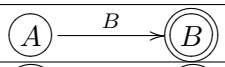
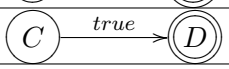
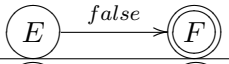
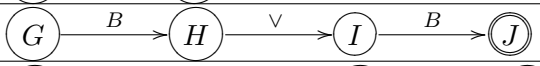
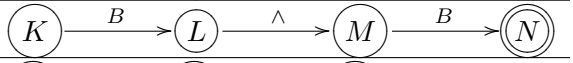
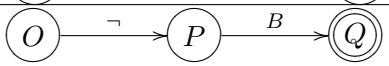
Oversættere, ugeopgave 3

Anders Bjerg Pedersen (andersbp@me.com)

29. november 2009

Opgave 1

Vi konstruerer først NFA'er for grammatikken fra opgave 3.22 med produktionen $B' \to B$ tilføjet:

	Produktion	**NFA**
0	$B' \to B$	
1	$B \to true$	
2	$B \to false$	
3	$B \to B \vee B$	
4	$B \to B \wedge B$	
5	$B \to \neg B$	

Vi tilføjer nu ε-transitioner til de states, der har en nonterminal som overgang:

Tilstand	ε-transition
A	C,E,G,K,O
G	C,E,G,K,O
I	C,E,G,K,O
K	C,E,G,K,O
M	C,E,G,K,O
P	C,E,G,K,O

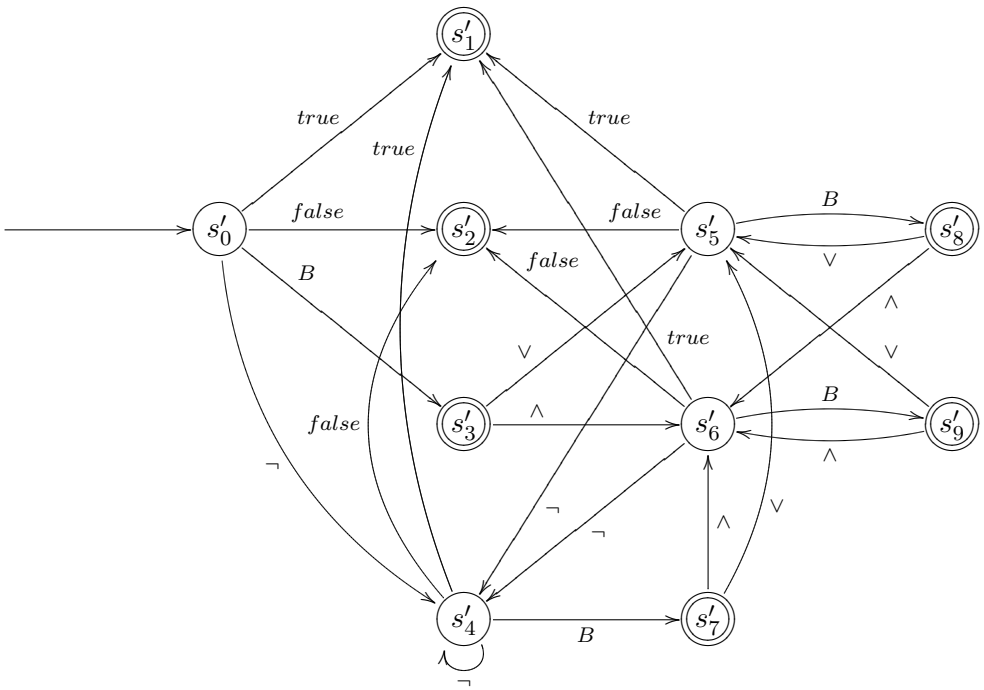
Vi anvender nu den sædvanlige algoritme til at konstruere en DFA ud fra vores NFA'er og ε-transitionerne (vi medtager ikke alle detaljer, da vi jo har prøvet det her en gang eller to før...). Vi har:

$$
\begin{aligned}
s'_0 &= \varepsilon\text{-}closure(\{A\}) = \{A, C, E, G, K, D\} \\
move(s'_0, true) &= \varepsilon\text{-}closure(\{D\}) = \{D\} = s'_1 \\
move(s'_0, false) &= \varepsilon\text{-}closure(\{F\}) = \{F\} = s'_2 \\
move(s'_0, B) &= \varepsilon\text{-}closure(\{B, H, L\}) = \{B, H, L\} = s'_3 \\
move(s'_0, \neg) &= \varepsilon\text{-}closure(\{P\}) = \{P, C, E, G, K, O\} = s'_4
\end{aligned}
$$

