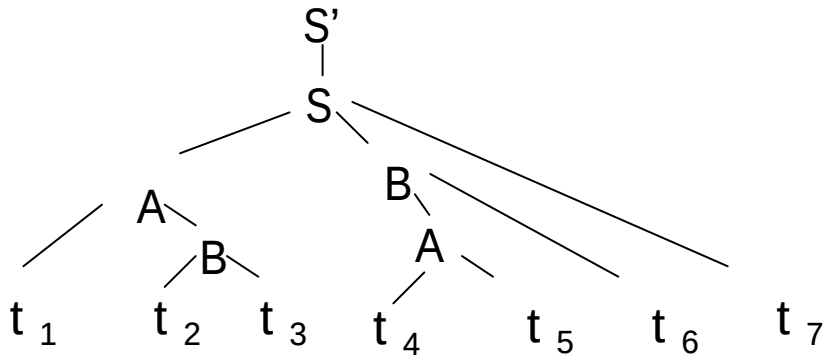
$S' \rightarrow S$

$S \rightarrow A\ B\ t_7 \mid ....$

$A \rightarrow t_4\ t_5 \mid t_1\ B \mid$

$B \rightarrow t_2\ t_3 \mid A\ t_6 \mid ....$

Anta at grammatikken er entydig, og at vi *kjenner syntaks-treet* for setningen:




LR-parsering :

- Ha en ”stakk” for *det som er lest* (altså *omvendt* av LL-parsering med stakk)
- Gjør ”reduksjonen” av et subtre når det ligger ”på toppen av” stakken

# Prinsippet og datastrukturen LR-parsering II

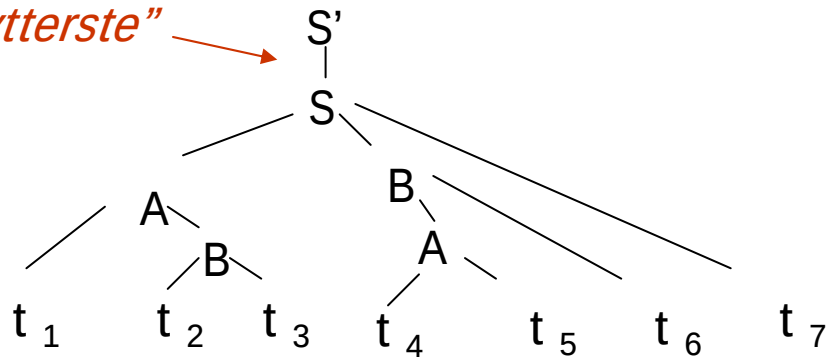
*Sett på ny "ytterste" produksjon*

$S' \rightarrow S$

$S \rightarrow A\ B\ t_7 \mid ....$

$A \rightarrow t_4\ t_5 \mid t_1\ B \mid$

$B \rightarrow t_2\ t_3 \mid A\ t_6 \mid ....$




Anta at grammatikken er entydig, og anta at vi kjenner syntaks-treet for setningen:

- Ha en "stakk" for det som er lest (og "redusert"!)
- Gjør "reduksjonen" av et subtre når subtreet ligger "på toppen av" stakken.
- Da erstatter vi dette med den ikke-terminalen som produserte dette subtreet.
- Dette tilsvarer en bestemt produksjon brukt "omvendt"

Start-situasjonen:

| stakk | input |
|---|---|
| $ | $t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ \$$ |

Slutt-situasjonen:

| stakk | input |
|---|---|
| $ S' | $ |

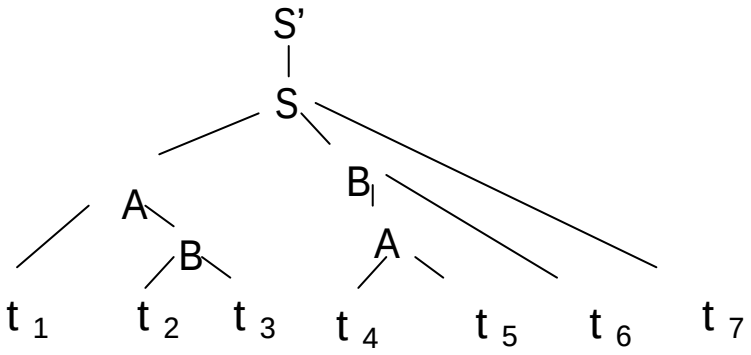
# Prinsippet med LR-parsering III

**S' → S**

**S → A B $t_7$ | ....**

**A → $t_4$ $t_5$ | $t_1$ B |**

**B → $t_2$ $t_3$ | A $t_6$ | ....**




| stakk | input |
|---|---|
| $ | $t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$$ |
| $ $t_1$ | $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$ $ |
| $ $t_1$ $t_2$ | $t_3$ $t_4$ $t_5$ $t_6$ $t_7$ $ |
| $ $t_1$ $t_2$ $t_3$ | $t_4$ $t_5$ $t_6$ $t_7$ $ |
| $ $t_1$ B | $t_4$ $t_5$ $t_6$ $t_7$ $ |
| $ A | $t_4$ $t_5$ $t_6$ $t_7$ $ |
| $ A $t_4$ | $t_5$ $t_6$ $t_7$$ |
| $ A $t_4$ $t_5$ | $t_6$ $t_7$ $ |
| $ A A | $t_6$ $t_7$ $ |
| $ A A $t_6$ | $t_7$ $ |
| $ A B | $t_7$ $ |
| $ A B $t_7$ | $ |
| $ S | $ |

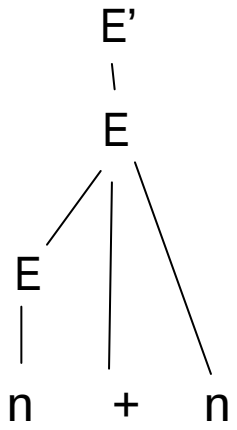
- Får to typer steg:
  - *Reduksjon* (på toppen av stakken med bestemt A → α)
  - *Lesing* (”skift”) av input over til stakken
- Dersom man kjenner syntakstreet for den aktuelle setning, er det lett å angi de rette stegene.
- MEN: Hvordan gjøre dette underveis uten å kjenne resten av input ??
- *Stakk + input:* Går gjennom en høyre-avledning, i omvendt rekkefølge

Husk at nå skal vi redusere input (bottom up) til startsymbolet S',
IKKE produsere input fra startesymbolet (slik vi gjorde ”top-down” i kap 4)

# Eksempel på LR-parsering

$E' \rightarrow E$

$E \rightarrow E + \mathbf{n} \mid \mathbf{n}$

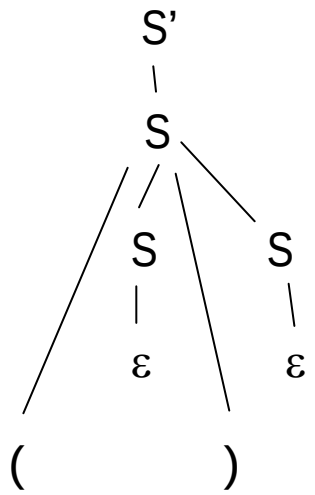



**Boka:** (Stakk + input) utgjør et stadium i en høyre-avledning. Den neste reduksjonen som skal gjøres, kalles situasjonens ”handle” (håndtak).

| | Parsing stack | Input | Action |
|---|---|---|---|
| 1 | $ | **n** + **n** $ | shift |
| 2 | $ **n** | + **n** $ | reduce $E \rightarrow \mathbf{n}$ |
| 3 | $ $E$ | + **n** $ | shift |
| 4 | $ $E$ + | **n** $ | shift |
| 5 | $ $E$ + **n** | $ | reduce $E \rightarrow E + \mathbf{n}$ |
| 6 | $ $E$ | $ | reduce $E' \rightarrow E$ |
| 7 | $ $E'$ | $ | accept |

# Eksempel på LR-parsering

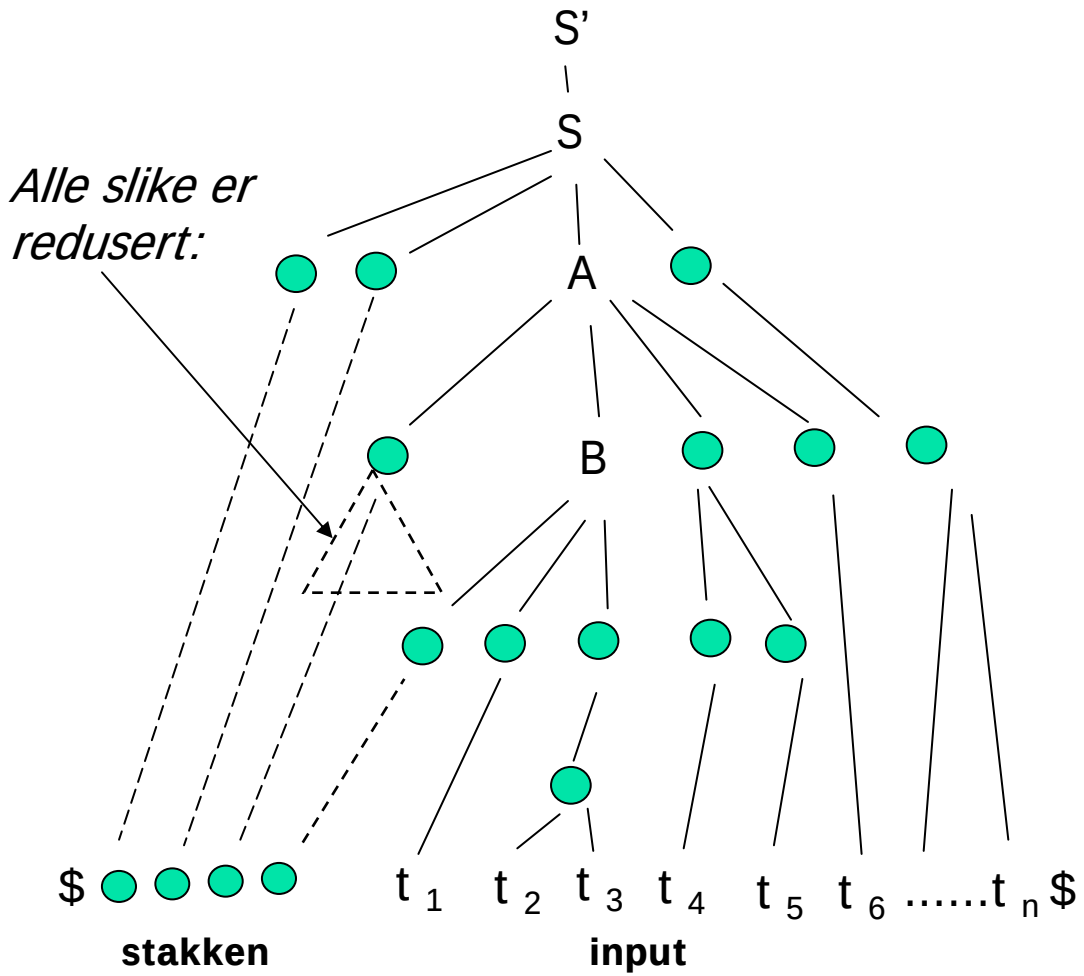
$S' \rightarrow S$
$S \rightarrow ( \ S \ ) \ \ S \mid \varepsilon$




