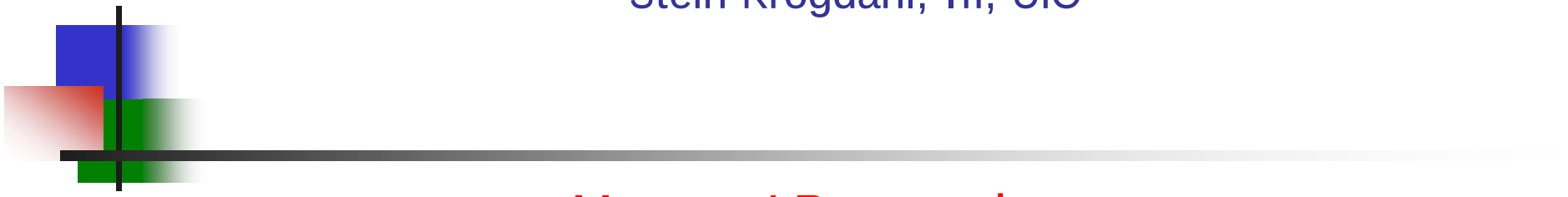
INF5110 – 26/2-2014

Kap. 5, Del 2

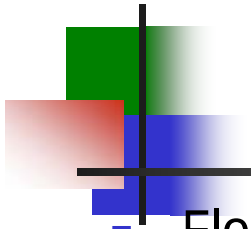
Stein Krogdahl, Ifi, UiO



Mer om LR-parsering

Har også igjen noen foiler fra sist

- NB: Oppgaver til kap 4 og 5 blir lagt ut på undervisningsplanen, og blir gjennomgått onsdag 5/3
- Begynner kanskje med oppgaver fredag 28/2 (beskjed gis)
- Oblig 1 er lagt ut.
- Det blir en intro til Oblig 1 ved Henning Berg fredag 28/2



Flertydige grammatikker og LR-parsering

- Flertydige grammatikker er aldri LR(0), SLR(1), LALR(1), LR(1), eller LR(k) for noen k. De er heller aldri LL(1)!
- Dette kommer av at om en grammatikk er flertydig så må det på et punkt i parseringen av en flertydig setning være flere «riktige» valg, men slik er det jo ikke for LR(k)- eller LL(k)-grammatikker
- 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-grammatikker: Angi presedens, assosiativitet e.l.
  - For "dangeling 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 → *if-stmt* | **other**

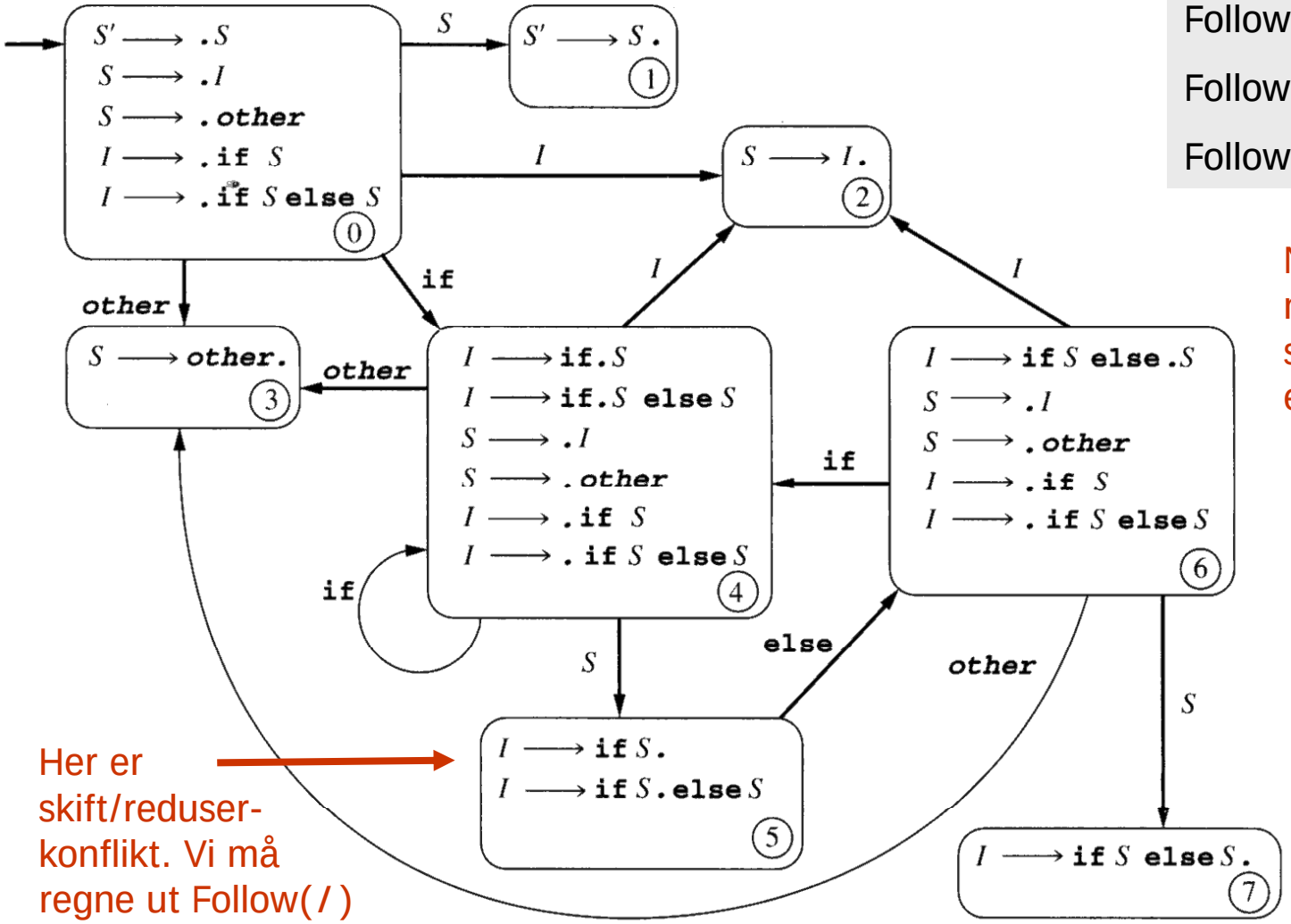
if-stmt → **if (** *exp* **)** *statement*

