INF5110 – oppgave 2 på eksamen v04 med teori

Arne Maus
Ifi

Oppgave 2

Oppgave 2

Vi har gitt grammatikken G, der x, y og z er terminal-symboler, A og B er ikketerminal-symboler, og A er startsymbol:

```
A ->  B x | B
B ->  y B | z
```

Spørsmål 2.a

Finn First- og Follow-mengdene til ikketerminal-symbolene i G. Du behøver ikke angi stegene i algoritmen, bare angi svaret (men det er desto viktigere at du ikke gjør slurvefeil).

FirstMengder

terminal

Def { First (A) = { a | finnes avledning A ⇒* a α }
Dessuten: Om A "er utnullbar", så er ε ∈ First(A)

Per def er da: First(a) = { a }
Def "A er utnullbar" ⇔ A ⇒* ε

Let X be a grammar symbol (a terminal or nonterminal) or ε. Then the set **First(X)** consisting of terminals, and possibly ε, is defined as follows.

1. If X is a terminal or ε, then First(X) = {X}.
2. If X is a nonterminal, then for each production choice $X \rightarrow X_1X_2 \ldots X_n$, First($X$) contains First($X_1$) − {$\varepsilon$}. If also for some $i < n$, all the sets First(X_1), . . . , First(X_i) contain ε, then First(X) contains First(X_{i+1}) − {ε}. If all the sets First(X_1), . . . , First(X_n) contain ε, then First(X) also contains ε.

Now define **First(α)** for any string $\alpha = X_1X_2 \ldots X_n$ (a string of terminals and nonterminals), as follows. First(α) contains First(X_1) − {ε}. For each $i = 2, \ldots, n$, if First(X_k) contains ε for all $k = 1, \ldots, i-1$, then First(α) contains First(X_i) − {ε}. Finally, if for all $i = 1, \ldots, n$, First(X_i) contains ε, then First(α) contains ε.

```
for all nonterminals A do First(A) := {};
while there are changes to any First(A) do
  for each production choice A → X1X2 ... Xn do
    k := 1; Continue := true;
    while Continue = true and k <= n do
      add First(Xk) − {ε} to First(A);
      if ε is not in First(Xk) then Continue := false;
      k := k + 1;
    if Continue = true then add ε to First(A);
```

Algoritme:

- Gjør steg 1.
- Gjenta steg 2 til alle Firstmengdene har stabilisert seg.

Eks. 4.9 Beregning av First-mengde

(1) *exp* → *exp addop term*
(2) *exp* → *term*
(3) *addop* → **+**
(4) *addop* → **-**
(5) *term* → *term mulop factor*
(6) *term* → *factor*
(7) *mulop* → *****
(8) *factor* → **(** *exp* **)**
(9) *factor* → **number**

Grammar rule	Pass 1	Pass 2	Pass 3
exp → *exp addop term*			
exp → *term*			First(*exp*) = {(, **number**}
addop → **+**	First(*addop*) = {**+**}		
addop → **-**	First(*addop*) = {**+**, **-**}		
term → *term mulop factor*			
term → *factor*		First(*term*) = {(, **number**}	
mulop → *****	First(*mulop*) = {*****}		
factor → **(** *exp* **)**	First(*factor*) = {**(**}		
factor → **number**	First(*factor*) = {(, **number**}		

Kan bare fylle ut en tabell

	First
exp	
addop	
term	
multop	
factor	

Beregning av Followmengder

β γ *virkårlige streneger*

Def Follow (A) = { a | finnes avledning S $ ⇒* β A a γ }

Dvs. Det finnes en utledning fra startsymbolet S
hvor A kommer rett før a (a er en terminal)

Given a nonterminal A, the set **Follow(A)**, consisting of terminals, and possibly $, is defined as follows.

