

INF5110 – 12/2-2013

Kap. 5, Del 2

Stein Krogdahl, Ifi, UiO

Dagens temaer:

Noen foiler igjen fra forrige gang

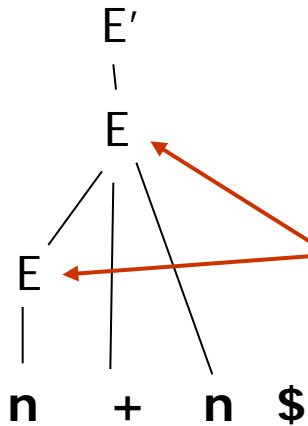
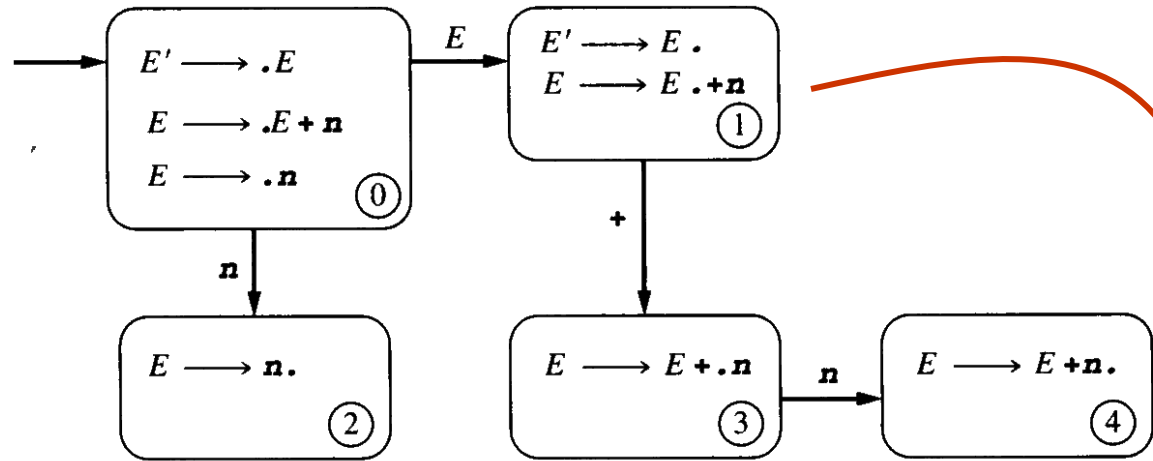
SLR(1), LR(1)- og LALR(1)-grammatikker

- NB: Oppgaver til kap 4 og 5 er lagt ut på undervisningsplanen
- Begynner antakeligvis på oppgaver til kap. 4 neste gang
- Oblig 1 kommer senest i starten av neste uke
- Det blir da en intro til disse ved Henning Berg

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

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

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



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

$n + n \$$

$+ n \$$

$+ n \$$

$n \$$

$\$$

$\$$

$\$$

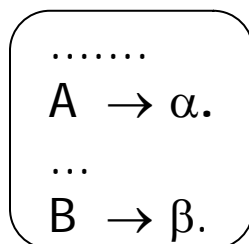
Skal skifte

Skal redusere med $E' \rightarrow E$

Løsning: Vi ser på neste input-symbol!

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

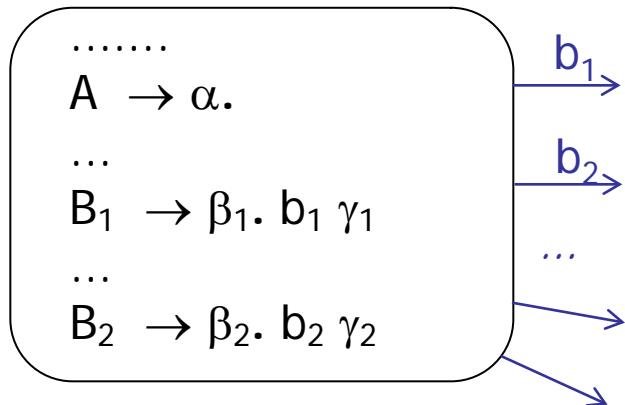
- Svært få grammatikker er LR(0)
- Ved å se på Follow-mengdene kan vi få en mye sterkere algoritme
- Tar også nå utgangspunkt i LR(0)-DFA'en
- Tabell-oppsettet er nesten likt, men nå må "reducer-linjene" fra LR(0)-tabellene, spesifiseres for hvert mulig «neste input-symbol» (= «token»)



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)