| **if (** *exp* **)** *statement* **else** *statement*

exp → **0** | **1**

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

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



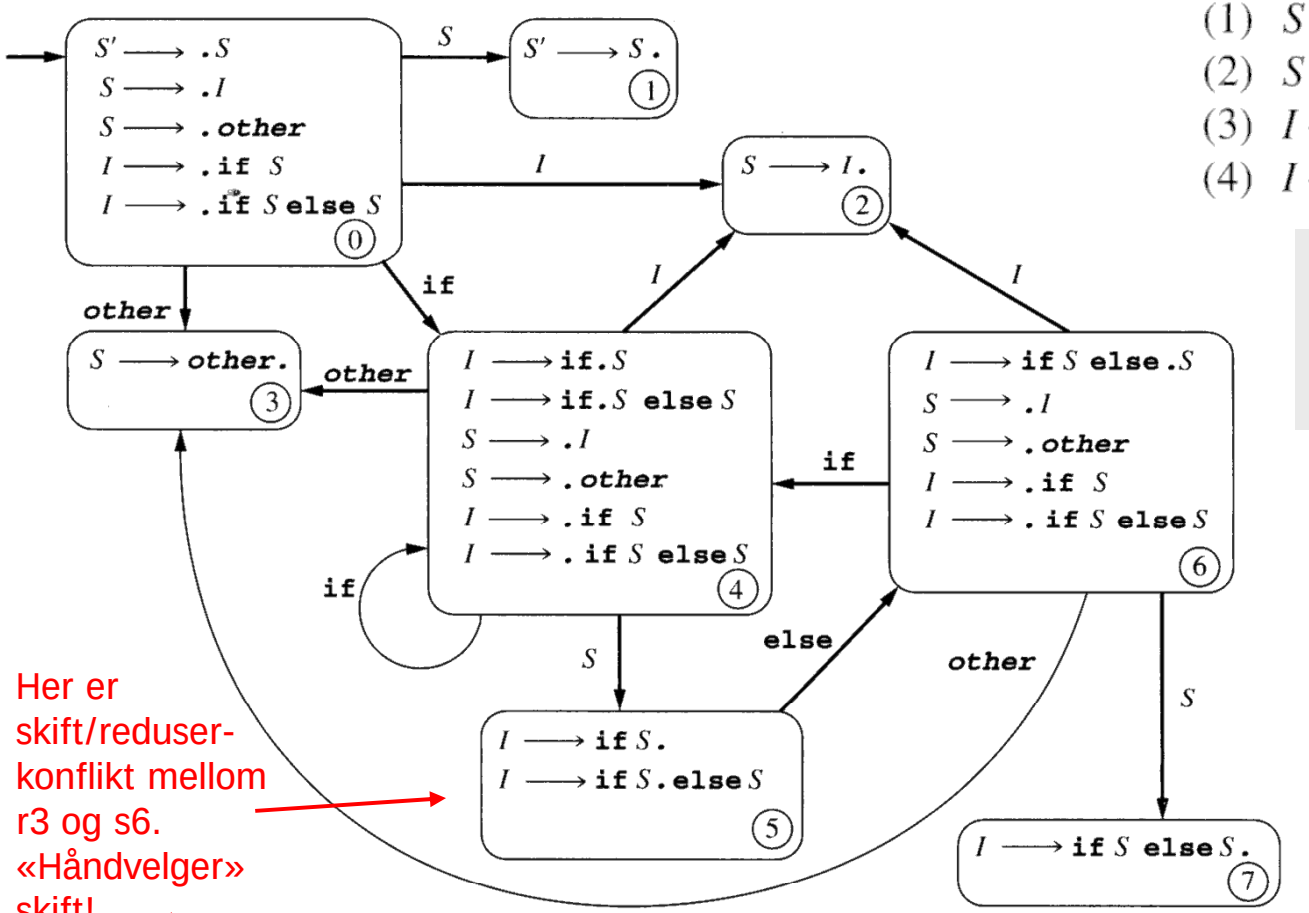
Follow(S) = { $, **else** }

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(/)



(1) $S \rightarrow I$
(2) $S \rightarrow$ **other**
(3) $I \rightarrow$ **if** S
(4) $I \rightarrow$ **if** S **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 og Goto-tabellen gir da ny topptilstand

Her er skift/reduser-konflikt mellom r3 og s6. «Håndvelger» 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

