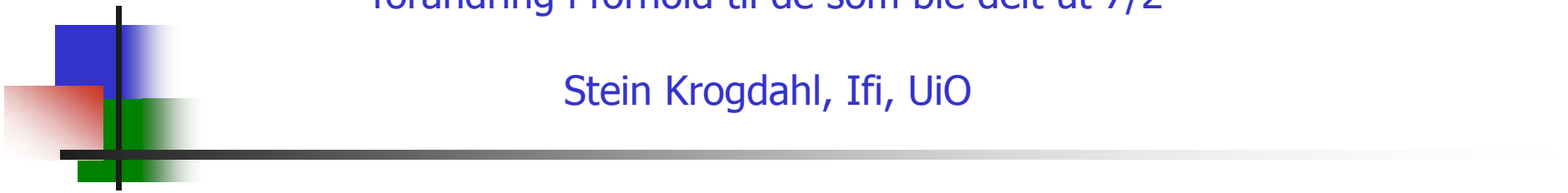
INF5110, 12/2-2008

Dagens tema:

Fortsette på kap. 5: LR-parsering

Her er foiler helt fra starten av kap 5, og det er litt forandring i forhold til de som ble delt ut 7/2

Stein Krogdahl, Ifi, UiO



Videre fremover:

14/2: Oppgaver til kap. 4, og antakeligvis mer kap. 5
Oppgavene ligger på undervisningsplan og på neste foil

Hvordan det blir neste uke kommer senest torsdag

Oppgaver til INF 5110, kapittel 4 (hårfint forandret 12/2, kl 11)
Gjennomgås torsdag 14. febr. 2008

Oppgave 1 (Mye repetisjon): Gitt gram.: exp → exp op exp | (exp) | **num**
op → + | - | * | / | ** | < | =

a) Grammatikken over er opplagt flertydig. Lag en entydig grammatikk for språket ut fra at følgende tilleggsregler:
   - ** (opphøying) har presedens 3 (høyest) og er høyre-assossiativ
   - * og / har presedens 2, og er venstre-assosiaiv
   - + og - har presedens 1 og er venstre-assosiativ
   - < og = har presedens 0, og er ikke-assosiativ

b) Se på grammatikken du fant under a), og skriv et syntaksdiagram (med løkker der det passer) for hver ikke-terminal. Del opp "op"- terminalene på hensiktsmessig måte.

c) Lag recursive-descent prosedyrer for å sjekke programmet (med while-setninger der det passer) ut fra grammatikken fra b). Du kan bruke både "match(token)" og "gettoken()" fra boka (som begge setter neste symbol inn i variablen "token").

d) Ut fra svaret på c), legg til trebyggings-setninger i prosedyren som behandler en sekvens av **, slik at treet får riktig høyre-assosiativ form.

e) Ta hele grammatikken fra a), og gjør den fri for venstreassosiativitet, og gjør all mulig venstrefaktorisering (men behold entydighet).

f) Sjekk om grammatikken fra e) er LL(1).

Oppgave 2: Skriv om prosedyrene midt på side 162, slik at de produserer trær (og ikke tall)

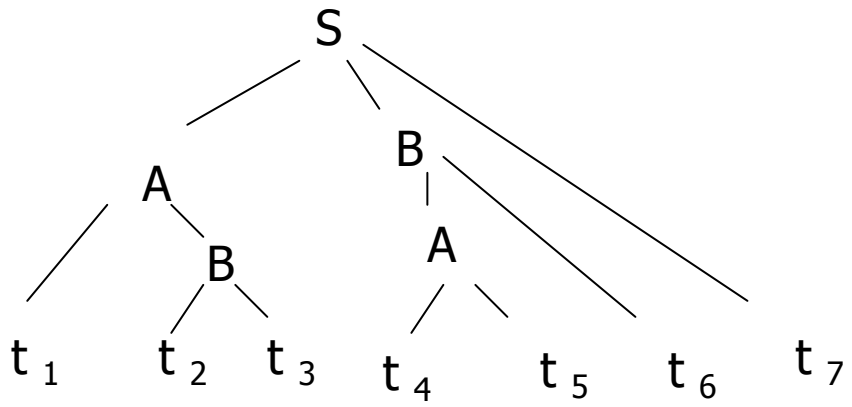
Oppgave 3: Sjekk om grammatikken "S → (S) S | ε" er LL(1)

Oppgave 4: Gitt gram.: exp → exp **+** exp | **(**exp**)** | **if** exp **then** exp **else** exp | **var**

a) Lag en entydig grammatikk for dette språket, der + skal være venstreassosiativ, og der "if x then y else z+u" skal bety "if x then y else (z+u)". Forsøk også å lage en grammatikk med den "motsatte" tolkningen: at det betyr "(if x then y else z)+u".

b) Hvorfor får vi ikke noe "dagling else"-problem her?

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



LR-parsering og grammatikker

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

-Autimatisert

-YACC, Bison, CUP (LALR(1))

Prinsippet og datastrukturen for LR-parsering

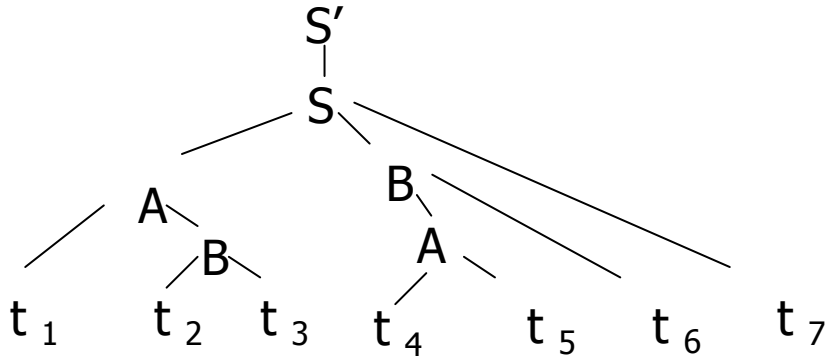
$S' \rightarrow S$

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

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

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

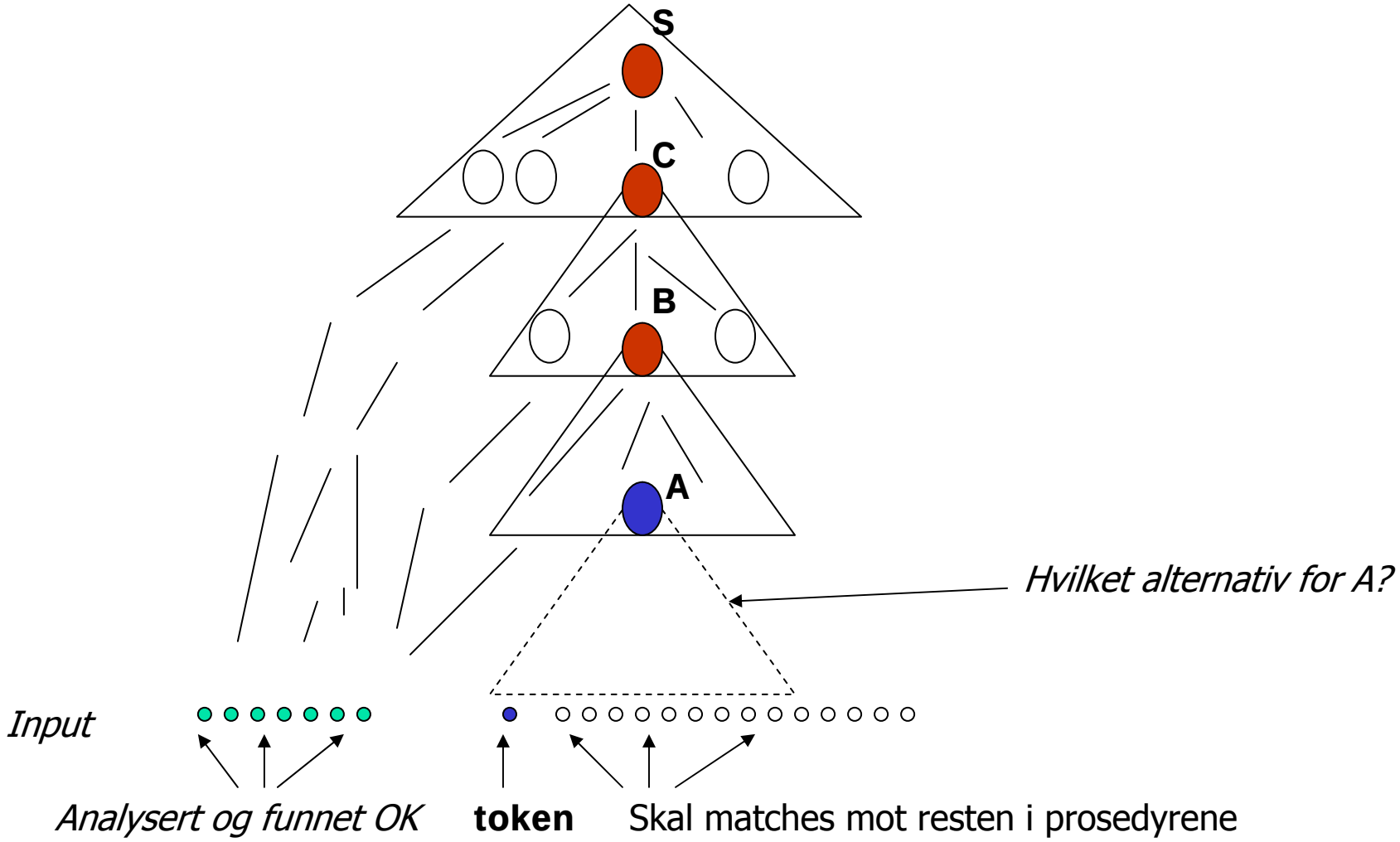
Anta at grammatikken er entydig, og at vi *kjenner syntaks-treet* for setningen:



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

Til sammenlikning: ”Top down”-parsering



Prinsippet og datastrukturen LR-parsering II

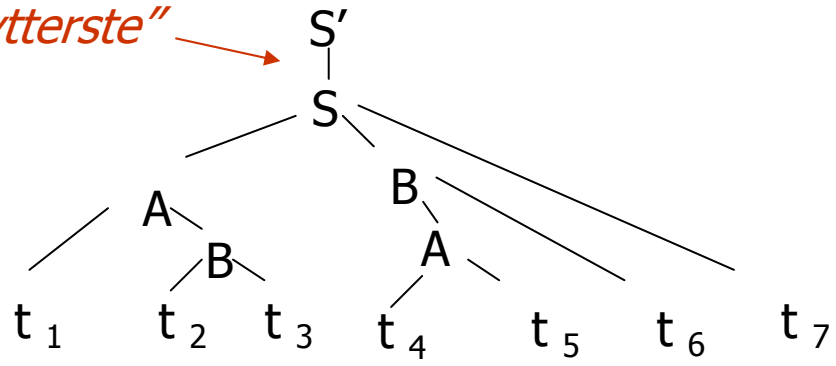
$S' \rightarrow S$

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

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

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

Sett på ny "ytterste" produksjon



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:

$	t_1 t_2 t_3 t_4 t_5 t_6 t_7 $
stakk	input

Slutt-situasjonen:

$ S'	$
stakk	input

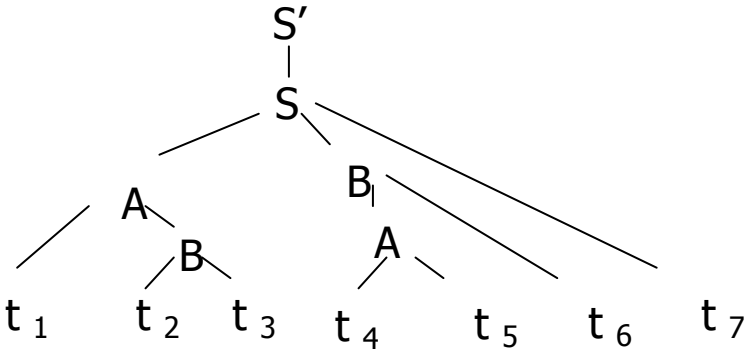
Prinsippet med LR-parsering III

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	$

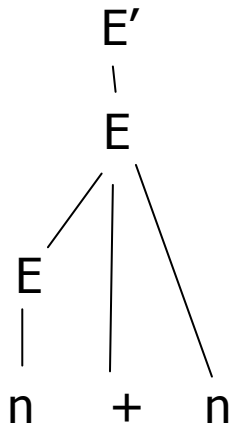
- Får to typer steg:
  - *Reduksjon* (på toppen av stakken med bestemt A → α)
  - *Lesing* ("skift") av input over til stakken
- Dersom man kjenner syntakstreet for den aktuelle setning, er det lett å angi de rette stegene.
- MEN: Hvordan gjøre dette underveis uten å kjenne resten av input ??
- *Stakk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

$E' \rightarrow E$

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

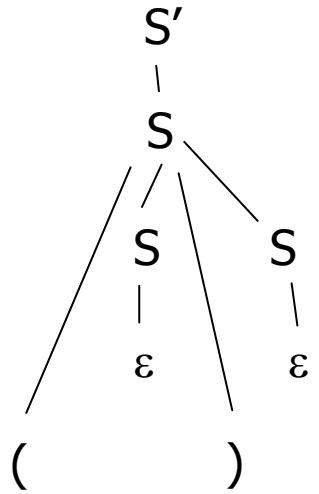


Boka: (Stakk + input) utgjør et stadium i en høyre-avledning. Den neste reduksjonen som skal gjøres, kalles situasjonens ”handle” (håndtak).

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

Eksempel på LR-parsering

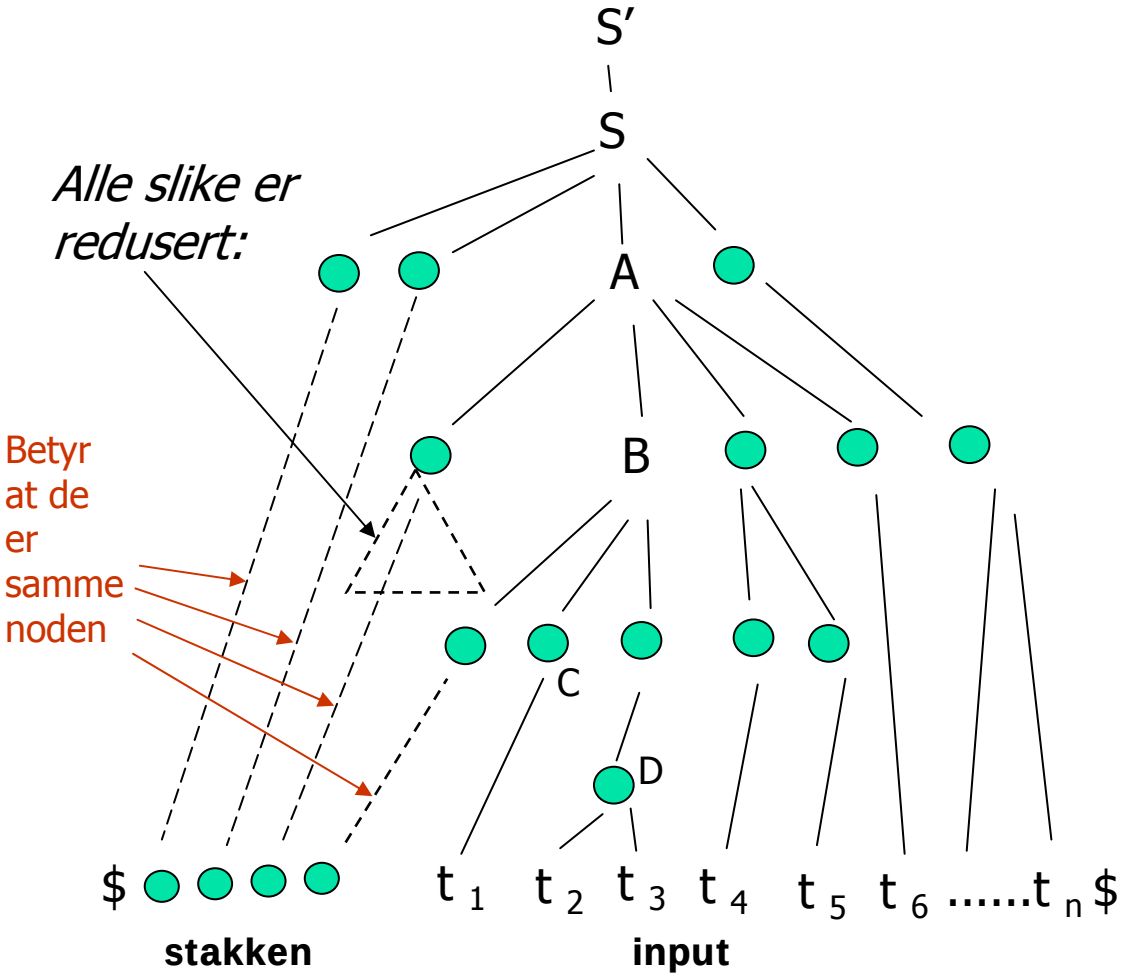
$S' \rightarrow S$
$S \rightarrow (\; S \;) \;\; S \mid \varepsilon$



NB: $S \rightarrow \varepsilon$
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



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

Det neste reduksjonen som blir gjort er reduksjon med

C -> t1

Deretter (etter noen skift):

D -> t2 t3

Gitt en entydig grammatikk G.