1. If A is the start symbol, then $ is in Follow(A).
2. If there is a production $B \rightarrow \alpha A \gamma$, then First($\gamma$) − {$\varepsilon$} is in Follow($A$).
3. If there is a production $B \rightarrow \alpha A \gamma$ such that ε is in First (γ), then Follow(A) contains Follow(B).

```
Follow(start-symbol) := {$} ;
for all nonterminals A ≠ start-symbol do Follow(A) := {} ;
while there are changes to any Follow sets do
  for each production A → X1X2 . . . Xn do
    for each Xi that is a nonterminal do
      add First(Xi+1Xi+2 . . . Xn) − {ε} to Follow(Xi)
      (* Note: if i=n, then Xi+1Xi+2 . . . Xn = ε *)
      if ε is in First(Xi+1Xi+2 . . . Xn) then
        add Follow(A) to Follow(Xi)
```

Algoritme:

- **Gjør steg 1.**
- **Gjenta steg 2 og 3 til alle Follow-mengdene har stabilisert seg.**

Beregning av Followmengder 4.12

(1) *exp* → *exp addop term*
(2) *exp* → *term*
(3) *addop* → **+**
(4) *addop* → **-**
(5) *term* → *term mulop factor*
(6) *term* → *factor*
(7) *mulop* → ***
(8) *factor* → **(** *exp* **)**
(9) *factor* → **number**

First(*exp*) = { **(**, **number** }
First(*term*) = { **(**, **number** }
First(*factor*) = { **(**, **number** }
First(*addop*) = {**+**, **-**}
First(*mulop*) = {***}

Follow(*exp*) = {$, **+**, **-**, **)** }
Follow(*addop*) = { **(**, **number** }
Follow(*term*) = {$, **+**, **-**, ***, **)** }
Follow(*mulop*) = { **(**, **number** }
Follow(*factor*) = {$, **+**, **-**, ***, **)** }

Grammar rule	Pass 1	Pass 2
exp → *exp addop term*	Follow(*exp*) = {$, +, -} Follow(*addop*) = { (, number} Follow(*term*) = {$, +, -}	Follow(*term*) = {$, +, -, *,) }
exp → *term*		
term → *term mulop factor*	Follow(*term*) = {$, +, -, *} Follow(*mulop*) = { (, number} Follow(*factor*) = {$, +, -, *}	Follow(*factor*) = {$, +, -, *,) }
term → *factor*		
factor → **(** *exp* **)**	Follow(*exp*) = {$, +, -,) }	

Kan bare fylle ut en tabell

	Follow
exp	
addop	
term	
multop	
factor	

Oppgave 2 a

```
Oppgave 2

Vi har gitt grammatikken G, der x, y og z er terminal-symboler, A og B
er ikketerminal-symboler, og A er startsymbol:

  A ->  B x | B
  B ->  y B | z

Spørsmål 2.a

Finn First- og Follow-mengdene til ikketerminal-symbolene i G.  Du
behøver ikke angi stegene i algoritmen, bare angi svaret (men
det er desto viktigere at du ikke gjør slurvefeil).
```

Svar: First(A) = {y,z}, First(B) = {y,z}
Follow(A) = {$}, Follow(B) = {x,$}

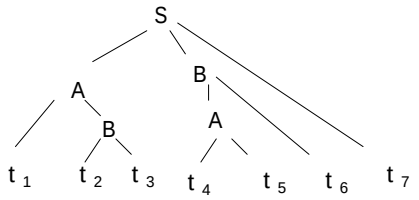
Oppgave 2b

```
A ->  B x | B
B ->  y B | z
```

```
Spørsmål 2.b

Tegn opp LR(0)-DFAen for G, etter først å ha utvidet (``augumented'')
G på standard måte.
```

