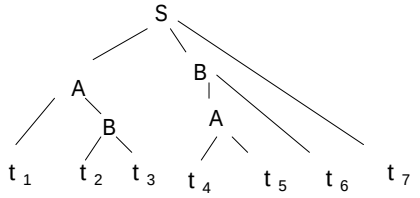
Kap. 5 del 1 –
Intro til ”parsering nedenfra-og-opp” samt LR(0) og SLR(1) –grammatikker
INF5110 – v2006

Arne Maus,
Ifi UiO

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



LR-parsering og grammatikker

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

-Autimatisert

-YACC, Bison (LALR(1))

Prinsippet med LR-parsering

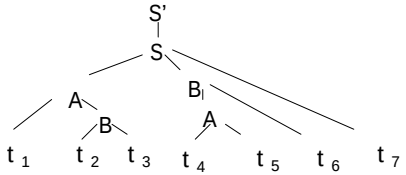
$S' \rightarrow S$

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

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

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

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



LR-parsering :

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

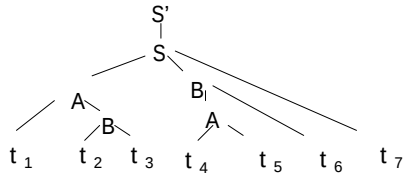
Prinsippet med LR-parsering II

$S' \rightarrow S$

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

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

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



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

- Ha en ”stakk” for det som er lest
- Gjør ”reduksjonen av et subtre når subtreet ligger ”på toppen av” stakken.
- Da erstatter vi det på stakken med den ikke-terminalen (som produserte dette subtreet)

Start-situasjonen:

$ $t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ \$$

stakk input

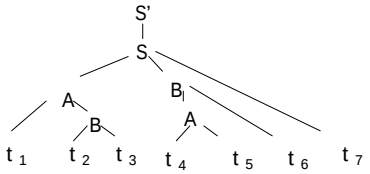
Slutt-situasjonen:

$ 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 | ....



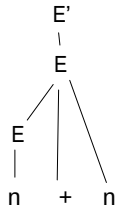
stakk	input
$	t_1 t_2 t_3 t_4 t_5 t_6 $t_7$$
$ t_1	t_2 t_3 t_4 t_5 t_6 t_7 $
$ t_1 t_2	t_3 t_4 t_5 t_6 t_7 $
$ t_1 t_2 t_3	t_4 t_5 t_6 t_7 $
$ t_1 B	t_4 t_5 t_6 t_7 $
$ A	t_4 t_5 t_6 t_7 $
$ A t_4	t_5 t_6 $t_7$$
$ A t_4 t_5	t_6 t_7 $
$ A A	t_6 t_7 $
$ A A t_6	t_7 $
$ A B	t_7 $
$ A B t_7	$
$ S	$

- Stakk + input: Høyre-avledninger i omvendt rekkefølge
- Får to typer steg:
  - Reduksjon (på toppen av stakken med A → α)
  - Lesing ("skift") av input til stakken
- Dersom man kjenner syntakstreet, er det lett å angi de rette stegene.
- MEN: Hvordan gjøre dette underveis uten å kjenne resten av input ??

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

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

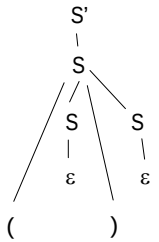


Boka: (Stakk + input) utgjør et stadium i en høyre-avledning. Den neste reduksjonen som skal gjøres, kalles situasjonens "handle" (håndtak).

	Parsing stack	Input	Action
1	$	n + n $	shift
2	$ n	+ n $	reduce E → n
3	$ E	+ n $	shift
4	$ E +	n $	shift
5	$ E + n	$	reduce E → E + n
6	$ E	$	reduce E' → E
7	$ E'	$	accept