**NB**: S → ε
blir litt ”rar”

| | Parsing stack | Input | Action |
|---|---|---|---|
| 1 | $ | ( ) $ | shift |
| 2 | $ ( | ) $ | reduce $S \rightarrow \varepsilon$ |
| 3 | $ ( $S$ | ) $ | shift |
| 4 | $ ( $S$ ) | $ | reduce $S \rightarrow \varepsilon$ |
| 5 | $ ( $S$ ) $S$ | $ | reduce $S \rightarrow ( \ S \ ) \ S$ |
| 6 | $ $S$ | $ | reduce $S' \rightarrow S$ |
| 7 | $ $S'$ | $ | accept |

# Typisk situasjon under LR-parsering




Legg merke til forskjellen fra LL(1)-parsering med eksplisitt stakk:

LL: Et ”bilde av” det vi *forventer å finne* ligger på stakken

LR: Et ”bilde av” det vi *har lest* ligger på stakken

$s_1 s_2 s_3 s_4 \ldots\ldots s_k$ ⟸ lest input

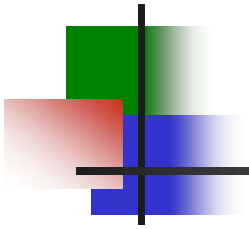
Gitt en entydig grammatikk G.

# Denne behandles som følger for å lage en LR-parser:

(Ikke alt er like tydelig beskrevet i boka, men bare boka er pensum)

- Vi ser på de mulig stakker som kan opptre, og ser disse som strenger over alfabetet {terminaler, ikke-terminaler}. Vi ser altså på:

  {S | S utgjør stakken på ett eller annet stadium i LR-parsering av en setning i L(G)}
- Dette språket viser seg å være regulært, og kan beskrives av en NFA der:
  - Tilstandene angis av ”itemer” av formen: $A \rightarrow X\ Y\ \mathbf{.}\ Z$
  - Kantene kan beskrives nokså greit (kommer snart)
- Denne NFA-en gjør vi om til en DFA på vanlig måte (subset construction fra kap. 2)
- Tilstandene i denne DFA-en vil altså være **mengder av itemer**
- **LR-parseringens hovedsetning (uformelt):**
  - Gitt en lovlig stakk og anta at den fører DFA-en angitt over til en (aksepterende!) tilstand T:
  - Da vil mengden av *itemer* i T angi de mulige ”lokale forhold” vi har ved det punktet vi nå er i i parseringen.
  - Det er ofte flere muligheter/valg (ett for hvert item), siden vi ikke har tatt hensyn til resten av input.



# Itemer: Hver produksjon i BNF-grammatikken gir opphav til et antall slike.

- For produksjonen :

  $A \rightarrow \alpha\, X\, \beta$

- Lager vi itemer med punktum på alle plasser (punktumet er et Meta-symbol):

  $A \rightarrow .\, \alpha\, X\, \beta$

  $A \rightarrow \alpha\, .\, X\, \beta$

  $A \rightarrow \alpha\, X\, .\, \beta$

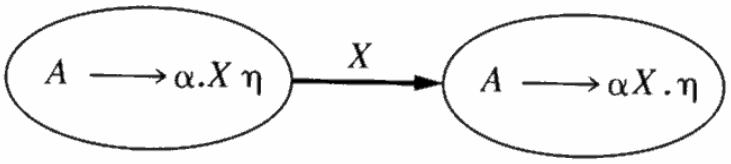
  $A \rightarrow \alpha\, X\, \beta\, .$

  *Disse produksjonene med punktum kalles **itemer***

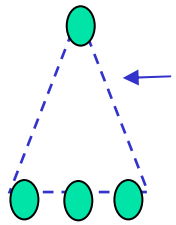
- Punktumet angir grovt sett situasjonen i ”dypeste del” av parseringen i øyeblikket:
  - Det til venstre for punktum er gjenkjent fra input, enten bare lest eller at deler av det er redusert til en ikke-terminal
  - Det til høyre for punktum har vi enda ikke sett/lest

# Kantene i NFA-en

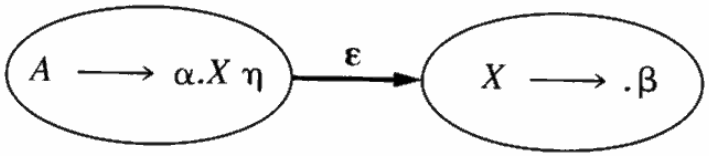
*Dette er ikke fullt forklart i boka*

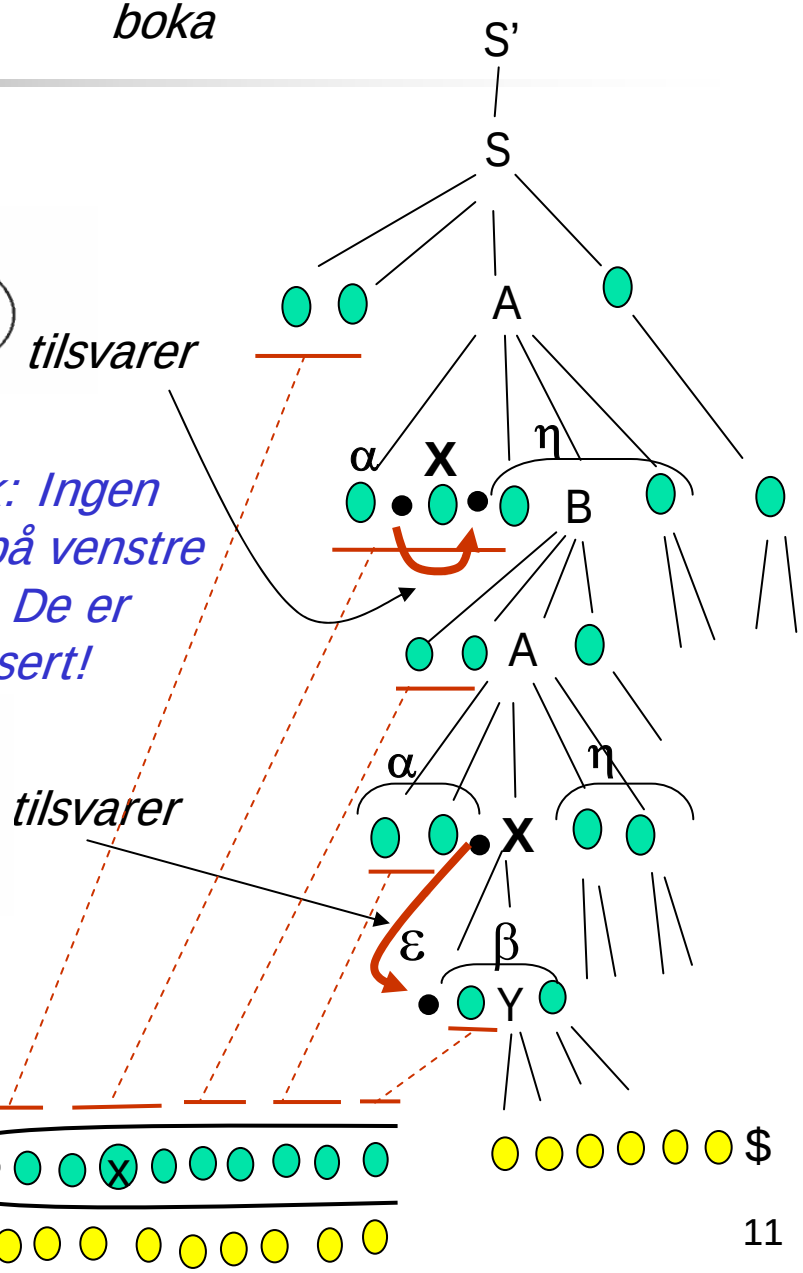



Gitt en grammatikk G og en situasjon under passering av en setning s i L(G).