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



LR-parsering og grammatikker

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

Prinsippet med LR-parsering

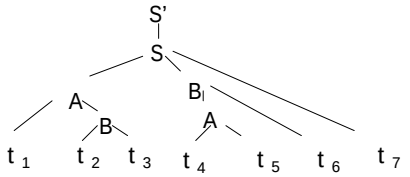
$S' \rightarrow S$

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

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

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

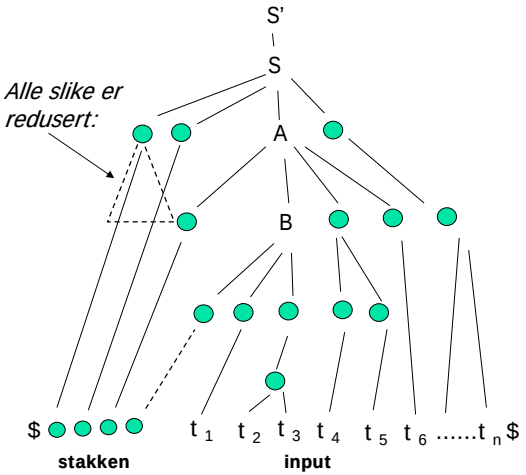
Anta at grammatikken er entydig, og at vi kjenner syntskstreet for setningen:



LR-parsering :

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

Typisk situasjon under LR-parsering



$s_1 s_2 s_3 s_4 \ldots\ldots s_k$ ⟵ lest 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):
  $A \rightarrow \bullet \alpha X \beta$
  $A \rightarrow \alpha \bullet X \beta$
  $A \rightarrow \alpha X \bullet \beta$
  $A \rightarrow \alpha X \beta \bullet$
- 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***

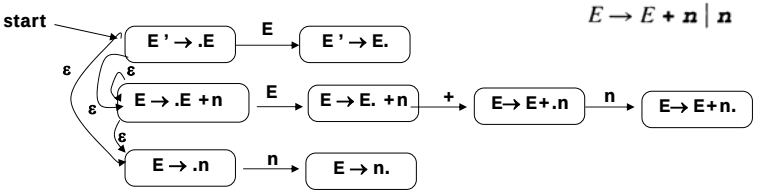
Gitt entydig grammatikk G.

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

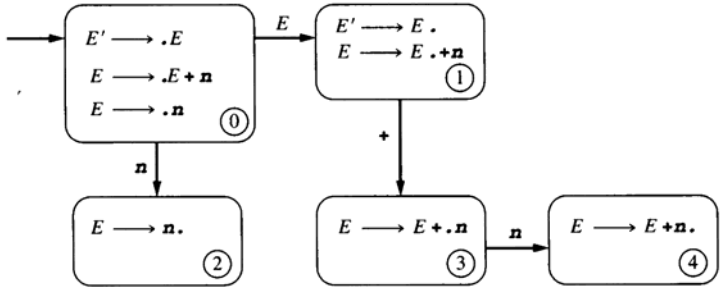
- Vi ser på de mulig stakker som kan opptre, og ser disse som strenger over alfabetet{terminaler, ikke-terminaler}. Vi ser altså på:
  {S | S utgjør stakken på ett eller annet stadium i LR-parsering av en setning i L(G)}
- Dette språket viser seg å være regulært, og kan beskrives av en NFA der:
  - Tilstandene 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).

LR(0) – NFA (litt mer systematisk enn boka)

$E' \to E$
$E \to E + n \mid n$



LR(0) - DFA



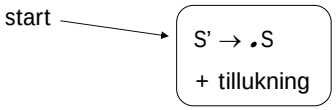
Hvordan sette opp LR(0) - DFA'en direkte fra grammatikken

- Tillukning av item-mengde I:
  - Dersom:
    - A → α • Bγ er med i I

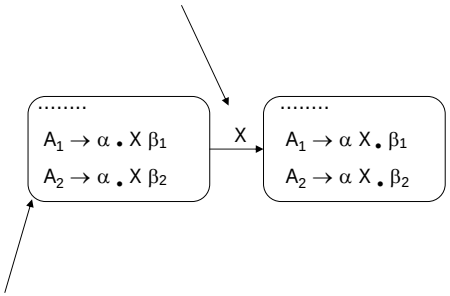
    B er en ikke-terminal
    - B →β 1 | β 2 | .....