Det er uinteressant at se på moves fra s'_1 og s'_2, da der ingen er (begge er accepterende tilstande uden ε-transitioner).

$$
\begin{aligned}
move(s'_3, \vee) &= \varepsilon\text{-}closure(\{I\}) = \{I, C, E, G, K, O\} = s'_5 \\
move(s'_3, \wedge) &= \varepsilon\text{-}closure(\{M\}) = \{M, C, E, G, K, O\} = s'_6 \\
move(s'_4, true) &= \varepsilon\text{-}closure(\{D\}) = \{D\} = s'_1 \\
move(s'_4, false) &= \varepsilon\text{-}closure(\{F\}) = \{F\} = s'_2 \\
move(s'_4, B) &= \varepsilon\text{-}closure(\{H, L, Q\}) = \{H, L, Q\} = s'_7 \\
move(s'_4, \neg) &= \varepsilon\text{-}closure(\{P\}) = \{P, C, E, G, K, O\} = s'_4 \\
move(s'_5, true) &= \varepsilon\text{-}closure(\{D\}) = \{D\} = s'_1 \\
move(s'_5, false) &= \varepsilon\text{-}closure(\{F\}) = \{F\} = s'_2 \\
move(s'_5, B) &= \varepsilon\text{-}closure(\{J, H, L\}) = \{J, H, L\} = s'_8 \\
move(s'_5, \neg) &= \varepsilon\text{-}closure(\{P\}) = \{P, C, E, G, K, O\} = s'_4 \\
move(s'_6, true) &= \varepsilon\text{-}closure(\{D\}) = \{D\} = s'_1 \\
move(s'_6, false) &= \varepsilon\text{-}closure(\{F\}) = \{F\} = s'_2 \\
move(s'_6, B) &= \varepsilon\text{-}closure(\{H, L, N\}) = \{H, L, N\} = s'_9 \\
move(s'_6, \neg) &= \varepsilon\text{-}closure(\{P\}) = \{P, C, E, G, K, O\} = s'_4 \\
move(s'_7, \vee) &= \varepsilon\text{-}closure(\{I\}) = \{I, C, E, G, K, O\} = s'_5 \\
move(s'_7, \wedge) &= \varepsilon\text{-}closure(\{M\}) = \{M, C, E, G, K, O\} = s'_6 \\
move(s'_8, \vee) &= \varepsilon\text{-}closure(\{I\}) = \{I, C, E, G, K, O\} = s'_5 \\
move(s'_8, \wedge) &= \varepsilon\text{-}closure(\{M\}) = \{M, C, E, G, K, O\} = s'_6 \\
move(s'_9, \vee) &= \varepsilon\text{-}closure(\{I\}) = \{I, C, E, G, K, O\} = s'_5 \\
move(s'_9, \wedge) &= \varepsilon\text{-}closure(\{M\}) = \{M, C, E, G, K, O\} = s'_6
\end{aligned}
$$

Vores seks accepterende tilstande er $F' = \{s'_1, s'_2, s'_3, s'_7, s'_8, s'_9\}$, hvilket giver os følgende trivielle DFA:



Vi tilføjer nu produktionen $B'' \to B'\$$ og beregner $FOLLOW$ for vores to nonterminaler:

$$FOLLOW(B') \subseteq FOLLOW(B)$$

$$\begin{aligned} FOLLOW(B') &= \{\$\} \\ FOLLOW(B) &= \{\$, \vee, \wedge\} \end{aligned}$$

Vi kan nu konstruere vores SLR-tabel og starter med at tilføje shifts på alle vores terminal-moves fra DFA'en og gos på alle vores nonterminal-moves. Derefter tilføjer vi reduces ud fra vores $FOLLOW$-mængder ovenfor og reglen på side 93 i bogen:

DFA-tilstand	*true*	*false*	∨	∧	¬	$	*B*
0	s1	s2			s4		g3
1			r1	r1		r1	
2			r2	r2		r2	
3			s5	s6		a	
4	s1	s2			s4		g7
5	s1	s2			s4		g8
6	s1	s2			s4		g9
7			s5/r5	s6/r5		r5	
8			s5/r3	s6/r3		r3	
9			s5/r4	s6/r4		r4	

Det ses nu, at der er seks konflikter, der skal løses via de givne præcedensregler fra opgave 3.22:

$$\text{Tilstand 7: } \begin{cases} \sout{s5} \ / \ \text{r5} & : & (\neg B) \vee B \\ \sout{s6} \ / \ \text{r5} & : & (\neg B) \wedge B \end{cases}$$