*Husk: Ingen slik på venstre side. De er redusert!*




*tilsvarer*

*tilsvarer*

*Stakk:*

*Opprinnelig input:*

# Eksempel på NFA

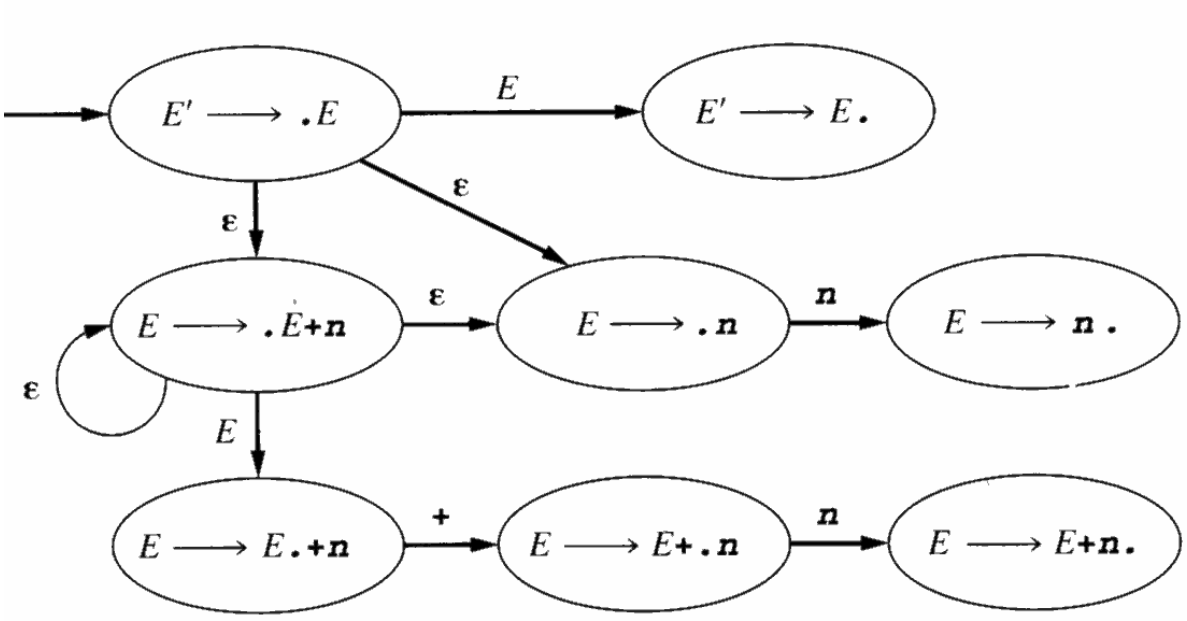
E ' → E

E → E + n

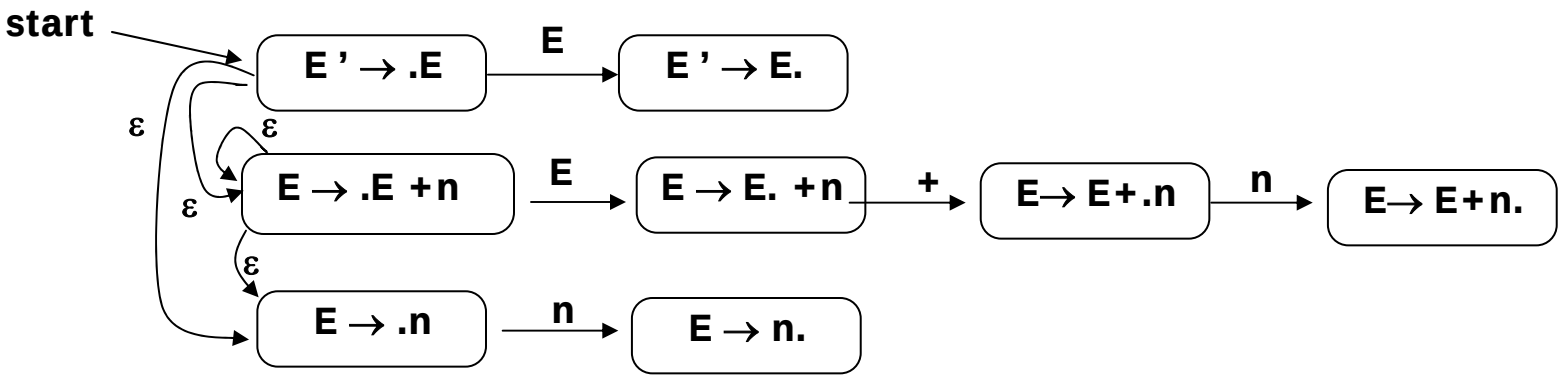
E → n

$E' \rightarrow .E$
$E' \rightarrow E.$
$E \rightarrow .E + n$
$E \rightarrow E.+ n$
$E \rightarrow E + .n$
$E \rightarrow E + n.$
$E \rightarrow .n$
$E \rightarrow n.$

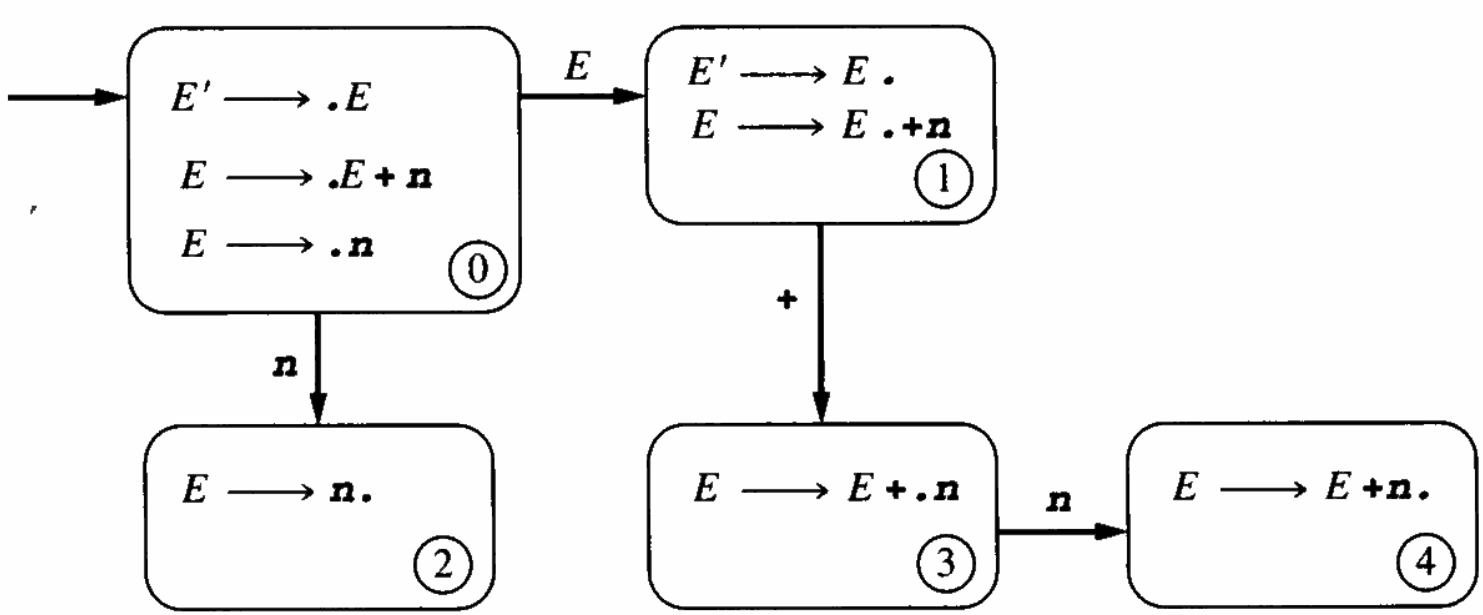
*Itemer (LR(0)itemer)*



LR(0)-NFA (litt mer systematisk enn boka):




LR(0)-DFA:

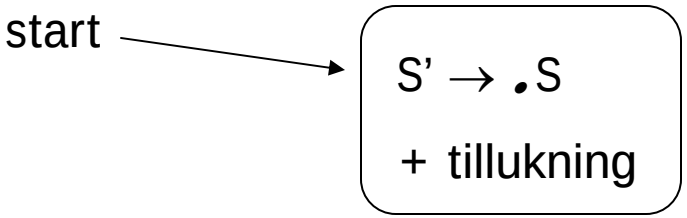


# Hvordan sette opp LR(0) - DFA'en direkte fra grammatikken

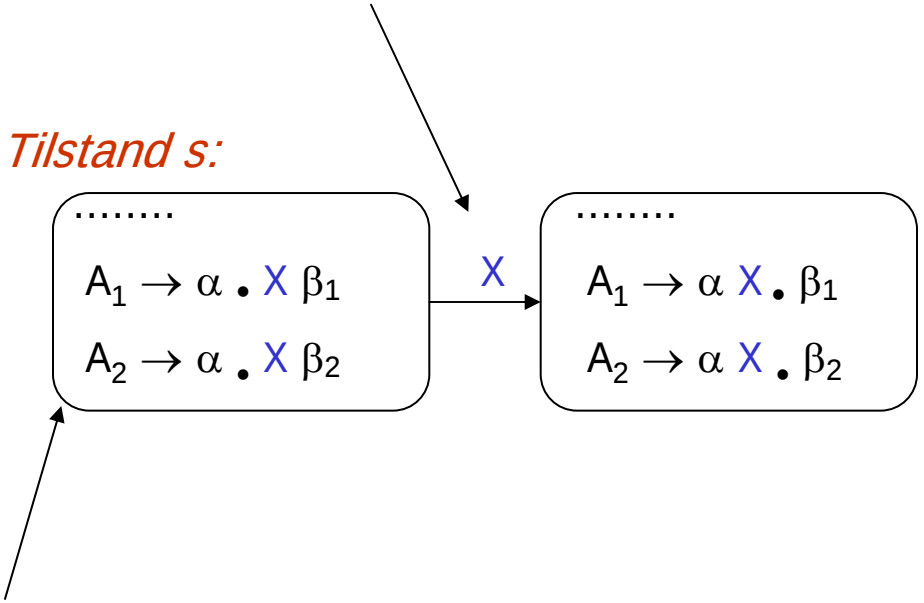
- Tillukning av item-mengde I:
  - Dersom:
    - $A \rightarrow \alpha \bullet B\gamma$ er med i I
    - B er en ikke-terminal
    - $B \rightarrow \beta_1 \mid \beta_2 \mid \ldots\ldots\ldots$

  Da er også Itemene
    - $B \rightarrow \bullet \beta_2$ .
    - $B \rightarrow \bullet \beta_1$

  med i I
- Start-tilstanden til LR(0)-DFA'en er:




- Tilstandsovergang ved symbol X fra tilstand s
  - X er terminal eller ikke-terminal




Få med *alle* av formen $A \rightarrow \alpha \bullet X\ \beta$

# Hvordan sette opp LR(0)-DFA'en direkte fra grammatikken - II

$$S' \rightarrow S$$
$$S \rightarrow ( S ) \quad S \mid \varepsilon$$

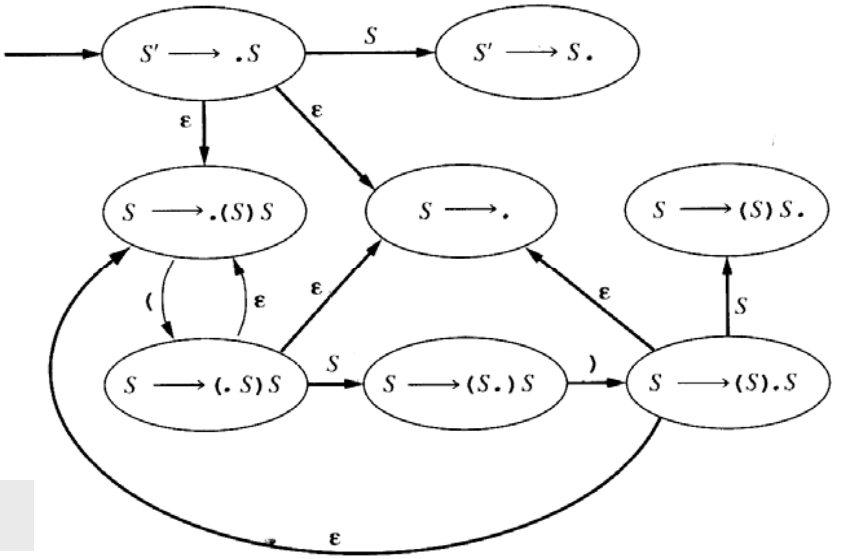
Merk at S → ε bare gir ett item, nemlig: S → •

