

INF5110

Kap. 5: Parsering nedenfra-og-opp  
(Bottom-up parsing)

Del 1

21/2-2014

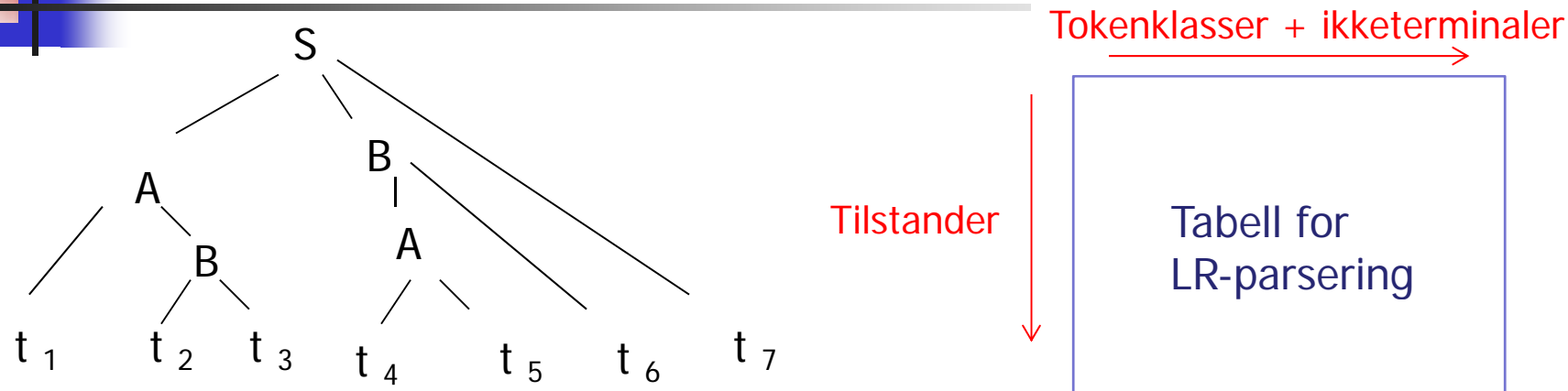
Stein Krogdahl

Ifi, UiO

**Angående Oblig 1:**

Blir lagt ut tirsdag/onsdag neste uke  
Oblig-ansvarlig Henning Berg orienterer 28/2  
Frist: Litt etter 15. mars

# "Bottom up" parsing (nedenfra-og-opp)

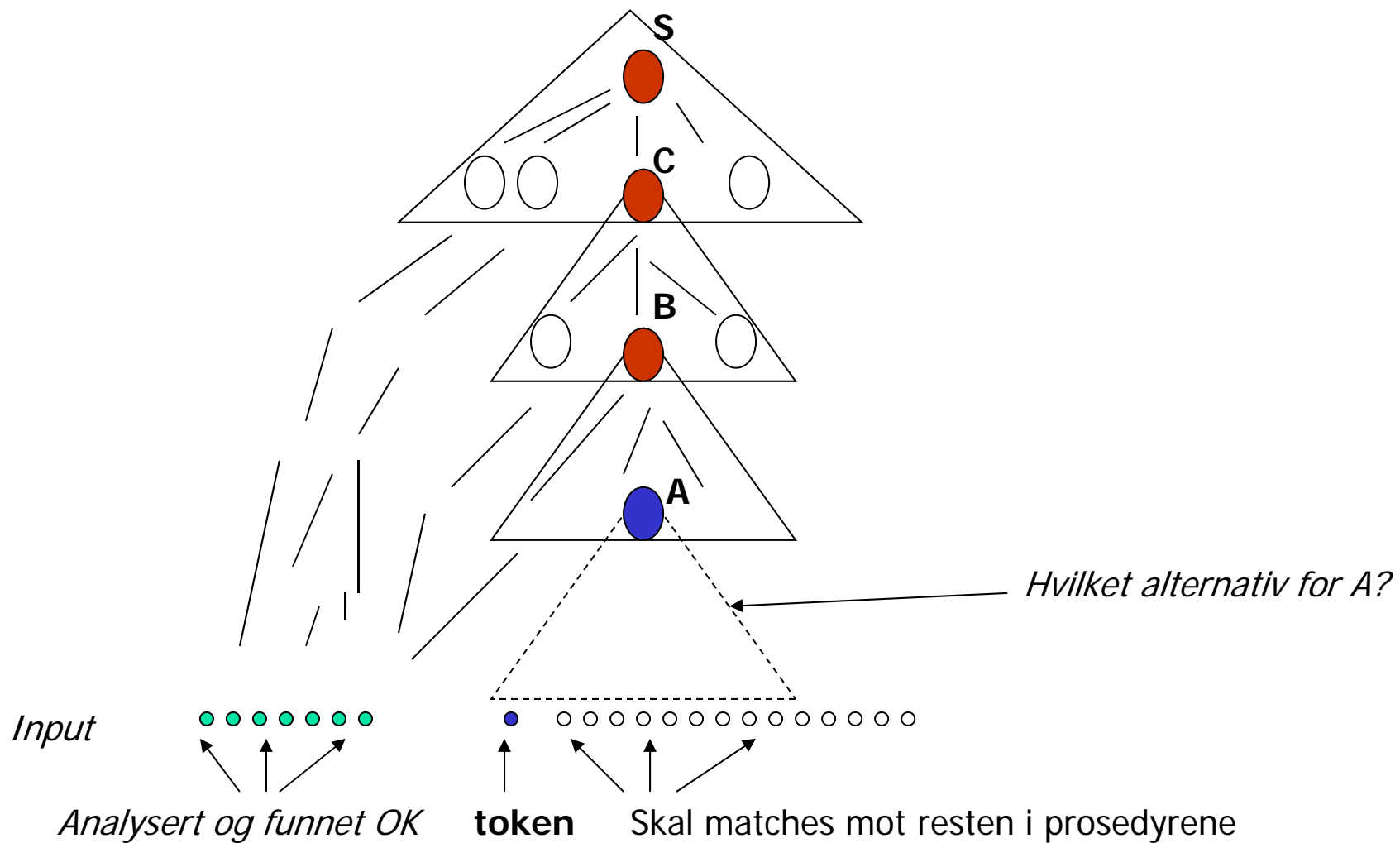


Metodene angitt under kan i rekkefølge ta "vanskeligere og vanskeligere" grammatikker. Hver «tilstand» er representert ved en rad i tabellen over.

- LR(0) Kan ta få grammatikker. Gir ca 300 "tilstander" for et vanlig programmeringsspråk. Skal se på denne som intro til SLR(1) og LALR(1).
- SLR(1) Kan ta de fleste vanlige grammatikker. Gir samme antall tilstander som LR(0)-metoden. Dette blir vår hovedmetode
- LALR(1) Kan ta litt flere grammatikker enn SLR(1). Gir samme antall tilstander som LR(0)-metoden. Denne skal vi bare se på prinsippene for.
- LR(1) Kan ta alle grammatikker det i det hele tatt er mulig å analysere ved å bare se ett symbol framover. Gir mange flere tilstander (ca 3000).

-Automatiserte verktøy som ut fra BNF-grammatikk leverer parserings-program:  
YACC, Bison, CUP ( alle bruker LALR(1)-teknikk)

# Til sammenlikning: "Top down"-parsering



# Datastrukturen for LR-parsering

$S' \rightarrow S$

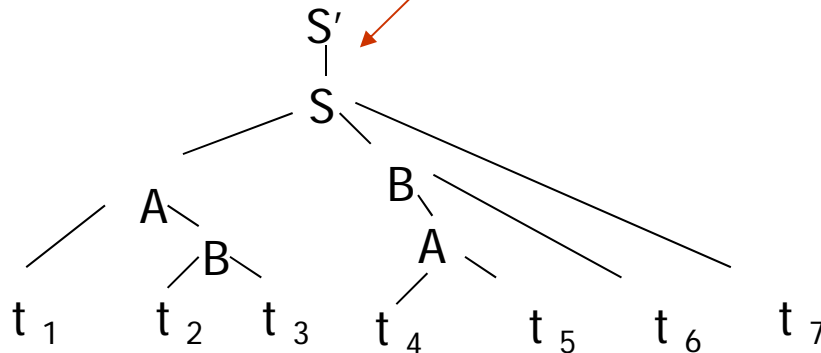
$S \rightarrow A B t_7 \mid \dots$

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

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

*Sett på ny "ytterste" produksjon. Derved vil startsymbolet  $S'$  aldri forkomme i en høyreside*

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



LR-parsering :

- Ha en "stakk" for *det som er lest*
- Gjør "reduksjonen" av et subtre når det ligger "på toppen av" stakken

# Mer om LR-parsering

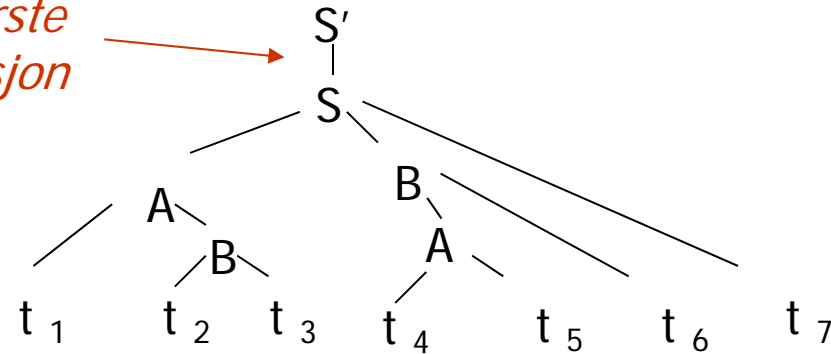
$S' \rightarrow S$

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

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

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

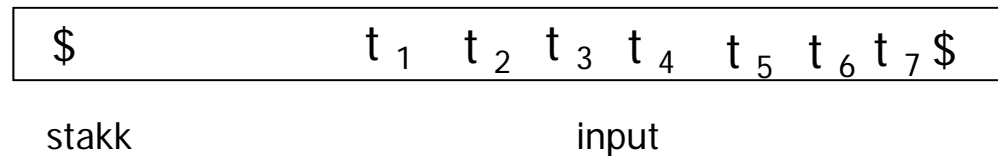
*Ny ytterste produksjon*



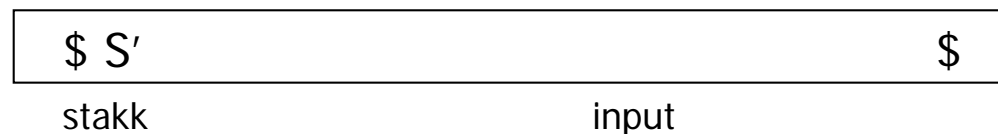
Anta at grammatikken er entydig, og anta (foreløpig!) at vi kjenner syntaks-treet for setningen:

- Ha en "stakk" for det som er lest (og er "reduert"!)
- Gjør "reduksjonen" av et subtre når subtreet ligger "på toppen av" stakken.
- Da erstatter vi dette med den ikke-terminalen som produserte dette subtreet.
- *En reduksjon*: En produksjon brukt "baklengs"