Eksempel på LR-parsering

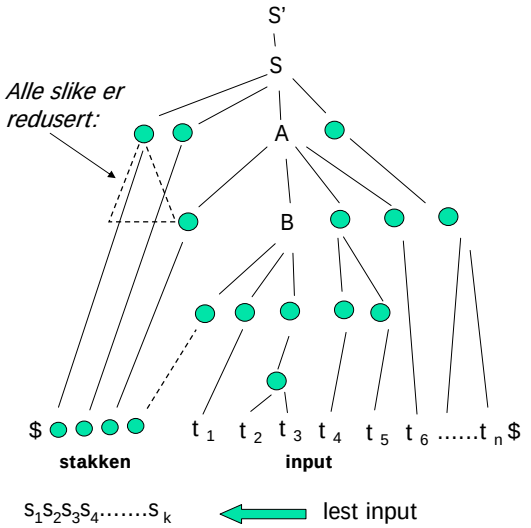
$S' \rightarrow S$
$S \rightarrow (S) \ S \mid \varepsilon$



NB: S → ε
blir litt "rart"

	Parsing stack	Input	Action
1	$	( ) $	shift
2	$ (	) $	reduce S → ε
3	$ (S	) $	shift
4	$ (S)	$	reduce S → ε
5	$ (S) S	$	reduce S → (S) S
6	$ S	$	reduce S' → S
7	$ S'	$	accept

Typisk situasjon under LR-parsering



Gitt entydig grammatikk G.

Denne behandles som følger for å lage en LR-parser:
(Ikke alt er like tydelig beskrevet i Boka, men bare boka er pensum)

- Vi ser på de mulig stakker som kan opptre, og ser disse som strenger over alfabetet{terminaler, ikke-terminaler}. Vi ser altså på:
  {S | S utgjør stakken på ett eller annet stadium i LR-parsering av en setning i L(G)}
- Dette språket viser seg å være regulært, og kan beskrives av en NFA der:
  - Tilstandene angis av "itemer" av formen: A →XY.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 hovedprinsipp (uformelt):**
  - Gitt en lovlig stakk og anta at den fører DFA-en til en (aksepterende) tilstand T:
  - Da vil mengden av *itemer* i T angi de mulige "lokale forhold" vi, i parseringen nå, kan ha ut fra dette stakk-innholdet (det kan være flere muligheter/valg, siden vi ikke har tatt hensyn til resten av input).

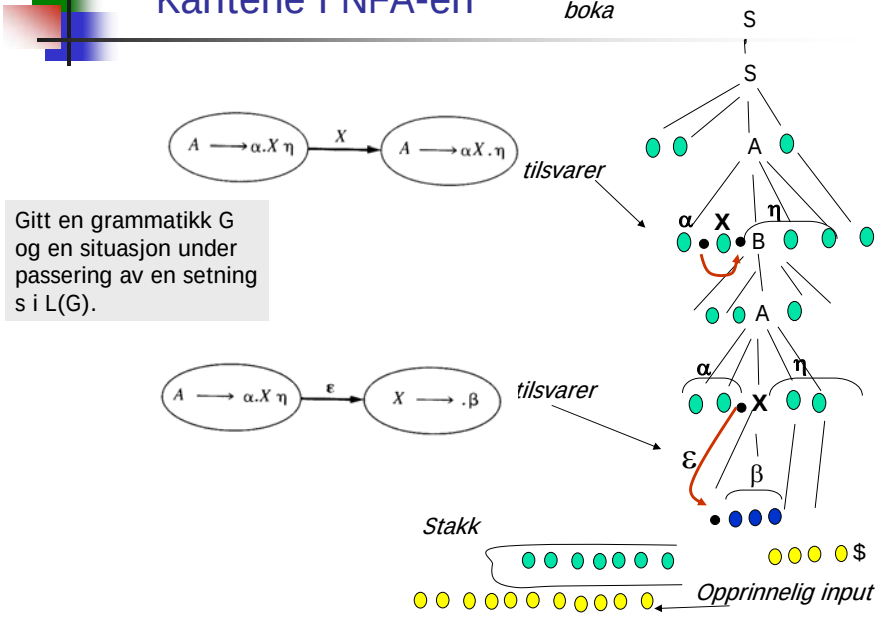
Items: Alle produksjonene i BNF-grammatikken omarbeides

- I stedenfor :
  $A \rightarrow \alpha X \beta$
- Legger vi inn nye produksjoner med punktum på alle plasser (punktumet er et Meta-symbol):
  $A \rightarrow .\alpha X \beta$
  $A \rightarrow \alpha . X \beta$
  $A \rightarrow \alpha X . \beta$
  $A \rightarrow \alpha X \beta .$
- Punktumet sier grovt sett hvilken del av den originale produksjonen vi prøver å jobbe med
  - 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