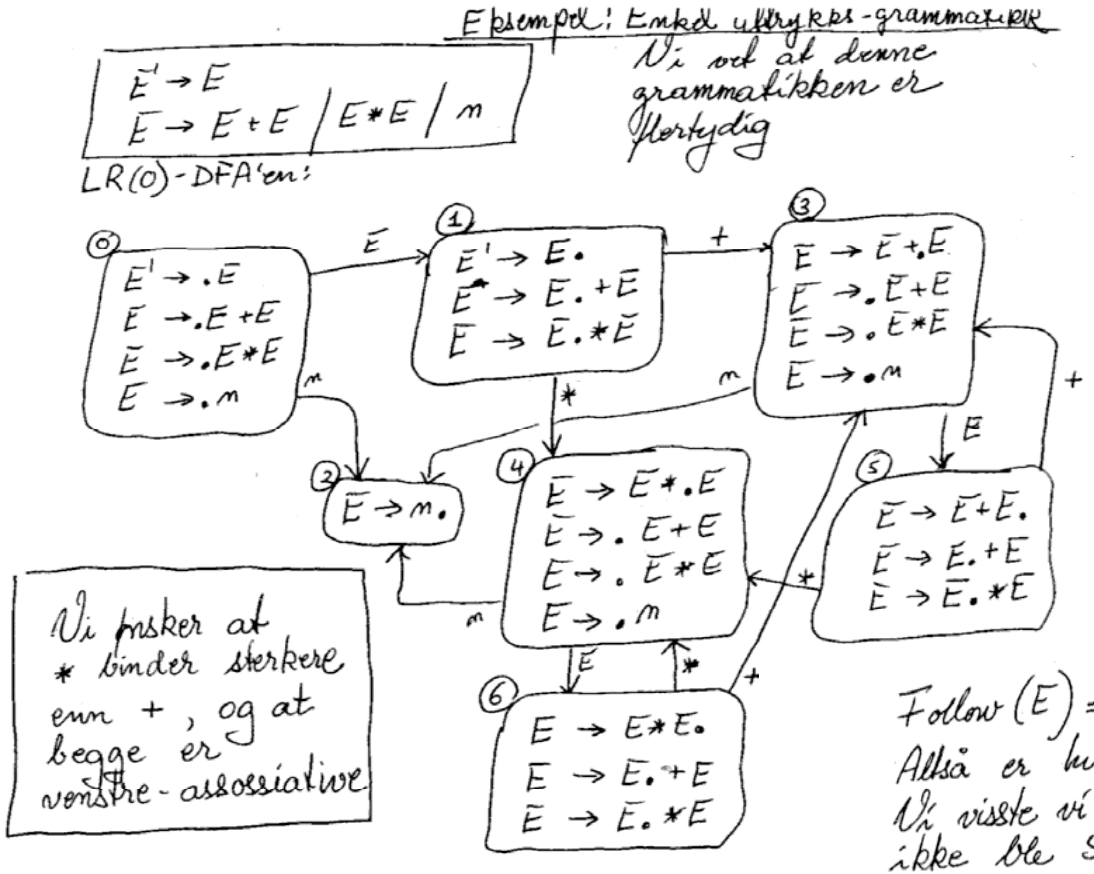
State	Input				Goto	
	if	**else**	**other**	**$**	*S*	*I*
0	s4		s3		1	2
1				accept		
2		r1		r1		
3		r2		r2		
4	s4		s3		5	2
5		s6		r3		
6	s4		s3		7	2
7		r4		r4		

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

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



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

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

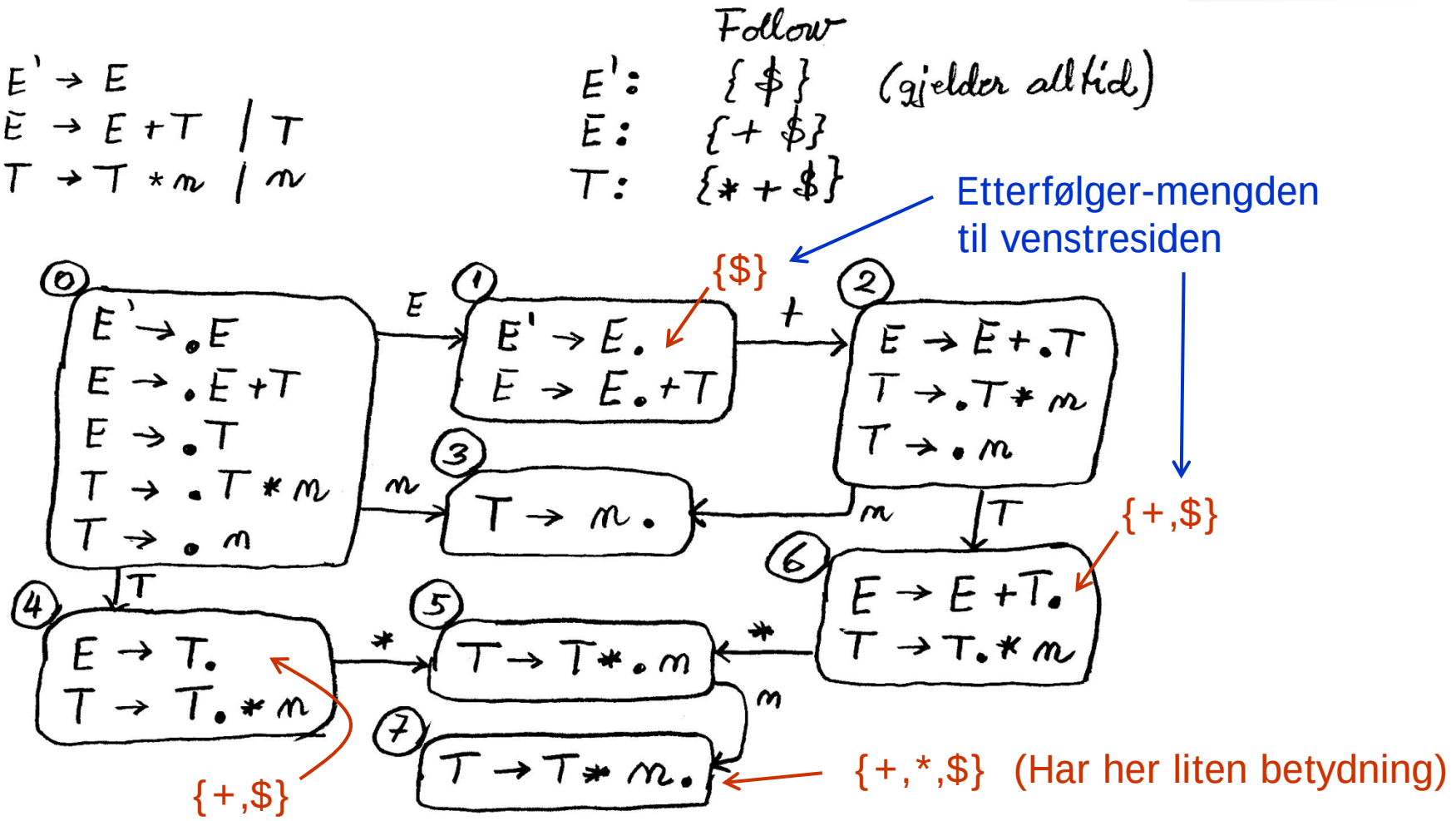
Tilstand 6: Stakk= .....E*E Input=
$: reduser, fordi skift ikke lovlig for $
+: reduser, fordi * har presedens over +
*: reduser, fordi * er venstreassossiativ)