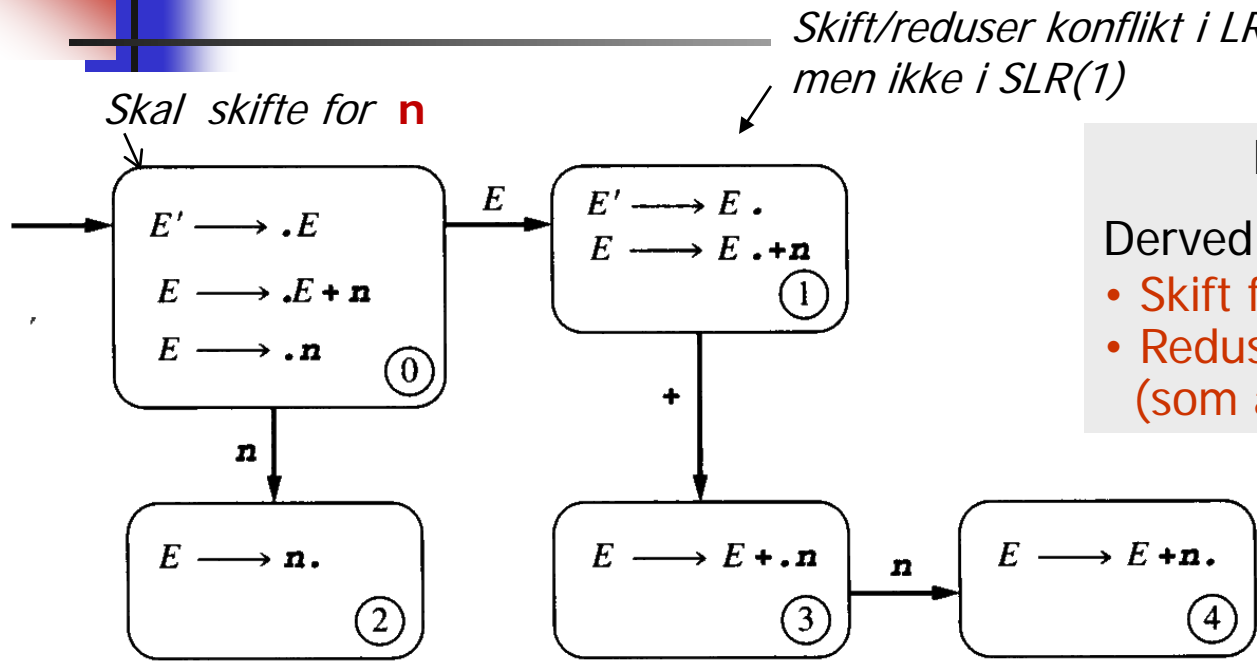


LR(0): Har her en (uløselig) skift/red.-konflikt

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 i denne mengden

"Complete item" = Har punktet til slutt

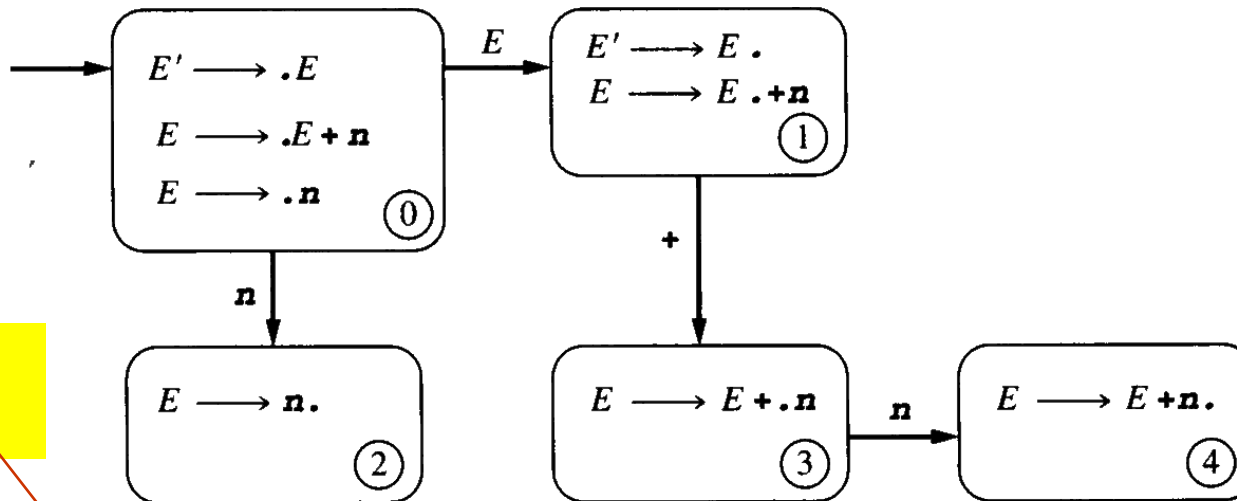
En tilsvarende formulering av SLR(1) kravet:
 Dersom følgende gir entydig algoritme, er grammatikken SLR(1)

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 $A \rightarrow \alpha.X\beta$, 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 $\$$.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 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 $B \rightarrow \alpha.A\beta$, 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 samme linje. ("Accept" er egentlig reduksjon med $A' \rightarrow A$)

'n' ikke i Follow(E)

State	Input			Goto
	n	+	\$	
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 LR(1)-grammatikk

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 ($E \rightarrow E + n$)	

Kan også se på gale setninger som:

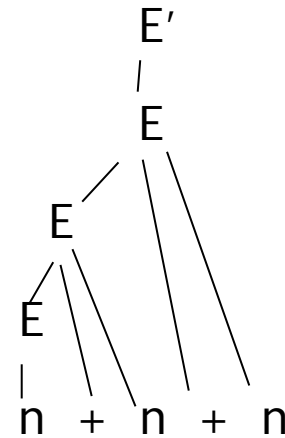
+ n \$

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



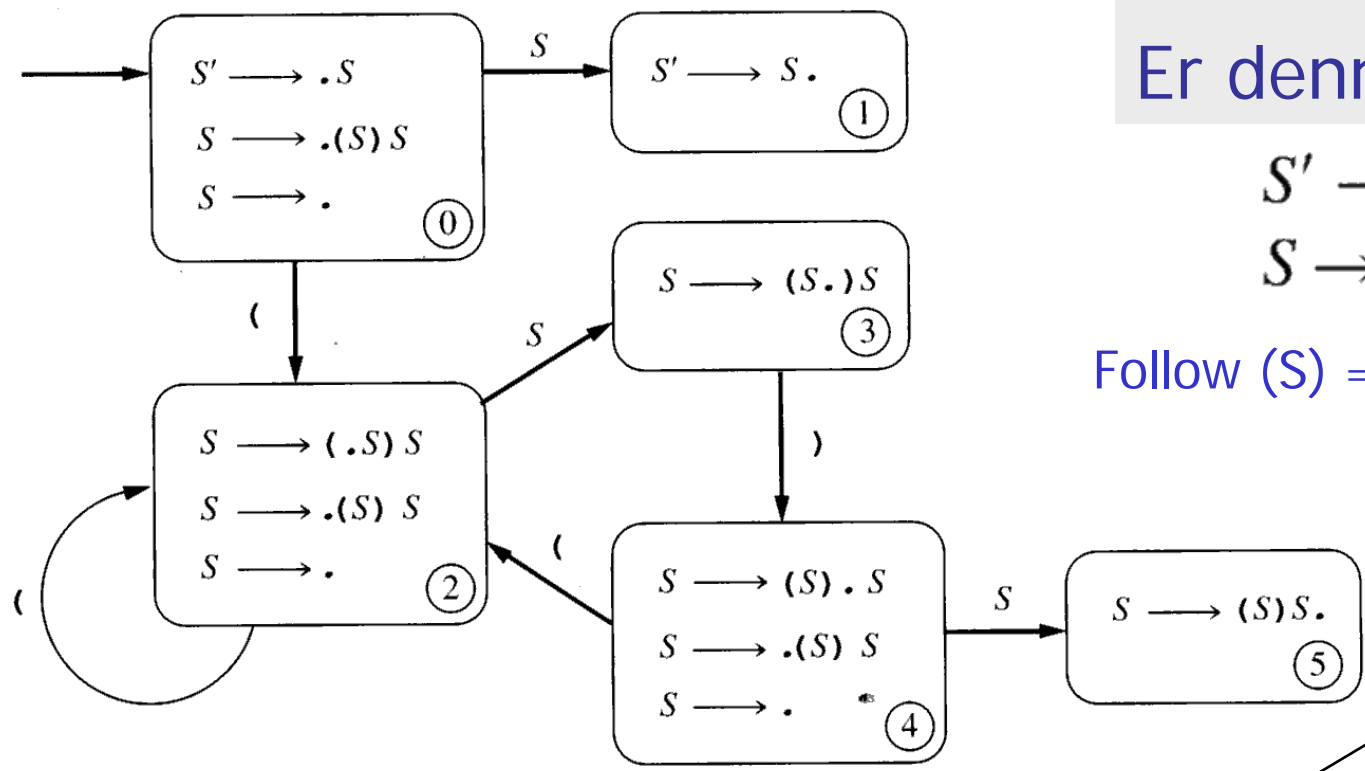
Merk at man ikke behøver å teste om det er mer input, siden man får accept bare foran \$ 7

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)	



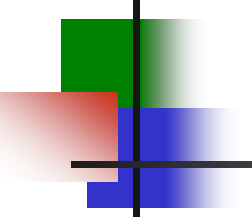
SLR(k) – mulig å lage teori for dette, men lite praktisk å få ut av det

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 First_k and 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 .



Flertydige grammatikker er aldri SLR(1) eller LR(1) (og de er altså heller ikke LL(1)!)