  Da er også Itemene
    - B → • β 2 .
    - B → • β 1

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



- Tilstandsovergang ved X fra s
  - X er terminal eller ikke-terminal



Få med alle av formen A → α • X β

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

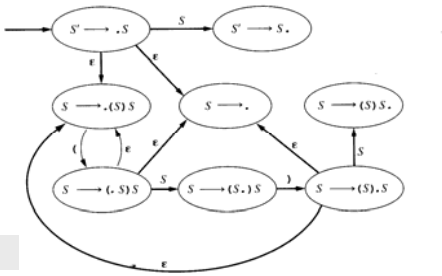
$S' \to S$
$S \to (S) \ S \mid \varepsilon$

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

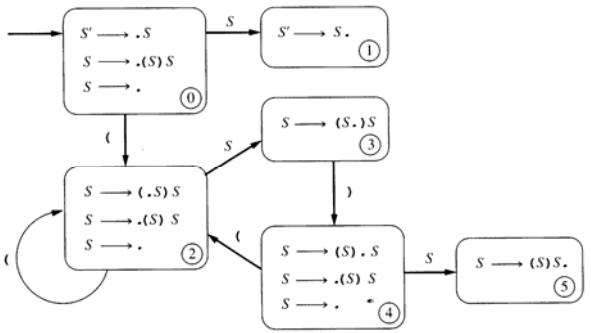
S → •
(Altså ikke: S → • ε
S → ε •)

$S' \to .S$
$S' \to S.$
$S \to .(S)S$
$S \to (.S)S$
$S \to (S.)S$
$S \to (S).S$
$S \to (S)S.$
$S \to .$

LR(0) – NFA:

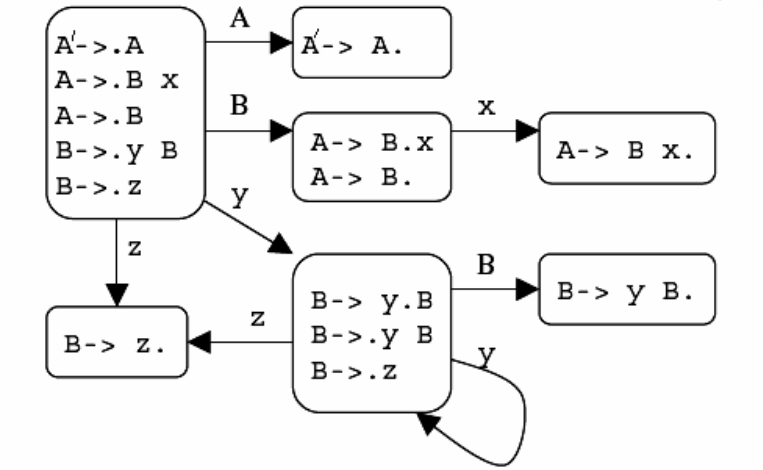


LR(0) – DFA:



2b - LR(0)-DFA

A' → A
A -> B x | B
B -> y B | z



LR(0) ??

Spørsmål 2.c

Avgjør følgende spørsmål, og gi en kort forklaring til hver av dem. Forsøk å foreta avgjørelsen så enkelt som mulig, helst uten å lage parseringstabellen.

2.c.1: Er G en LR(0)-grammatikk?

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 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

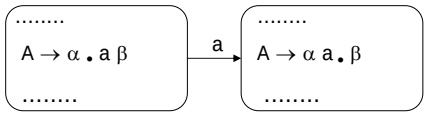
2. If state s contains any complete item (an item of the form $A \to \gamma.$), then the action is to reduce by the rule $A \to \gamma$. A reduction by the rule $S' \to S$, where S is the start state, is equivalent to acceptance, provided the input is empty, and error if the input is not empty. In all 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.)