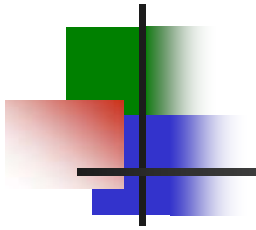
Denne behandles som følger for å lage en LR-parser:

(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 alle tilstander er aksepterende:
  - Tilstandene angis av "itemer" av formen: A $\rightarrow$ X Y **.** 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):**
  - 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 angitt 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.
  - 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***

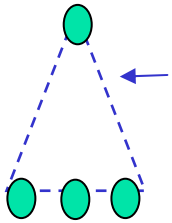
- Punktumet angir grovt sett situasjonen i ”dypeste del” av parseringen i øyeblikket:
  - Det til venstre for punktum er gjenkjent fra input, enten bare lest eller at deler av det er redusert til en ikke-terminal
  - Det til høyre for punktum har vi enda ikke sett/lest

Kantene i NFA-en I

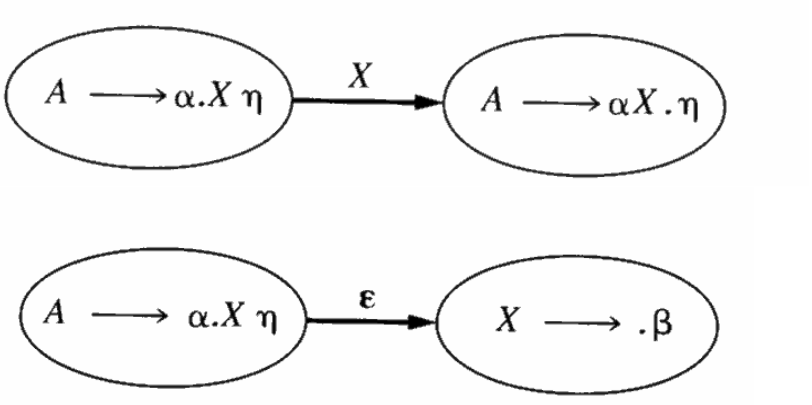
Dette er ikke fullt forklart i boka, og er ikke pensum

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).

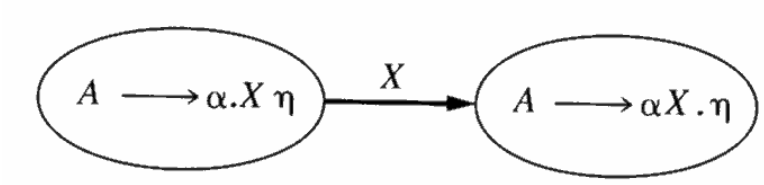


Husk: Ingen slik på venstre side. De er redusert!



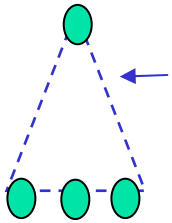
Kantene i NFA-en II

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

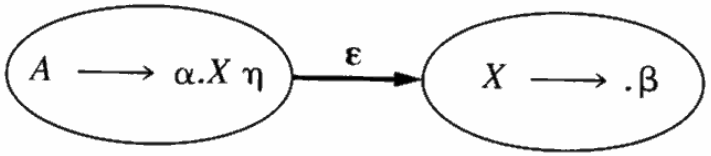


tilsvarer

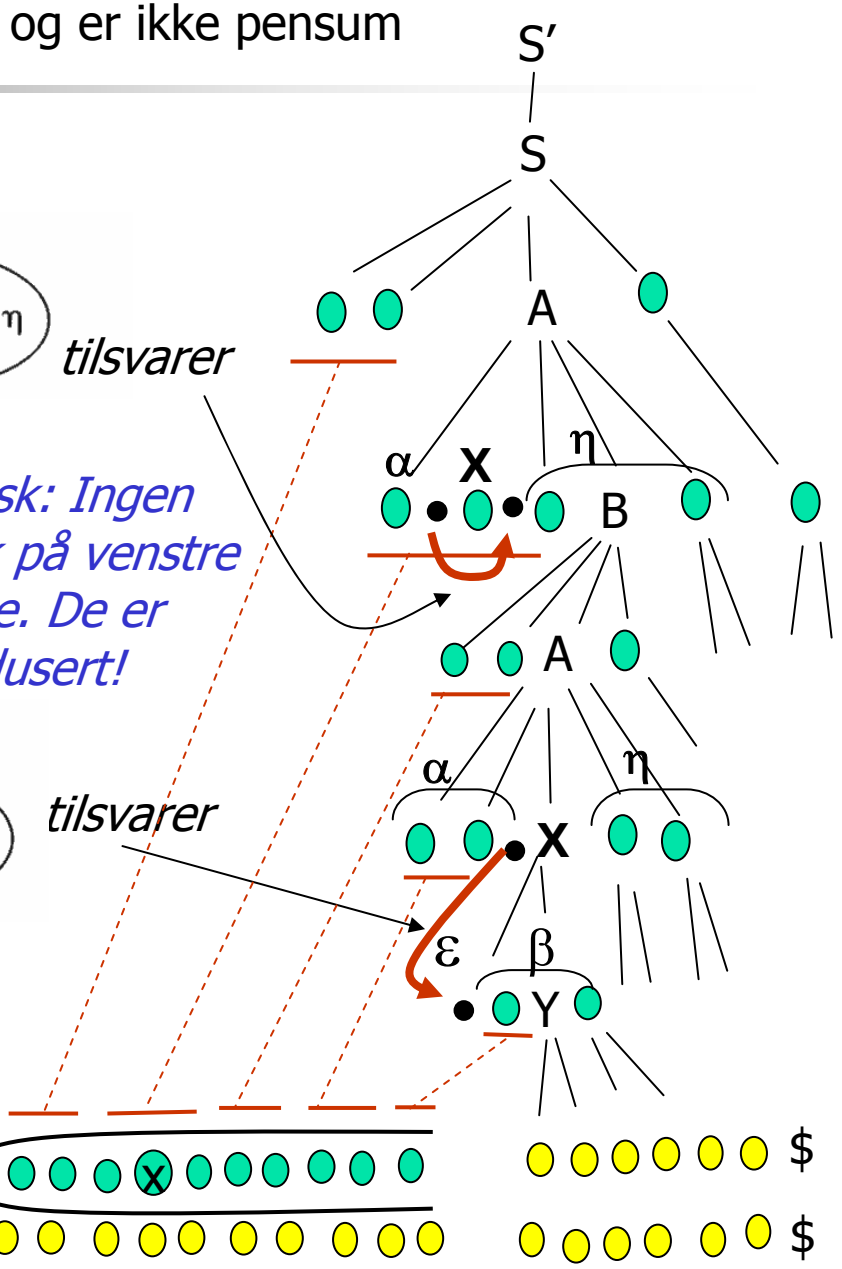
Gitt en grammatikk G og en situasjon under passering av en setning s i L(G).



