UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i	INF5110 — Kompilatortteknikk
Eksamensdag:	7. Juni 2017
Tid for eksamen:	09:00 – 13:00
Oppgavesettet er på 12 sider.	
Vedlegg:	3 sider
Tillatte hjelpemidler:	Alle trykte og skrevne.

Kontroller at oppgavesettet er komplett før
du begynner å besvare spørsmålene.

Noen generelle råd og vink.

- Les hele oppgaveset før du starter, for å få overblikk så du kan bruke tiden fornuftig.
- Skriv tydelig, og tegn illustrasjoner på en oversiktlig og leselig måte.
- Gi presise og forståelige forklaringer!
- Du kan besvare oppgavene 4, 5, og delvis 6b ved å fylle ut bilaget og legge det ved resten av besvarelsen (i den "hvite versjonen").

Lykke til!

(Fortsettes på side 2.)

Oppgave 1 Front-end faser av en kompilator (vekt 8%)

Front-enden av en kompilator inneholder vanligvis de følgende fasene: den leksikalske fasen (= "scanner"), den syntaktiske fasen (= "parser"), og den semantiske fasen. Fasene kontrollerer og behandler språkets elementer på hver sin måte. For hver av de følgene "språkreglene", angi hvilke av de nevnte fasene som er best egnet for å kontrollerer den. Hvis mer enn én fase er egnet for å behandle en regel (på en rimelig måte), forklar.

(i) Funksjoner må bli kalt med det *riktige antall argumenter*.

(ii) Underscore "`_`" er tillat *midt i* en "identifier", men ikke i begynnelsen og ikke i slutten (dvs, "`my_id`" er tillat men ikke "`_id`').

(iii) Hver variabel må bli deklarert *før* den bli brukt.

(iv) En tilordningssetning ("assignment statement") må bli avsluttet med et semikolon "`;`".

(Fortsettes på side 3.)

Oppgave 2 Regulære uttrykk (vekt 12%)

Remote file identifers ser ut, i den mest generelle formen, som:

```
user@hostname:filename
```

Mer presist: delene av identifikatorene består av *ord*, som er sekvenser av én eller flere *bokstaver* og *sifre* ("words", "letters" og "digits"). *User*-delen består av et *enkelt* ord. Et *hostname* består av én eller flere ord, atskilt av *punktum*-tegn (som i `www.uio.no`). I henhold til Unix konvensjoner består et *filename* av én eller flere ord, atskilt av skråstrek tegn ("/"); en innledende og/eller en avsluttende skråstrek er valgfri. "`User@`" delen er valgfri og kan bli utelatt. Hele "`user@hostname:`" delen kan også bli utelatt, medregnet kolon "`:`". "`User@`" delen er tillat bare hvis "`hostname:`" delen er tilstede også.

Spørsmålet er nå: spesifiser formen av "remote file identifiers" med *regulære uttrykk*. Bruk gjerne en av de mer "brukervennlige" versjonene av regulære uttrykk.

(Fortsettes på side 4.)

Oppgave 3 Top-down parsering (vekt 24%)

Anta at vi har følgende kontekst-frie grammatikk:

$$\begin{array}{lcl} S & \rightarrow & ABA \\ A & \rightarrow & B\mathbf{c} \mid \mathbf{d}A \mid \boldsymbol{\epsilon} \\ B & \rightarrow & \mathbf{e}A \end{array} \tag{1}$$

3a First- and follow-mengdene (vekt 10%)

Angi *First*- og *Follow*-mengdene for ikke-terminal-symbolene.

3b LL(1)-parsering (vekt 14%)

Tegn opp en LL(1)-parseringstabell for grammatikken. Er grammatikken LL(1)?

(Fortsettes på side 5.)