## Start-situasjonen:

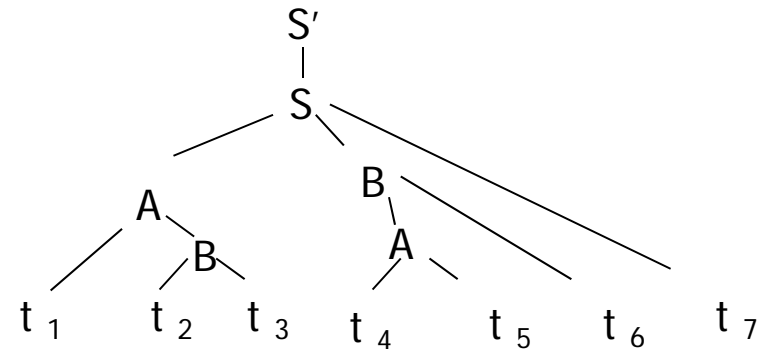


## Slutt-situasjonen:



Prinsippet for LR-parsing og operasjonene: **skift** og **redusér**

$S' \rightarrow S$   
 $S \rightarrow A B t_7 \mid \dots$   
 $A \rightarrow t_4 t_5 \mid t_1 B \mid$   
 $B \rightarrow t_2 t_3 \mid A t_6 \mid \dots$



stakk	input
\$	t <sub>1</sub> t <sub>2</sub> t <sub>3</sub> t <sub>4</sub> t <sub>5</sub> t <sub>6</sub> t <sub>7</sub> \$
\$ t <sub>1</sub>	t <sub>2</sub> t <sub>3</sub> t <sub>4</sub> t <sub>5</sub> t <sub>6</sub> t <sub>7</sub> \$
\$ t <sub>1</sub> t <sub>2</sub>	t <sub>3</sub> t <sub>4</sub> t <sub>5</sub> t <sub>6</sub> t <sub>7</sub> \$
\$ t <sub>1</sub> t <sub>2</sub> t <sub>3</sub>	t <sub>4</sub> t <sub>5</sub> t <sub>6</sub> t <sub>7</sub> \$
\$ t <sub>1</sub> B	t <sub>4</sub> t <sub>5</sub> t <sub>6</sub> t <sub>7</sub> \$
\$ A	t <sub>4</sub> t <sub>5</sub> t <sub>6</sub> t <sub>7</sub> \$
\$ A t <sub>4</sub>	t <sub>5</sub> t <sub>6</sub> t <sub>7</sub> \$
\$ A t <sub>4</sub> t <sub>5</sub>	t <sub>6</sub> t <sub>7</sub> \$
\$ A A	t <sub>6</sub> t <sub>7</sub> \$
\$ A A t <sub>6</sub>	t <sub>7</sub> \$
\$ A B	t <sub>7</sub> \$
\$ A B t <sub>7</sub>	\$
\$ S	\$

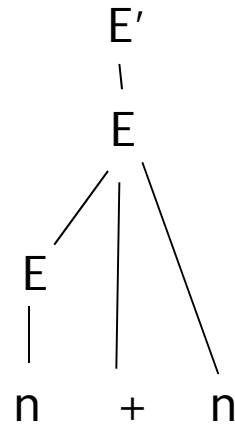
- Det blir to typer steg:
  - *Skift* ("les") neste input over til stakken
  - *Reduksjon* på toppen av stakken med bestemt regel  $A \rightarrow \alpha$
- Dersom man kjenner syntakstreet for den aktuelle setning, er det lett å angi de rette stegene.
- **MEN: Hvordan i all verden gjøre dette underveis uten å kjenne verken:**
  - resten av input
  - eller det fulle syntakstre

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

# Eksempel på LR-parsering (når vi kjenner treet!)

$$E' \rightarrow E$$

$$E \rightarrow E + n \mid n$$



Høyre-avledning (men håpløst eksempel på det!)

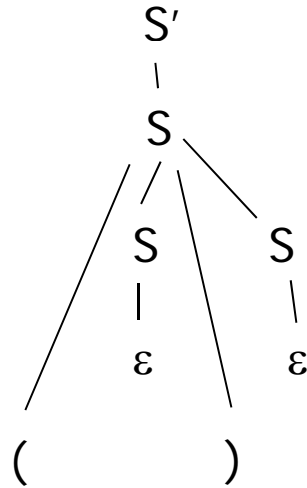
I boka, men vi stresser det ikke:

- Den neste reduksjonen som skal gjøres, kalles situasjonens **"handle"** (håndtak).
- "stakk + input" utgjør et stadium i en **høyre-avledning**, men de kommer i **omvendt rekkefølge** av en høyre-avledning

	Parsing stack	Input	Action
1	\$	<b>n + n</b> \$	↑ shift
2	\$ <b>n</b>	<b>+ n</b> \$	→ reduce $E \rightarrow n$
3	\$ <b>E</b>	<b>+ n</b> \$	↑ shift
4	\$ <b>E +</b>	<b>n</b> \$	↑ shift
5	\$ <b>E + n</b>	\$	reduce $E \rightarrow E + n$
6	\$ <b>E</b>	\$	reduce $E' \rightarrow E$
7	\$ <b>E'</b>	\$	accept

# Eksempel på LR-parsering

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

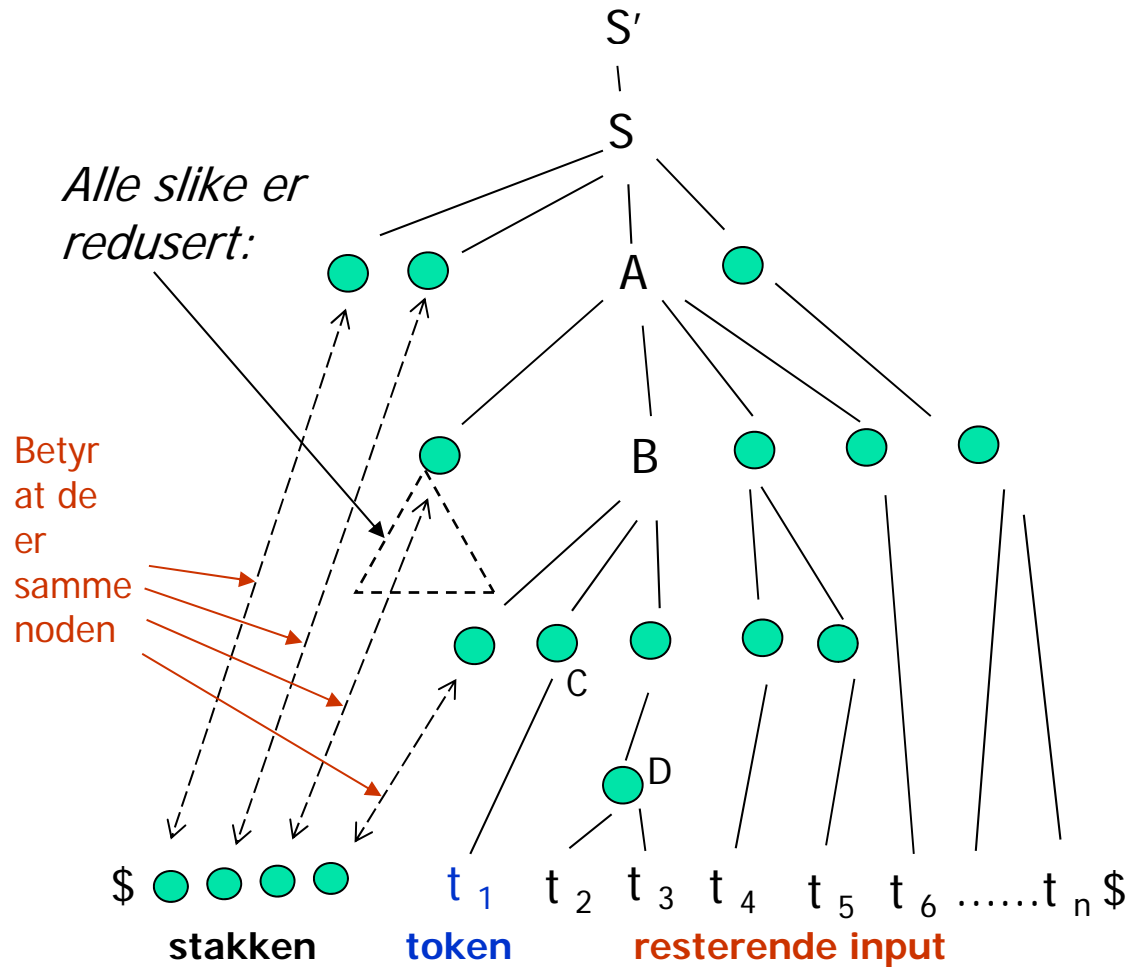


**NB:**  $S \rightarrow \varepsilon$  blir litt "rar":  
 Ved reduksjon med denne dukker en ikke terminal bare opp på enden av stakken, her og her.

	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



$S_1 S_2 S_3 S_4 \dots S_k$

lest input (stakken er redusert utgave av dette)

Det neste reduksjonen som blir gjort er reduksjon med:

$C \rightarrow t_1$

Deretter (etter noen skift-operasjoner) reduser med:

$D \rightarrow t_2 \ t_3$

Gitt en entydig grammatikk  $G$ .

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

(Ikke alt er beskrevet i boka, og bare det i boka er pensum)

- Vi ser på *alle* mulig stakker som kan opptre, og ser disse som strenger over alfabetet:  $\{\text{terminaler}\} \cup \{\text{ikke-terminaler}\}$ . Vi ser altså på språket (= streng-mengden):  
 $\{\text{String } s \mid s \text{ utgjør stakken på ett eller annet stadium i LR-parsering av en eller annen setning i } L(G)\}$
- Dette språket viser seg å være **regulært**, og kan beskrives av en NFA der **alle tilstander er aksepterende**:
  - Tilstandene angis av "itemer" av formen:  $A \rightarrow X Y \cdot Z$  (mer presist: "LR(0)-itemer")
  - Kantene (tilstandsovergangene) 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 DFA-en er mengder av NFA-tilstander, og vil her altså være **mengder av itemer**
- **LR-parseringens hovedsetning (uformelt, gjentas senere):**
  - Se på en situasjon under LR-parsering av en setning, der vi har strengen  $S$  på stakken.
  - Strengen  $S$  vil da aksepteres av DFA-en omtalt over. Anta at den bringer oss til DFA-tilstanden  $T$ .
  - Da vil mengden av *itemer* i  $T$  angi de mulige "lokale forhold" vi har ved det punktet vi nå er i i parseringen.
  - Selv om grammatikken er entydig er det ofte flere muligheter/valg (ett for hvert item). Dette kommer av at vi ennå ikke har sett på resten av input.



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

---

- For produksjonen :

$A \rightarrow X Y Z$

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

$A \rightarrow \cdot X Y Z$

$A \rightarrow X \cdot Y Z$

$A \rightarrow X Y \cdot Z$

$A \rightarrow X Y Z \cdot$

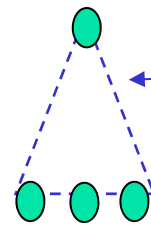
- Punktumet angir grovt sett hvor vi er hen i "dypeste aktive del" av parseringen i øyeblikket:
  - Det til venstre for punktum er lest fra input. Enten er det bare lest eller deler av inputen er redusert til ikke-terminaler på stakken.
  - Det til høyre for punktum har vi enda ikke sett/lest.
  - Det første symbolet i den videre input ligger i variabelen «token»

# Kantene i NFA-en

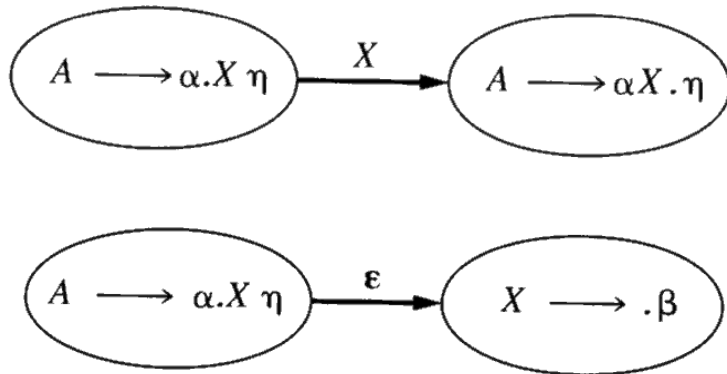
Dette er ikke fullt forklart i boka, og er ikke pensum

*Skal lage en NFA som aksepterer alle slike "linjer", og intet annet*

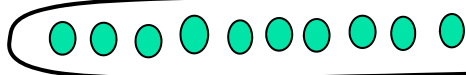
Gitt en grammatikk  $G$  og en situasjon under passering av en setning  $s$  i  $L(G)$ .