Avslutning

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

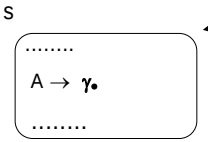
LR(0) - grammatikken og LR(0) – algoritmen.
Navngir "topp-tilstandene" og legger dem på stakken.

1)



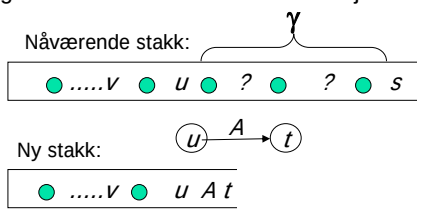
Angir at neste skritt kan være skift, og at dette er lovlig om neste input er a (ny topptilstand blir t)

2)



Kalles et "slutt-item" (complete item)

Angir at neste skritt kan være reduksjon A → γ



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) ?

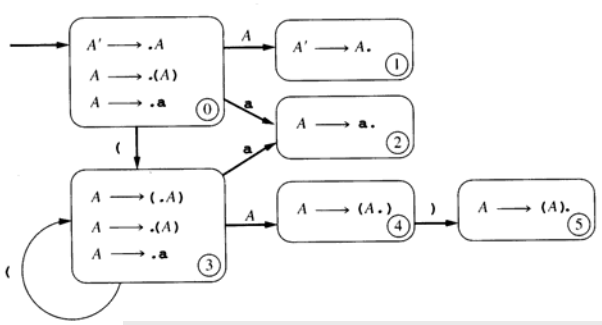
Ser på de tre grammatikkene våre – først A:

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

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

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

A gir følgende LR(0) – DFA:

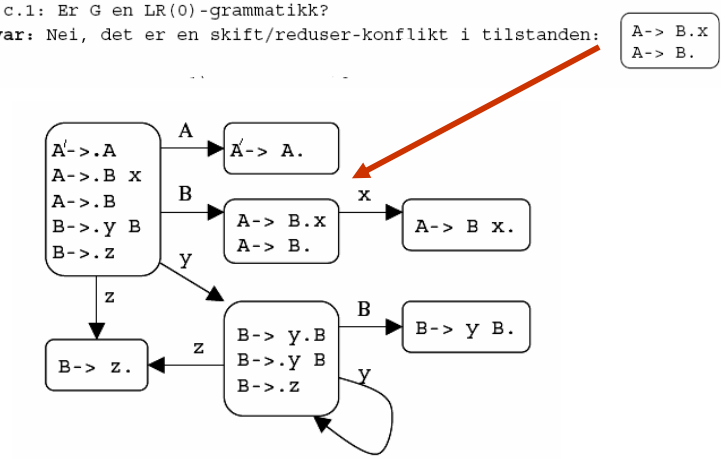


Tils	Mulig aksjoner:
0	Bare skift for "(","a"
1	Bare red. med A' → A
2	Bare red. med A → a
3	Bare skift for "(","a"
4	Bare skift for ")"
5	Bare red. med A → (A)

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

Oppgave 2 - LR(0) ?

2.c.1: Er G en LR(0)-grammatikk?
Svar: Nei, det er en skift/reduser-konflikt i tilstanden:



SLR(1)

2.c.2: Er G en SLR(1)- grammatikk?

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 $\rightarrow \alpha.$
...
B $\rightarrow \beta.$

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

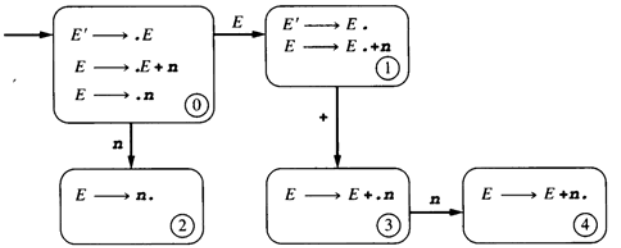
.....
A $\rightarrow \alpha.$
...
$B_1 \rightarrow \beta_1. b_1 \gamma_1$
...
$B_2 \rightarrow \beta_2. b_2 \gamma_2$

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 $A \rightarrow \alpha.X\beta$ 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 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 $A \rightarrow \alpha.X\beta$, 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.