(Altså ikke: ~~S → • ε~~

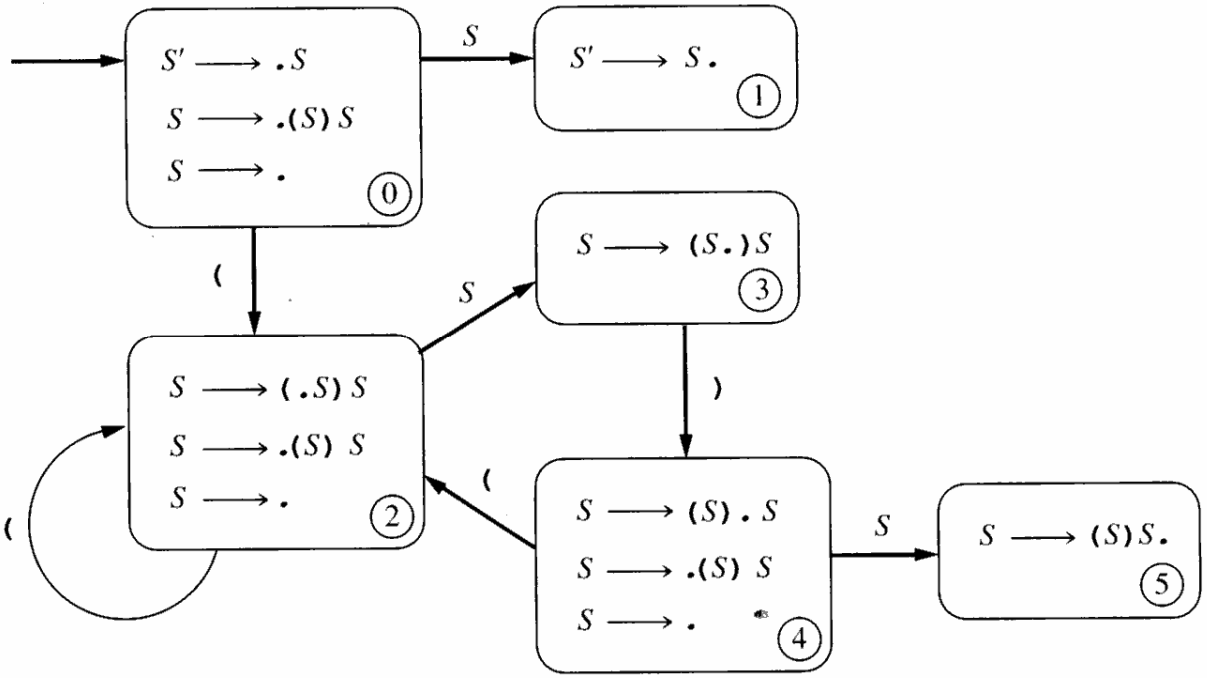
og ~~S → ε •~~)

$$S' \rightarrow .S$$
$$S' \rightarrow S.$$
$$S \rightarrow .(S)S$$
$$S \rightarrow (.S)S$$
$$S \rightarrow (S.)S$$
$$S \rightarrow (S).S$$
$$S \rightarrow (S)S.$$
$$S \rightarrow .$$

LR(0) – NFA:

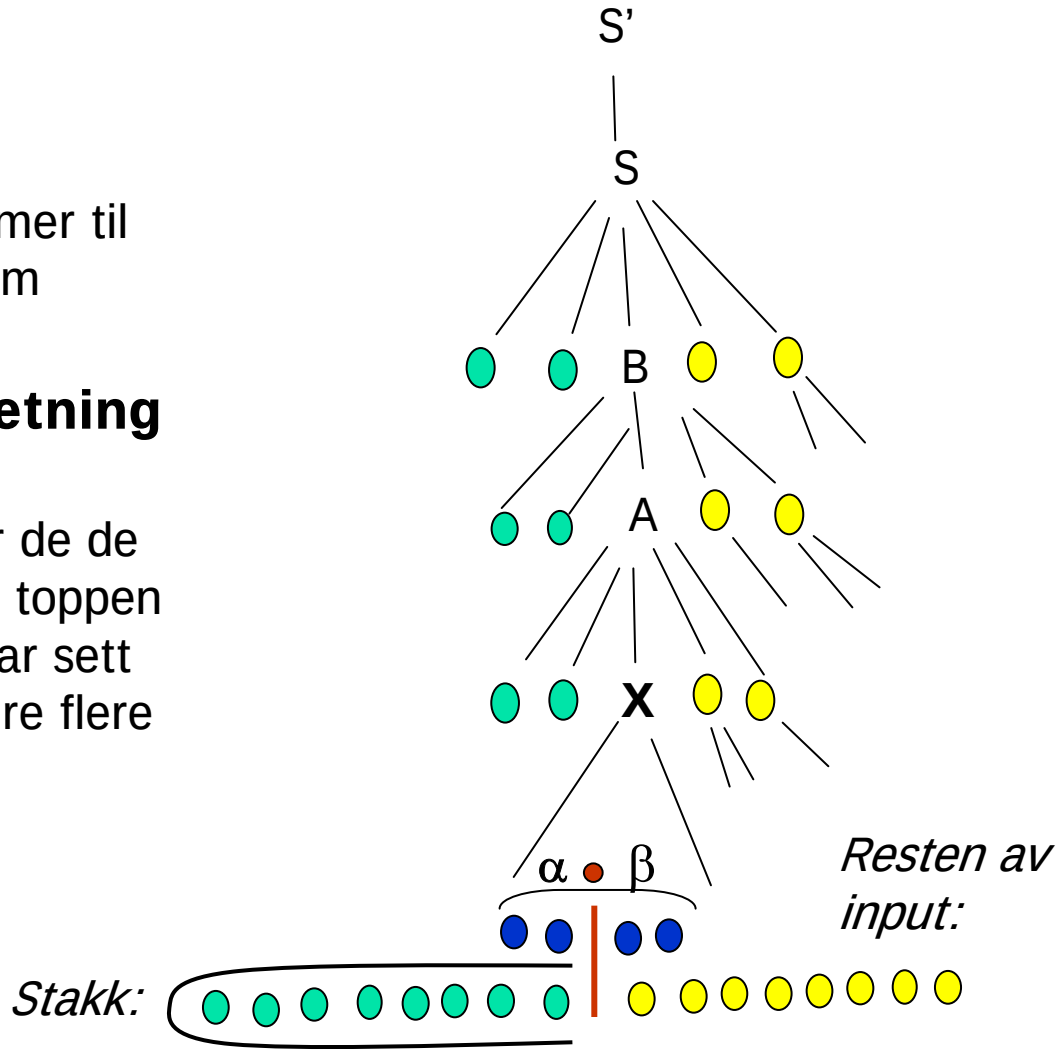



LR(0) – DFA:



# Hva sier ”topp-tilstanden”?

- Topp-tilstanden:
  - Den DFA-tilstanden vi kommer til ved å la stakken gå gjennom DFA’en.
- **LR-parseringens hovedsetning** (litt løselig, bevises ikke):
  - Itemene i topptilstanden er de de mulige ”lokale forhold” ved toppen av stakken (Siden vi ikke har sett resten av input kan det være flere muligheter)

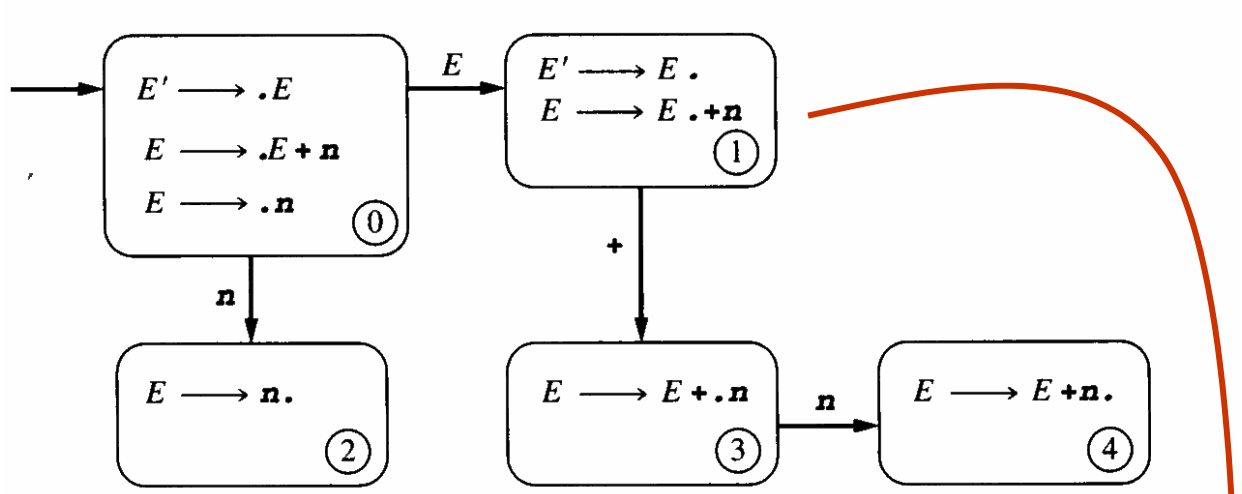


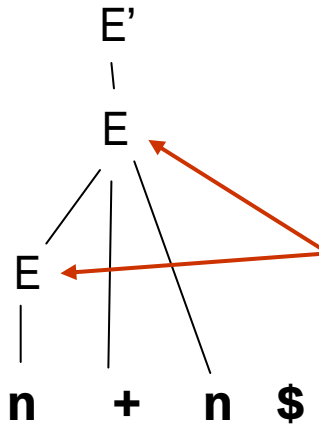

Dersom itemet: $X \rightarrow \alpha \bullet \beta$ er med i topp-tilstanden kan situasjonen være som angitt