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

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

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



- 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!

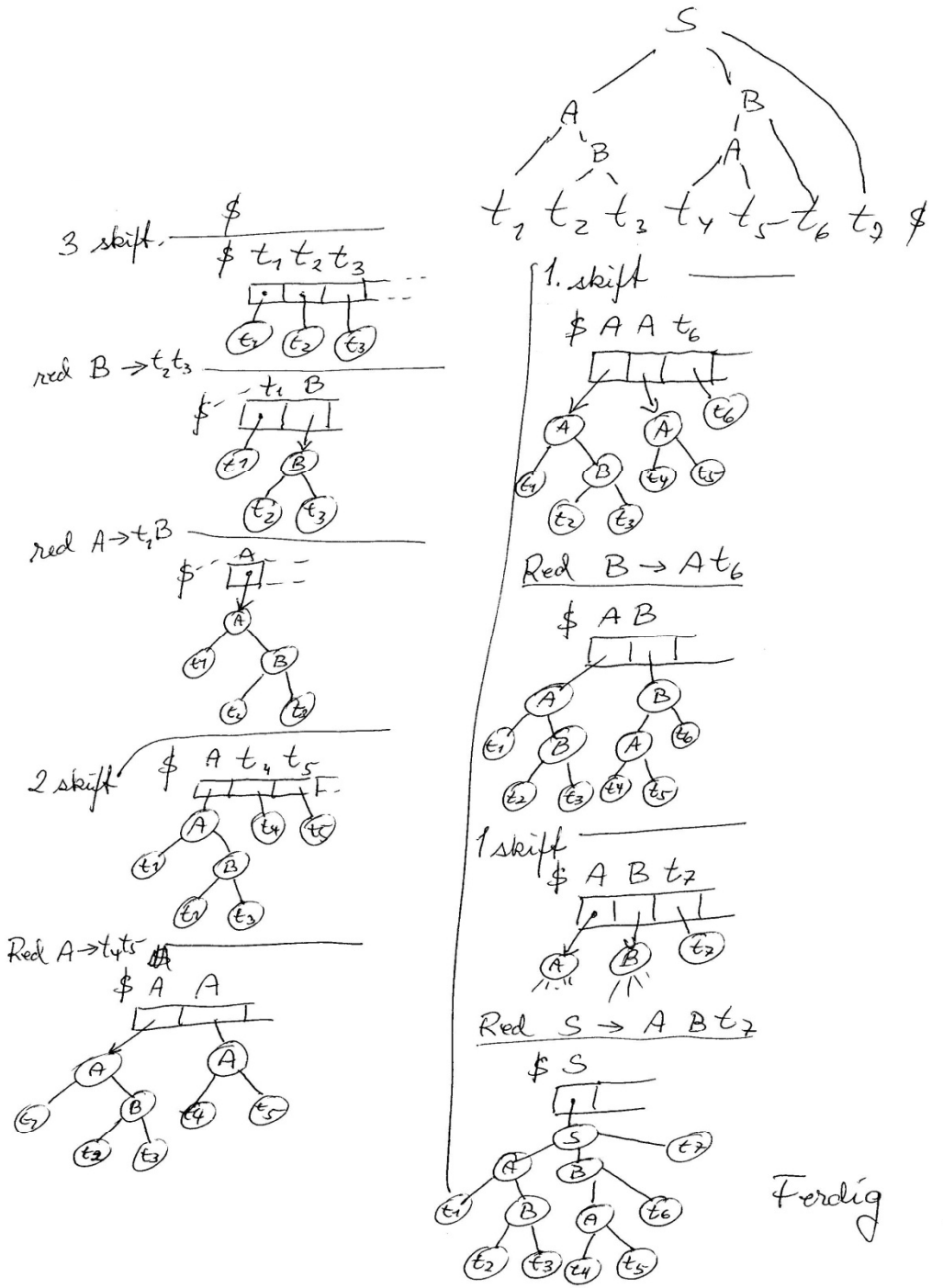
Ved skift:

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

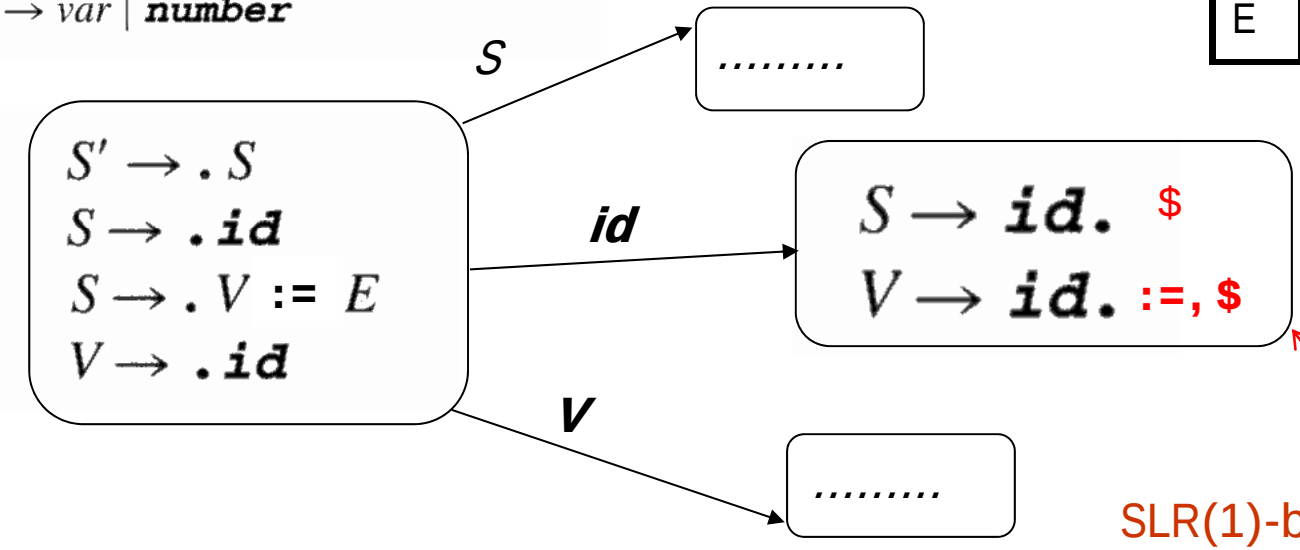


Grammatikk som ikke er SLR(1)

stmt → *call-stmt* | *assign-stmt*
call-stmt → **identifier**
assign-stmt → *var* **:=** *exp*
var → *var* **[** *exp* **]** | **identifier**
exp → *var* | **number**

$S \rightarrow \mathbf{id} \mid V \mathbf{:=} E$
$V \rightarrow \mathbf{id}$
$E \rightarrow V \mid \mathbf{n}$

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



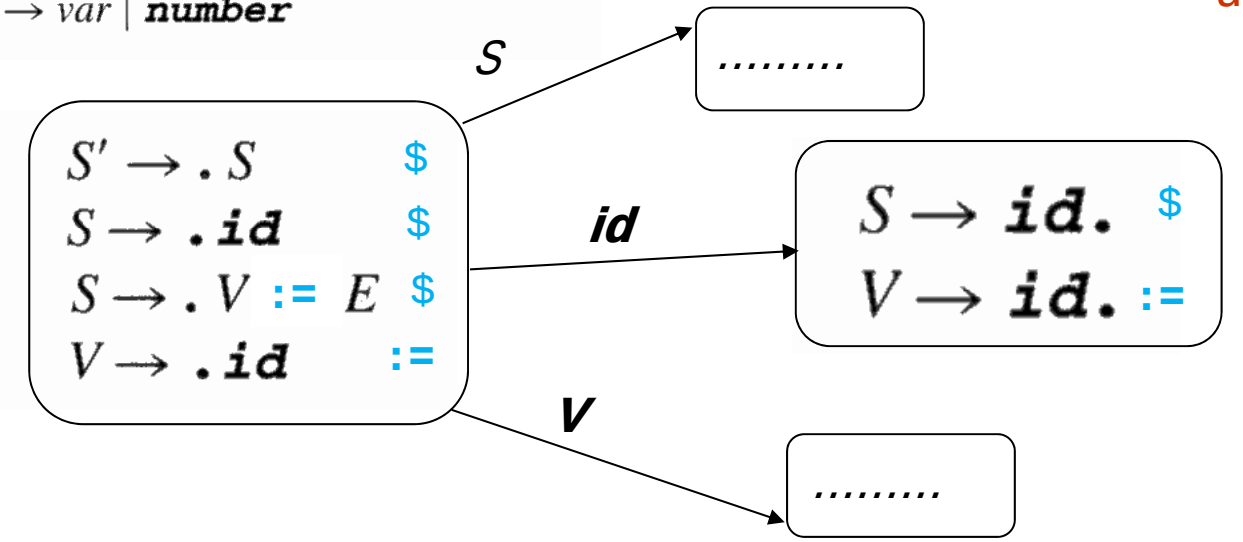
SLR(1)-betrakning: Gir her reduser/reduser-konflikt for input = $. Se First og Follow over.