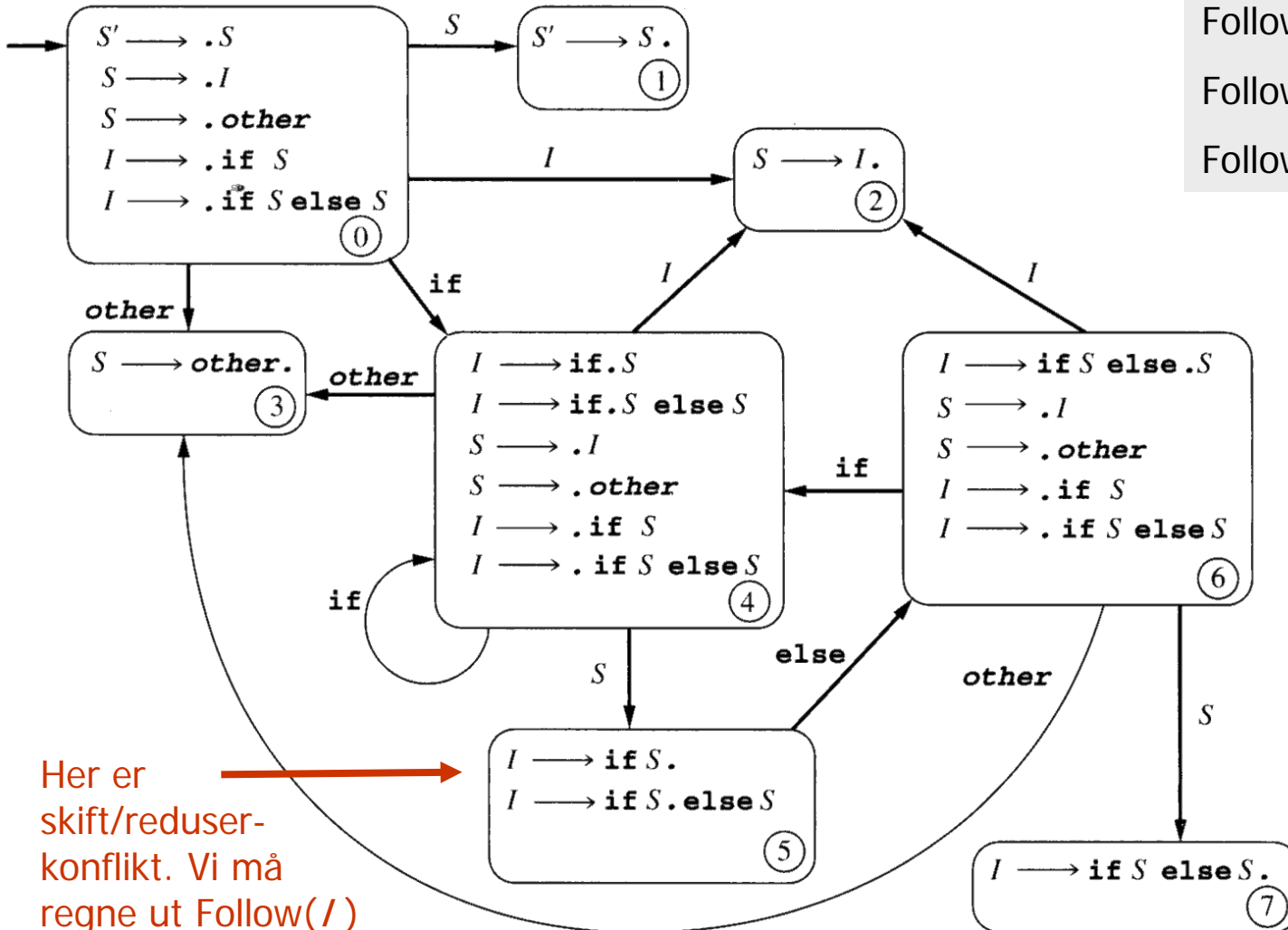
- Flertydige grammatikker er heller ikke SLR(k) eller LR(k) for noen k
- LR(0)-DFA'er for flertydige grammatikker vil derfor alltid ha "uløselige" konflikter, samme hvor mange symboler man få lov å se framover
 - Og det gjelder også LR(1)-DFA'er, LR(2)-DFA'er, osv
- **Men:** Konfliktene kan ofte likevel løses med presedens, assosiativitet, og andre regler.
 - For uttrykks-grammetikker: Angi presedens, assosiativitet e.l.
 - For "dangling else"-problemet: Om det er tvil, *gjøre en skift* (ikke redusér!).
- Vanlig strategi i Yacc, CUP etc.:
 - Skift/Redusér-konflikter: *Velger skift* om intet annet er angitt

Eksempel med flertydig grammatikk

$statement \rightarrow if-stmt \mid other$
 $if-stmt \rightarrow \mathbf{if} (exp) statement$
 $\quad \mid \mathbf{if} (exp) statement \mathbf{else} statement$
 $exp \rightarrow 0 \mid 1$

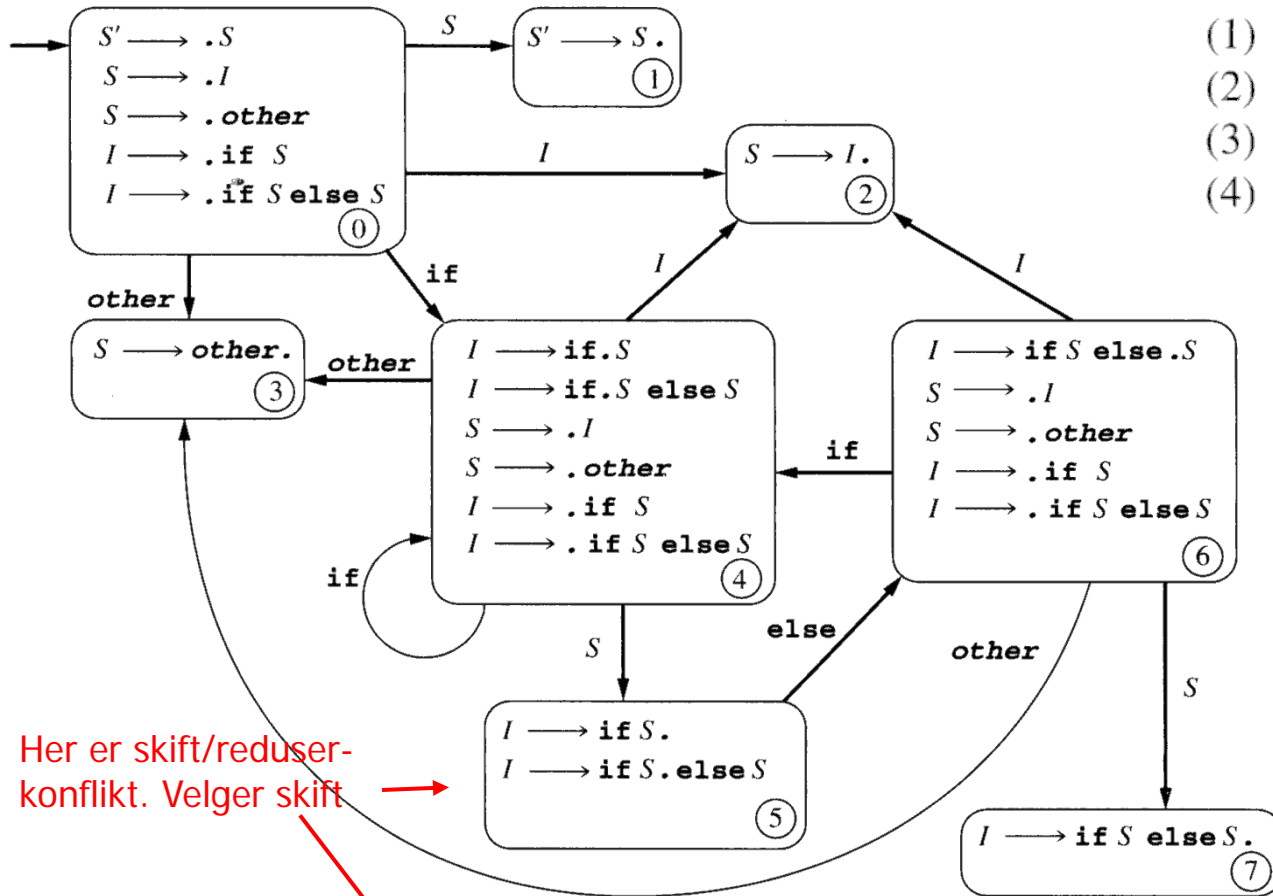
$S \rightarrow I \mid other$
 $I \rightarrow \mathbf{if} S \mid \mathbf{if} S \mathbf{else} S$

Follow(S) = { \$, else }
 Follow(I) = { \$, else }
 Follow(S') = { \$ }



NB: Her *måtte* det bli minst én konflikt, siden grammatikken er flertydig.