Disse produksjonene med punktum kalles ***items***

Kantene i NFA-en

Dette er ikke fullt forklart i boka



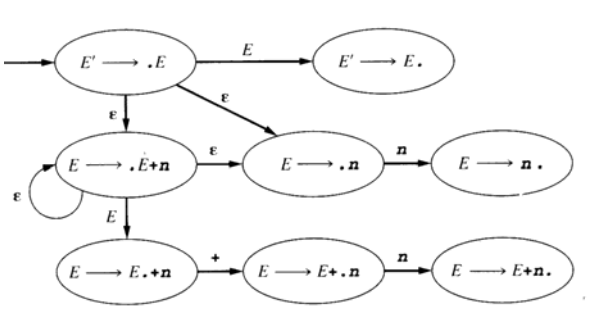
Gitt en grammatikk G og en situasjon under passering av en setning s i L(G).

Eksempel på NFA

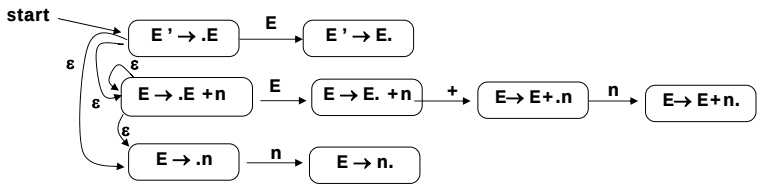
E ' → E
E → E + n
E → n

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

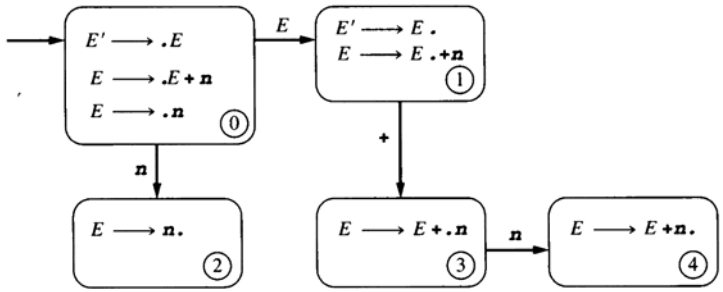
Itemer (LR(0) - itemer)



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

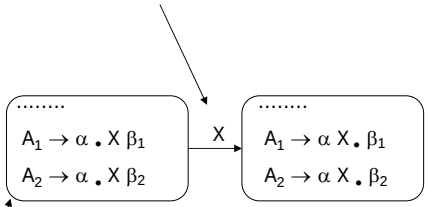
    B er en ikke-terminal
    - $B \rightarrow \beta_1 \mid \beta_2 \mid \ldots\ldots$

  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:

start → S' → •S + tillukning

- Tilstandsovergang ved X fra 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

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

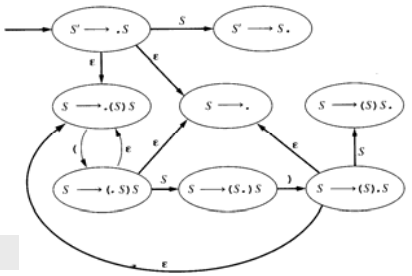
Merk at $S \rightarrow \varepsilon$ bare gir ett item, nemlig:

$S \rightarrow \bullet$

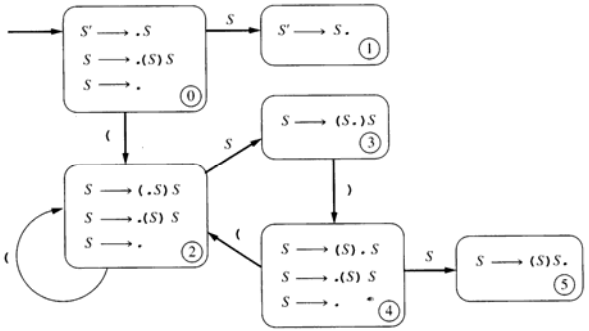
(Altså ikke: $S \rightarrow \bullet \varepsilon$, ~~$S \rightarrow \varepsilon \bullet$~~)