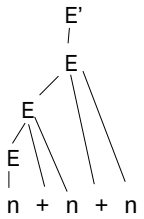
Dette er nytt i forhold til LR(0).

Tabell-oppsett for SLR(1)-grammatikk

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 ($E \rightarrow E + n$)	

Parsering av setningen: n + n + n

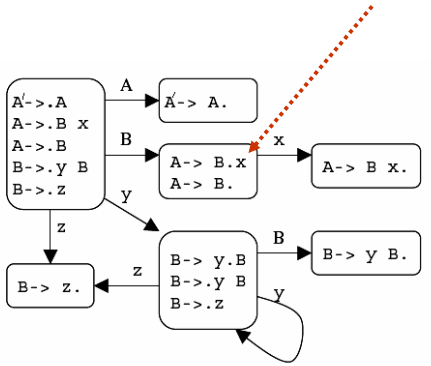
	Parsing stack	Input	Action
1	$ 0	n + n + n $	shift 2
2	$ 0 n 2	+ n + n $	reduce $E \rightarrow n$
3	$ 0 E 1	+ n + n $	shift 3
4	$ 0 E 1 + 3	n + n $	shift 4
5	$ 0 E 1 + 3 n 4	+ n $	reduce $E \rightarrow E + n$
6	$ 0 E 1	+ n $	shift 3
7	$ 0 E 1 + 3	n $	shift 4
8	$ 0 E 1 + 3 n 4	$	reduce $E \rightarrow E + n$
9	$ 0 E 1	$	accept



Direkte ut fra tabellen

2.c.2: Er G en SLR(1)- grammatikk?

Svar: Ja, konflikten kan løses med SLR(1)-metoden siden Follow(A) ikke innholder x.



Svar: First(A) = {y,z}, First(B) = {y,z}
Follow(A) = {$}, Follow(B) = {x,$}

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

```
2.c.3: Er G en LR(1)-grammatikk?
Svar: Ja enhver SLR(1)-grammatikk er også LR(1)

2.c.4: Er G en LALR(1)-grammatikk?
Svar: Ja enhver SLR(1)-grammatikk er også LALR(1)
```

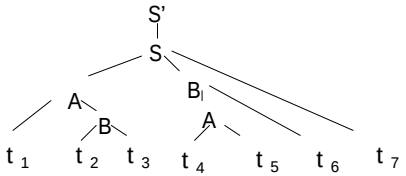
Prinsippet med LR-parsering II

$S' \rightarrow S$

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

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

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



Anta at grammatikken er entydig, og anta at vi kjenner syntakstreet 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:

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

Slutt-situasjonen:

stakk	input
$ S'	$

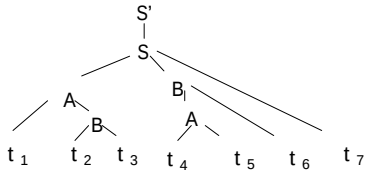
Prinsippet med LR-parsering III

$S' \rightarrow S$

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

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

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



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 $A \rightarrow \alpha$)
  - 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)

Spørsmål 2.d

Vi er interessert i LR(1)-DFAen for (den utvidede) G. Du skal ikke tegne hele, men bare start-tilstanden med de riktige LR(1)-itemer inni. Du kan slå sammen LR(1)-itemer med samme LR(0)-del om du vil det (som ved LALR(1)-DFAer).