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



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

Follow(S) = { \$, else }
 Follow(I) = { \$, else }
 Follow(S') = { \$ }

I tabellen betyr:
s3 – skift fra input til stakk. Legg så tilstand 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 konflikt (tilstand 5) løst på standard måte

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) $S \rightarrow \mathbf{other}$
- (3) $I \rightarrow \mathbf{if} S$
- (4) $I \rightarrow \mathbf{if} S \mathbf{else} S$

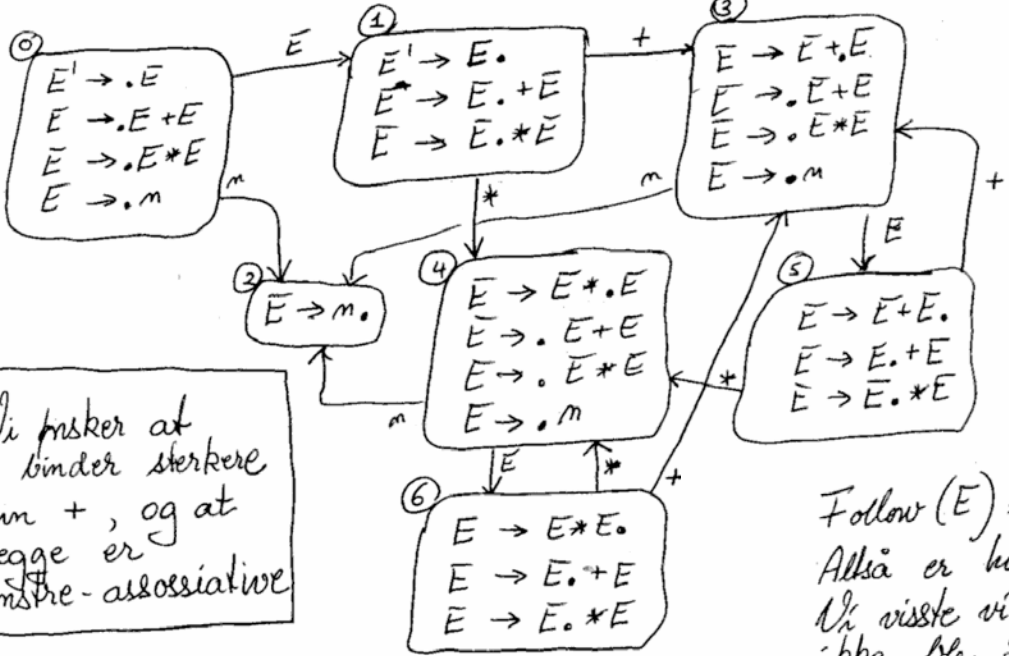
Mer bruk av flertydige grammatikker ved LR-parsing (ikke i boka, men pensum)

Eksempel: Enkel uttrykk-grammatikk

$E' \rightarrow E$
 $E \rightarrow E + E \mid E * E \mid m$

Vi vet at denne grammatikken er flertydig

LR(0)-DFA'en:



Vi ønsker at * binder sterkere enn +, og at begge er venstre-assosiative

Fordel ved flertydige grammatikker:
 De er som regel enklere å sette opp, se f.eks. til venstre her, og tidligere grammatikker for if-setningen

Konflikter må oppstå, men:
 man kan løse mange konflikter ved å angi presedens, assosiativitet, m.m. Dette kan f.eks. gjøres i CUP og Yacc

Follow(E) = {+, *, \$}
 Alltså er hverken 5 eller 6 SLR-tilstander.
 Vi visste vi måtte få konflikter slik at grammatikken ikke ble SLR-spen. Ingen flertydige grammatikker er SLR.
 Hva skal vi så gjøre i tilstand 5 og 6?

Tilstand 5: **Stakk=E+E** Input=
 \$: reduser, fordi skift ikke lovlig for \$
 +: reduser, fordi + er venstreassosiativ
 *: skift, fordi * har presedens over +

Tilstand 6: **Stakk=E*E** Input=
 \$: reduser, fordi skift ikke lovlig for \$
 +: reduser, fordi * har presedens over +
 *: reduser, fordi * er venstreassosiativ

Hva om også **? (høyreass.). Blir oppgave!

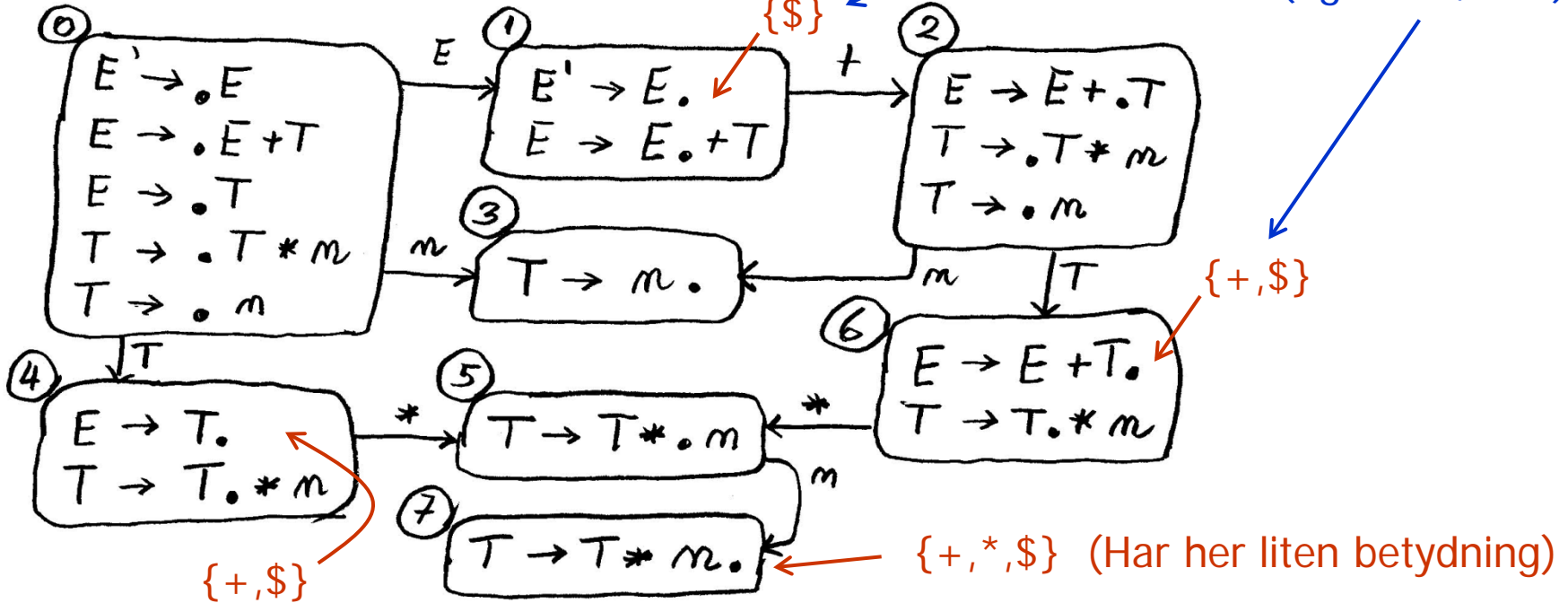
	m	+	*	\$	E
0	s2				1
1		s3	s4	accept	
2		r(E→m)	r(E→m)	r(E→n)	5
3	s2				6
4	s2				
5		r(E→E+E)	s4	r(E→E+E)	
6		r(E→E*E)	r(E→E*E)	r(E→E*E)	