# Hva sier topp-tilstanden? Eksempel:

$E' \rightarrow E$

$E \rightarrow E + n \mid n$



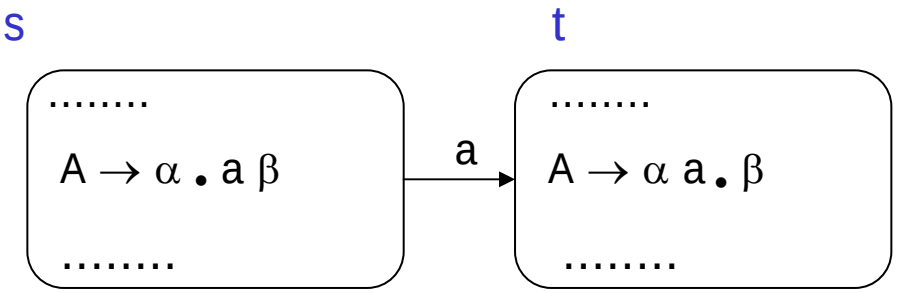

| | |
|---|---|
| $ | n + n $ |
| $ n | + n $ |
| $ E | + n $ |
| $ E + | n $ |
| $ E + n | $ |
| $ E | $ |
| $ E' | $ |

Skal skifte

Skal redusere med E'→E

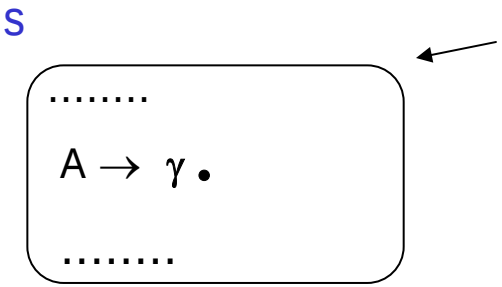
# LR(0)-grammatikker og LR(0)-algoritmen:

1)




Angir at neste skritt *kan* være skift, og at dette er lovlig om neste input er a (ny topptilstand blir t)

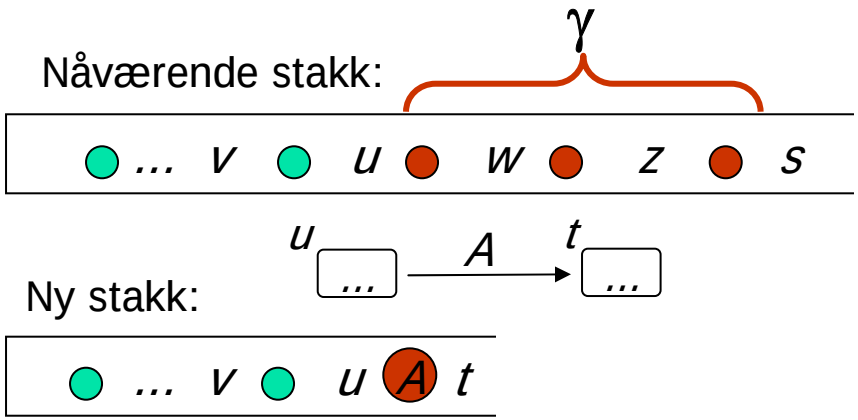
2)




Kalles et "slutt-item" (complete item)

Angir at neste skritt *kan* være reduksjon A → γ

Vi holder tilsandene mellom stakk-symbolene




*N.B :* **s,t,u,v,w,z** står for nummere på tilstander fra DFAen. Disse ligger på stakken sammen med det vi foreløpig har skiftet inn fra innput og evt. redusert dette til.

LR(0) –grammatikk:
Dersom dette gir entydige aksjoner for alle tilstander (kan=må), er grammatikken LR(0)!

Da har vi en grei algoritme!

Reduser-steget: Pop av det som tilsvarer γ (og mellomliggende tilstander), og push på A, og finn riktig ny topptilstand ut fra u og A

# Er eksempel-grammatikkene i Kap 5 LR(0) ?

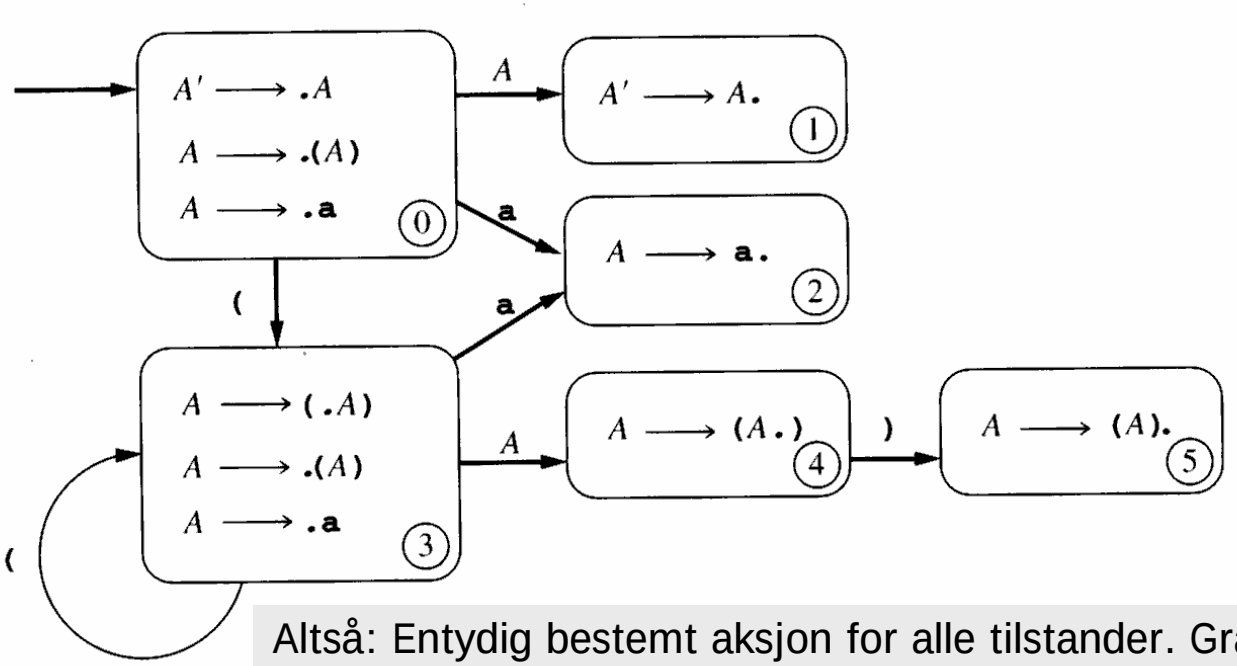
Ser på de tre grammatikkene våre – først A:

$$A \rightarrow ( \; A \; ) \mid \mathbf{a}$$

$$S' \rightarrow S$$
$$S \rightarrow ( \; S \; ) \;\; S \mid \varepsilon$$

$$E' \rightarrow E$$
$$E \rightarrow E + \mathbf{n} \mid \mathbf{n}$$

A gir følgende LR(0) – DFA:




| Tilst. | Mulig aksjoner: |
|---|---|
| 0 | Bare skift, for "(","a" |
| 1 | Bare red. med A' → A |
| 2 | Bare red. med A → a |
| 3 | Bare skift, for "(","a" |
| 4 | Bare skift, for ")" |
| 5 | Bare red. med A → (A) |

Altså: Entydig bestemt aksjon for alle tilstander. Grammatikken er LR(0)

MERK: Der det reduksjon må det ikke være tvil om med hvilken produksjon!

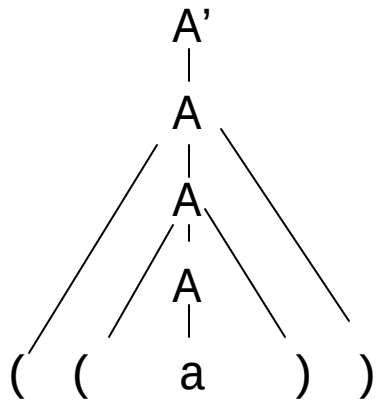
# Tabell-oppsett for en LR(0) - grammatikk:

| State | Action | Rule | Input | | | Goto |
|---|---|---|---|---|---|---|
| | | | ( | **a** | ) | *A* |
| 0 | shift | | 3 | 2 | | 1 |
| 1 | reduce | $A' \rightarrow A$ | | | | |
| 2 | reduce | $A \rightarrow$ **a** | | | | |
| 3 | shift | | 3 | 2 | | 4 |
| 4 | shift | | | | 5 | |
| 5 | reduce | $A \rightarrow$ ( *A* ) | | | | |

Hvis en reduksjon bringer oss tilbake til tilstand 0 eller 3, sier Goto hvilken tilstand A gir.

Parsering av setningen: ((a))

| | Parsing stack | Input | Action |
|---|---|---|---|
| 1 | \$ 0 | ((**a**)) \$ | shift |
| 2 | \$ 0 ( 3 | (**a**)) \$ | shift |
| 3 | \$ 0 ( 3 ( 3 | **a**)) \$ | shift |
| 4 | \$ 0 ( 3 ( 3 **a** 2 | )) \$ | reduce $A \rightarrow$ **a** |
| 5 | \$ 0 ( 3 ( 3 *A* 4 | )) \$ | shift |
| 6 | \$ 0 ( 3 ( 3 *A* 4 ) 5 | ) \$ | reduce $A \rightarrow$ ( *A* ) |
| 7 | \$ 0 ( 3 *A* 4 | ) \$ | shift |
| 8 | \$ 0 ( 3 *A* 4 ) 5 | \$ | reduce $A \rightarrow$ ( *A* ) |
| 9 | \$ 0 *A* 1 | \$ | accept |




Skal her redusere med
A' → A, og input tom: Ferdig

# Parsering av noen gale strenger for: $A \rightarrow ( A ) \mid a$

| State | Action | Rule | Input | | | Goto |
|---|---|---|---|---|---|---|
| | | | ( | a | ) | A |
| 0 | shift | | 3 | 2 | | 1 |
| 1 | reduce | $A' \rightarrow A$ | | | | |
| 2 | reduce | $A \rightarrow$ **a** | | | | |
| 3 | shift | | 3 | 2 | | 4 |
| 4 | shift | | | | 5 | |
| 5 | reduce | $A \rightarrow ( A )$ | | | | |

| | |
|---|---|
| $ 0 | ( ( a ) $ |
| $ 0 ( 3 | ( a ) $ |
| $ 0 ( 3 | ( a ) $ |
| $ 0 ( 3 ( 3 | a ) $ |
| $ 0 ( 3 ( 3 a | ) $ |
| $ 0 ( 3 ( 3 A ) 5 | $ |
| $ 0 ( 3 ( 3 A 4 | $ |

| | |
|---|---|
| $ 0 | ( ) $ |
| $ 0 ( 3 | ) $ |

# En (litt ruskete) formulering av LR(0)-kravet: Dersom dette gir en entydig algoritme er grammatikken LR(0):

**s er en DFA-tilstand med flere itemer**

***The LR(0) parsing algorithm.*** Let $s$ be the current state (at the top of the parsing stack). Then actions are defined as follows:

1. If state $s$ contains any item of the form $A \to \alpha . X \beta$, where $X$ is a terminal, then the action is to shift the current input token onto the stack. If this token is $X$, and state $s$ contains item $A \to \alpha . X \beta$, then the new state to be pushed on the stack is ~~the state containing the item $A \to \alpha X . \beta$~~. If this token is not $X$ for some item in state $s$ of the form just described, an error is declared.