Men situasjonen kan her reddes

stmt → *call-stmt* | *assign-stmt*
call-stmt → **identifier**
assign-stmt → *var* **:=** *exp*
var → *var* **[** *exp* **]** | **identifier**
exp → *var* | **number**

$S \rightarrow \mathbf{id} \mid V \mathbf{:=} E$
$V \rightarrow \mathbf{id}$
$E \rightarrow V \mid \mathbf{n}$

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



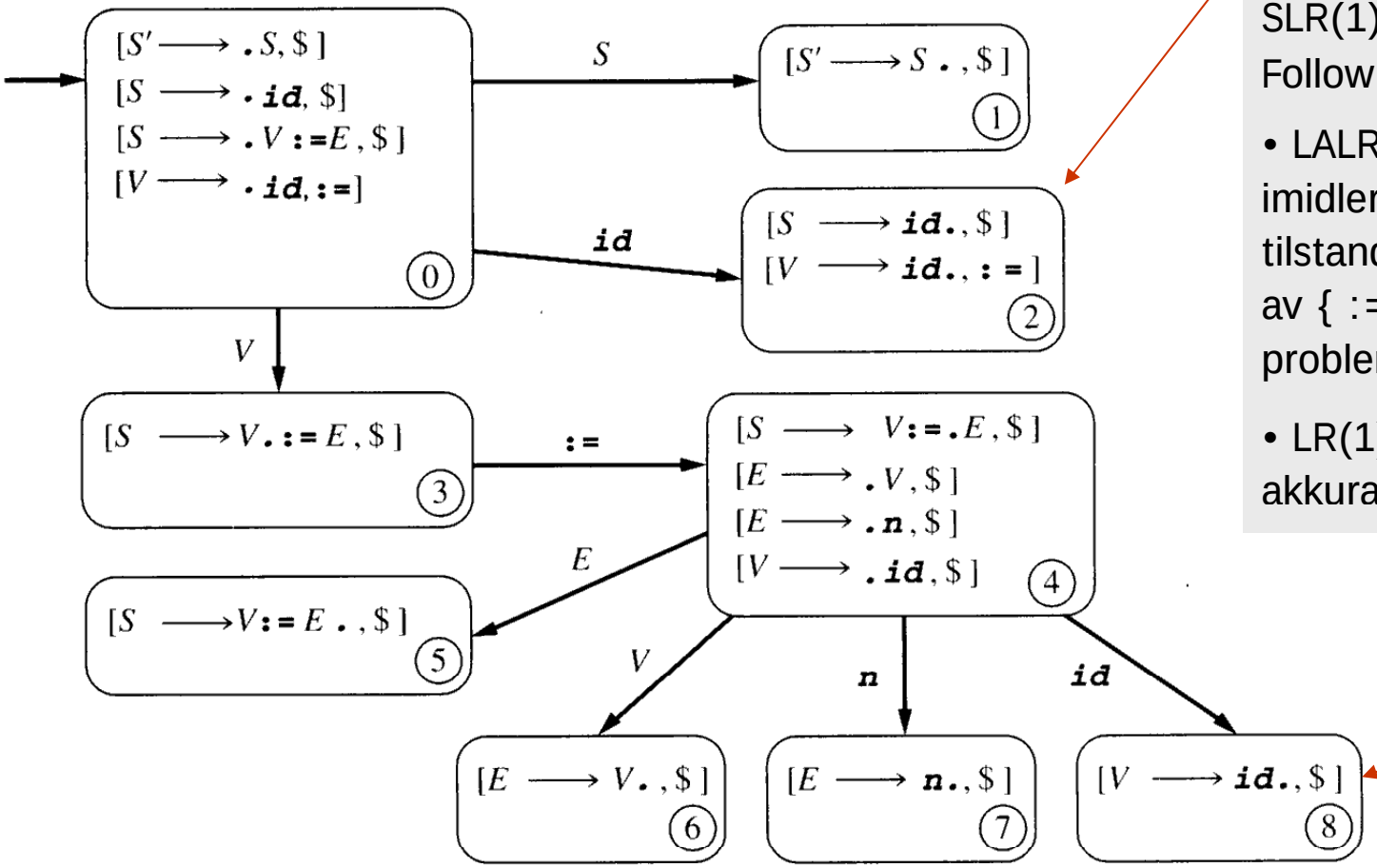
	First	Follow
S	id	$
V	id	:=, $
E	id, n	$

- 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 etterfølge-mengde for V -> id. over, og da er det ingen konflikt!
- Vi sier da at vi finner *LALR(1)-DFA’en* til dette språket. Da er det en etterfølgermengde bak hvert item i hver tilstand

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 LALR(1) og LR(1)-itemer ser slik ut:

[A -> α. β , x]



- Her var det konflikt ved SLR(1), siden Follow(V) = { :=, $}.
- LALR(1)-betraktning viser imidlertid at i denne tilstanden kan V bare følges av { :=}, og da har vi ikke problemer.
- LR(1)-betrakning 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")

a = "*look-ahead*"

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-symboler. Derfor MANGE tilstander.

LR(1)-itemer:

$[A \rightarrow \alpha \,.\, \beta \,,\, \mathbf{a}]$

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 $\beta = \varepsilon$)

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 $[A \rightarrow \alpha X . \gamma, a]$.

NFA: $[A \rightarrow \alpha \,.\, \mathbf{X}\gamma, \mathbf{a}] \xrightarrow{\mathbf{X}} [A \rightarrow \alpha \mathbf{X} \,.\, \gamma, \mathbf{a}]$

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 $[B \rightarrow .\beta, b]$ for every production $B \rightarrow \beta$ and *for every token* b in First(γa).