Til sammenlikning: Den tilsvarende entydige grammatikken, med innbakt prioritet og venstre-assosiativitet

$E' \rightarrow E$
 $E \rightarrow E + T \mid T$
 $T \rightarrow T * m \mid m$

Follow
 $E': \{ \$ \}$ (gjelder alltid)
 $E: \{ + \$ \}$
 $T: \{ * + \$ \}$

Etterfølger-mengden til venstresiden (også her, osv.)



- LR(0)-DFA'en her har 8, og ikke 7 tilstander (som den flertydige)
- Grammatikken er, som vi ser, en SLR(1)-grammatikk

Om å bygge tre ved LR-parsering.

Vi bruker den parallelle variabelstakken som CUP tilbyr!

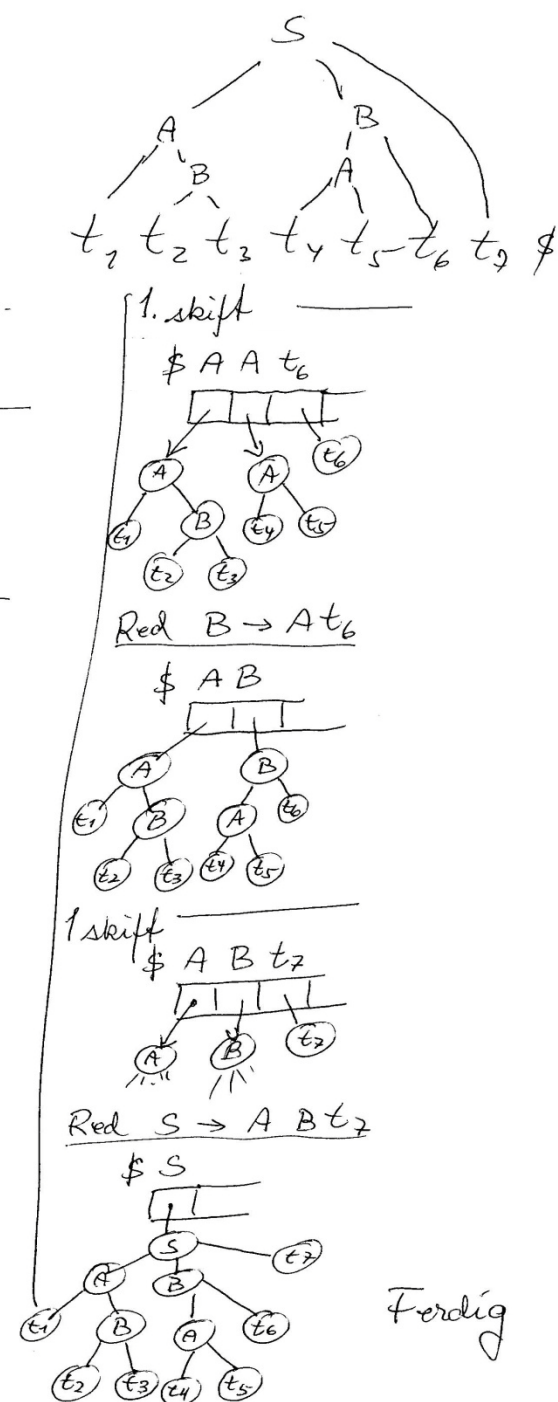
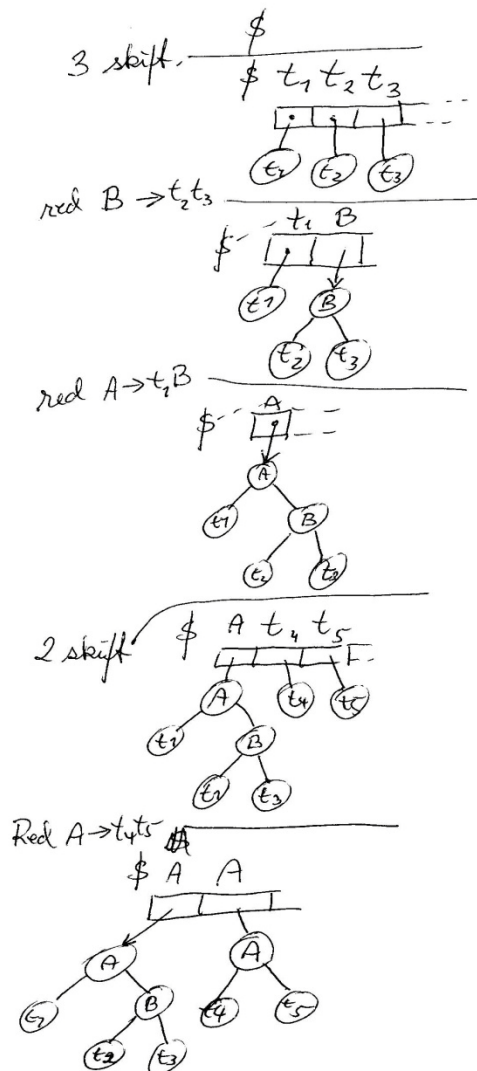
Ved skift:

Man skifter, og lager en ny node for den innkommende terminalen

Ved reduksjon:

Man lager en ny node for venstresiden i produksjonen og henger under denne det som lå tilsvarende høyresiden på stakken

Konklusjon: Etter hvert som man reduserer det tenkte treet dukker det opp igjen som et fysisk tre under stakken