   **t, where s $\xrightarrow{X}$ t**

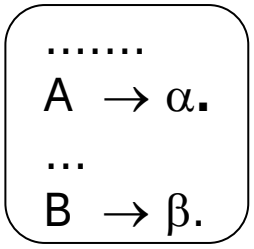
2. If state $s$ contains any complete item (an item of the form $A \to \gamma .$), then the action is to reduce by the rule $A \to \gamma$. A reduction by the rule $S' \to S$, where $S$ is the start state, is equivalent to acceptance, provided the input is empty, and error if the input is not empty. In all other cases, the new state is computed as follows. Remove the string $\gamma$ and all of its corresponding states from the parsing stack (the string $\gamma$ must be at the top of the stack, according to the way the DFA is constructed). Correspondingly, back up in the DFA to the state from which the construction of $\gamma$ began (this must be the state uncovered by the removal of $\gamma$). Again, by the construction of the DFA, this state must contain an item of the form $B \to \alpha . A \beta$. Push $A$ onto the stack, and push (as the new state) the state ~~containing the item $B \to \alpha A . \beta$~~. (Note that this corresponds to following the transition on $A$ in the DFA, which is indeed reasonable, since we are pushing $A$ onto the stack.)

   Avslutning

   **t, where u $\xrightarrow{A}$ t**

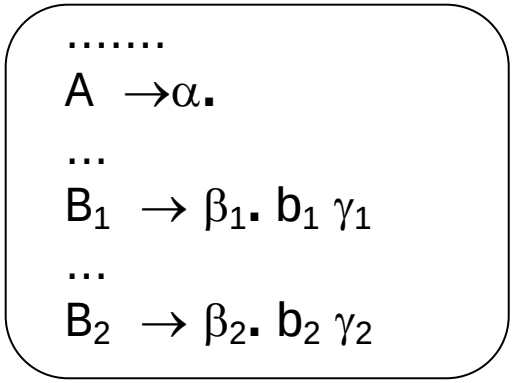
# SLR(1) - grammatikker, SLR(1) - algoritmer

- Svært få grammatikker er LR(0)
- Ved å se på Follow-mengdene kan vi få en mye sterkere algoritme
- Tar også utgangspunkt i LR(0) DFA-en
- Tabellene er nesten like, men nå må "reduser-linjene" spesifiseres for hvert mulig "neste input-symbol".




LR(0): Har her (uløselig) red/red-konflikt

SLR(1): Dersom: Follow(A) ∩Follow(B) = ∅ så kan konflikten løses ved å se på neste input

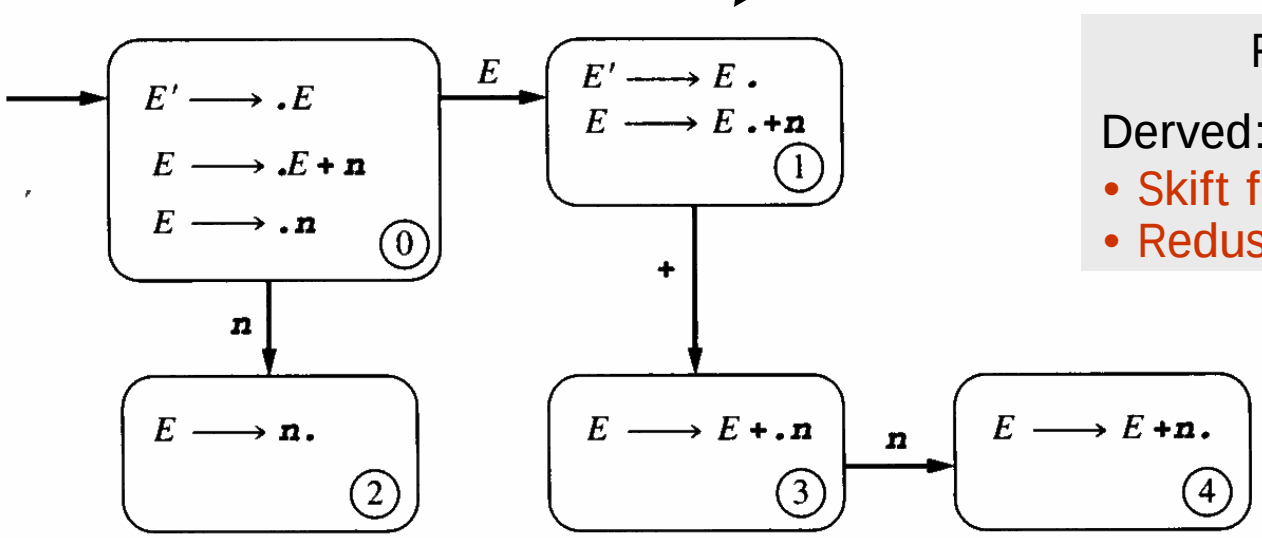



LR(0) : Har her (uløselig) shift/red - konflikt

SLR(1): Dersom: Follow(A) ∩{$b_1$,$b_2$} = ∅ så kan konflikten løses ved å se på neste input-symbol

# Er denne grammatikken SLR(1)

*Skift/reduser konflikt i LR(0), men ikke i SLR(1)*




Follow(E') = { $ }

Derved:

- Skift for '+'
- Reduser for '$', med E' → E

<u>En grei måte å formulere SLR(1)-kravet:</u>

For alle DFA-tilstander s skal gjelde:

1. For any item $A \rightarrow \alpha.X\beta$ in $s$ with $X$ a terminal, there is no complete item $B \rightarrow \gamma.$ in $s$ with $X$ in Follow($B$).
2. For any two complete items $A \rightarrow \alpha.$ and $B \rightarrow \beta.$ in $s$, Follow($A$) ∩ Follow($B$) is empty.

*Ville ellers ha shift / reduser -konflikt ved input X*

*Ville ellers ha reduser / reduser -konflikt ved input i denne mengden*

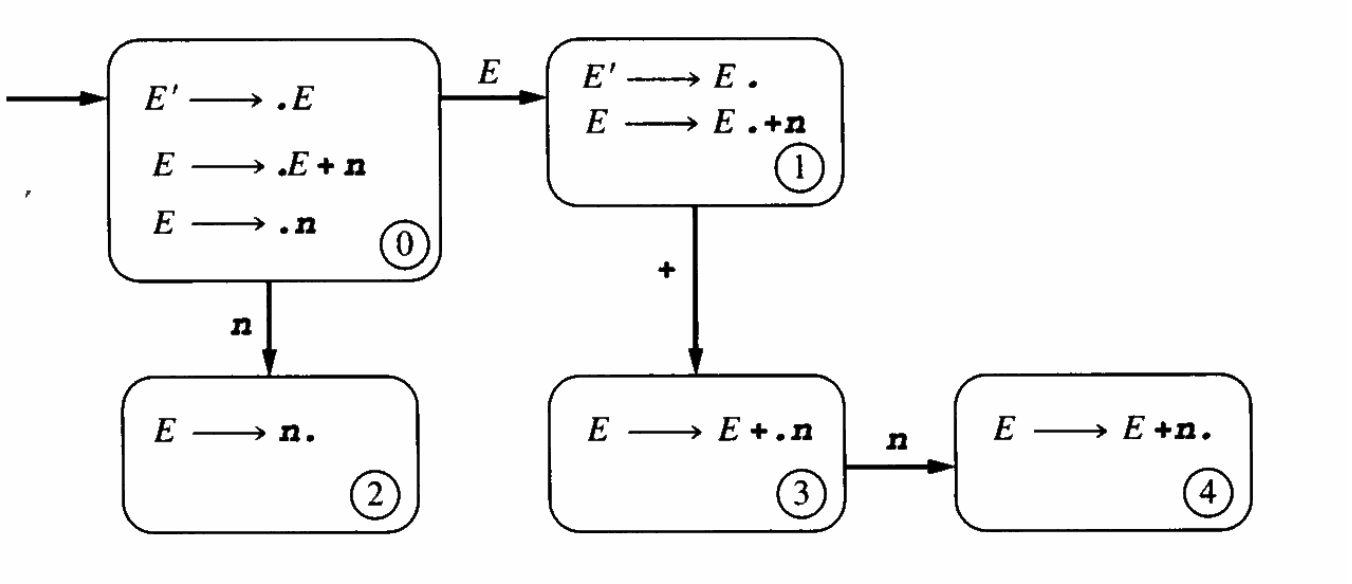
## En tilsvarende formulering av SLR(1) kravet
## Dersom følgende gir entydig algoritme, er grammatikken SLR(1)

***The SLR(1) parsing algorithm.*** Let $s$ be the current state (at the top of the parsing stack). Then actions are defined as follows:

1. If state $s$ contains any item of the form $A \to \alpha . X \beta$, where $X$ is a terminal, and $X$ is the next token in the input string, then the action is to shift the current input token onto the stack, and the new state to be pushed on the stack is the state ~~containing the item $A \to \alpha X . \beta$~~. **t, where s $\xrightarrow{X}$ t**
2. If state $s$ contains the complete item $A \to \gamma .$, and the next token in the input string is in Follow($A$), then the action is to reduce by the rule $A \to \gamma$. A reduction by the rule $S' \to S$, where $S$ is the start state, is equivalent to acceptance; this will happen only if the next input token is \$.[4] In all other cases, the new state is computed as follows. Remove the string $\gamma$ and all of its corresponding states from the parsing stack. Correspondingly, back up in the DFA to the state from which the construction of $\gamma$ began. By construction, this state must contain an item of the form $B \to \alpha . A \beta$. Push $A$ onto the stack, and push the state ~~containing the item $B \to \alpha A . \beta$~~. **t, where u $\xrightarrow{A}$ t**
3. If the next input token is such that neither of the above two cases applies, an error is declared.