Oppgave 4 Bottom-up parsering (vekt 20%)

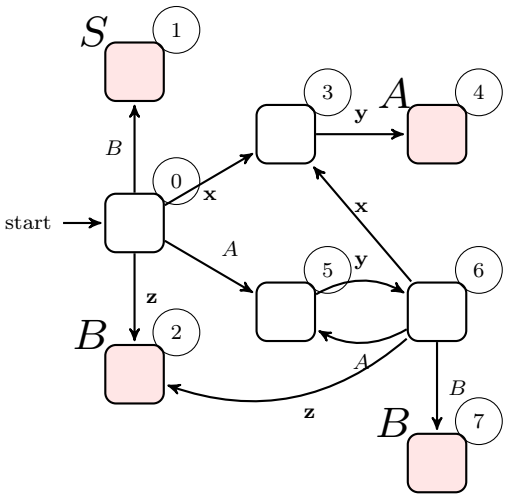


Figure 1: LR(0)-DFA (tilstandene mangler "items")

4a LR(0)-DFA (vekt 14%)

Anta en kontekst-fri grammatikk med ikke-terminalene S (som start-symbol), A og B, og terminaler **x**, **y** og **z**. Se på den LR(0)-DFA, som er angitt i Figur 1. Her er alle tilstandene og transisjonene angitt, med start-tilstand 0. Også angitt er tilstander som — i den fullstendige automaten — inneholder et *"complete item",* dvs., en item med "**.**"-tegn på slutten. I tillegg: for tilstander som inneholder "complete items", er det tilsvarende ikke-terminal-symbolet *på venstre siden* av den "complete item" vist.

For eksempel, tilstand nummer 1 er ment å inneholde et "complete item" med S på venstre siden av produksjonen, dvs., et item som ser ut som

$$S \rightarrow \text{<right-hand-side>} \ \textbf{.}$$

Tilsvarende for tilstandene 2,4 og 7.

Spørsmålet er nå: fyll ut den manglende informasjonen, dvs., fyll inn "items" som utgjør tilstandene $0-7$ på en slik måte at den utfylte DFAen er resultatet av den vanlige LR(0)-DFA konstruksjon for grammattiken (ved å fylle in "items" riktig, gjenoppbygger du også implisitt selve grammatikken).

Automaten er gjentatt i "*vedlegg*". Du kan bruke det for tegne løsningen din. Vi anbefaler å skissere løsningen først på et ekstra ark før du koperer den inn (leselig).

4b Klassifisering (vekt 6%)

For grammatikken i problem 4a: Er den LR(0), LR(1), SLR(1), LALR(1)?

(Fortsettes på side 6.)

Oppgave 5 Attributgrammatikker (vekt 16%)

Anta vi har følgende grammatikk:

$$
\begin{aligned}
\textit{program} &\rightarrow \textbf{prog}\ \textit{stmt-seq} \\
\textit{stmt-seq} &\rightarrow \textit{stmt}\ \textit{stmt-seq} \\
\textit{stmt-seq} &\rightarrow \textit{stmt} \\
\textit{stmt} &\rightarrow \textbf{do var = const upto const begin}\ \textit{stmt-seq}\ \textbf{end} \\
\textit{stmt} &\rightarrow \textbf{assign}
\end{aligned}
$$

Grammatikken spesifiserer et (veldig enkelt) spåk, som kan brukes til å iterere gjennom sekvenser av setninger, spesielt tilordninger. Terminal-symboler er angitt med **fet** skrift, ikke-terminal-symboler i *kursiv*. En **do**-løkke er spesifisert med en løkke-variabel som går fra en nedre til en øvre grense (begge to er konstanter). Grammatikken har ingen produksjon for tilordninger, siden det er uviktig for oppgavene her. Det betyr at tilordningene (representert av **assign**) er behandlet som terminal-symboler.

Listing 1 viser et eksempel-program (i er token verdien for **var**, analogt for 1 and 42).

Listing 1: Eksempel-program

```
1 prog
2   do i = 1 upto 42
3   begin
4      assign
5      assign
6    end
7    assign
```

Gi en attributtgrammatikk som bestemmer for hvert tilordning i et program

hvor mange ganger tilordningen blir utført

når man kjører programmet.

Anta at terminal-symbolet, som representerer konstanter, har et attributt `val` for dens konstante verdi (et heltall). Resultatet skal bli funnet i et attributt for **assign**-symbolet, (gjerne `iterated` eller `i` for kort). Du kan også bruke det samme attributtet for andre symboler av grammatikken, hvis det passer for din løsning.

Anta at for alle løkkene er den nedre grensen $\textbf{const}_0$ mindre eller lik den øvre grensen $\textbf{const}_1$; det er ikke nødvendig å sjekke det i din løsning. Sørg for at de etterspurte attributt-verdiene er beregnet eksakt (ikke bare mer eller mindre korrekt, pluss/minus én).