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

LR(0) – NFA:



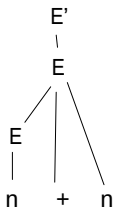
LR(0) – DFA:



Hva sier "topp-tilstanden"

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

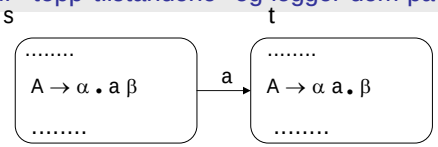
$E' \rightarrow E$
$E \rightarrow E + n \mid n$



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

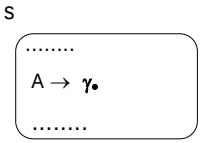
LR(0) - grammatikken og LR(0) – algoritmen.
Navngir ”topp-tilstandene” og legger dem på stakken.

1)



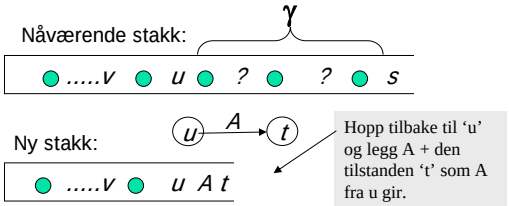
Angir at neste skritt kan være skift, og at dette er lovlig om neste input er a (ny topptilstand blir t)

2)



Kalles et ”slutt-item” (complete item)

Angir at neste skritt kan være reduksjon A → γ



Hopp tilbake til ‘u’ og legg A + den tilstanden ‘t’ som A fra u gir.

N.B : **s,t,u,v** 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, er grammatikken LR(0)!

Da har vi en grei algoritme.

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

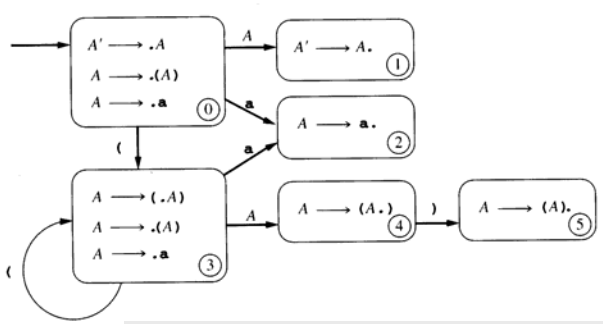
Ser på de tre grammatikkene våre – først A:

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

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

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

A gir følgende LR(0) – DFA:



Tils	Mulig aksjoner:
0	Bare skift for ”(”,”a”
1	Bare red. med A’ → A
2	Bare red. med A → a
3	Bare skift for ”(”,”a”
4	Bare skift for ”)”
5	Bare red. med A → (A)

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

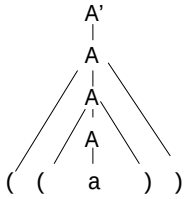
Tabell-oppsett for en LR(0) - grammatikk:

State	Action	Rule	Input			Goto
			(	a	)	A
0	shift		3	2		1
1	reduce	$A' \rightarrow A$				
2	reduce	$A \rightarrow \mathbf{a}$				
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 $A \rightarrow \mathbf{a}$
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 er tom

Noen gale strenger (for A → ..)

som før

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

$ 0	( ) $
$ 0 (3	) $

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

s er en DFA-tilstand med flere item

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 t, where $s \xrightarrow{X} t$ ~~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.
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 t, where $u \xrightarrow{A} t$ ~~containing the item $B \to \alpha A . \beta$~~. (Note that this corresponds to following the transition on A in the DFA, which is indeed reasonable, since we are pushing A onto the stack.)

Avslutning

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-ene
- Tabellene er nesten like, men nå må "reduser-linjene" spesifiseres for hver input.