**Dette er nytt i forhold til LR(0).**

# Tabell-oppsett for SLR(1)-grammatikk




| State | Input | | | Goto |
|---|---|---|---|---|
| | n | + | $ | E |
| 0 | s2 | | | 1 |
| 1 | | s3 | accept | |
| 2 | | r ($E \rightarrow n$) | r ($E \rightarrow n$) | |
| 3 | s4 | | | |
| 4 | | r ($E \rightarrow E + n$) | r ($E \rightarrow E + n$) | |

# Parsering for SLR(1)-grammatikk

| State | Input | | | Goto |
|---|---|---|---|---|
| | **n** | **+** | **$** | *E* |
| 0 | s2 | | | 1 |
| 1 | | s3 | accept | |
| 2 | | r ($E \rightarrow$ **n**) | r ($E \rightarrow$ **n**) | |
| 3 | s4 | | | |
| 4 | | r ($E \rightarrow E$ **+ n**) | r ($E \rightarrow E$ **+ n**) | |

Kan også se på gale setninger som:

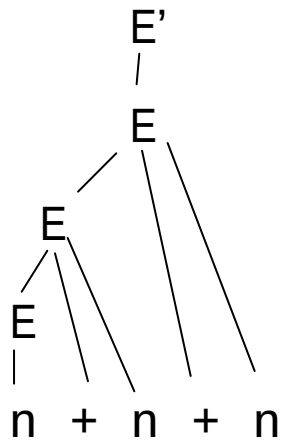
+ n $

n n $

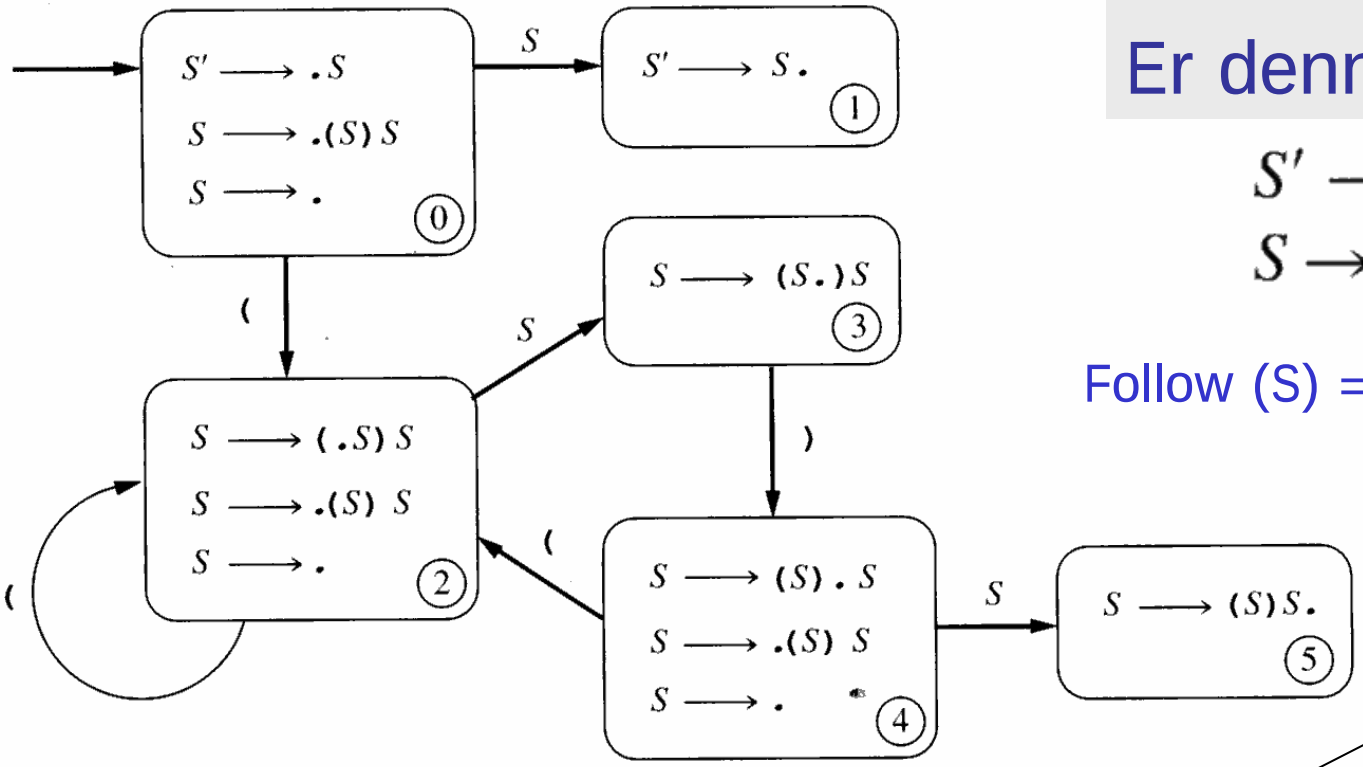
n + $

Parsering av setningen: n + n + n

| | Parsing stack | Input | Action |
|---|---|---|---|
| 1 | $ 0 | **n + n + n** $ | shift 2 |
| 2 | $ 0 **n** 2 | **+ n + n** $ | reduce $E \rightarrow$ **n** |
| 3 | $ 0 *E* 1 | **+ n + n** $ | shift 3 |
| 4 | $ 0 *E* 1 **+** 3 | **n + n** $ | shift 4 |
| 5 | $ 0 *E* 1 **+** 3 **n** 4 | **+ n** $ | reduce $E \rightarrow E$ **+ n** |
| 6 | $ 0 *E* 1 | **+ n** $ | shift 3 |
| 7 | $ 0 *E* 1 **+** 3 | **n** $ | shift 4 |
| 8 | $ 0 *E* 1 **+** 3 **n** 4 | $ | reduce $E \rightarrow E$ **+ n** |
| 9 | $ 0 *E* 1 | $ | accept |




Direkte ut fra tabellen




# Er denne SLR(1) ?

$$S' \rightarrow S$$
$$S \rightarrow ( \; S \; ) \quad S \mid \varepsilon$$

Follow (S) = **{** ), $ **}**

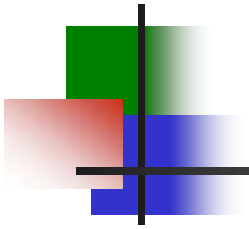
Til senere:
Dette får vi i SLR(1), men ikke i LALR(1). Begge oppdager feilen, men LALR gjør det noe tidligere.

| State | Input | | | Goto |
|---|---|---|---|---|
| | ( | ) | $ | S |
| 0 | s2 | r ($S \rightarrow \varepsilon$) | r ($S \rightarrow \varepsilon$) | 1 |
| 1 | | | accept | |
| 2 | s2 | r ($S \rightarrow \varepsilon$) | r ($S \rightarrow \varepsilon$) | 3 |
| 3 | | s4 | | |
| 4 | s2 | r ($S \rightarrow \varepsilon$) | r ($S \rightarrow \varepsilon$) | 5 |
| 5 | | r ($S \rightarrow ( \; S \; ) \; S$) | r ($S \rightarrow ( \; S \; ) \; S$) | |

# Parsering av setningen: ( ) ( ) $

| | Parsing stack | Input | Action |
|---|---|---|---|
| 1 | $ 0 | ( ) ( ) $ | shift 2 |
| 2 | $ 0 ( 2 | ) ( ) $ | reduce $S \rightarrow \varepsilon$ |
| 3 | $ 0 ( 2 $S$ 3 | ) ( ) $ | shift 4 |
| 4 | $ 0 ( 2 $S$ 3 ) 4 | ( ) $ | shift 2 |
| 5 | $ 0 ( 2 $S$ 3 ) 4 ( 2 | ) $ | reduce $S \rightarrow \varepsilon$ |
| 6 | $ 0 ( 2 $S$ 3 ) 4 ( 2 $S$ 3 | ) $ | shift 4 |
| 7 | $ 0 ( 2 $S$ 3 ) 4 ( 2 $S$ 3 ) 4 | $ | reduce $S \rightarrow \varepsilon$ |
| 8 | $ 0 ( 2 $S$ 3 ) 4 ( 2 $S$ 3 ) 4 $S$ 5 | $ | reduce $S \rightarrow (S)S$ |
| 9 | $ 0 ( 2 $S$ 3 ) 4 $S$ 5 | $ | reduce $S \rightarrow (S)S$ |
| 10 | $ 0 $S$ 1 | $ | accept |

| State | Input | | | Goto |
|---|---|---|---|---|
| | ( | ) | $ | $S$ |
| 0 | s2 | r ($S \rightarrow \varepsilon$) | r ($S \rightarrow \varepsilon$) | 1 |
| 1 | | | accept | |
| 2 | s2 | r ($S \rightarrow \varepsilon$) | r ($S \rightarrow \varepsilon$) | 3 |
| 3 | | s4 | | |
| 4 | s2 | r ($S \rightarrow \varepsilon$) | r ($S \rightarrow \varepsilon$) | 5 |
| 5 | | r ($S \rightarrow$ ( $S$ ) $S$) | r ($S \rightarrow$ ( $S$ ) $S$) | |



# Flertydige grammatikker er aldri SLR(1) eller LR(1)

- Er heller ikke SLR(k) eller LR(k) for noen k
- LR(0)-DFA'ene vil derfor alltid ha "uløselige" konflikter
- **Men**: Konfliktene kan oftest løses med intuisjon og fornuft, f.eks. ved å angi presedens og assosiativitet
- F.eks. vanlig strategi i Yacc etc. (antakeligvis også i CUP):
  - Skift/Reduser – konflikter:

    Bruker skift, om intet annet er angitt

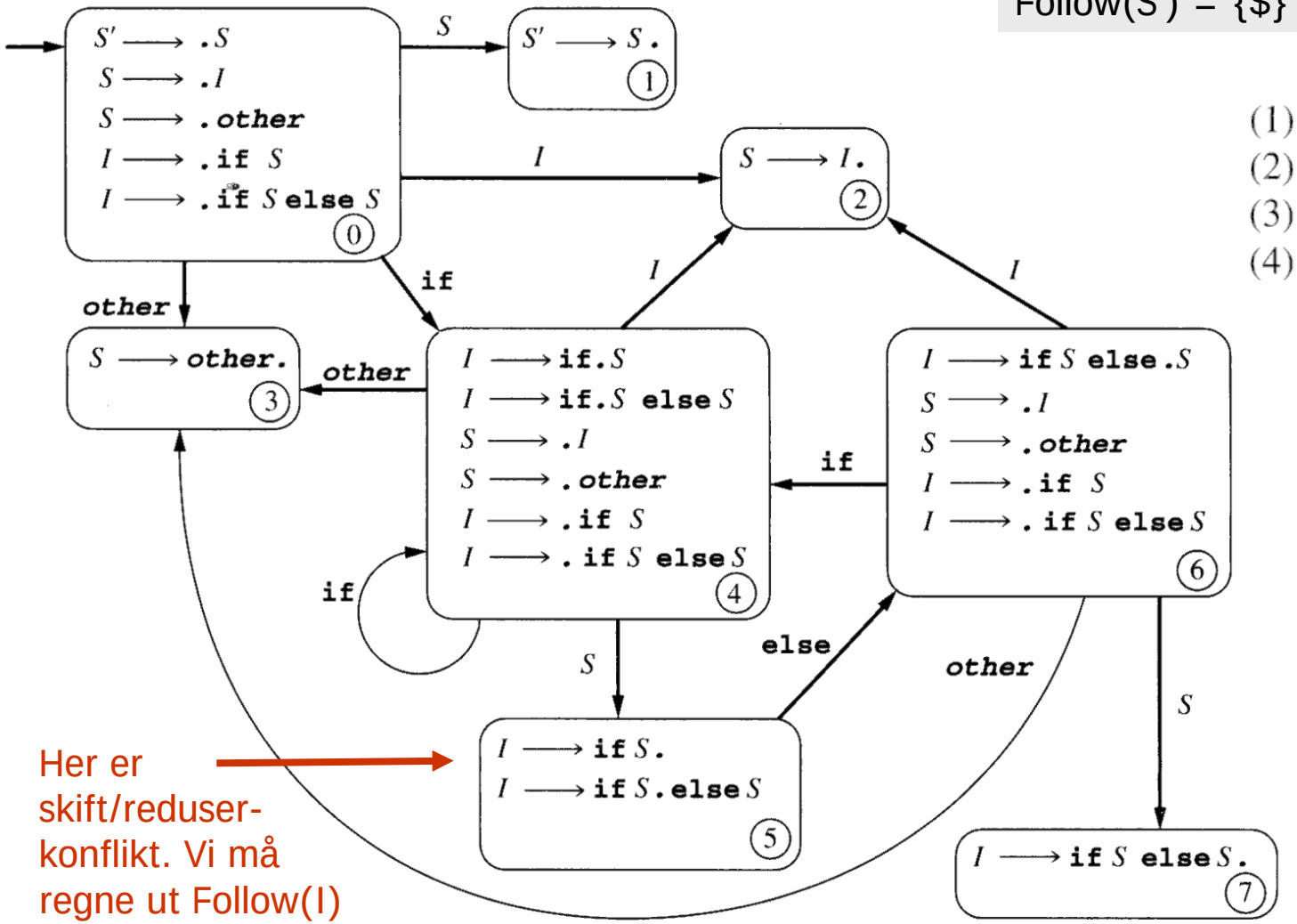
    (man kan ofte angi valg eksplisitt)

*statement* → *if-stmt* | **other**
*if-stmt* → **if** ( *exp* ) *statement*
| **if** ( *exp* ) *statement* **else** *statement*
*exp* → **0** | **1**

$S \rightarrow I \mid \textbf{other}$
$I \rightarrow \textbf{if}\ S \mid \textbf{if}\ S\ \textbf{else}\ S$

Follow(S) = Follow(I) = { $, **else** }

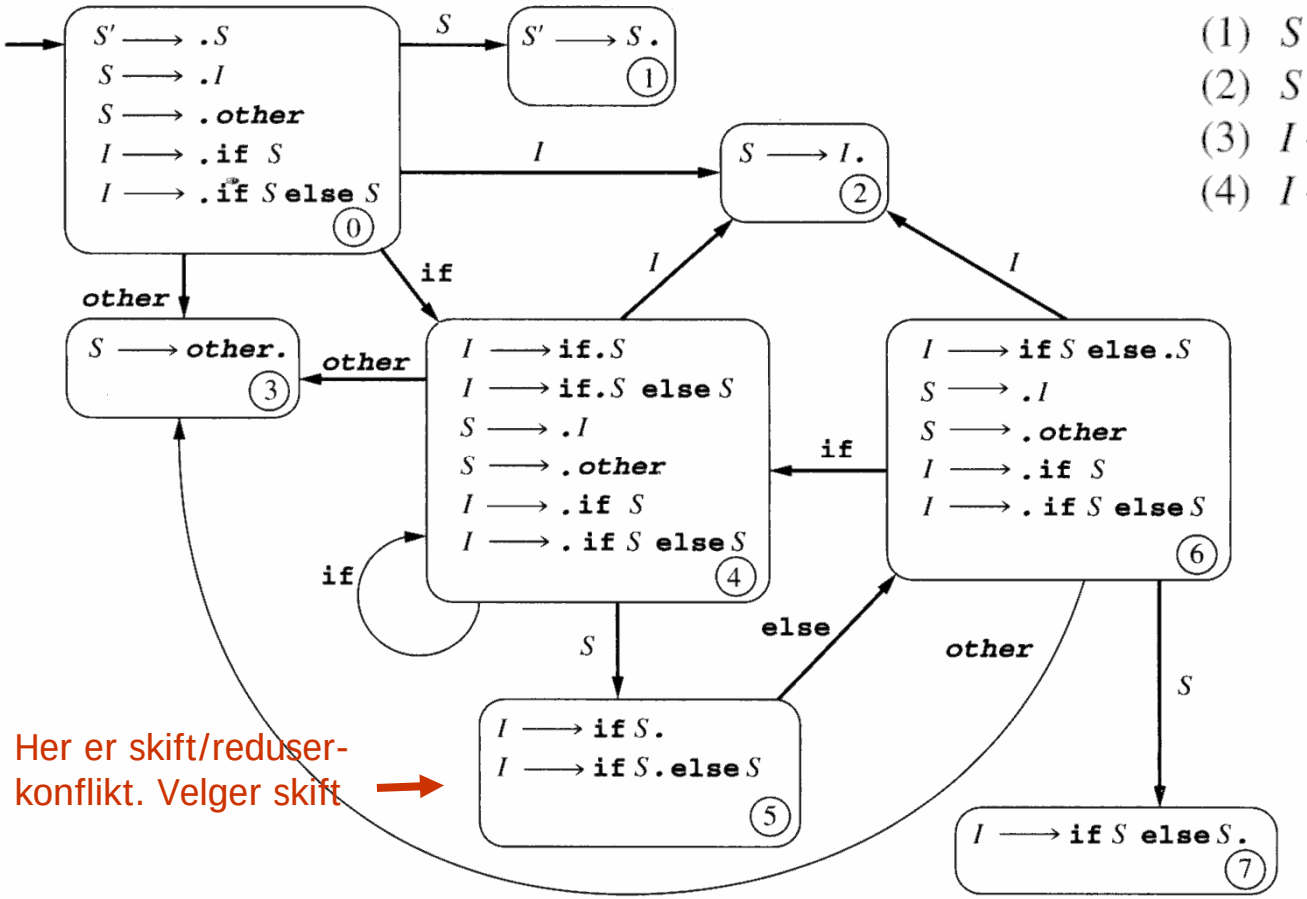
Follow(S') = {$}




(1) $S \rightarrow I$
(2) $S \rightarrow \textbf{other}$
(3) $I \rightarrow \textbf{if}\ S$
(4) $I \rightarrow \textbf{if}\ S\ \textbf{else}\ S$

Her er skift/reduser-konflikt. Vi må regne ut Follow(I)

NB: Det måtte bli minst én konflikt, siden grammatikken er flertydig.




(1) $S \rightarrow I$
(2) $S \rightarrow$ **other**
(3) $I \rightarrow$ **if** $S$
(4) $I \rightarrow$ **if** $S$ **else** $S$

Follow(S) = Follow(I) = { $, **else** }

Follow(S') = {$}

Her er skift/reduser-konflikt. Velger skift

| State | Input | | | | Goto | |
|---|---|---|---|---|---|---|
| | **if** | **else** | **other** | **$** | *S* | *I* |
| 0 | s4 | | s3 | | 1 | 2 |
| 1 | | | | accept | | |
| 2 | | r1 | | r1 | | |
| 3 | | r2 | | r2 | | |
| 4 | s4 | | s3 | | 5 | 2 |
| 5 | | s6 | | r3 | | |
| 6 | s4 | | s3 | | 7 | 2 |
| 7 | | r4 | | r4 | | |

# SLR(1) tabell med ”hånd-løst” konflikt (tilstand 5)

| State | Input | | | | Goto | |
|---|---|---|---|---|---|---|
| | **if** | **else** | **other** | **$** | *S* | *I* |
| 0 | s4 | | s3 | | 1 | 2 |
| 1 | | | | accept | | |
| 2 | | r1 | | r1 | | |
| 3 | | r2 | | r2 | | |
| 4 | s4 | | s3 | | 5 | 2 |
| 5 | | s6 | | r3 | | |
| 6 | s4 | | s3 | | 7 | 2 |
| 7 | | r4 | | r4 | | |

Her betyr:

**s4** – skift **if** fra input til stakk. Legg så 4 på toppen av stakken

**r3** – reduser ved regel 3 i grammatikken. Tilstanden på toppen av (den reduserte) stakken gir da ny tilstand ved ”Goto”

## Parsering:

| | |
|---|---|
| $ 0 | if if other else other $ |
| $ 0 if 4 | if other else other $ |
| $ 0 if 4 if 4 | other else other $ |
| $ 0 if 4 if 4 other 3 | else other $ |
| $ 0 if 4 if 4 S 5 | else other $ |
| $ 0 if 4 if 4 S 5 else 6 | other $ |

(1) $S \rightarrow I$
(2) $S \rightarrow$ **other**
(3) $I \rightarrow$ **if** $S$
(4) $I \rightarrow$ **if** $S$ **else** $S$