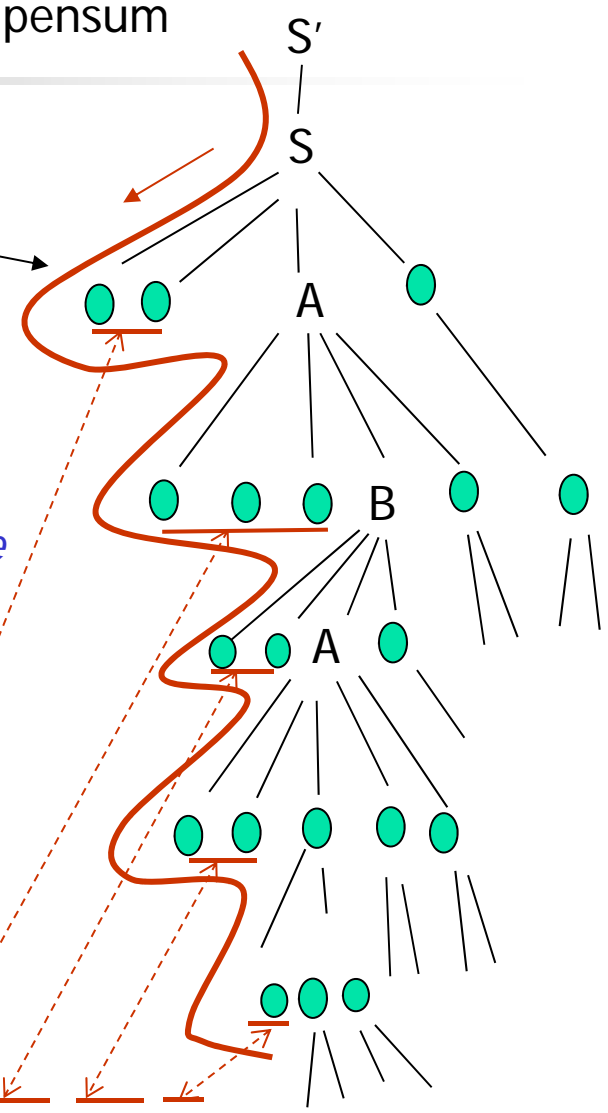
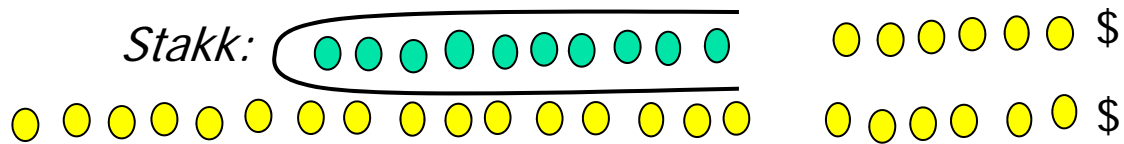
*Husk: Ingen slik på venstre side. De er redusert!*



Stakk:

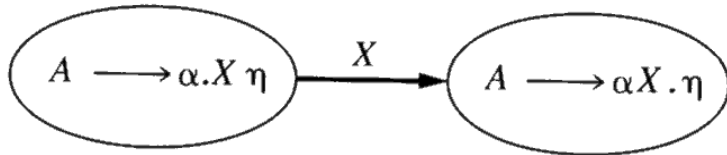


Opprinnelig input:

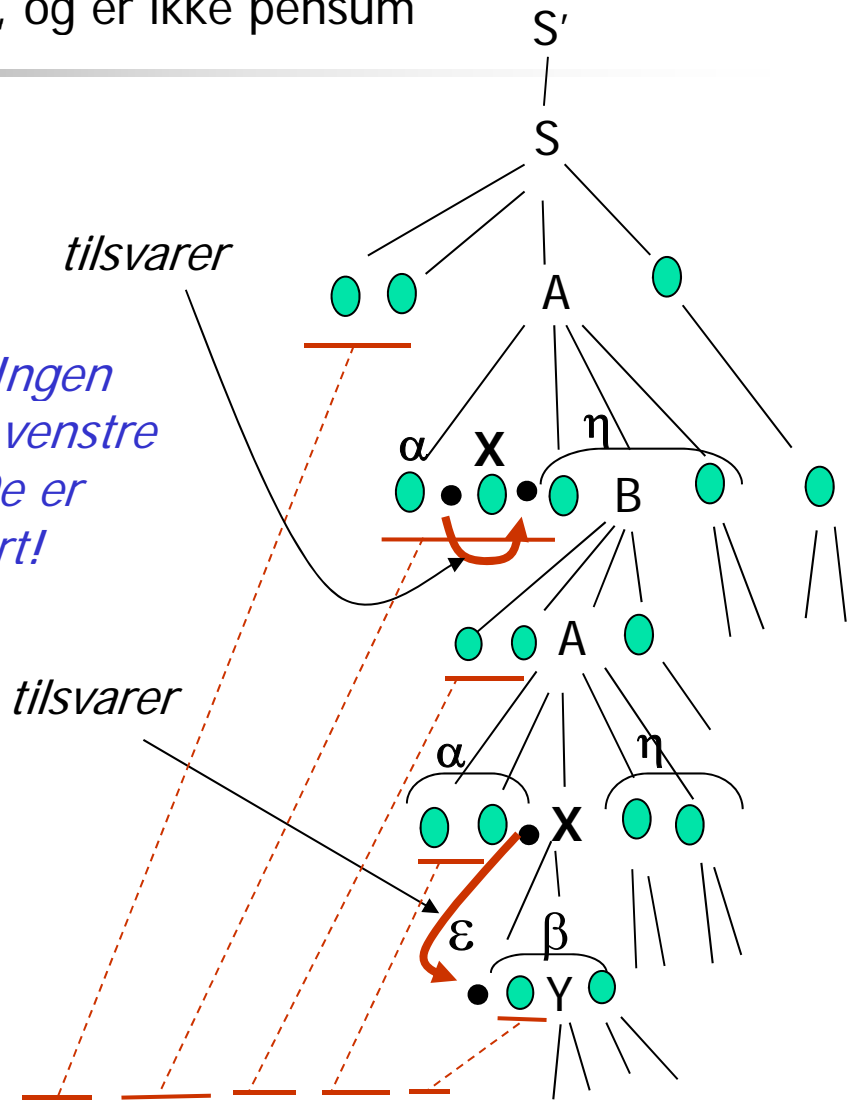
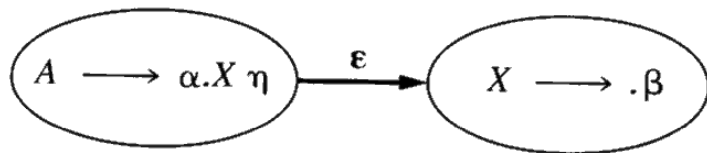


# Mer: Kantene i NFA-en

Dette er ikke fullt forklart i boka, og er ikke pensum



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



Må vise:

- (1) For enhver situasjon vil juletre-kanten (=stakken) kunne fremstilles av NFA-en.
- (2) For enhver vei gjennom NFA-en vil vi kunne lage en situasjon med denne kanten (=stakken)

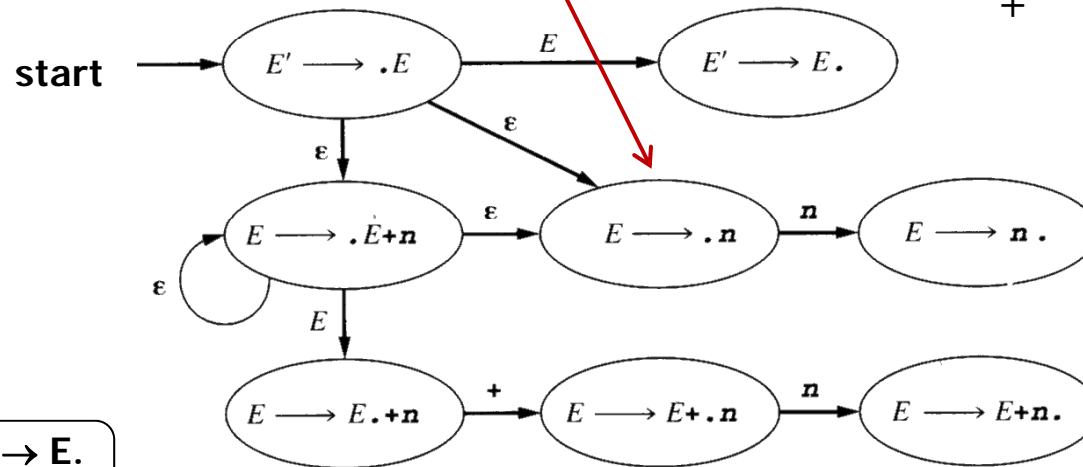
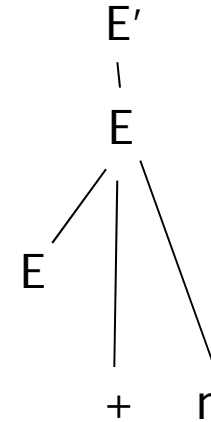


# NFA'en som beskriver alle mulige stakker

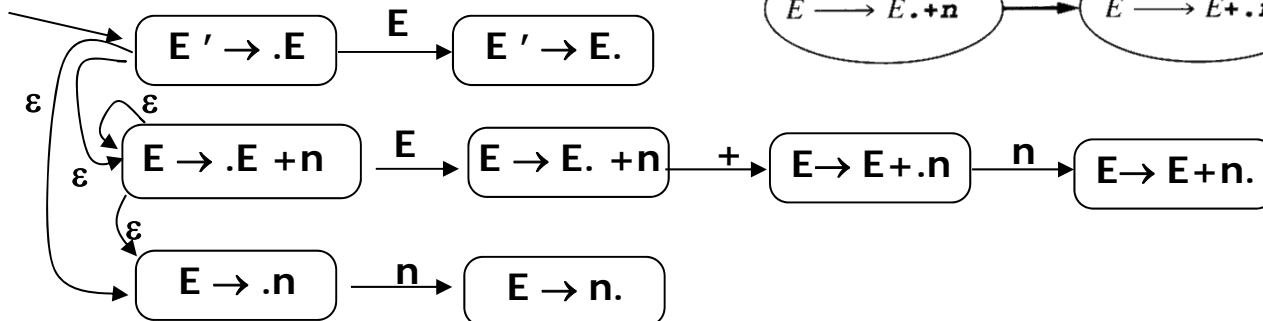
$E' \rightarrow E$   
 $E \rightarrow E + n$   
 $E \rightarrow n$