NFA: $[A \rightarrow \alpha \,.\, \mathbf{B}\gamma, \mathbf{a}] \xrightarrow{\varepsilon} [B \rightarrow \,.\, \beta, \mathbf{b}]$

For alle
$B \rightarrow \beta_1 \mid \beta_2 \mid ..$
og alle
$b \in$ First(γa)

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

$[A \rightarrow \alpha \,.\, \mathbf{B}, \mathbf{a}] \xrightarrow{\varepsilon} [B \rightarrow \,.\, \beta, \mathbf{a}]$

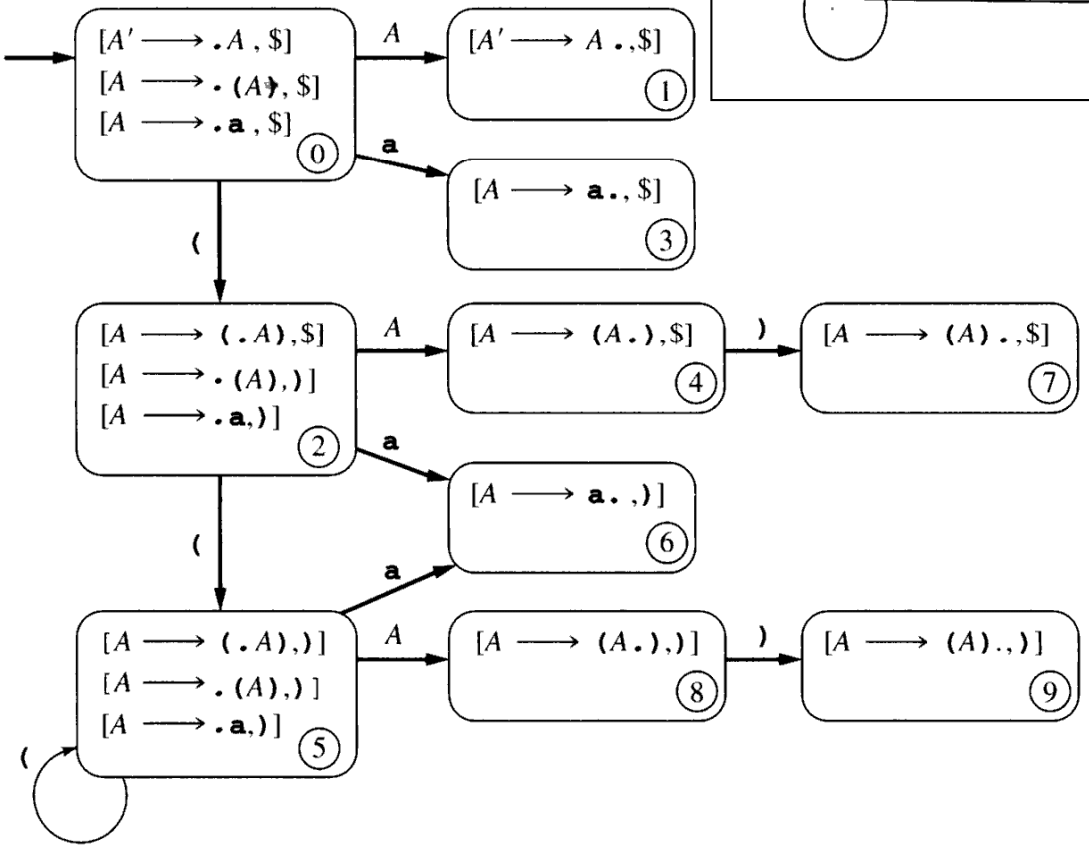
For alle $B \rightarrow \beta_1 \mid \beta_2 \mid ..$

Ikke pensum

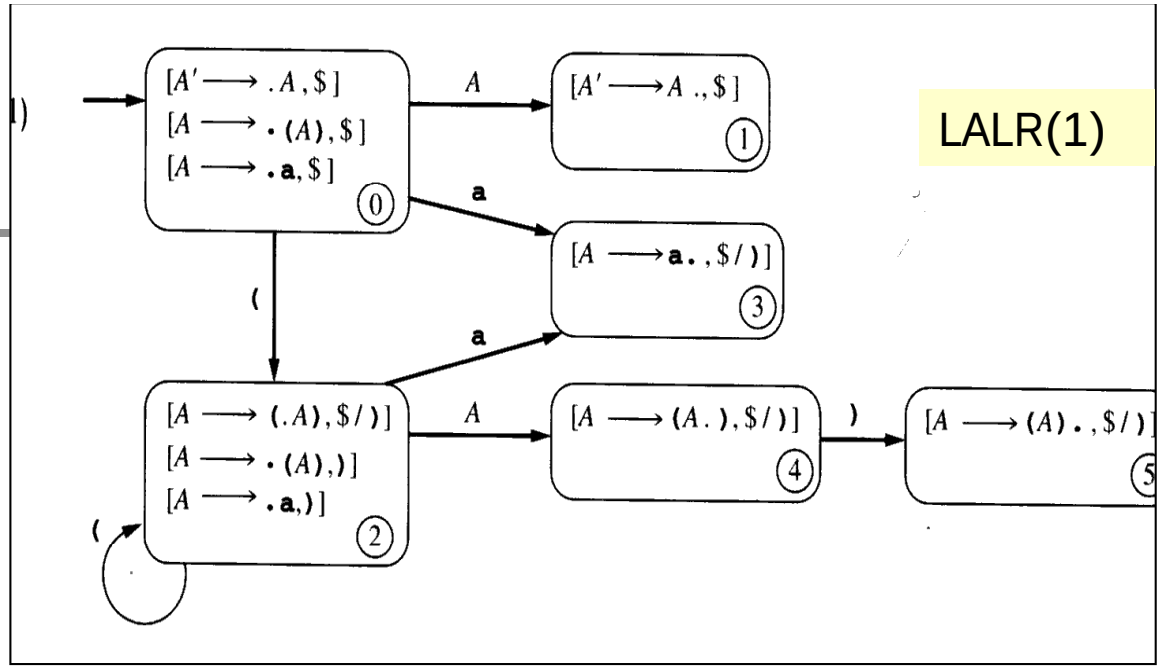
Sammenligning av LR(1)- og LALR(1)-DFA'er for samme grammatikk:

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

LR(1)



LALR(1)

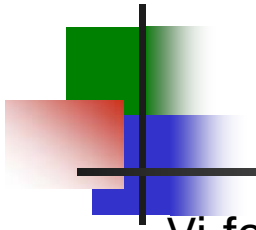


Vi har her slått sammen alle tilstander med samme «core» (se neste side). Antall tilstander blir da som for LR(0)-DFA-en

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

Denne metoden 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 →α.β, a]** og **[A →α.β, b]** . Dette gir bare ett LR(0)-item i kjernen, nemlig : **A →α.β**
- 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 ut-kanter 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 →α. , a b c ...]** 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.