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



tilsvarer



Eksempel på NFA

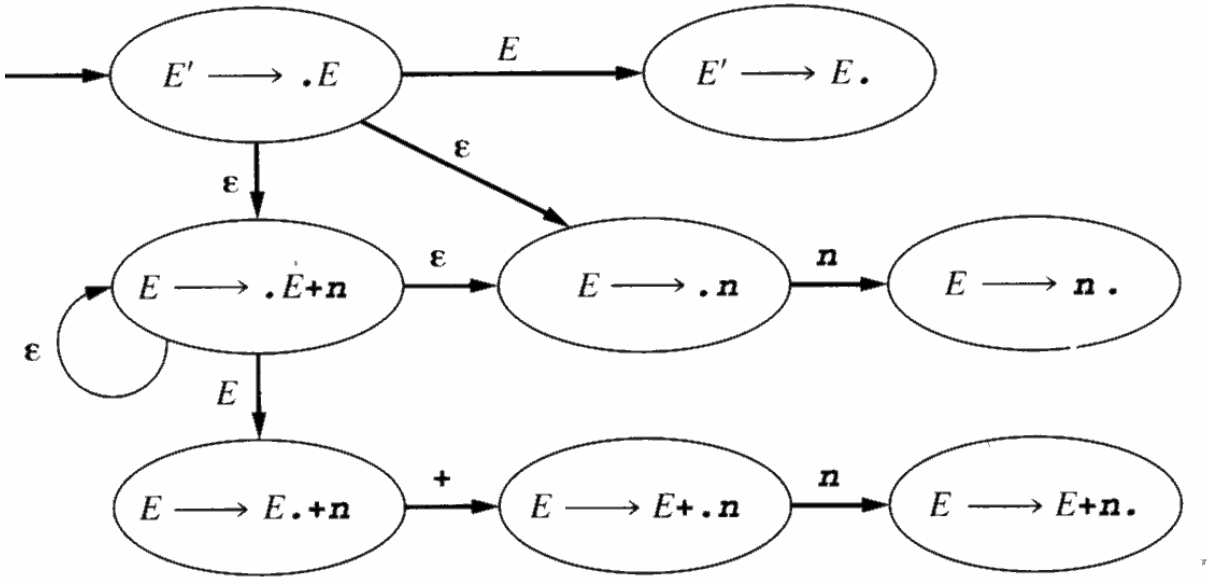
E ′ → E

E → E + n

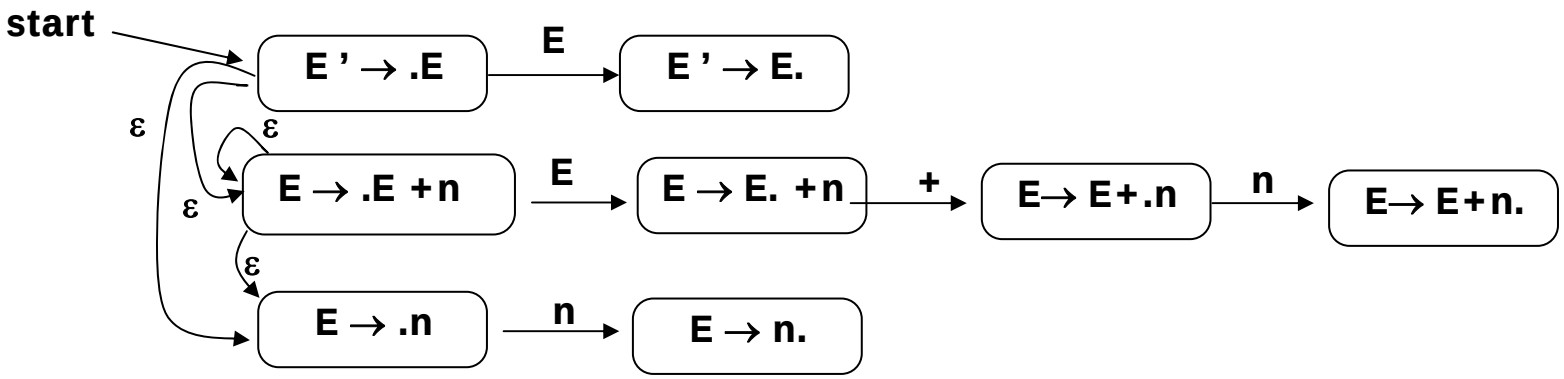
E → n

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

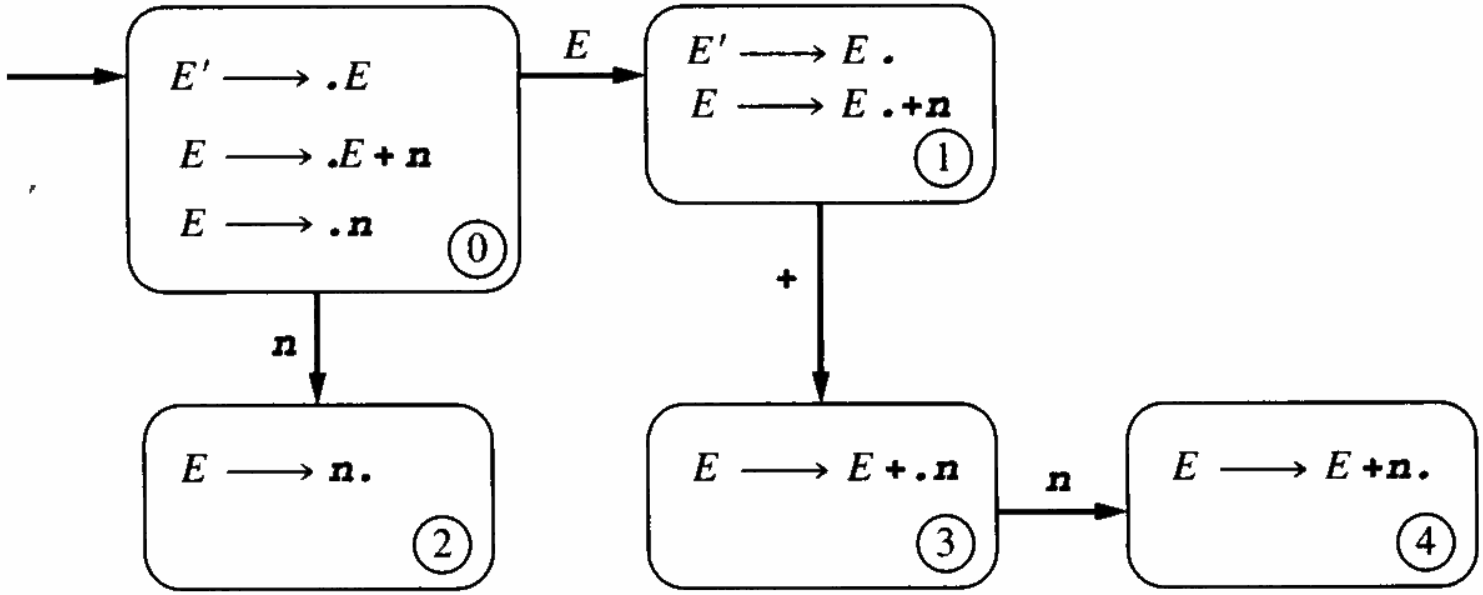
Itemer (LR(0)itemer)



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



LR(0)-DFA:

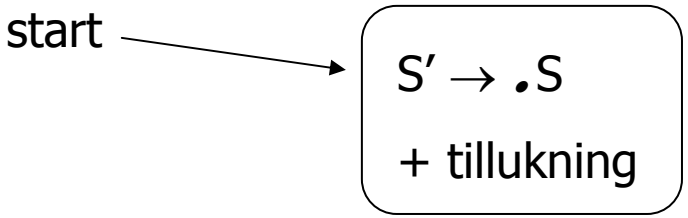


Hvordan sette opp LR(0) - DFA'en direkte fra grammatikken

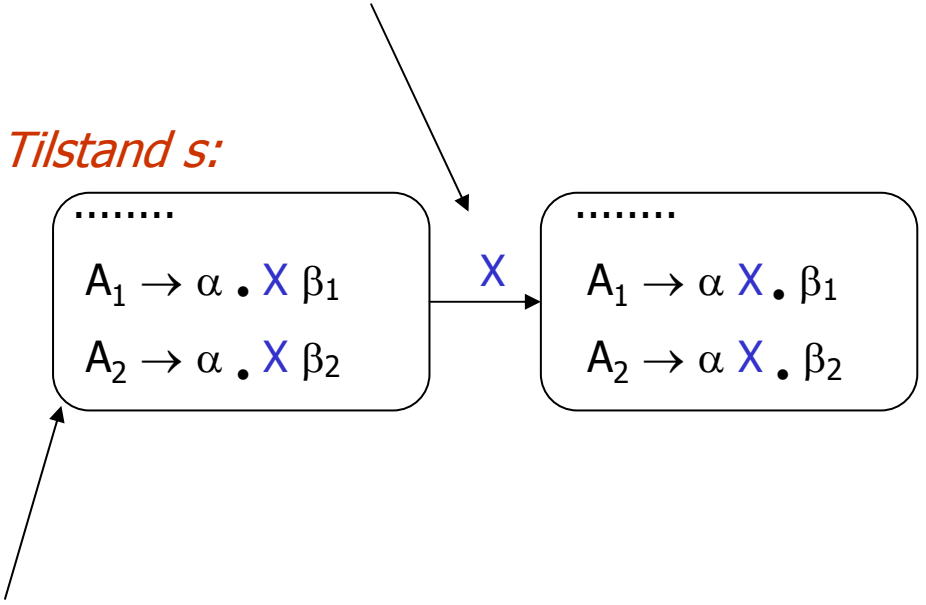
- Tillukning av item-mengde I:
  - Dersom:
    - $A \rightarrow \alpha \bullet B\gamma$ er med i I
    - B er en ikke-terminal
    - $B \rightarrow \beta_1 \mid \beta_2 \mid \ldots\ldots\ldots$