(Fortsettes på side 7.)

	productions/grammar rules			semantic rules
0	*program*	→	**prog** *stmt-seq*	
1	*stmt-seq*$_0$	→	*stmt stmt-seq*$_1$	
2	*stmt-seq*	→	*stmt*	
3	*stmt*	→	**do var =** **const**$_0$ **upto const**$_1$ **begin** *stmt-seq* **end**	
4	*stmt*	→	**assign**	

Gi svaret ved å fylle ut tabellen. Du kan bruke tabellen i vedlegget (ved å rive arket av og levere det sammen med den hvite besvarelsen).

(Fortsettes på side 8.)

Oppgave 6 Kodegenerering (vekt 20%)

6a Kodegenerering og optimalisering (vekt 8%)

Se på den følgende transformasjon av tre-adresse-mellomkode, som er illustrert i et enkelt eksempel:

Listing 2: Før

```
1 if t == 0
2 then
3     x = y + z;
4     <rest of then-branch>
5 else
6     x = y + z;
7     <rest of else-branch>
8 endif
```

Listing 3: Etter

```
1 x = y + z;
2 if t == 0
3 then
4     <rest of then-branch>
5 else
6     <rest of else-branch>
7 endif
```

Idéen her er at man flytter "felles instruksjoner" (som tilordningen `x = y + z` i eksemplet) *foran* en if-then-else-sentning (hvis dette ikke forandrer kodens betydning). Tre-adresse-koden i deloppgavene her støtter if-then-else-konstruksjonen i stedet for *if-goto*-konstruksjonen som i forelesningen og i deloppgaven 6b.

Anta kodegenerering slik dette er fremstilt på forelesningene og i notatet fra Kapittel 9 i boken *Compilers: Principles, Techniques, and Tools, A. V. Aho, R. Sethi, and J. D. Ullman, 1986* (den gamle "dragon book").

Anta videre at kodegeneratoren har tilgang til *lokal* "liveness" informasjon, dvs. liveness informasjon per basisblokk ("basic block"), mens global liveness *ikke* er tilgjengelig.

Under disse antagelsene, hva er mulige effekter av transformasjonen på kvaliteten of koden som er generert? Diskutér spørsmålet med hensyn til *kostnadsmodellen* fra forelesnigene/notatet.

Merk: vi forventer ingen eksakte to-adresse-kode-sekvenser some ble generert eller detaljerte beregninger av kostnader, bare en diskusjon av hvilke konsekvenser transformasjonen kan har på faktorene i kostnadsmodellen.

6b Global analyse (vekt 12%)

Se på programmet i Listing 4 i tre-adresse-kode. Vi skiller her ikke mellom vanlige variabler og temporære variabler ("temporaries").

(i) Gi basisblokker (*basic blocks*) av koden ved å oppgi start og slutt-linjer for hvert blokk, og nummerere blokker som B_0, B_1, etc. Du kan gjerne bruke koden i vedlegg for å tegne horisontale linjer som viser grensene av blokkene. Ikke glem å vise B_i-nummerne for blokkene.

(ii) Tegn opp "control-flow" grafen til programmet og bruk B_0, B_1, ... for

(Fortsettes på side 9.)

å betegne nodene til grafen.

(iii) Ha control-flow grafen en "loop"? Begrepet "loop" i en control-flow graf er ment som det bli brukt i forelesningene.

(iv) Angi *inLive* og *outLive* informasjonen for hver blokk (gjerne i form av en tabell).