Generelt om LR(1) - parsering

a, b = *look-ahead*

Hovedidé: Vil holde greie på under oppsett av NFA og DFA hva som kan stå bak en produksjon i den sammenhengen den står

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

NFA: **[A →α . Xγ, a]** —X→ **[A →αX . γ, 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 $[A \to \alpha . B \gamma, a]$, where B is a nonterminal, there are ε-transitions to items $[B \to .\beta, b]$ for every production $B \to \beta$ and *for every token* b in First(γa).

NFA: **[A →α . Bγ, a]** —ε→ **[A → . β, b]**

For alle:
B →β 1 | β 2 | ..
og alle
b ∈ First(γa)

Spesialtilfelle (γ = ε):

[A →α . Bγ, a] —ε→ **[A → . β, a]**

For alle B →β 1 | β 2 | ..

LR(1) – itemene:

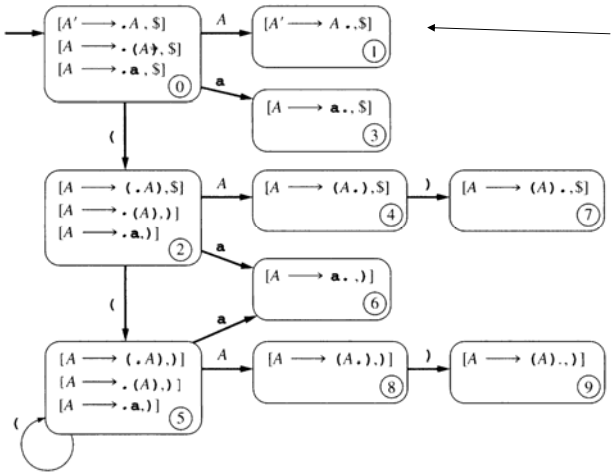
[A →α . Xγ, a]

Angir at a kan komme etter A →αβ i denne aktuelle sammenhengen

(Altså **ikke** bak punktumet, med mindre β = ε)

Oppsett av LR(1) – DFA
(uten å gå veien om LR(1) – NFA !)

Starttilstanden er tillukningen av itemet [A' → . A, $]



Grammatikk:
A' → A
A → (A)
A → a

A' → A. opptrer bare med look-ahead $ (angir aksept)

To tilstander i LR(1)-DFA'en regnes som like bare om de har nøyaktig den samme mengde av LR(1) – itemer (medregnet look-ahead symbolene)

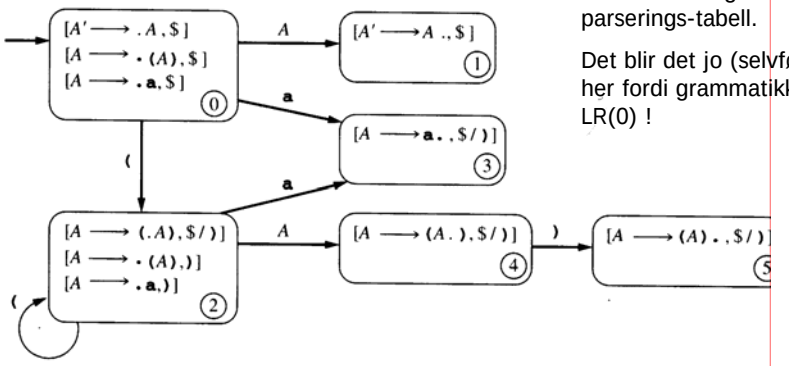
se tilstand 2 og 5

Overgang fra LR(1) til LALR(1)

- "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 item i kjernen : **A →α.β**
- Observasjoner:
  - Kjernen i alle LR(1) – tilstander er en LR(0) – tilstand
  - To LR(1)-tilstander med samme kjerne har samme kanter ut, og tilvarende ut-kanter fører til tilsander med samme kjerne.
    Disse er LALR(1) - DFA'en

Kan lage LALR(1)- DFA'en direkte

- Lag LR(0)-DFA'en (sett av plass til "look-ahead" – symboler.
- Sett inn lookahead $ på [A' →.A,]
- Fyll på med det som "må med" av lookahead'er i en kompletteringsprosess, til det stopper



Grammatikken er LALR(1) dersom dette gir en entydig parserings-tabell.