    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

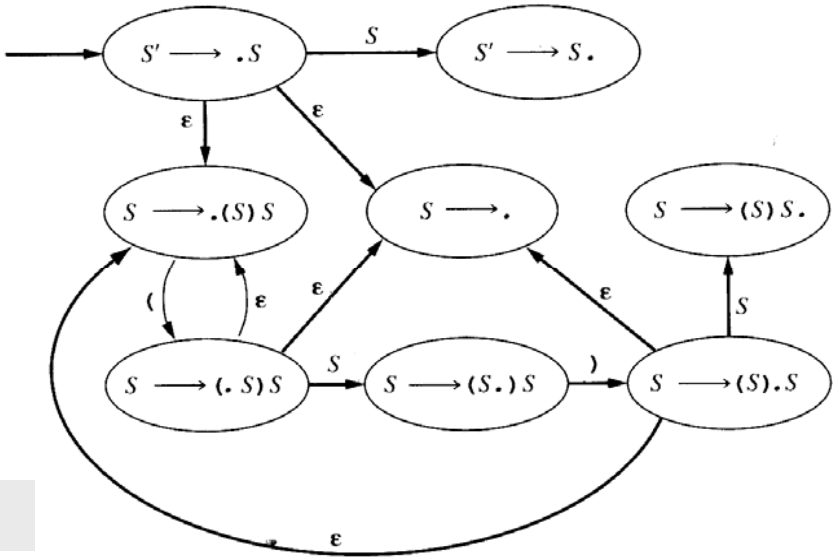
$$S' \rightarrow S$$
$$S \rightarrow (\ S \ ) \quad S \mid \varepsilon$$

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

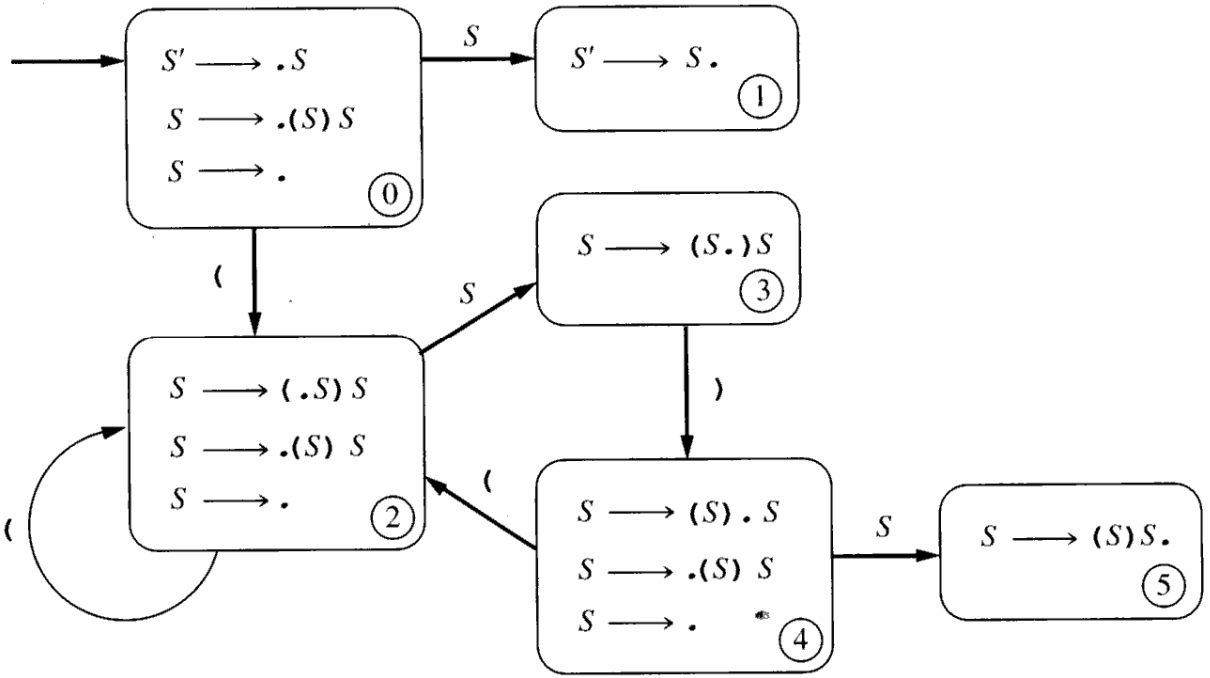
(Altså ikke: ~~S → • ε~~ og ~~S → ε •~~)

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

LR(0) – NFA:

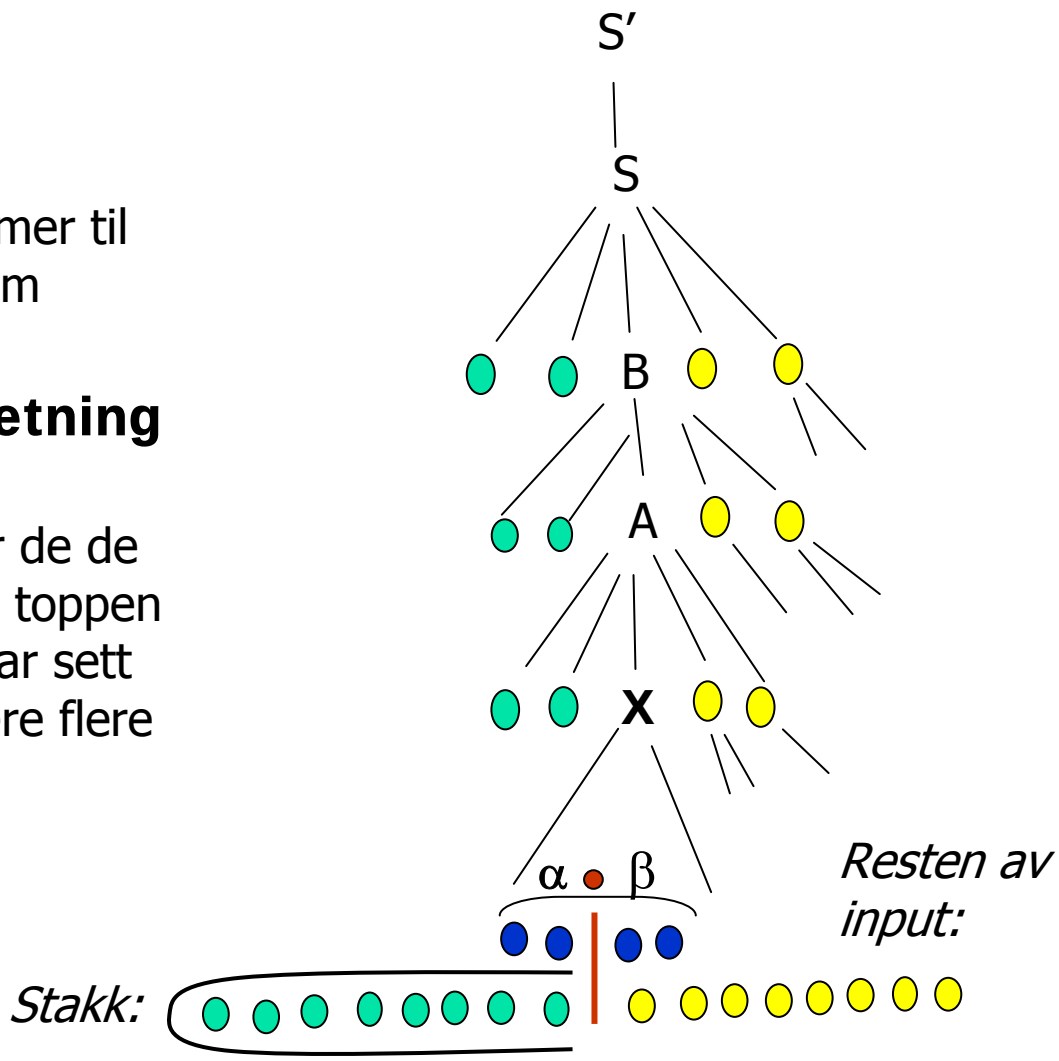


LR(0) – DFA:



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)

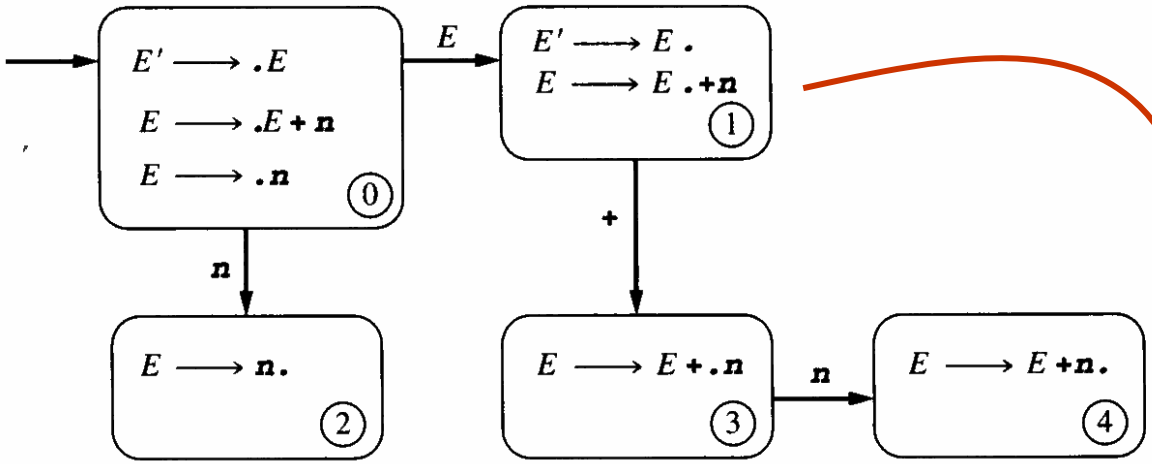


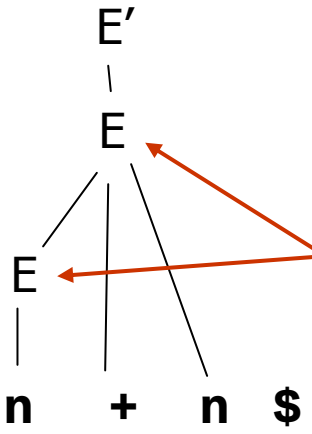
Dersom itemet: $\mathbf{X} \rightarrow \alpha \bullet \beta$ er med i topp-tilstanden kan situasjonen være som angitt