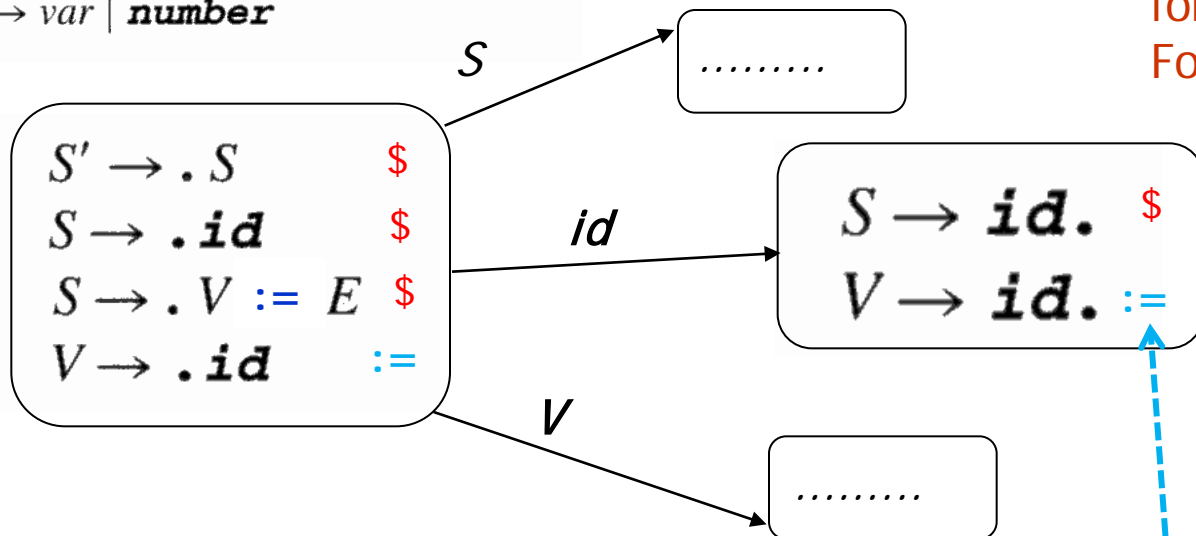
Ferdig

Grammatikk som **ikke** er SLR(1)

$stmt \rightarrow call-stmt \mid assign-stmt$
 $call-stmt \rightarrow identifier$
 $assign-stmt \rightarrow var := exp$
 $var \rightarrow var [exp] \mid identifier$
 $exp \rightarrow var \mid number$

$S \rightarrow id \mid V := E$
 $V \rightarrow id$
 $E \rightarrow V \mid n$

SLR(1): Gir her reduser/reduser-konflikt for input = \$. Se First og Follow under.



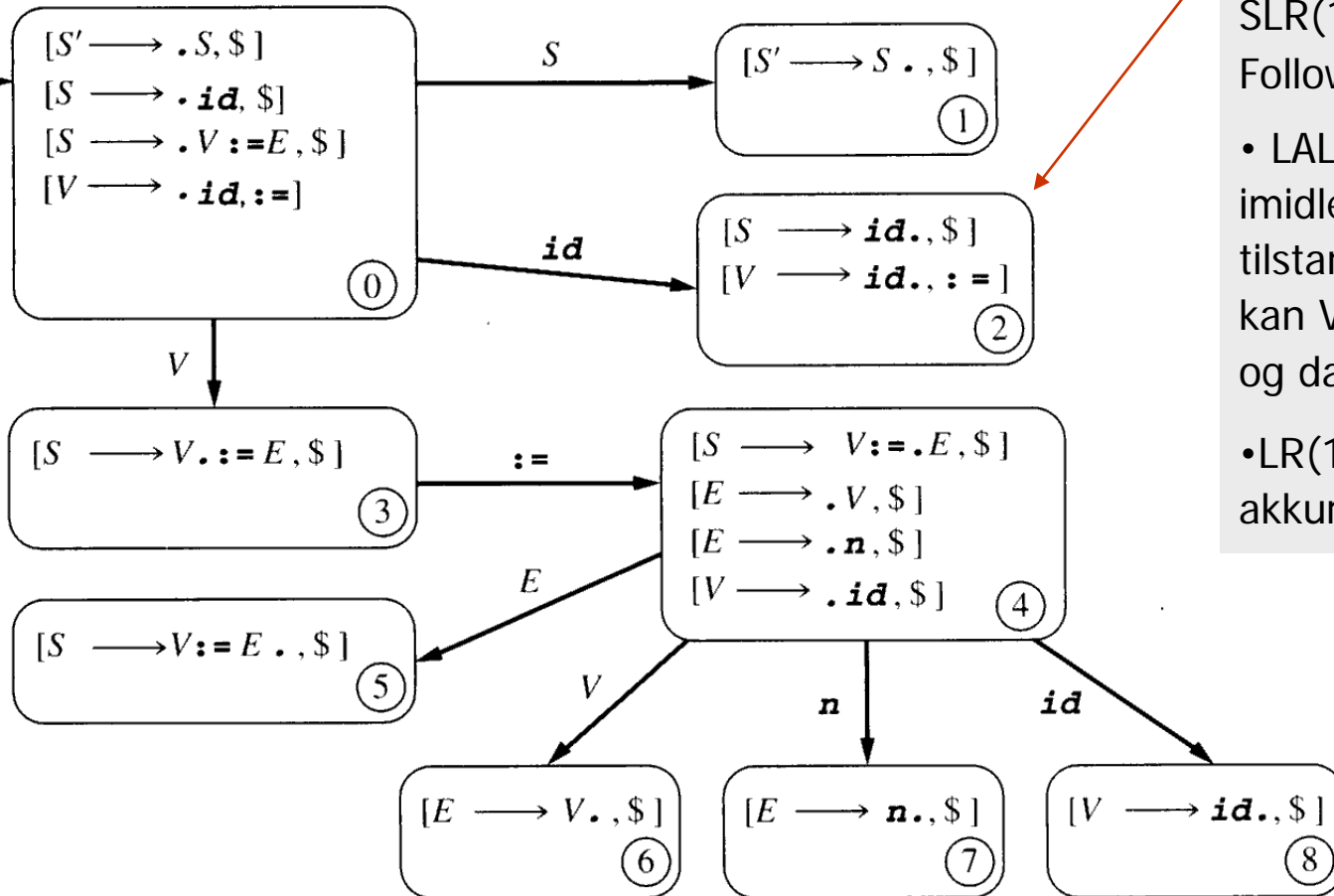
	First	Follow
S	id	\$
V	id	:=, \$
E	Id, n	\$

- **Men:** Det viser seg at vi kan løse denne ved å være nøyere med "etterfølgermengder" *under konstruksjon* av DFA'en. Da får vi her bare := som etterfølgemengde for $V \rightarrow id$. Over, og da er det ingen konflikt!
- Vi sier da at vi finner *LALR(1)-DFA'en*

LALR(1)-DFA (og LR(1)-DFA) for gram. på forrige side. Da løser det seg i tilstand 2, og i alle andre tilstander.

Pensum: Skal vite at LRLR(1) og LR(1)-itemer ser slik ut:

$[A \rightarrow \alpha . \beta , x]$



- Her var det konflikt ved SLR(1), siden $\text{Follow}(V) = \{ :=, \$ \}$.
- LALR(1)-betraktning viser imidlertid at i denne tilstanden (sammenhengen) kan V bare følges av $\{ := \}$, og da har vi ikke problemer.
- LR(1)-betraktning gir akkurat den samme DFA'en

Her er situasjonen hvor V kan følges av \$, men her er det ikke problematisk.

Grammatikken er altså LALR(1) (og dermed også LR(1)!))

Full LR(1)-parsering (Ikke pensum)

(det beste vi kan få til med én "Lookahead")

Hovedidé: Vi vil, under oppsett av NFA og DFA, ha LR(1)-itemer som fra starten inneholder et etterfølger-symbol.

OG: Tilstander er bare like om alle itemene også har samme etterfølger-symbol. Derfor MANGE tilstander.

