TENTAMEN COMPILERCONSTRUCTIE
Maandag 15 december 2014, 14:00 – 17:00 uur

Dit tentamen bestaat uit 5 opgaven.

Als je het antwoord op een onderdeel niet weet, en je hebt dat antwoord nodig bij een later onderdeel, dan kun je het antwoord 'kopen' bij de docent.

Als er bij een opgave gevraagd wordt om uitleg of motivatie van je antwoord, is het belangrijk dat je die ook geeft.

---

1. Wat verstaan we bij *lexical analysis* onder

   - een *token*
   - een *lexeme*
   - een *pattern*

   Maak duidelijk wat de verschillen tussen de drie termen zijn.

---

2. Laat α een string van symbolen uit een context-vrije grammatica zijn, en laat A een variabele zijn. Leg duidelijk (en volledig) uit wat FIRST(α) en FOLLOW(A) zijn. Wat zit er in?

---

3. Beschouw de context-vrije grammatica G met startvariabele S en de volgende producties:

$$\begin{aligned} S &\rightarrow aAb \mid abB \\ A &\rightarrow bB \mid cc \\ B &\rightarrow ac \mid Sb \end{aligned}$$

   (a) Bepaal voor elke variabele in G de FOLLOW-verzameling.

   (b) Construeer de LR(0)-automaat bij grammatica G.

   (c) Construeer de SLR *parsing table* bij grammatica G.

   (d) Is G een SLR grammatica? Motiveer je antwoord.

---

4. Bij het genereren van drie-adres code voor boolese expressies en *flow-of-control* instructies kunnen we gebruik maken van labels voor adressen waar Goto-instructies naartoe moeten springen. Zowel de code als de labels kunnen we tijdens de vertaling doorgeven als attributen van de variabelen in de grammatica.

   Bekijk het volgende stukje uit een syntax-directed definition voor het genereren van drie-adres code:

Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$
	$P.code = S.code \;\; \|\| \;\; label(S.next)$
$S \rightarrow$ **while** $(B)\ S_1$	$begin = newlabel()$
	$B.true = newlabel()$
	$B.false = S.next$
	$S_1.next = begin$
	$S.code = label(begin) \;\; \|\| \;\; B.code \;\; \|\| \;\; label(B.true)$
	$\|\| \;\; S_1.code \;\; \|\| \;\; gen('\texttt{goto}'\ begin)$
$B \rightarrow B_1 \&\& B_2$	$B_1.true = newlabel()$
	$B_1.false = B.false$
	$B_2.true = B.true$
	$B_2.false = B.false$
	$B.code = B_1.code \;\; \|\| \;\; label(B_1.true) \;\; \|\| \;\; B_2.code$
$B_1 \rightarrow$ **id**$_1$ **rel** **id**$_2$	$B_1.code = gen('\texttt{if}'\ \mathbf{id}_1.addr\ \mathbf{rel}.op\ \mathbf{id}_2.addr\ '\texttt{goto}'\ B_1.true)$
	$\|\| \;\; gen('\texttt{goto}'\ B_1.false)$
$S_1 \rightarrow$ **id**$_1 =$ **id**$_2$ **binop** **num**	$S_1.code = gen(\mathbf{id}_1.addr = \mathbf{id}_2.addr\ \mathbf{binop}.op\ \mathbf{num}.val)$

   Hierin komt de variabele S (en ook S_1) overeen met een instructie, en de variabele B (en ook B_1 en B_2) met een boolese expressie. Het token **rel** staat voor een relationale operator, en **binop** staat voor een (rekenkundige) binaire operator. Zowel P, S als B heeft een attribuut *code*, S heeft daarnaast een attribuut *next*, en B heeft attributen *true* en *false*.

   (a) Leg voor elk van de vier attributen uit waar het voor staat.

   (b) Welk(e) van de vier attributen is/zijn inherited en welk(e) synthesized?

   (c) Teken de afleidingsboom (*parse tree*) bij bovenstaande grammatica (met startvariabele P) voor het volgende 'programma':

```
while (x!=y && x<max)
  x = x*2
```

   (d) Bepaal voor elke variabele in de afleidingsboom van het vorige onderdeel de waarde van zijn attributen *code*, *next*, *true* en *false* (uiteraard alleen de attributen die voor een variabele van toepassing zijn). Nummer de gebruikte labels $L1, L2, \ldots$

   (e) Leg uit wat er volgens de syntax-directed definition gebeurt (de semantische regels) bij de productie

$$S \rightarrow \textbf{while}\ (B)\ S_1$$

   Leg ook uit waarom dat gebeurt.

   N.B.: het gaat er hier om wat er in het algemeen gebeurt bij deze productie, en niet zozeer wat er gebeurt in het geval van ons voorbeeld'programma'.

5. Om een array A met integers op posities $0 \ldots n-1$ te sorteren, kunnen we gebruik maken van BubbleSort. Een mogelijke implementatie van BubbleSort ziet er als volgt uit:

```
for (grens=n-1;grens>=1;grens--)
  for (i=0;i<=grens-1;i++)
  { if (A[i]>A[i+1])
    { tmp = A[i];
      A[i] = A[i+1];
      A[i+1] = tmp;
    }
  }
```

Als we aannemen dat een integer vier bytes in beslag neemt, kan een rechttoe-rechtaan vertaling van de binnenste for-lus in dit algoritme in drie-adres code het volgende opleveren:

```
(1)  i = 0
(2)  t1 = grens-1
(3)  iffalse i <= t1 goto 22
(4)  t2 = 4*i
(5)  t3 = a[t2]
(6)  t4 = i+1
(7)  t5 = 4*t4
(8)  t6 = a[t5]
(9)  iffalse t3 > t6 goto 20
(10) t7 = 4*i
(11) tmp = a[t7]
(12) t8 = 4*i
(13) t9 = i+1
(14) t10 = 4*t9
(15) t11 = a[t10]
(16) a[t8] = t11
(17) t12 = i+1
(18) t13 = 4*t12
(19) a[t13] = tmp
(20) i = i+1
(21) goto 2
(22) ...
```

(a) Welke instructies in bovenstaande drie-adres code zijn de *leaders*?

(b) Teken de *flow graph* met de basic blocks bij regels 1–21 van bovenstaande drie-adres code. Nummer de basic blocks $B_1, B_2, \ldots$.

(c) Je moet de drie-adres code in regels 1–21 hierboven nu stapsgewijs gaan optimaliseren. Bij iedere stap moet je kort aangeven wat je doet, en moet je voor de basic blocks die bij die stap veranderen de complete nieuwe blokken geven. Je mag gebruik maken van de volgende soorten transformaties (voor zover ze te gebruiken zijn):

- *constant folding*
- *local common-subexpression elimination*
- *global common-subexpression eliminiation*

- *copy propagation*
- *dead-code elimination*
- *code motion*
- *reduction in strength*
- *induction-variable elimination*

Noem bij iedere stap het soort transformatie dat je gebruikt hebt. Geef ook de complete nieuwe flow graph die het eindresultaat is van de optimalisatiestappen.