Hva sier topp-tilstanden? Eksempel:

$E' \rightarrow E$

$E \rightarrow E + n \mid n$

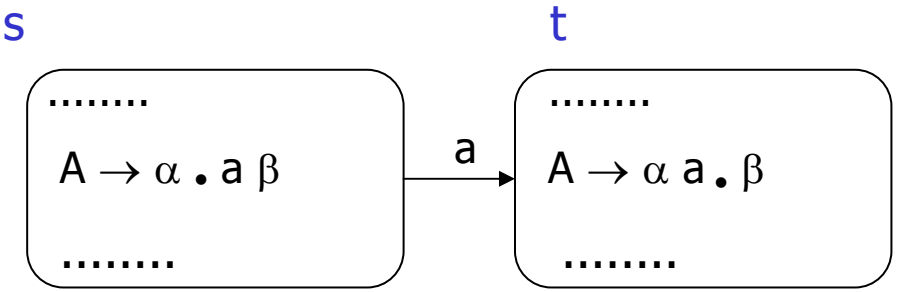




$		n + n $
$ n		+ n $
$ E	Skal skifte	+ n $
$ E +		n $
$ E + n		$
$ E	Skal redusere med E′→E	$
$ E′		$

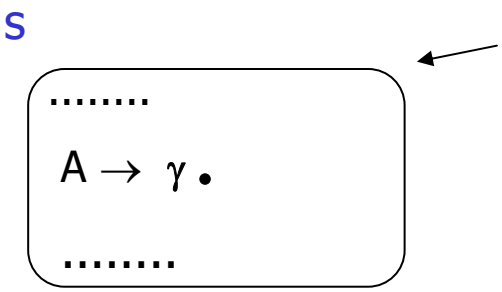
LR(0)-grammatikker og LR(0)-algoritmen:

1)



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

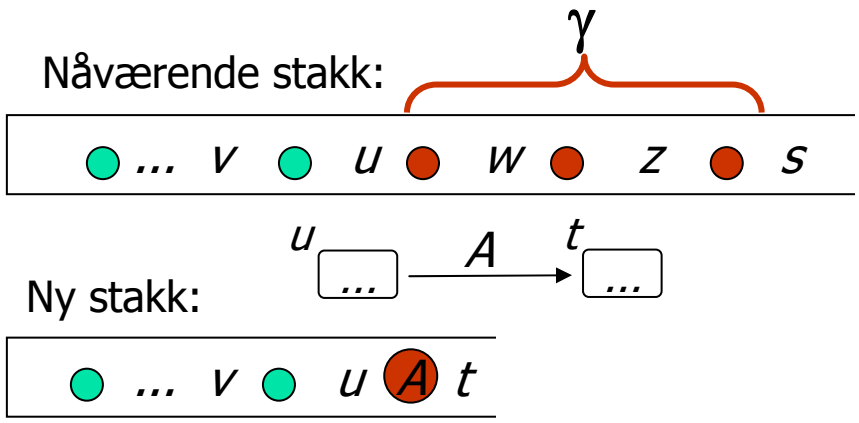
2)



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) ?

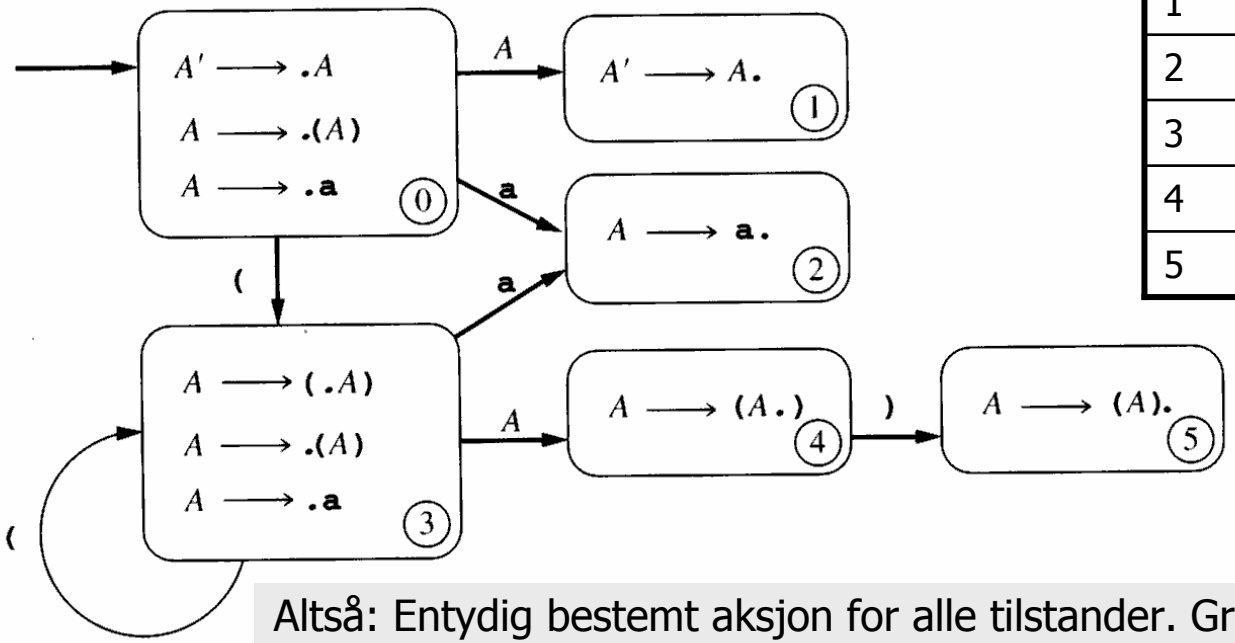
Ser på de tre grammatikkene våre – først A:

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

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

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

A gir følgende LR(0) – DFA:



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

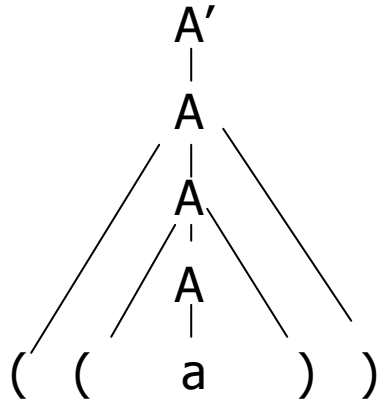
Tabell-oppsett for en LR(0) - grammatikk:

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	) $

Viktig: *Skifter aldri noe ulovlig inn på stakken!*

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 γ and all of its corresponding states from the parsing stack (the string γ 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 γ began (this must be the state uncovered by the removal of γ). 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.)