Det blir det jo (selvfølgelig) her fordi grammatikken er jo LR(0) !

Spørsmål 2.d

Vi er interessert i LR(1)-DFAen for (den utvidede) G. Du skal ikke tegne hele, men bare start-tilstanden med de riktige LR(1)-itemer inni. Du kan slå sammen LR(1)-itemer med samme LR(0)-del om du vil det (som ved LALR(1)-DFAer).

```
[A'->.A, $]
[A->.B x ,$]
[A->.B ,$]
[B->.y B ,x]
[B->.z ,x]
[B->.y B ,$]
[B->.z ,$]
```

eller

```
[A'->.A, $]
[A->.B x ,$]
[A->.B ,$]
[B->.y B ,x/$]
[B->.z ,x/$]
```

A' → A
A -> B x | B
B -> y B | z

FirstMengder

terminal

Def { First (A) = { a | finnes avledning A ⇒* a α }
Dessuten: Om A "er utnullbar", så er ε ∈ First(A)

Per def er da: First(a) = { a }
Def "A er utnullbar" ⇔ A ⇒* ε

Let X be a grammar symbol (a terminal or nonterminal) or ε. Then the set **First(X)** consisting of terminals, and possibly ε, is defined as follows.

1. If X is a terminal or ε, then First(X) = {X}.
2. If X is a nonterminal, then for each production choice $X \rightarrow X_1X_2 \ldots X_n$, First($X$) contains First($X_1$) − {$\varepsilon$}. If also for some $i < n$, all the sets First(X_1), . . . , First(X_i) contain ε, then First(X) contains First(X_{i+1}) − {ε}. If all the sets First(X_1), . . . , First(X_n) contain ε, then First(X) also contains ε.

Now define **First(α)** for any string $\alpha = X_1X_2 \ldots X_n$ (a string of terminals and nonterminals), as follows. First(α) contains First(X_1) − {ε}. For each $i = 2, \ldots, n$, if First(X_k) contains ε for all $k = 1, \ldots, i-1$, then First(α) contains First(X_i) − {ε}. Finally, if for all $i = 1, \ldots, n$, First(X_i) contains ε, then First(α) contains ε.

```
for all nonterminals A do First(A) := {};
while there are changes to any First(A) do
  for each production choice A → X1X2 ... Xn do
    k := 1; Continue := true;
    while Continue = true and k <= n do
      add First(Xk) − {ε} to First(A);
      if ε is not in First(Xk) then Continue := false;
      k := k + 1;
    if Continue = true then add ε to First(A);
```

Algoritme:

- Gjør steg 1.
- Gjenta steg 2 til alle Firstmengdene har stabilisert seg.

Oppsummering for bottom-up

- Vi formulerer vår grammatikk som basal BNF
- Konflikter kan løses med:
  - omdefinere BNF'en (prøver kanskje det først)
  - eller ved direktiver til Yacc
  - eller løser det senere i semantisk analyse
  - N.B. Ikke **alle** konflikter kan løses av selv LR(1) – skriv om! (eks: A → a | aAa)
- Vi har ulike muligheter for å betrakte/parsere vår BNF:

	fordel	ulempe
LR(0)	Definerer DFA-tilstander som brukes av SLR og LALR	Mange (unødige) konflikter red/red og red/skift oppstår
SLR(1)	Klar forbedring av LR(0)	Få/ingen, men ikke så god som LALR
LR(1)	Mest nøyaktig (færrest) konflikter. Best feilhåndtering.	Svært mange tilstander (tabell på opp til 1 M felter for et ordentlig språk)
LALR(1)	Nesten like bra som LR(1)	Dårligere feilhåndtering (for mange reduksjoner før feil oppfattes)