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

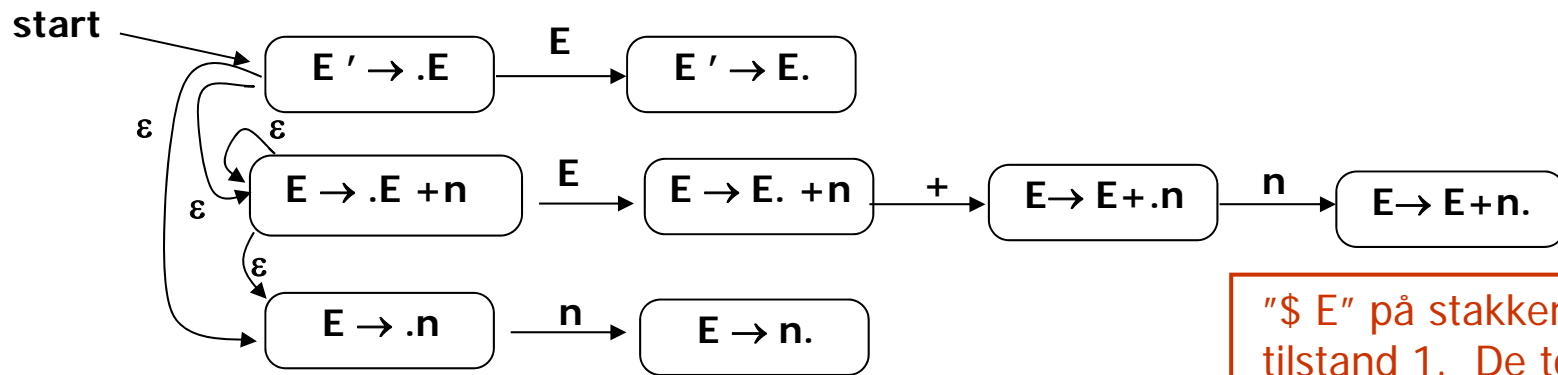
*Itemer (LR(0)-itemer)*



Den samme, i litt ryddigere utgave:



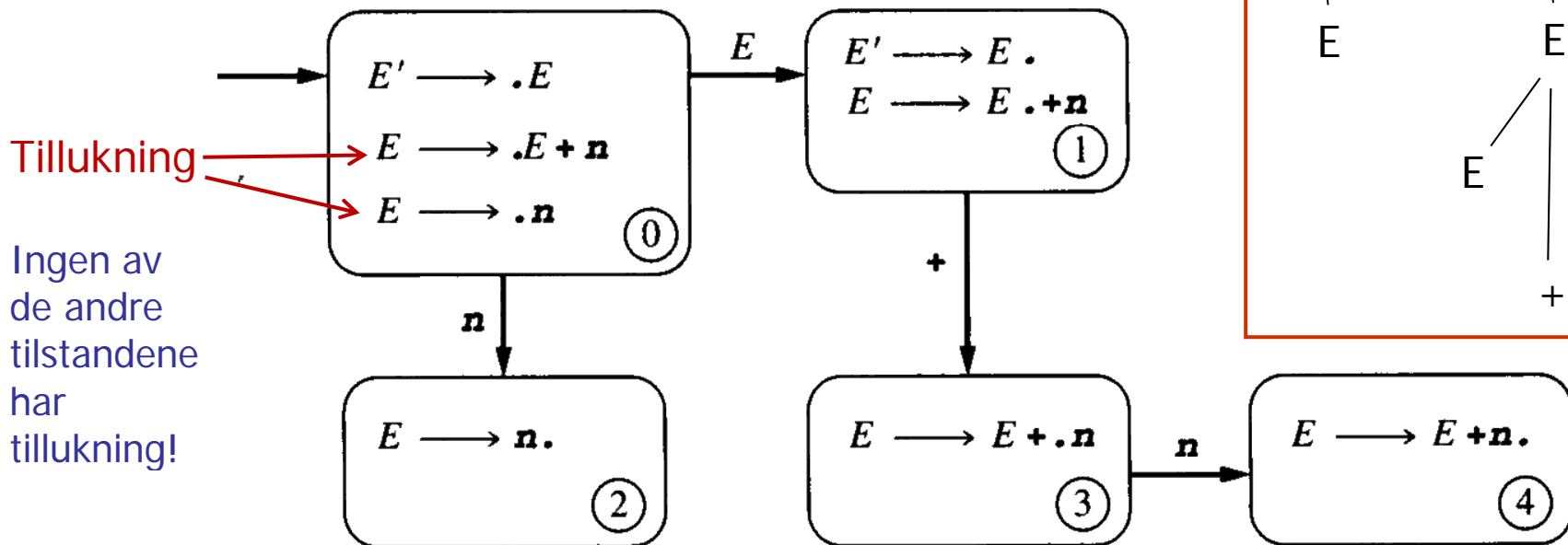
LR(0)-NFA (Altså litt mer systematisk enn i boka):



"\$ E" på stakken gir tilstand 1. De to mulighetene kan da eksemplifiseres ved:

The first parse tree shows  $E'$  deriving  $E$ . The second parse tree shows  $E'$  deriving  $E$ , which then derives  $E + n$ .

LR(0)-DFA, laget ved subset-konstruksjonen (kap 2):



# Hvordan sette opp LR(0)-DFA'en direkte fra grammatikken? (Rett fram bruk av subset-konstruksjonen. Bør trenes på!)

- Tillukning av item-mengde I:

- Dersom:

- $A \rightarrow \alpha \cdot B \gamma$  er med i I
- B er en ikke-terminal
- $B \rightarrow \beta_1 \mid \beta_2 \mid \dots$

Da er også Itemene:

- $B \rightarrow \cdot \beta_1$
- $B \rightarrow \cdot \beta_2$
- ...

med i I

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

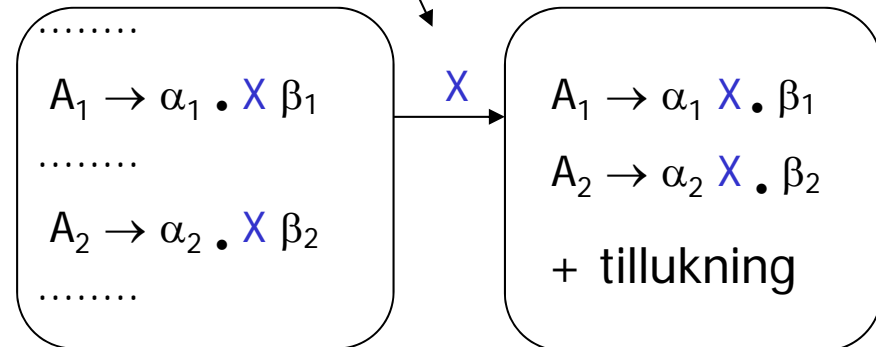
$S' \rightarrow \cdot S$

+ tillukning

- Tilstandsovergang ved symbol X fra tilstand I

- X er terminal eller ikke-terminal (behandles likt!)

Tilstand I:



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



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

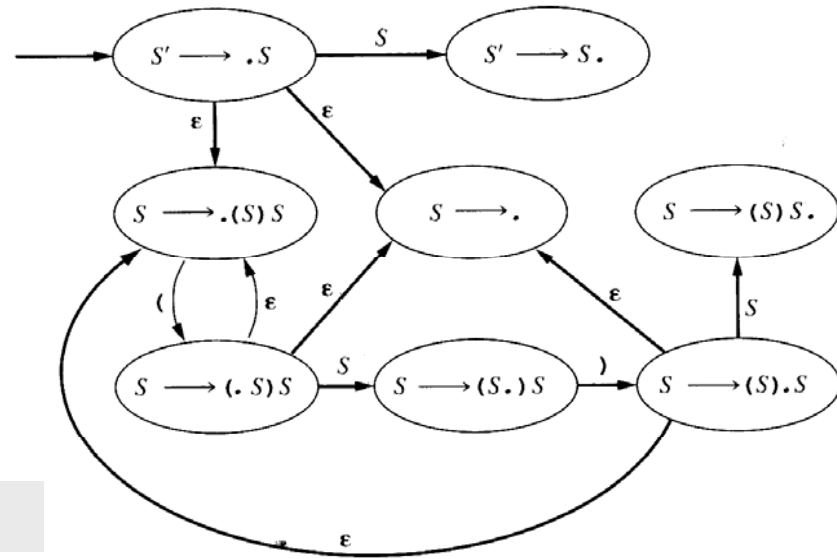
$S' \rightarrow S$   
 $S \rightarrow ( S ) S \mid \epsilon$

Merk at  $S \rightarrow \epsilon$  bare gir ett item, nemlig:  $S \rightarrow \bullet$

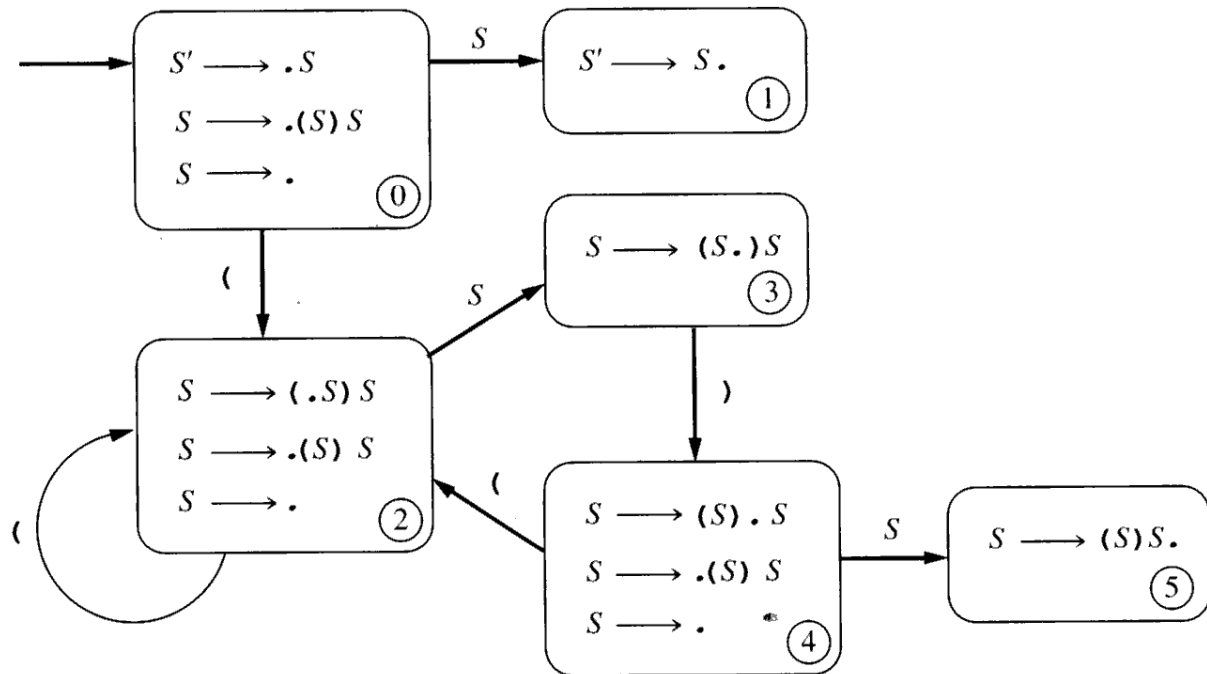
(Altså ikke:  ~~$S \rightarrow \bullet \epsilon$~~   
 og  ~~$S \rightarrow \epsilon \bullet$~~ )

- $S' \rightarrow \bullet S$
- $S' \rightarrow S \bullet$
- $S \rightarrow \bullet ( S ) S$
- $S \rightarrow ( \bullet S ) S$
- $S \rightarrow ( S \bullet ) S$
- $S \rightarrow ( S ) \bullet S$
- $S \rightarrow ( S ) S \bullet$
- $S \rightarrow \bullet$