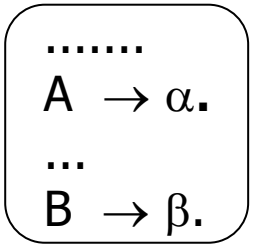
   t, where $u \xrightarrow{A} t$

Avslutning

Kan forekomme i flere tilstander

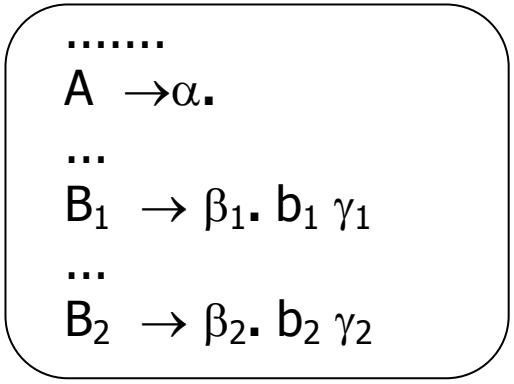
SLR(1) - grammatikker, SLR(1) - algoritmer

- 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

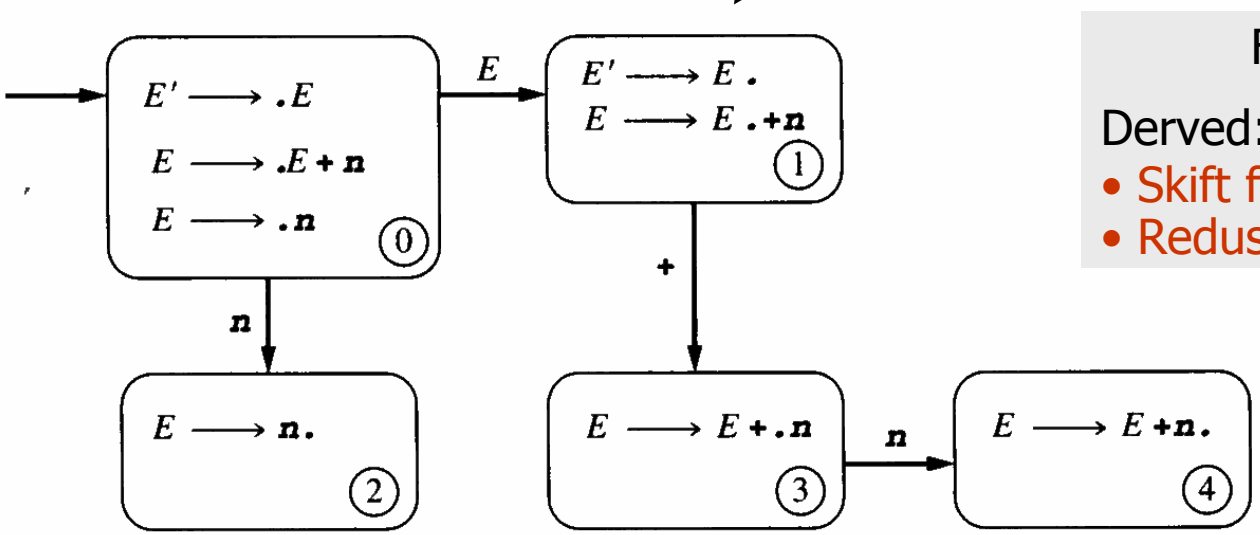


LR(0) : Har her (uløselig) shift/red - konflikt

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

"Complete item" = Har punktumet til slutt

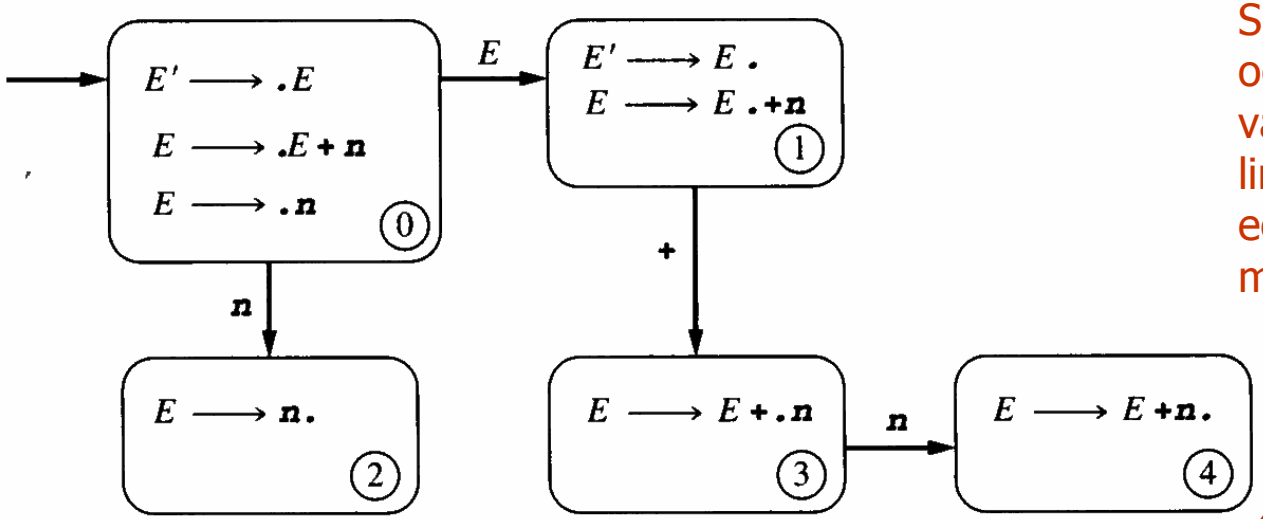
En tilsvarende formulering av SLR(1) kravet
Dersom følgende gir entydig algoritme, er grammatikken SLR(1)

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 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 \$.[4] In all other cases, the new state is computed as follows. Remove the string γ and all of its corresponding states from the parsing stack. Correspondingly, back up in the DFA to the state from which the construction of γ began. By construction, this state 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' -> A)

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 ($E \rightarrow E + n$)	

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

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:

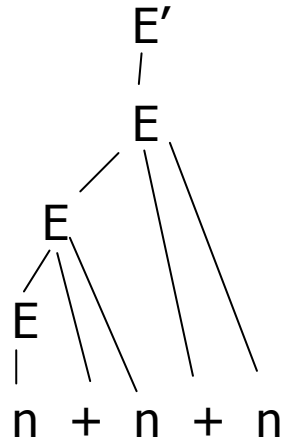
+ n $

n n $

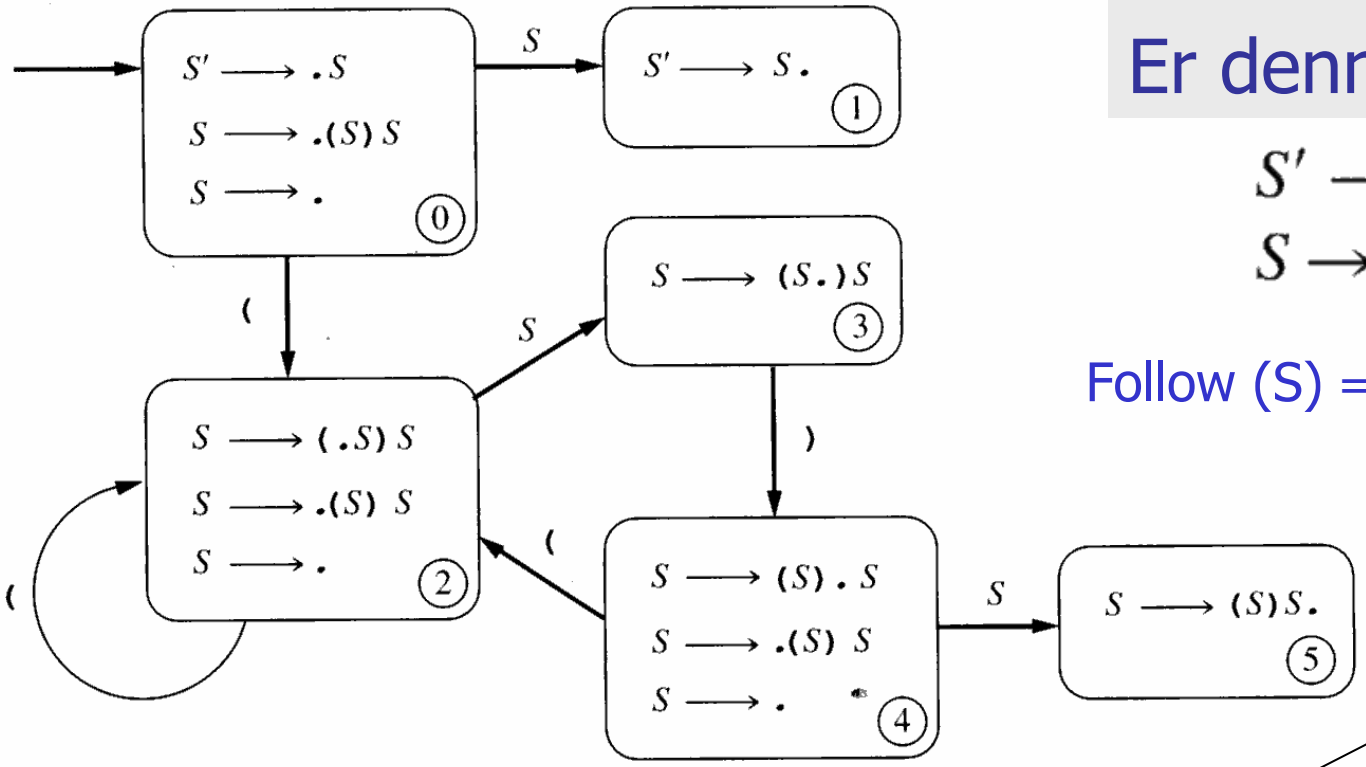
n + $

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) = {), $ }

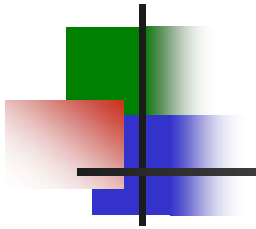
Til senere:
Dette får vi i SLR(1), men ikke i LALR(1). Begge oppdager feilen, men LALR gjør det noe tidligere.