Vi vælger altså et reduce i stedet for et shift i begge tilfælde.

$$\text{Tilstand 8: } \begin{cases} \text{s5} \ / \ \sout{r3} & : & B \vee (B \vee B) \\ \text{s6} \ / \ \sout{r3} & : & B \vee (B \wedge B) \end{cases}$$

Vi vælger altså et shift i stedet for et reduce i begge tilfælde.

$$\text{Tilstand 9: } \begin{cases} \sout{s5} \ / \ \text{r4} & : & (B \wedge B) \vee B \\ \text{s6} \ / \ \sout{r4} & : & B \wedge (B \wedge B) \end{cases}$$

Vi vælger altså et reduce i første tilfælde og et shift i andet tilfælde.

Dermed bliver vores færdige SLR-tabel som følger:

DFA-tilstand	*true*	*false*	$\vee$	$\wedge$	$\neg$	\$	B
0	s1	s2			s4		g3
1			r1	r1		r1	
2			r2	r2		r2	
3			s5	s6		a	
4	s1	s2			s4		g7
5	s1	s2			s4		g8
6	s1	s2			s4		g9
7			r5	r5		r5	
8			s5	s6		r3	
9			r4	s6		r4	

Opgave 2

Mit yndlingssprog er selvfølgelig Java[1]. Her er der mulighed for forskellige datastrukturer til brug, når man skal konstruere en dictionary. Standarden er et Map, der indeholder key-value par. Et Map har selfølgelig en constructor til at lave en ny tom symboltabel (og metoden `Map.isEmpty()` til at finde ud af, om tabellen er tom), samt (blandt andet) metoderne `Map.put(K key, V value)` (til at lave en ny binding, der erstatter den gamle) og `Map.get(Object key)` (til at slå op i vores dictionary - metoden returnerer NULL, hvis der intet findes).

For at håndtere forskellige scopes kan vi for eksempel anvende en stak som beskrevet i afsnit 4.2.3 i bogen. Til dette formål har Java også en direkte implementation af en stak, i form af klassen Stack. Vores Stack skal så bruges til at holde på markører for scope-ændringer samt de oprindelige bindings, der skal reetableres efter vi kommer tilbage til et ældre scope. Her kan vi anvende `Map.containsKey(Object key)` til at afgøre, om vi skal skifte scope.

(eller har jeg FULDSTÆNDIG misforstået opgaven?!!)

Opgave 3

Den her opgave er da godt nok også kryptisk formuleret. Som jeg har forstået den, så bliver man bedt om at vise, hvordan *EN* semi-persistent dictionary kan bruges til

[1] Se evt. API'en på http://java.sun.com/javase/6/docs/api/

at opbygge en symboltabel for et programmeringssprog med almindelige scoping-regler og static binding. Men det er jo præcis, hvad der bliver beskrevet i afsnit 4.2.3?!! Den skitserede semi-persistente dictionary givet i opgaven er altså fuldstændig irrelevant...?

Opgave 4

a)

Professor Juniper har sandsynligvis omskrevet til Chomsky normalform for at kunne lægge en øvre grænse på den tid det vil tage at parse en given tekststreng af længde n. Chomsky normalform har nemlig den egenskab, at dens syntakstræer er balancerede binære søgetræer. Denne datastruktur har den gode egenskab, at den længste vej til enhvert blad (som i Chomsky vil være en terminal) er af længde $O(\lg t)$. Derfor vil parsing af en streng af længde n kunne parses i $O(n \lg t)$.[2]

b)

Som beskrevet ovenfor er køretiden $O(n \lg t)$ for en parsing realistisk indenfor antagelsen om, at terminaler kan bearbejdes (hvad en der så ligger i det) i konstant tid. Ulempen ved at omdanne en grammatik til Chomsky normalform bør dog være, at antallet af produktioner og nonterminaler kan stige kraftigt. Nu vides det ikke, hvad der ligger i den preprocessing, der tager $O(t^2vp)$ tid, men den vil være forholdsvist irrelevant, da den øjensynligt kun foregår, når compileren oversættes – ikke hver gang der parses en tekststreng. Da antallet af terminaler jo ikke ændres, samt at vi ved fra sidste ugeopgave, at en LL(1)-parser kan køre i lineær tid, er tidsgrænserne sandsynligvis realistiske.

[2] Desuden har Chomsky normalformer den fordel, at de kun kan beskrive kontekstfrie grammatikker og omvendt. At Professor Juniper har omskrevet Mr. Olsens grammatik til en sådan beviser altså, at den er kontekstfri.