## LR(0) – NFA:



## LR(0) – DFA:



# Hva sier "topp-tilstanden"?

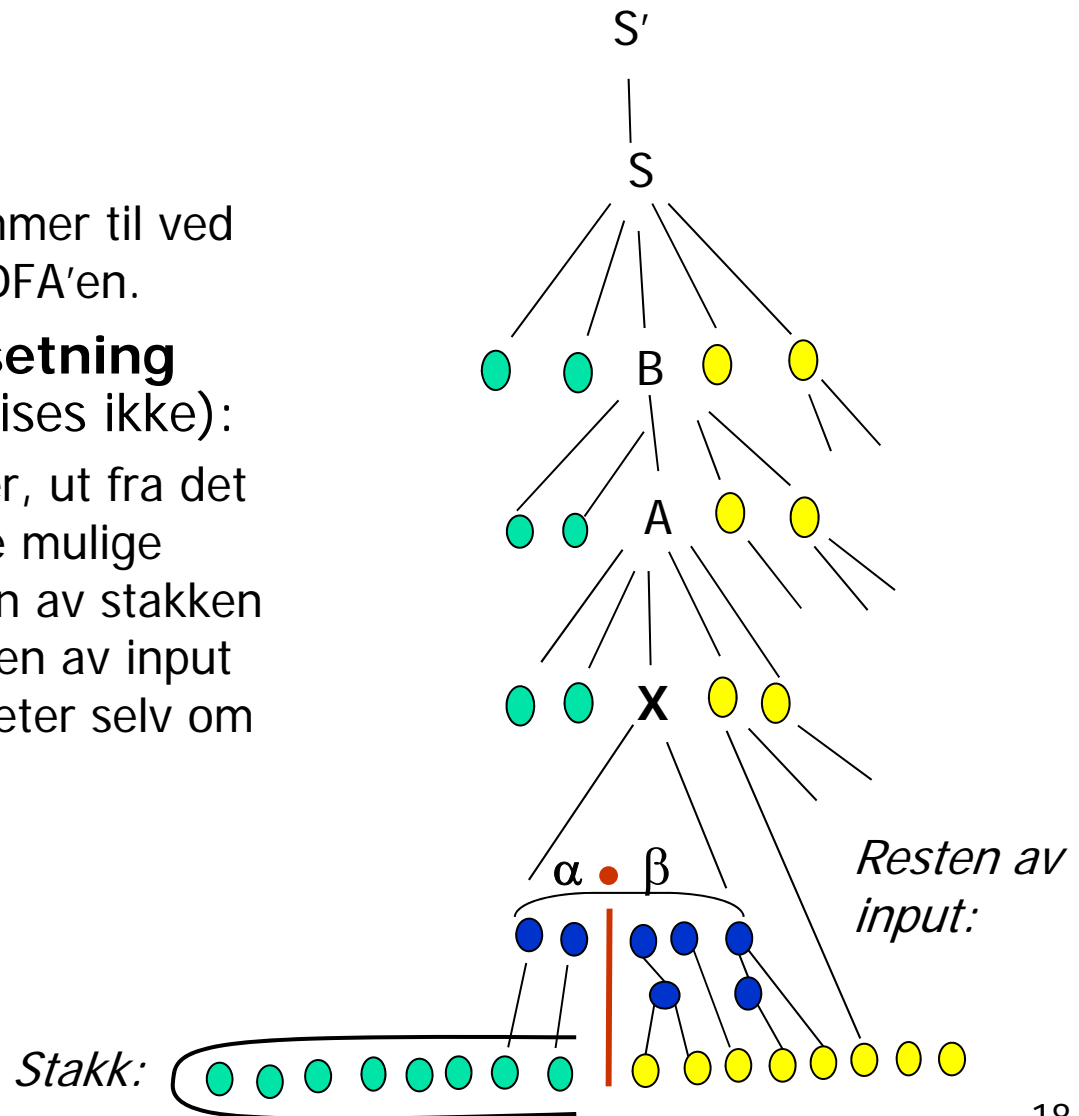
- **Topp-tilstanden:**

- Den DFA-tilstanden vi kommer til ved å la stakken gå gjennom DFA'en.

- **LR-parseringens hovedsetning**  
(En gang til, litt løselig, bevises ikke):

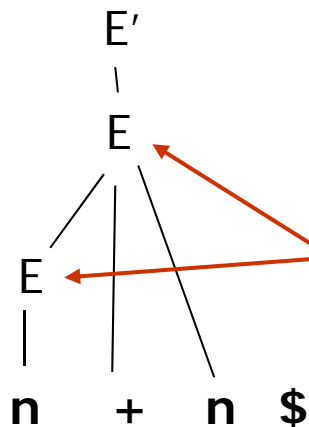
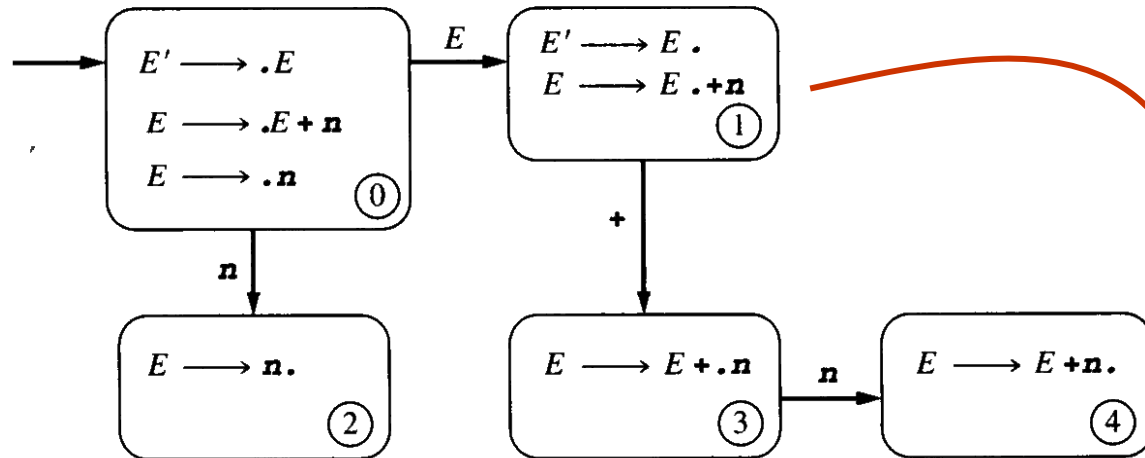
- Itemene i topp-tilstanden er, ut fra det vi har lest av input, *alle* de mulige "lokale forhold" ved toppen av stakken (Siden vi ikke har lest resten av input kan det være flere muligheter selv om grammatikken er entydig)

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



# Hva sier topp-tilstanden? Eksempel:

$E' \rightarrow E$   
 $E \rightarrow E + n \mid n$



$\$$	$E$	$n + n \$$
$\$ n$	$E +$	$+ n \$$
$\$ E$	$E + n$	$+ n \$$
$\$ E +$	$E$	$n \$$
$\$ E + n$	$E$	$\$$
$\$ E$	$E'$	$\$$
$\$ E'$	$\$$	$\$$

Skal skifte

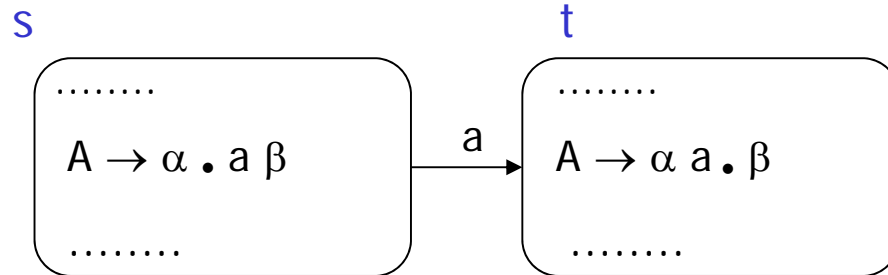
Skal redusere med  $E' \rightarrow E$

LR(0)-grammatikk: Topptilstanden bestemmer alltid entydig neste steg.

Derved: Denne grammatikken er ikke LR(0). For stakk "E" er neste skritt ubestemt. 19

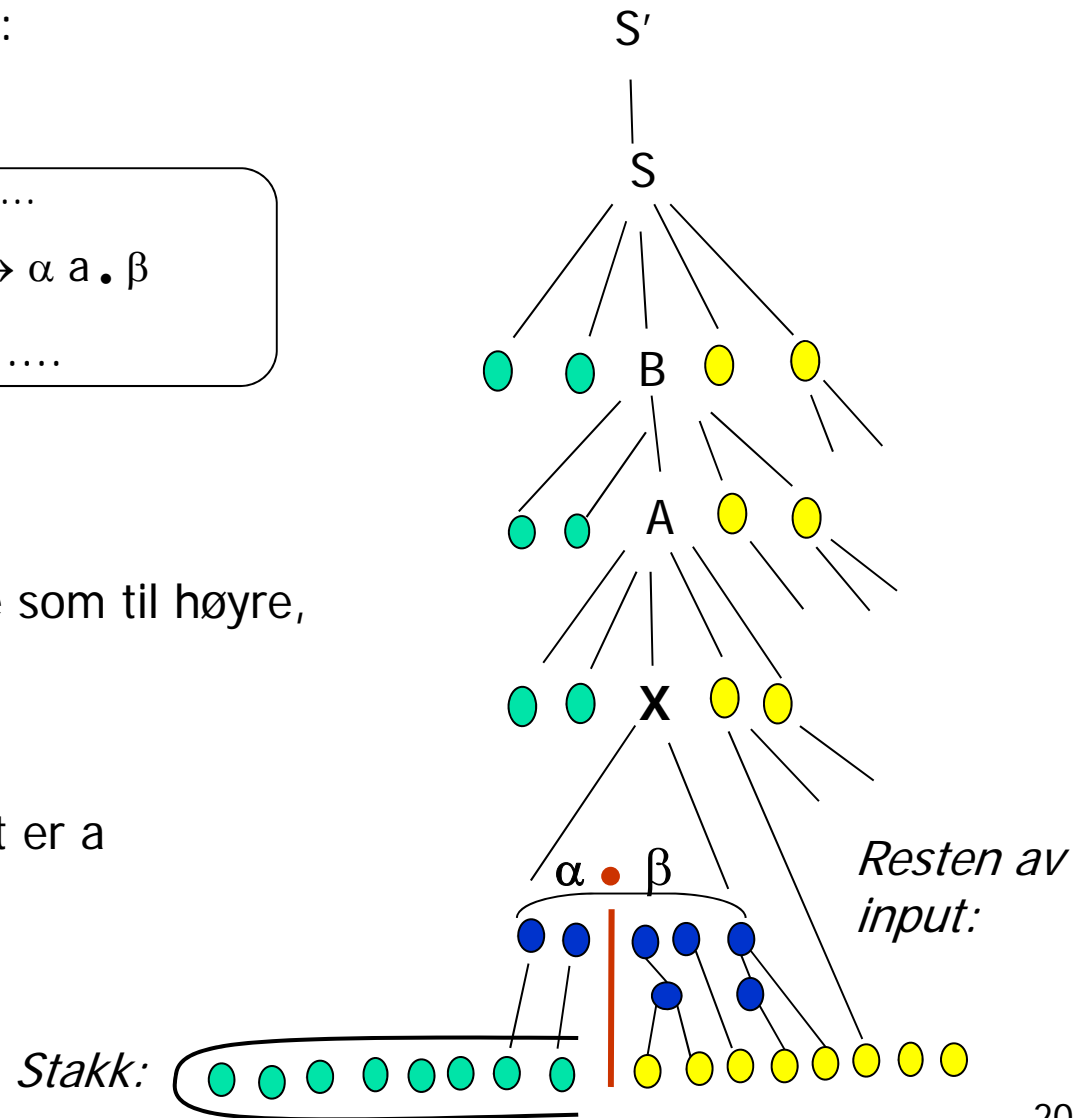
# Når er "skift" en mulighet?