Listing 4: Tre-adresse-kode

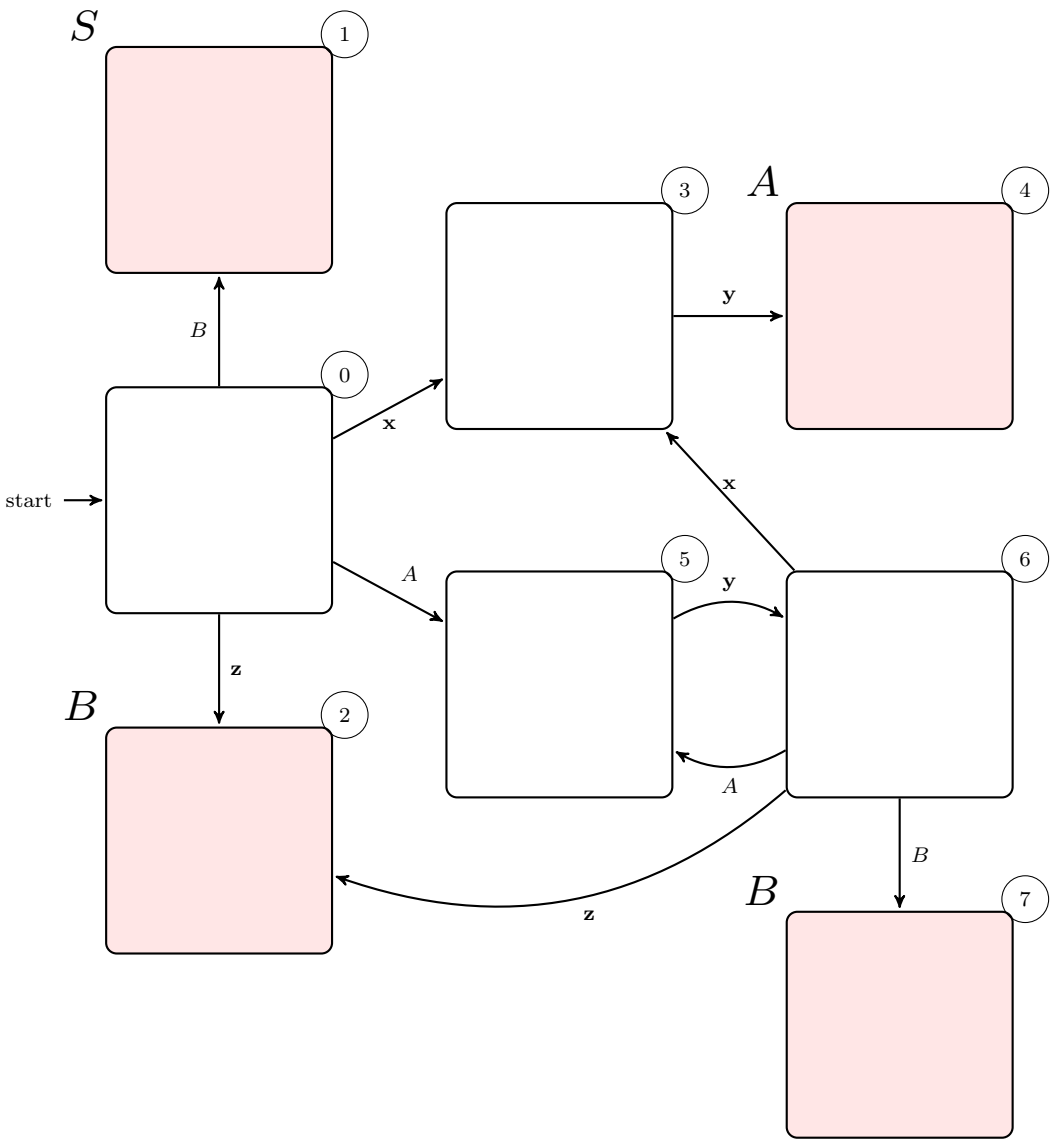
```
1  x = input
2  y = input
3  label L1
4  b = x + y
5  z = b*z
6  label L2
7  x = a + 1
8  if_false x goto L3
9  x = y + x
10 if_true z goto L5
11 goto L1
12 label L3
13 z = b * 2
14 goto L2
15 label L5
16 output x
```

(Fortsettes på side 10.)

Appendix: DFA for Problem 4

Candidate nr.: .....

Date: .....



(Fortsettes på side 11.)

Appendix: Skjema for Problem 5

Candidate nr.: .....

Date: .....

	productions/grammar rules	semantic rules
0	*program* → **prog** *stmt-seq*	
1	$stmt\text{-}seq_0$ → *stmt stmt-seq*$_1$	
2	*stmt-seq* → *stmt*	
3	*stmt* → **do var =** **const**$_0$ **upto const**$_1$ **begin** *stmt-seq* **end**	
4	*stmt* → **assign**	

(Fortsettes på side 12.)

Appendix: Koden for Problem 6b

Candidate nr.: .....

Date: .....

```
1  x = input
2  y = input
3  label L1
4  b = x + y
5  z = b*z
6  label L2
7  x = a + 1
8  if_false x goto L3
9  x = y + x
10 if_true z goto L5
11 goto L1
12 label L3
13 z = b * 2
14 goto L2
15 label L5
16 output x
```