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)
(og de er heller ikke LL(1)!)

- Er heller ikke SLR(k) eller LR(k) for noen k
- LR(0)-DFA'er for flertydige grammatikker 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 som vi skal bruke i obligene):
  - Skift/Reduser – konflikter:

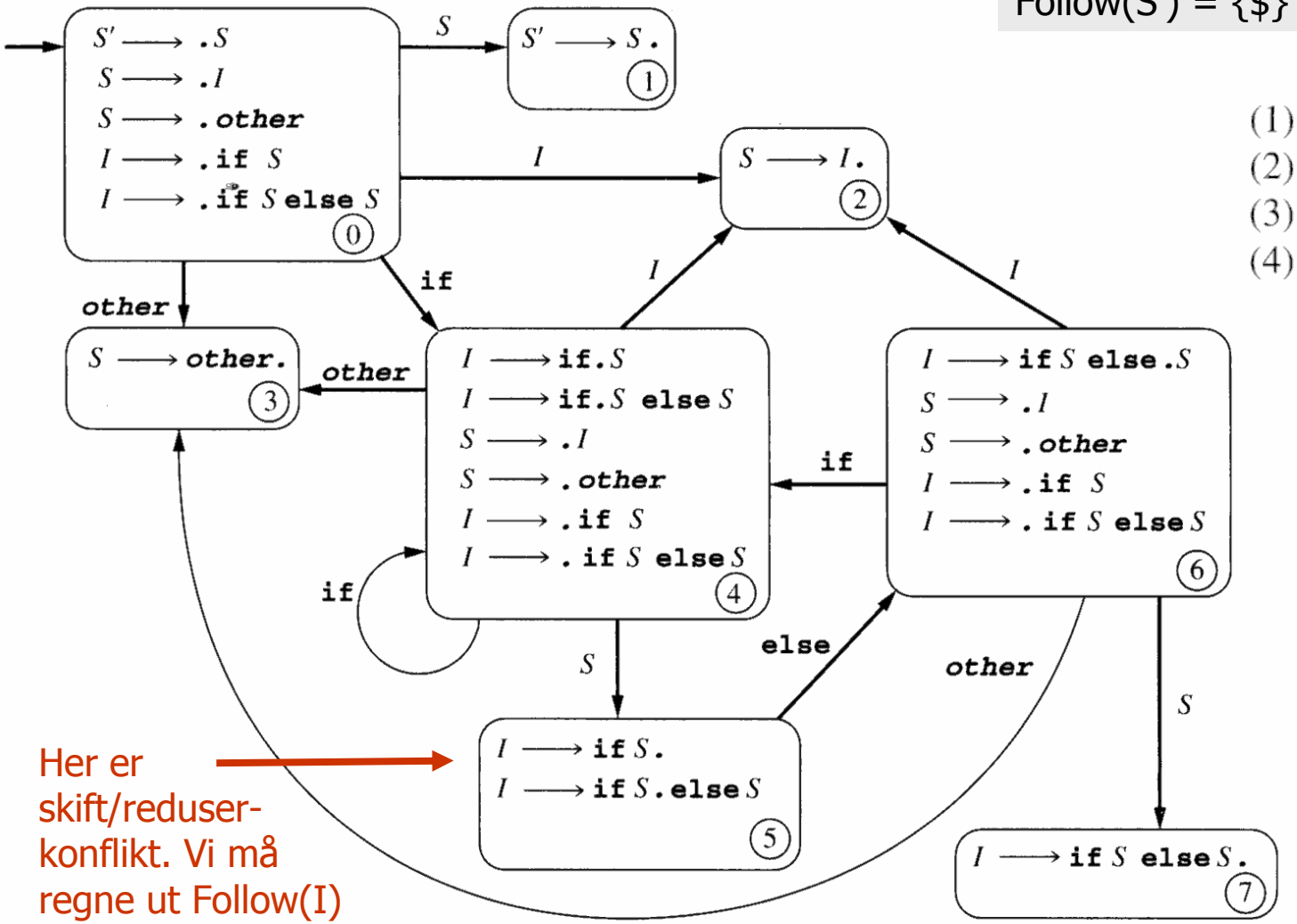
    Bruker skift, om intet annet er angitt

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** }

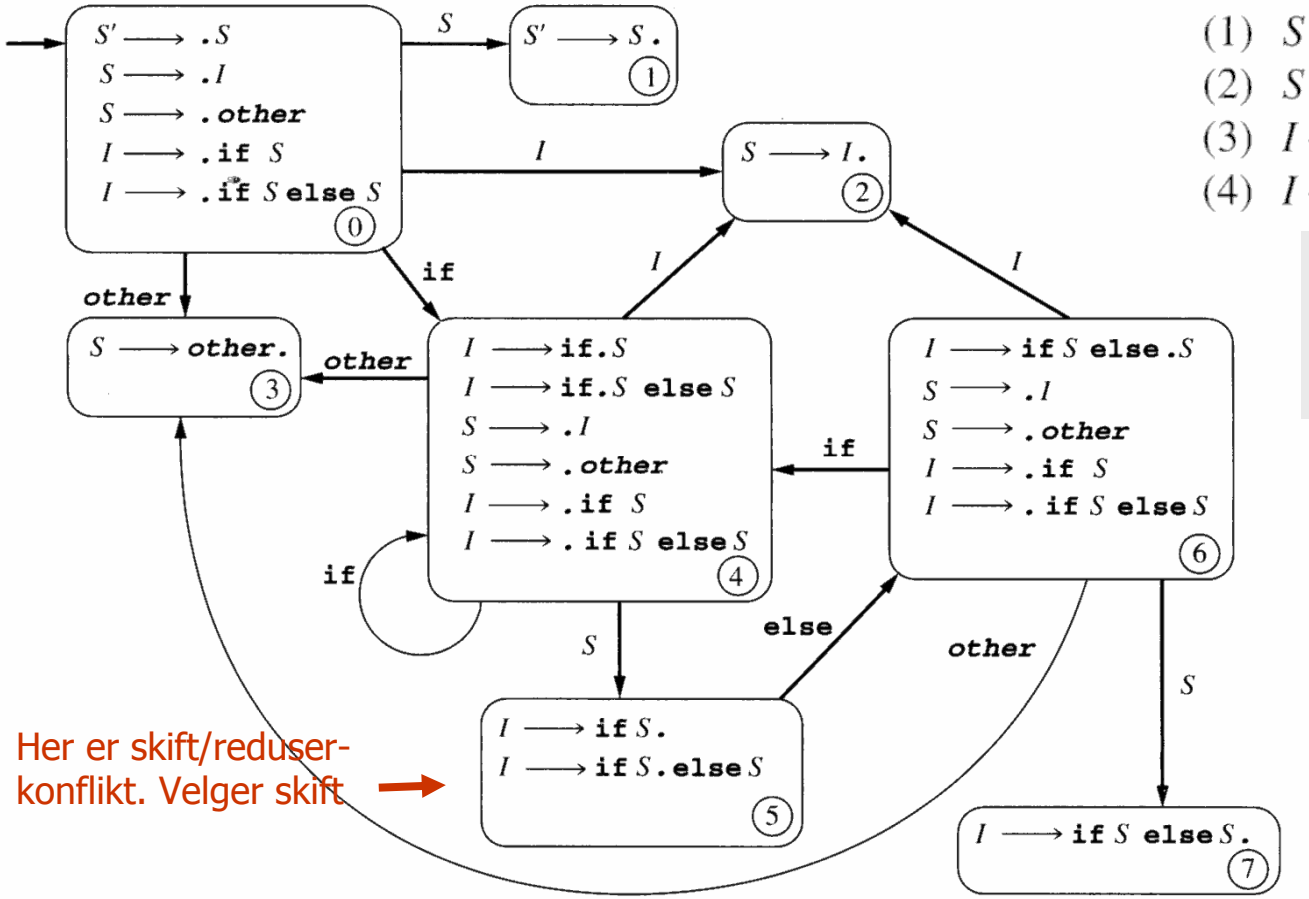
Follow(S') = {$}



(1) $S \rightarrow I$
(2) $S \rightarrow$ **other**
(3) $I \rightarrow$ **if** S
(4) $I \rightarrow$ **if** S **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') = {$}

I tabellen betyr betyr:

s3 – skift **if** fra input til stakk. Legg så 3 på toppen av stakken

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

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		

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