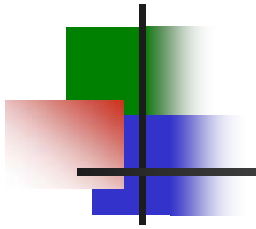
Oppsummering om LR-parsering (bottom-up)

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 (pga. flertydighet eller annet) 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 → a | 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?

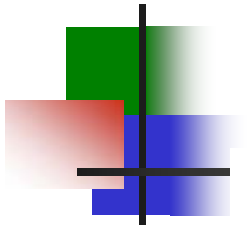
Husk: Når tabellen først er satt opp, er algoritmen for parsering alltid den samme



Når parseren oppdager feil

Foiler opprinnelig fra kap. 4, men vi utsatte dem.

- Minstekrav:
  - Tester løpende at programmet er OK, og gir fornuftig feilmelding ved feil (men stopper kanskje)
- Vanlig krav ved feil (”error recovery”):
  - At man gir fornuftig feilmelding.
  - ”Blar forbi feilen” til et sted man kan fortsette parseringen (ønske: bla forbi så lite som mulig!).
  - Ønsker også å ta opp tråden etter semantiske feil
  - Men det er etter *syntaksfeil* det er vanskeligst å ta opp tråden igjen.



Feilmeldinger m.m.

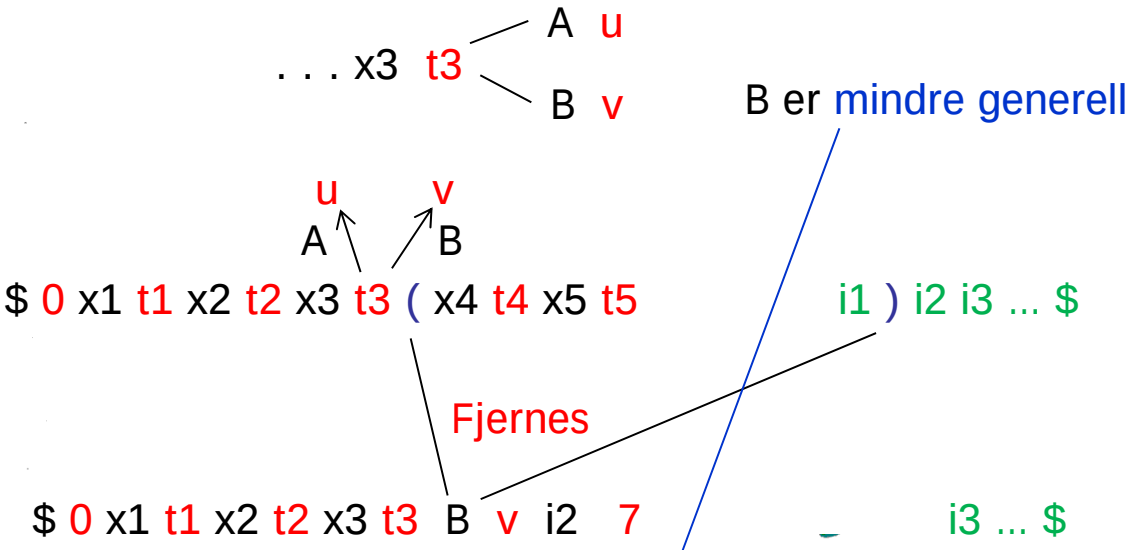
- Viktig:
  - Forsøke å unngå feilmeldinger som bare er følgefeil
  - Rapportere feil så tidlig som mulig, helst så snart det man har lest *ikke kan forlenges til et riktig program*
  - Man må passe på at man ikke blir gående i løkke *uten å lese noe fra input* (verken med eller uten rapportering feil)

- Hvilken feilmelding skal man gi? (eksempel fra LL(1)-analyse)
  - Anta at man ved ”factor” har valgt alternativet ” (*exp*) ”, og at man, når exp-metoden returnerer, ikke finner en ”) ”.
  - Hvilken feilmelding skal man da gi??
  - Man kan melde: ”Sluttparentes mangler”
  - Men dette kan ofte være forvirrende, f.eks. om det som stod var: (a + b c).
  - Her vil exp-metoden stoppe etter ”(a + b” med ”c” i «token», og det vil være forvirrende her med feilmeldingen ”Sluttparentes mangler”
  - Man bør derfor heller gi meldingen: ”Noe galt i et uttrykk, eller sluttparentes mangler”

Ved syntaksfeil under LR-parsering:

”Panic-mode” for LR-parsering (det eneste vi skal se på)

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



B er mindre generell

		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. Among the reduce actions, prefer one whose associated nonterminal is least general. — (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

Ta vekk én og én input, og gjenta 2

Eksempel: if-setning er mindre generell enn setning

Typisk Yacc-produsert parseringstabell

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

Grammatikk:

command → exp

exp → exp + term | exp – term | term

term → term * factor | factor

factor → 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-parsering kan gå i evig løkke

Vi bruker følgende grammatikk:

command → exp
exp → exp + term | exp – term | term
*term → term * factor | factor*
factor → NUMBER | (exp)

samt parseringstabellen som Yacc produserte for denne (tidligere lysark)

Parsering 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