Anta vi er i tilstand  $s$  under:



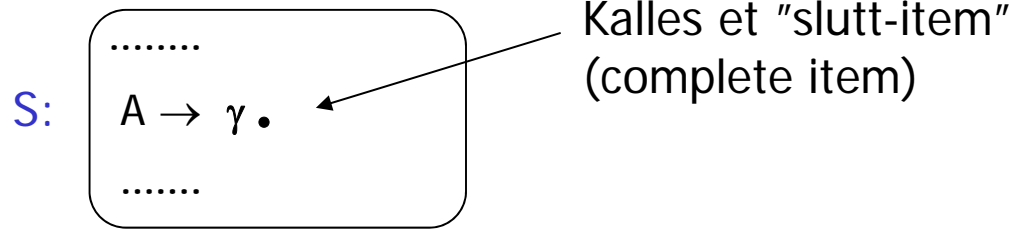
Angir at situasjonen *kan* være som til høyre, og derved:

- Neste skritt kan være skift
- Skift er lovlig om neste input er  $a$
- Ny topptilstand blir  $t$  i så fall



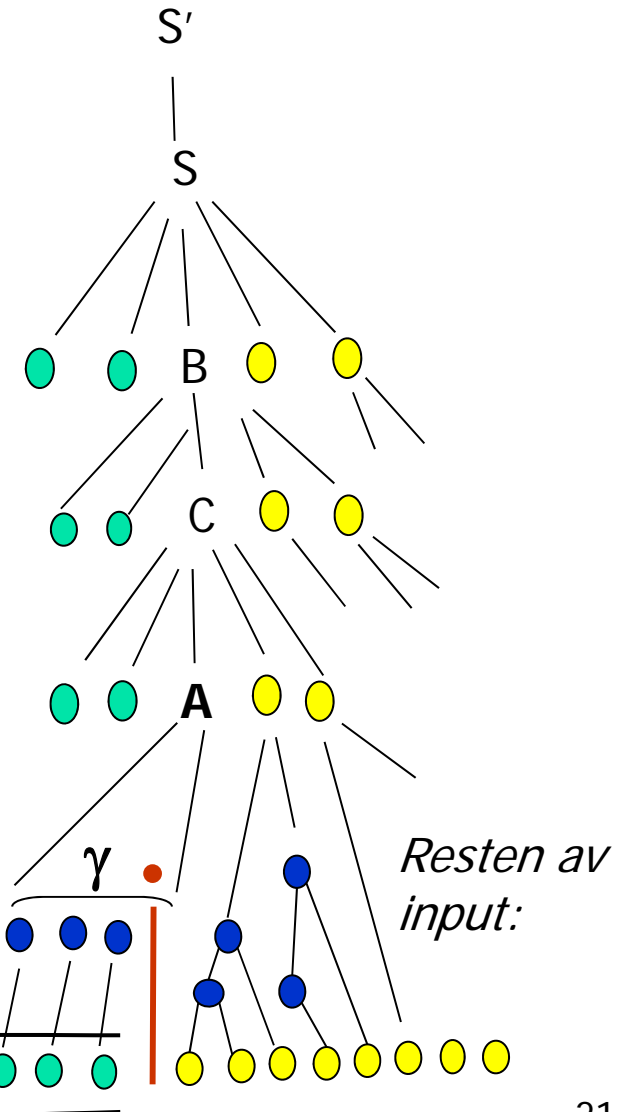
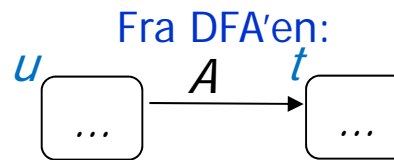
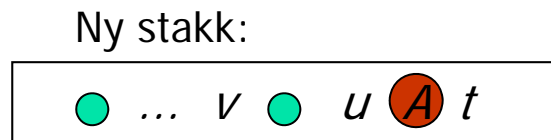
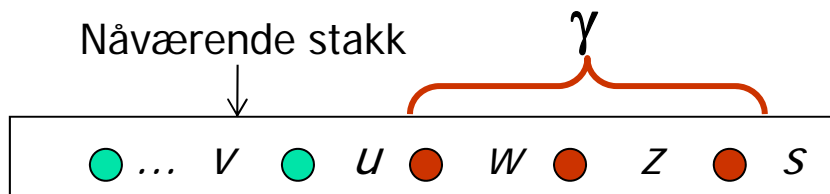
# Når er reduksjon (med en gitt produksjon) en mulighet?

Anta at topptilstanden er  $s$ , og at den er slik:



Angir at situasjonen *kan* være som til høyre, og at neste skritt derved *kan* være reduksjon med  $A \rightarrow \gamma$

**NB:** Vi holder tilstandene mellom stakk-symbolene



**Reduser-steget:** Pop av det som tilsvarer  $\gamma$  (og mellomliggende tilstander), og push på  $A$ , og finn riktig ny topptilstand  $t$  fra  $u$  og  $A$

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

Ser på tre grammatikker:

$$A \rightarrow ( A ) \mid a$$

$$S' \rightarrow S$$

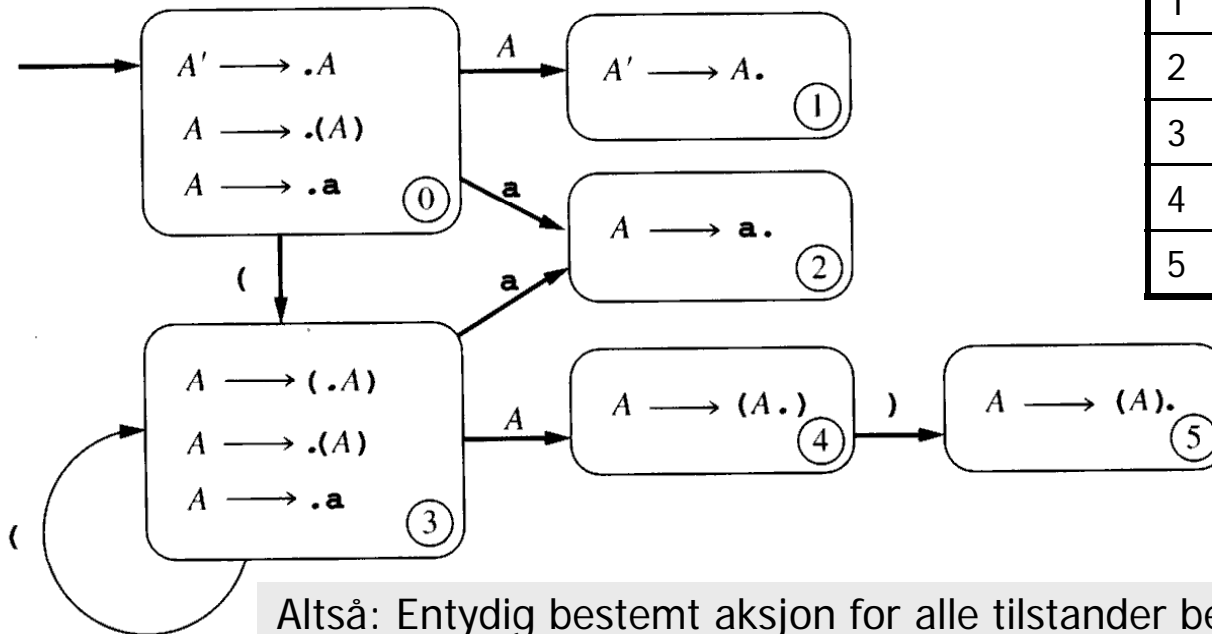
$$S \rightarrow ( S ) S \mid \varepsilon$$

$$E' \rightarrow E$$

$$E \rightarrow E + n \mid n$$

Har vi sett på:  
Er ikke LR(0)

Den første gir følgende LR(0) – DFA:



Tilst.	Mulig aksjoner:
0	Bare skift mulig, for "(" og "a"
1	Bare red. mulig, med $A' \rightarrow A$
2	Bare red. mulig, med $A \rightarrow a$
3	Bare skift mulig, for "(" og "a"
4	Bare skift mulig, for ")"
5	Bare red. mulig, med $A \rightarrow (A)$

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

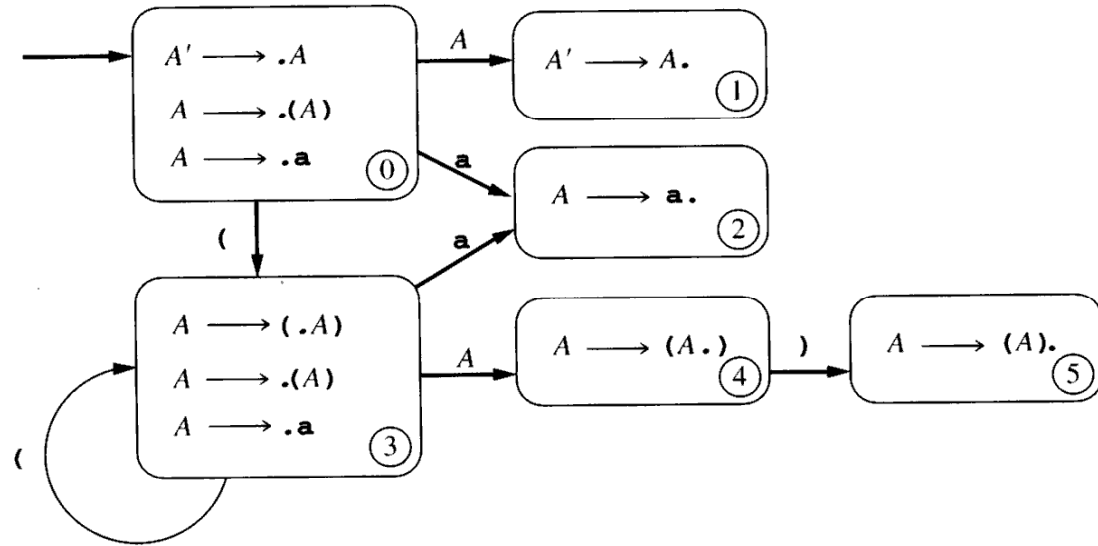
**MERK:** Der det er reduksjon må det ikke være tvil om med hvilken produksjon!

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

NB: Den er litt annerledes enn for SLR(1), LALR(1), LR(1), som alle er like

Grammatikk:

$A \rightarrow ( A ) \mid a$



State	Action	Rule	Input			Goto
			(	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	$A \rightarrow ( A )$				

# Parsering av en setning i en LR(0) – grammatikk

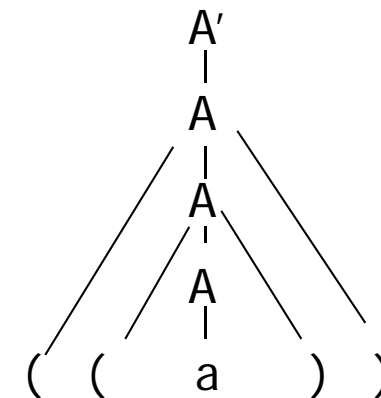
Setning: ((a))

State	Action	Rule	Input			Goto
			(	a	)	
0	shift		3	2		1
1	reduce	$A' \rightarrow A$				4
2	reduce	$A \rightarrow a$			5	
3	shift		3	2		1
4	shift				5	4
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  $A' \rightarrow A$ , og input tom: Ferdig



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

State	Action	Rule	Input			Goto
			(	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	) \$

Viktig: Skifter  
aldri noe ulovlig  
inn på stakken!

# En (litt ruskete) formulering av LR(0)-kravet: Dersom følgende 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  $A \rightarrow \alpha.X\beta$ , 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  $A \rightarrow \alpha.X\beta$ , then the new state to be pushed on the stack is  $t$ , where  $s \xrightarrow{X} t$  ~~the state containing the item  $A \rightarrow \alpha.X\beta$ .~~ If this token is not  $X$  for some item in state  $s$  of the form just described, an error is declared.
2. If state  $s$  contains any complete item (an item of the form  $A \rightarrow \gamma$ ), then the action is to reduce by the rule  $A \rightarrow \gamma$ . A reduction by the rule  $S' \rightarrow S$ , 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  $u$  uncovered by the removal of  $\gamma$ ). Again, by the construction of the DFA, this state  $u$  must contain an item of the form  $B \rightarrow \alpha.A\beta$ . Push  $A$  onto the stack, and push (as the new state) the state  $t$ , where  $u \xrightarrow{A} t$  ~~containing the item  $B \rightarrow \alpha.A\beta$ .~~ (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.)