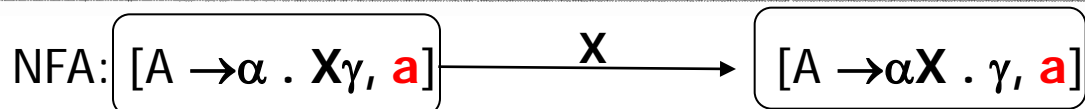
$a = \text{"look-ahead"}$

LR(1)-itemer:

$[A \rightarrow \alpha \cdot \beta, a]$

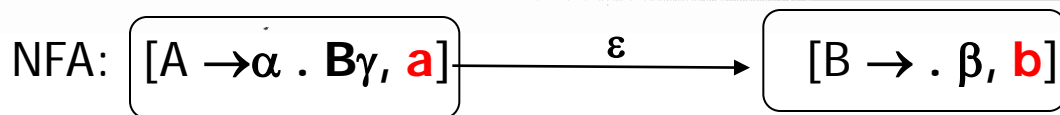
Angir at a kan komme etter $A \rightarrow \alpha\beta$ i den aktuelle sammenhengen. Altså at a kan komme bak *hele* $\alpha\beta$, (og *ikke* bak punktumet, med mindre $\beta = \epsilon$)

Definition of LR(1) transitions (part 1). Given an LR(1) item $[A \rightarrow \alpha.X\gamma, a]$, where X is any symbol (terminal or nonterminal), there is a transition on X to the item $[A \rightarrow \alpha X.\gamma, a]$.



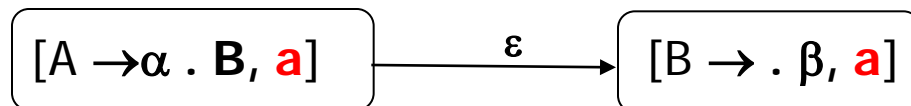
Vi flytter oss ett hakk framover i høyresiden, men produksjonen forblir i samme sammenheng

Definition of LR(1) transitions (part 2). Given an LR(1) item $[A \rightarrow \alpha.B\gamma, a]$, where B is a nonterminal, there are ϵ -transitions to items $[B \rightarrow \cdot\beta, b]$ for every production $B \rightarrow \beta$ and for every token b in $\text{First}(\gamma a)$.



For alle $B \rightarrow \beta_1 \mid \beta_2 \mid \dots$
og alle $b \in \text{First}(\gamma a)$

Spesialtilfelle, inkludert i det over ($\gamma = \epsilon$):



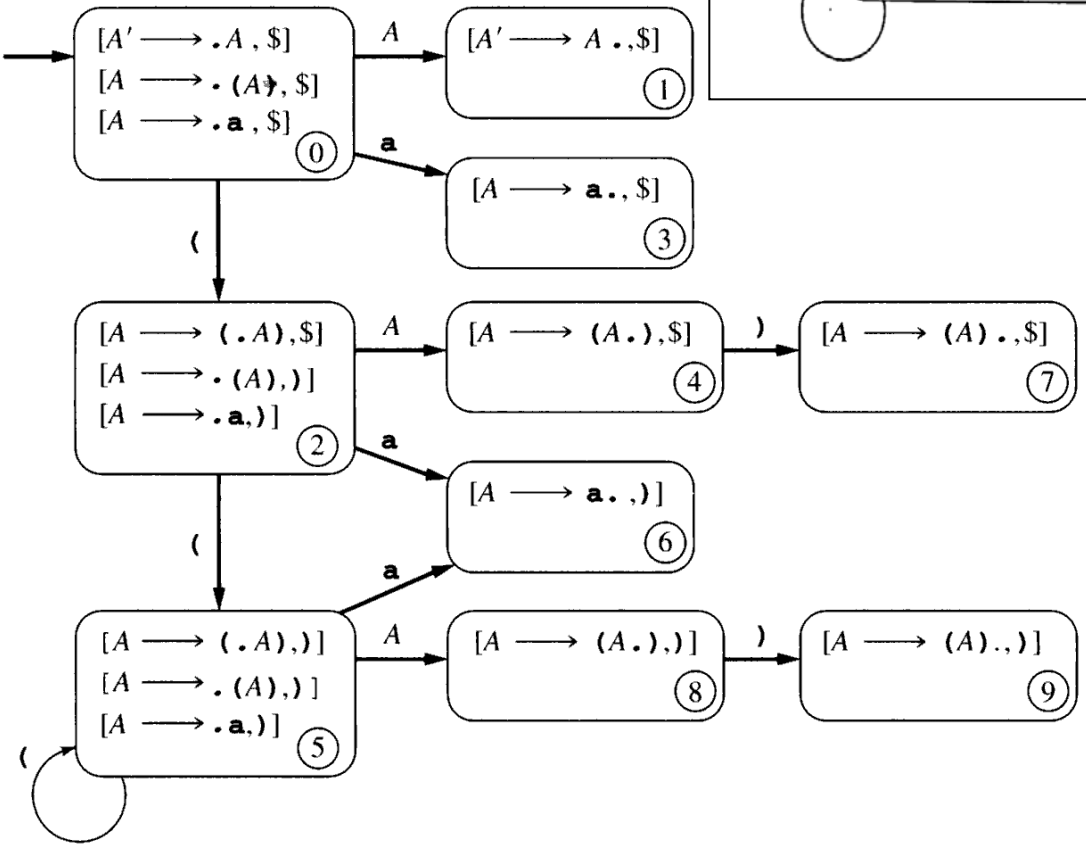
For alle $B \rightarrow \beta_1 \mid \beta_2 \mid \dots$

Ikke pensum

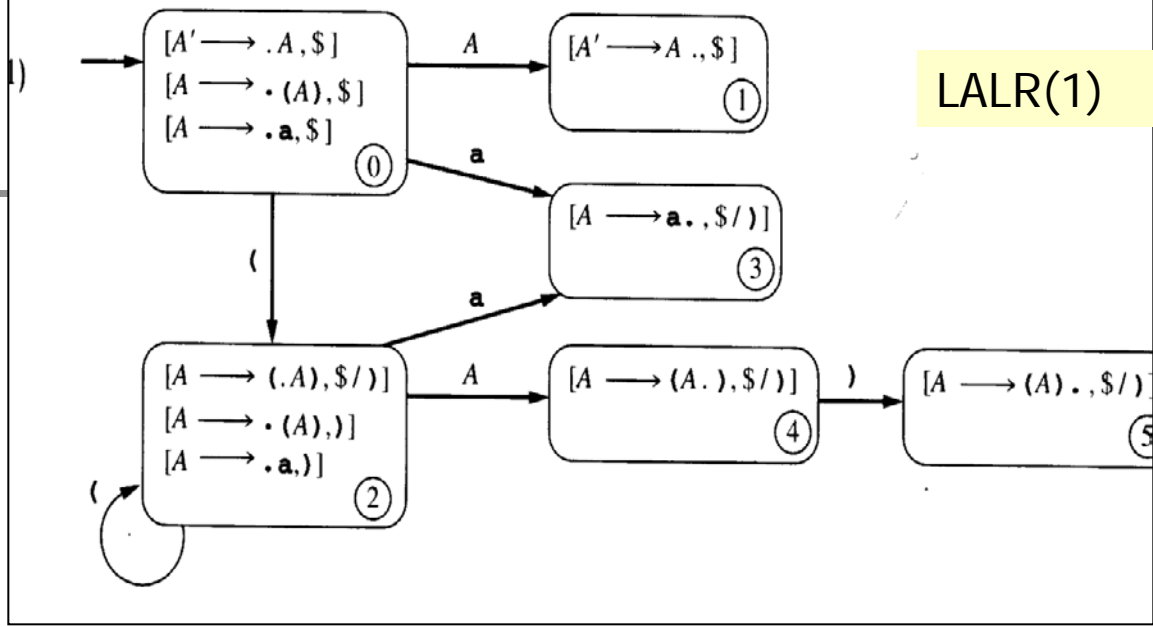
Sammenligning av LR(1)- og LALR(1)-DFA'er for samme grammatikk:

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

LR(1)



LALR(1)



Oppsett av LALR(1)-DFA'en ut fra LR(1)-DFA'en

Brukes ikke i praksis!