.....
A $\to \alpha .$
...
B $\to \beta .$

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

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

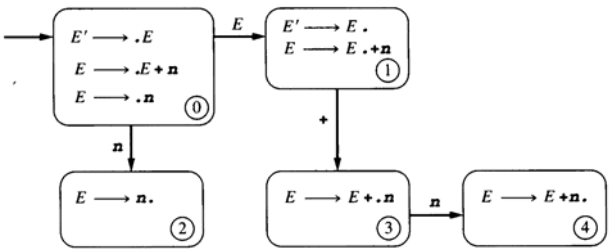
.....
A $\to \alpha .$
...
$B_1 \to \beta_1 . b_1 \gamma_1$
...
$B_2 \to \beta_2 . b_2 \gamma_2$

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

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

Er denne grammatikken SLR(1)

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



First (E) = {n}

Follow(E) = { +, $}

Follow(E') = {$}

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

For alle DFA-tilstander s skal gjelde:

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

Ville ellers ha shift / reduser -konflikt ved input av X (test hver X for seg)

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

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

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

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

Dette er nytt i forhold til LR(0).

Tabell-oppsett for SLR(1)-grammatikk

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

Kan også se på gale setninger som:

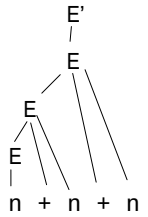
+ n $

n n $

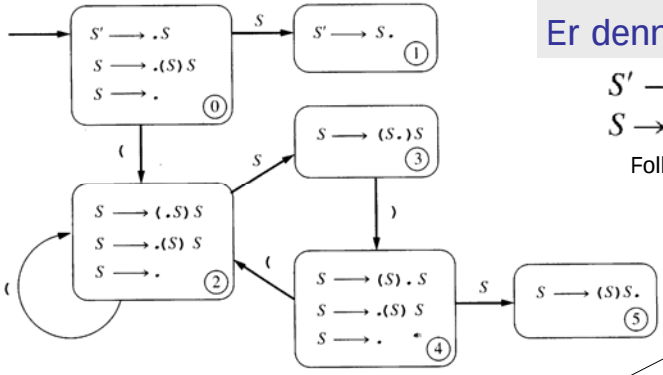
n + $

Parsering av setningen: n + n + n

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



Direkte ut fra tabellen



Er denne SLR(1) ?

$S' \to S$

$S \to (S) \;\; S \mid \varepsilon$

Follow (S) = {), $ }

Dette har vi i SLR(1), ikke i LALR(1) . Begge oppdager feilen, men LALR gjør det tidligere

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

Parsering av setningen: ( ) ( ) $

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

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

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

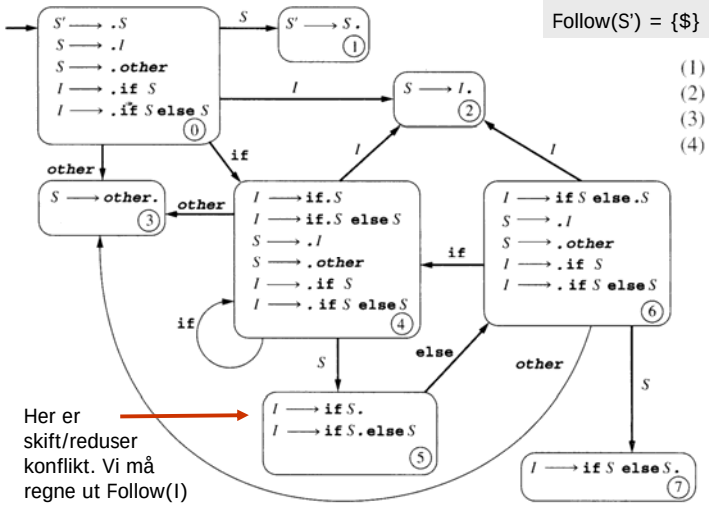
- Heller ikke SLR(k) eller LR(k) for noen k
- LR(0)-DFA-ene vil derfor alltid ha ”uløselige” konflikter
- **Men**: Konfliktene kan oftest løses med intuisjon og fornuft.
- Vanlig metode i Yacc etc:
  - Skift/Reduser – konflikter løse ved å bruke skift

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

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

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

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



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

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 $
$	

Her betyr:

- **s4** – skift (**if** fra input til stakk) Legg også 4 på stakken og gå til tilstand 4
- **r3** – bruk regel 3 i grammatikken og redusert toppen av stakken med den. Toppen av (den reduserte) stakken sier da ny tilstand

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