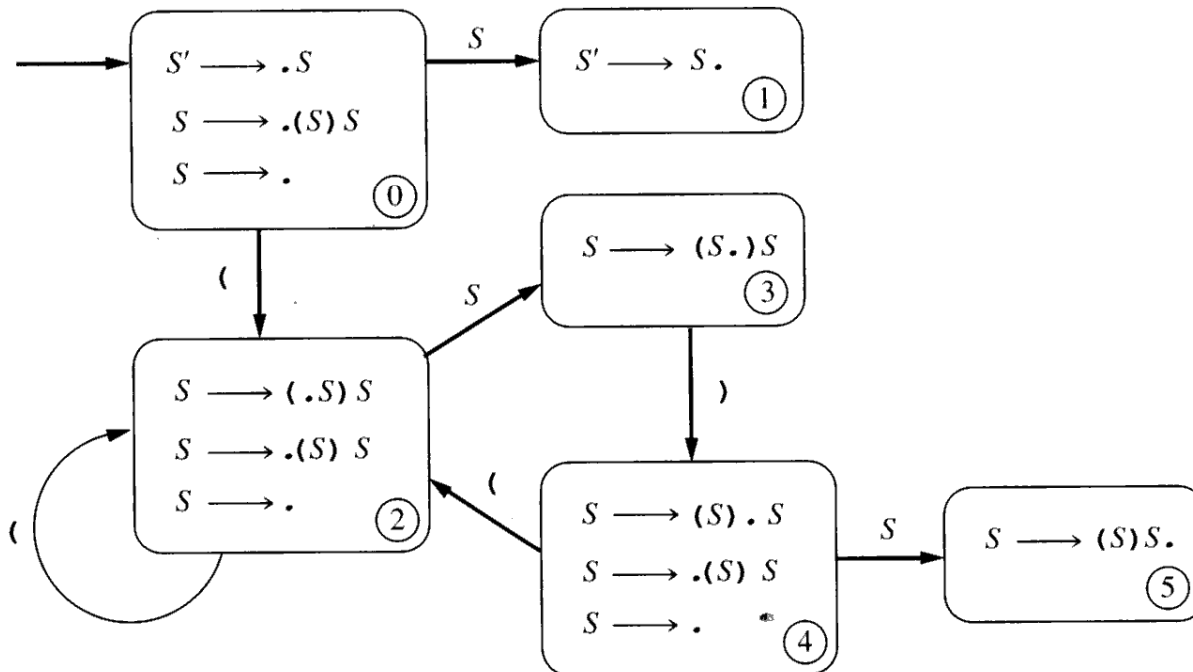
Kan forekomme i flere tilstander

Avslutning

Er denne grammatikken LR(0)?  
 Nei, pga. tilstandene 0, 2 og 4

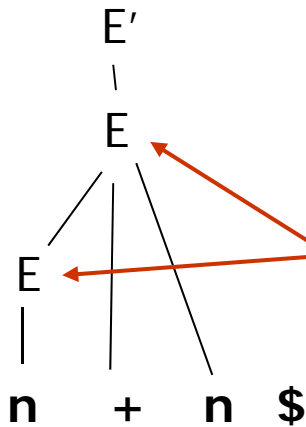
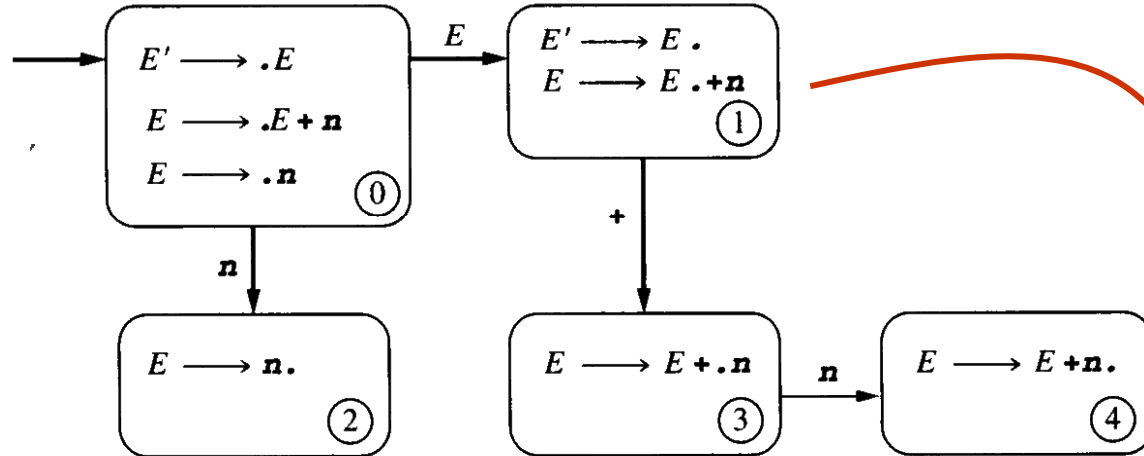
$S' \rightarrow S$   
 $S \rightarrow ( S ) S \mid \varepsilon$

LR(0) – DFA:



Er denne LR(0)? Nei, pga. tilstand 1:  
(har vi sett på tidligere)

$E' \rightarrow E$   
 $E \rightarrow E + n \mid n$



\$		$n + n$	\$
\$	$n$	$+ n$	\$
\$	$E$	$+ n$	\$
\$	$E +$	$n$	\$
\$	$E + n$		\$
\$	$E$		\$
\$	$E'$		\$

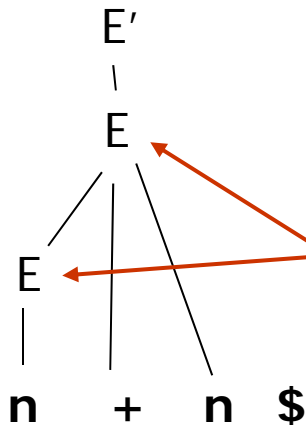
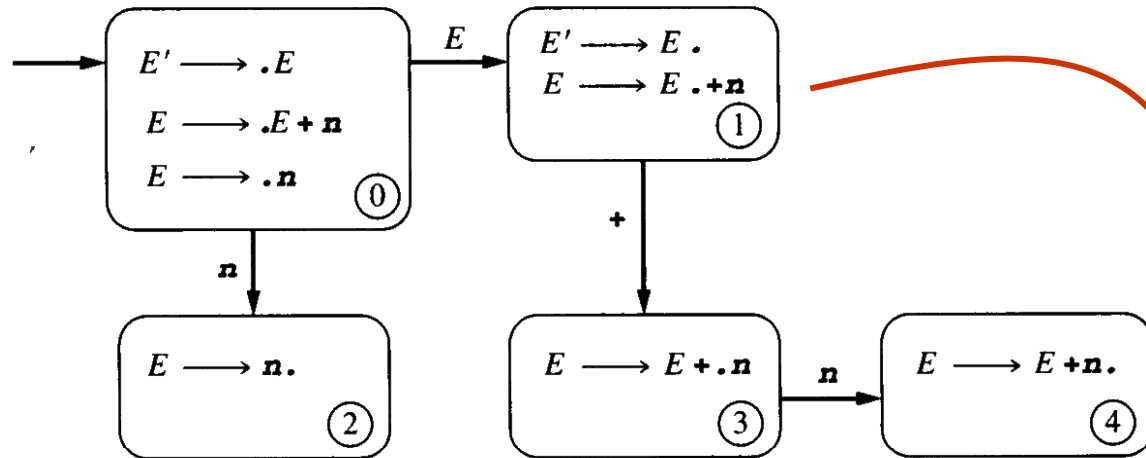
Annotations: "Skal skifte" (red) points to the transition from  $E$  to  $E +$ ; "Skal redusere med  $E' \rightarrow E$ " (red) points to the transition from  $E$  to  $E'$ .

*Derved: Denne grammatikken er ikke LR(0): For stakk "E" er neste skritt ubestemt* 28

# Er den LR(0)? Nei, pga. tilstand 1!

Men hvordan avgjøre hva man skal gjøre i tilstand 1??

$E' \rightarrow E$   
 $E \rightarrow E + n \mid n$



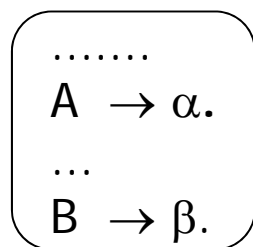
\$		$n + n$	\$
\$	$n$	$+ n$	\$
\$	$E$	$+ n$	\$
\$	$E +$	$n$	\$
\$	$E + n$		\$
\$	$E$		\$
\$	$E'$		\$

Annotations: "Skal skifte" (red) points to the transition from  $E$  to  $E +$ . "Skal redusere med  $E' \rightarrow E$ " (red) points to the transition from  $E$  to  $E'$ .

**Løsning:** Vi ser på neste input-symbol! Og den ligger i token-variablen

# 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 *nå* utgangspunkt i LR(0)-DFA'en
- Tabell-oppsettet er nesten likt, men nå må "reduser-linjene" fra LR(0)-tabellene, spesifiseres for hvert mulig «neste input-symbol» (= «token»)

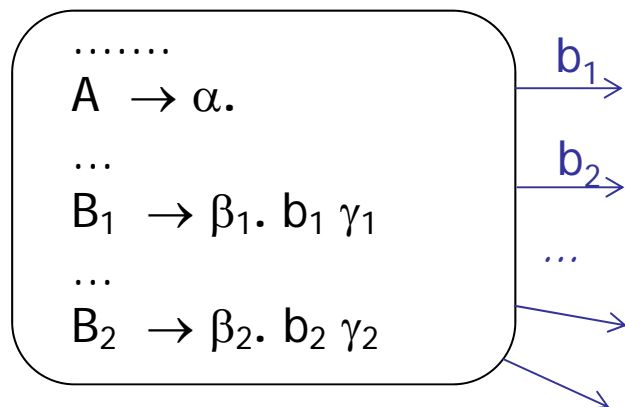


LR(0): Har her en (uløselig) red./red.-konflikt

SLR(1): Dersom:  $\text{Follow}(A) \cap \text{Follow}(B) = \emptyset$  så kan konflikten løses ved å se på neste input:

Om  $\text{token} \in \text{Follow}(A)$  reduser med  $A \rightarrow \alpha$

Om  $\text{token} \in \text{Follow}(B)$  reduser med  $B \rightarrow \beta$   
(ellers feil)