Ikke pensum

- "Core" (kjerne) av en LR(1)-tilstand:
 - Mengden av LR(0)-itemer, når man ser bort fra "look-ahead" itemene.
 - Merk at vi kan ha både $[A \rightarrow \alpha.\beta, a]$ og $[A \rightarrow \alpha.\beta, b]$. Dette gir bare ett LR(0)-item i kjernen, nemlig: $A \rightarrow \alpha.\beta$
- Observasjoner:
 - Kjernen i alle LR(1)-DFA-tilstander er en LR(0)-DFA-tilstand (for samme grammatikken)
 - To LR(1)-tilstander med samme kjerne har samme kanter ut, og tilsvarende utkanter fører til tilstander med samme kjerne.
- LALR(1)-DFA'en:
 - I LR(1)-DFA'en slår vi sammen alle tilstander med samme kjerne.
 - Ut fra observasjonen 2 over får vi da også konsistente kanter mellom disse tilstandene.
 - Vi sitter rett og slett med LR(0)-DFA'en
 - Men med det tillegg at det etter hvert item er satt på lookahead-symboler som er *unionen* av det som var i tilstandene som ble slått sammen.
 - Det *kan* da altså hende at lookahead-mengden i et slutt-item $[A \rightarrow \alpha. , a b c \dots]$ i LALR(1)-DFA'en er mindre enn etterfølgermengden til A , og dette gir større muligheter til å løse konflikter enn ved SLR(1)-betrakninger.

Oppsummering om LR-parsering (bottom-up)

OG: Det som står her om LALR(1) og LR(1) er det man skal vite om dette.

- Vi formulerer vår grammatikk som basal BNF
- Konflikter (f.eks. pga. flertydighet) kan løses med:
 - omredigere BNF'en (dog slik at den produserer samme språk!)
 - eller ved direktiver til CUP/Yacc/Bison (assosiativitet, presedens, etc.)
 - eller løse det senere i semantisk analyse (er: "(a and b) + 3" en semantisk feil?)
 - NB: Ikke **alle** konflikter *kan* løses selv av LR(1) – skriv om! (eks: $A \rightarrow a \mid a A a$)
- De forskjellige varianter av LR-grammatikker, den ene sterkere enn den andre:

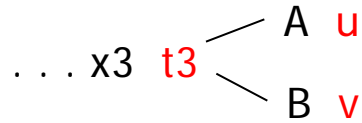
	Fordeler	Annet
LR(0)	Definerer DFA-tilstander som brukes av SLR og LALR	Mange (unødige) konflikter (red/red og skift/red) oppstår. Brukes neppe.
SLR(1)	Klar forbedring av LR(0), selv om den bare bruker det samme antall tilstander. Tabell typisk 50K felter	Ikke så god som LALR(1), men OK til det meste. Grei om man vil hånd-lage parser for liten grammatikk
LALR(1)	Nesten like bra som LR(1), men antall tilstander bare som for LR(0)	Brukes i de aller fleste automatiserte LR-parsere
LR(1)	Klarer alle grammatikker som teoretisk lar seg analysere ved én lookahead.	Svært mange tilstander (typisk tabell opp mot 1M felter for et reelt språk). Brukes?

Husk: Når tabellen først er satt opp, er algoritmen for parsering alltid den samme 22

Ved syntaksfeil:

"Panic-mode" for LR-parsing (det eneste vi skal se på)

Syntaksfeil:
fordi ruten $[t_5, i_1]$ er tom



\$ 0 x1 t1 x2 t2 x3 t3 (x4 t4 x5 t5 i1) i2 i3 ... \$

Fjernes

\$ 0 x1 t1 x2 t2 x3 t3 B v i2 7 i3 ... \$

	i1	i2	A	B
t3			u	v
t4	-		-	-
t5	-		-	-
u	-	r..		
v	-	s7		

1. Pop states from the parsing stack until a state is found with nonempty Goto entries. finner t3
2. If there is a legal action on the current input token i_1 from one of the Goto states, push that state onto the stack and restart the parse. If there are several such states, prefer a shift to a reduce. (u og v)
3. If there is no legal action on the current input token from one of the Goto states, advance the input until there is a legal action or the end of the input is reached.

Velger til slutt å pushe på B, som etter å ha fjernet i1 gir tilstand v

Eksempel: if-setning er mindre generell enn setning

} Ta vekk én og én input, og gjenta 2

Typisk Yacc-produsert parseringstabell

(merk påfyll av **ekstra** reduksjoner, som en plass-optimalisering i Yacc)

Grammatikk: command \rightarrow exp
 exp \rightarrow exp + term | exp - term | term
 term \rightarrow term * factor | factor
 factor \rightarrow NUMBER | (exp)

State	Input							Goto			
	NUMBER	(+	-	*)	\$	command	exp	term	factor
0	s5	s6						1	2	3	4
1							accept				
2	r1	r1	s7	s8	r1	r1	r1				
3	r4	r4	r4	r4	s9	r4	r4				
4	r6	r6	r6	r6	r6	r6	r6				
5	r7	r7	r7	r7	r7	r7	r7				
6	s5	s6							10	3	4
7	s5	s6								11	4
8	s5	s6								12	4
9	s5	s6									13
10			s7	s8		s14					
11	r2	r2	r2	r2	s9	r2	r2				
12	r3	r3	r3	r3	s9	r3	r3				
13	r5	r5	r5	r5	r5	r5	r5				
14	r8	r8	r8	r8	r8	r8	r8				

Panic-mode for LR-parsing kan gå i evig løkke

Vi bruker følgende grammatikk:

$command \rightarrow exp$
 $exp \rightarrow exp + term \mid exp - term \mid term$
 $term \rightarrow term * factor \mid factor$
 $factor \rightarrow NUMBER \mid (exp)$

samt parsingstabellen som Yacc produserte for denne (tidligere lysark)

Parsing med feil input "(n n)":

```
$ 0          ( n n ) $
$ 0 ( 6      n n ) $
$ 0 ( 6 n 5   n ) $
$ 0 ( 6 F 4   n ) $
$ 0 ( 6 T 3   n ) $
$ 0 ( 6 E 10  n ) $
$ 0 ( 6 F 4   n ) $
... gjentar seg selv
```

Feil, siden [10, n] er tomt. Videre:

- 10 har ingen goto, så E 10 poppes av
- 6 har goto for

E	T	F
---	---	---
- ville gå til tilstand

10	3	4
----	---	---
- ved input n gir dette

-	r4	r6
---	----	----

(ingen skift, dessverre!)
- Av T og F velger vi F, som er den minst generelle, og pusher derfor på F 4

Men da er vi tilbake til en tilstand vi har vært i (uten å ha lest noe input), og ting vil da bare gjenta seg selv.

Mulig løsning (meget løslig):

- Hold greie på om du kommer tilbake til samme tilstand, og gjør noe spesielt:
- Ta da mer bort fra stakken, og forsøk igjen
- Kanskje: *Forlange* en skift-mulighet for å sette i gang igjen