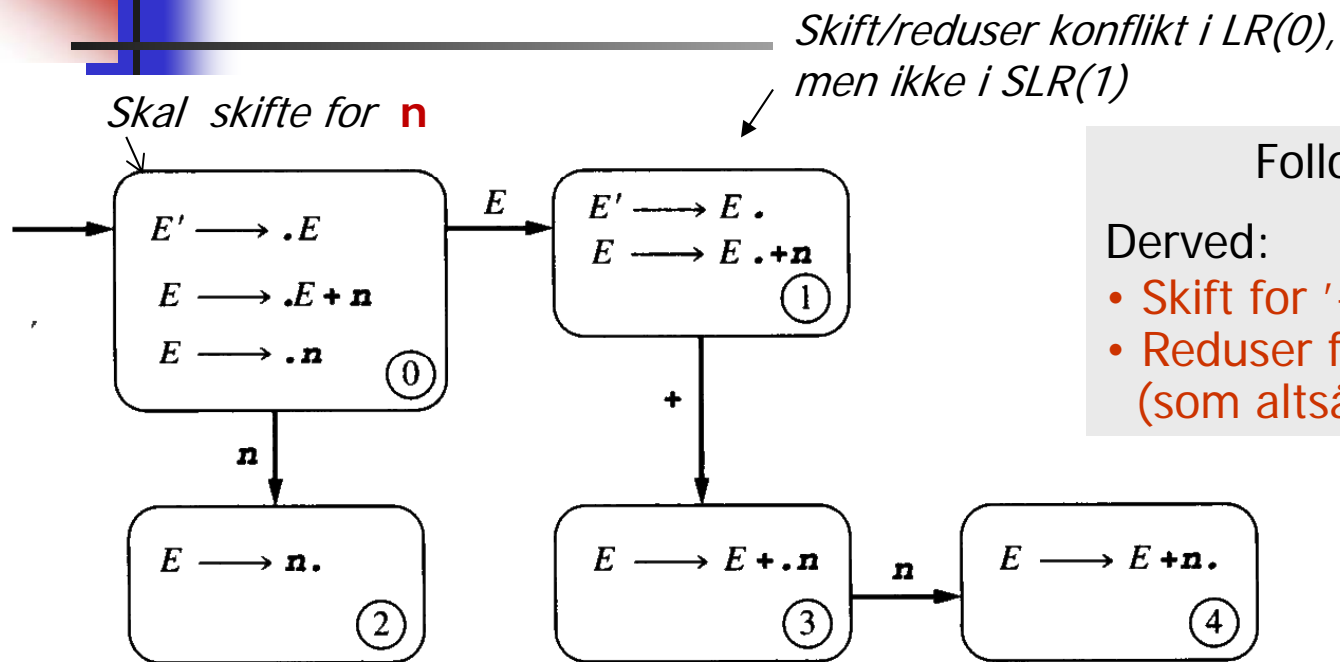
LR(0): Har her en (uløselig) skift/red.-konflikt

SLR(1): Dersom:  $\text{Follow}(A) \cap \{b_1, b_2, \dots\} = \emptyset$  så kan konflikten løses ved å se på neste input:

Om  $\text{token} \in \text{Follow}(A)$ : reduser med  $A \rightarrow \alpha$

For  $\text{token} = b_1, b_2, \dots$ : skift (input avgjør ny tilstand)  
(ellers feil)

# Er denne grammatikken SLR(1)?



Follow(E') = { \$ }

Derived:

- Skift for '+'
- Reduser for '\$', med  $E' \rightarrow E$  (som altså er accept)

## SLR(1)-kravet slik det er formulert i boka:

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( $A$ )  $\cap$  Follow( $B$ ) is empty.

Ville ellers ha skift /  
reduser -konflikt ved  
input  $X$

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

"Complete item" = Har punktumet til slutt



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

---

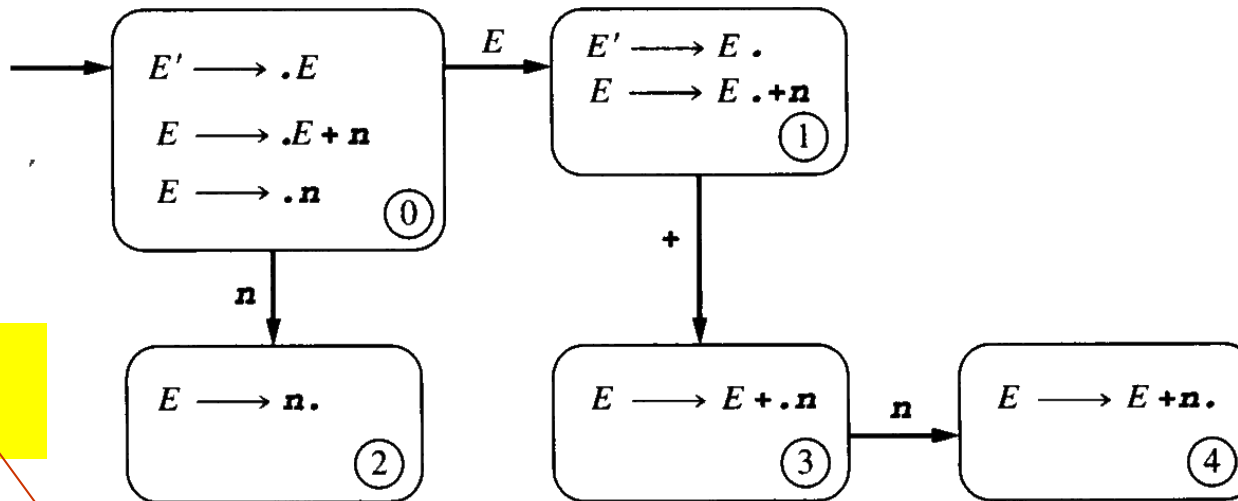
**The SLR(1) parsing algorithm.** Let  $s$  be the current state (at the top of the parsing stack). Then actions are defined as follows:

1. If state  $s$  contains any item of the form  $A \rightarrow \alpha.X\beta$ , 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  $A \rightarrow \alpha.X\beta$~~   $t$ , where  $s \xrightarrow{X} t$
2. If state  $s$  contains the complete item  $A \rightarrow \gamma.$ , and the next token in the input string is in  $\text{Follow}(A)$ , then the action is to reduce by the rule  $A \rightarrow \gamma$ . A reduction by the rule  $S' \rightarrow S$ , where  $S$  is the start state, is equivalent to acceptance; this will happen only if the next input token is  $\$$ .<sup>4</sup> 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 ~~containing the item  $B \rightarrow \alpha.A\beta$~~   $u$  must contain an item of the form  $B \rightarrow \alpha.A\beta$ . Push  $A$  onto the stack, and push the state ~~containing the item  $B \rightarrow \alpha.A\beta$~~   $t$ , 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



SLR(1): Både skift og reduser kan være på sammen linje. ("Accept" er egentlig reduksjon med  $A' \rightarrow A$ )

'n' ikke i Follow(E)

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

SLR(1)-kravet på en annen måte: Denne tabellen må være entydig!

# Parsering for SLR(1)-grammatikk

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

Kan også se på gale setninger som:

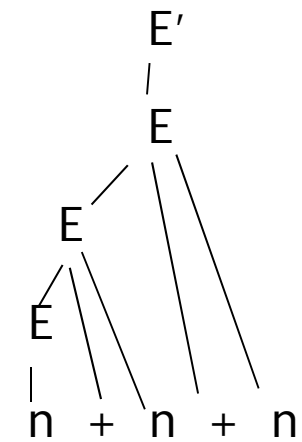
+ n \$

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



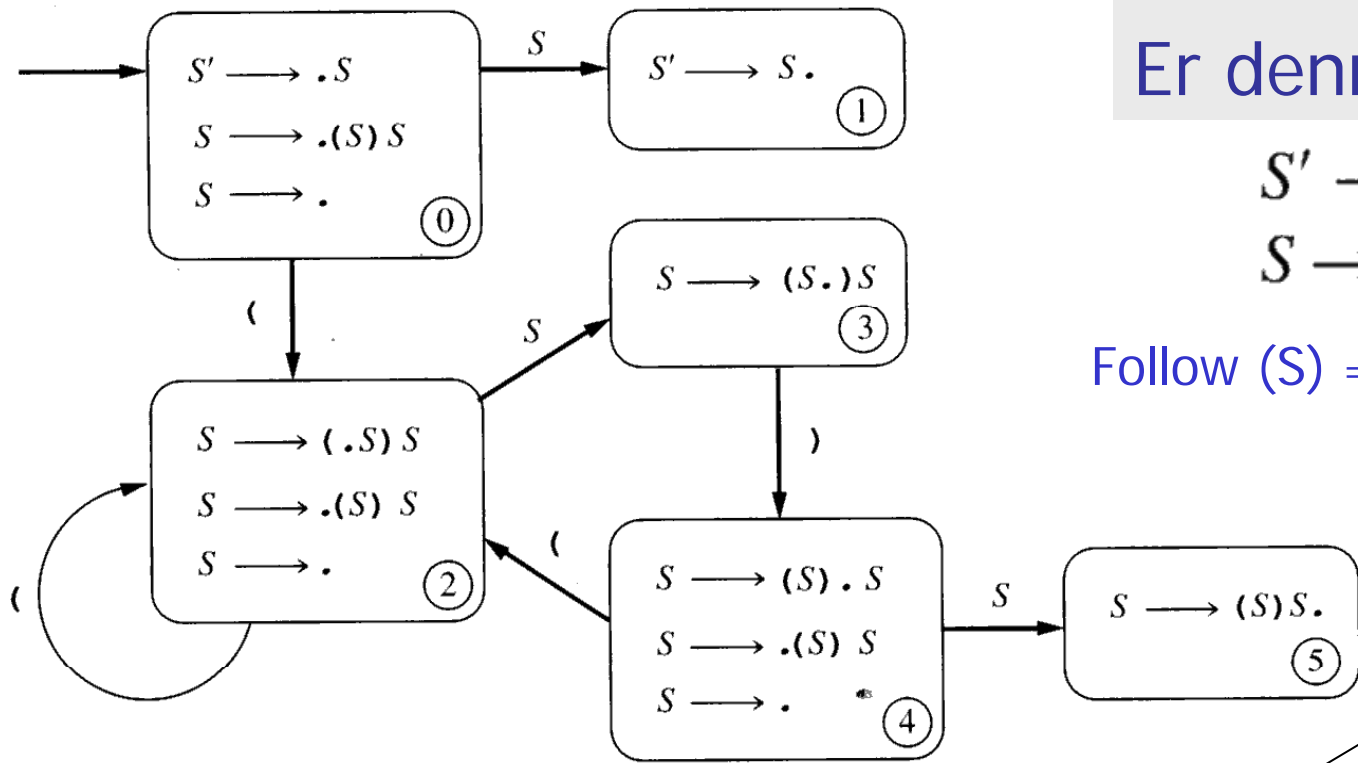
Merk at man ikke behøver å teste om det er mer input, siden man får accept bare foran \$ 34

Er denne SLR(1) ?

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

Follow (S) = { ), \$ }

**LITT RAR:**  
 Dette får vi i SLR(1), men ikke i LALR(1). Begge oppdager feilen, men SLR gjør det etter noen ekstra reduksjoner



State	Input			Goto
	(	)	\$	S
0	s2	r(S → ε)	r(S → ε)	1
1			accept	
2	s2	r(S → ε)	r(S → ε)	3
3		s4		
4	s2	r(S → ε)	r(S → ε)	5
5		r(S → ( S ) S)	r(S → ( S ) S)	



SLR(k) – mulig å lage teori for dette, men lite praktisk å få ut av det

---

### 5.3.4 SLR( $k$ ) Grammars

As with other parsing algorithms, the SLR(1) parsing algorithm can be extended to SLR( $k$ ) parsing where parsing actions are based on  $k \geq 1$  symbols of lookahead. Using the sets  $\text{First}_k$  and  $\text{Follow}_k$  as defined in the previous chapter, an SLR( $k$ ) parser uses the following two rules:

1. If state  $s$  contains an item of the form  $A \rightarrow \alpha.X\beta$  ( $X$  a token), and  $Xw \in \text{First}_k(X\beta)$  are the next  $k$  tokens 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  $A \rightarrow \alpha X.\beta$ .
2. If state  $s$  contains the complete item  $A \rightarrow \alpha.$ , and  $w \in \text{Follow}_k(A)$  are the next  $k$  tokens in the input string, then the action is to reduce by the rule  $A \rightarrow \alpha$ .

SLR( $k$ ) parsing is more powerful than SLR(1) parsing when  $k > 1$ , but at a substantial cost in complexity, since the parsing table grows exponentially in size with  $k$ .