HERTENTAMEN COMPILERCONSTRUCTIE
Donderdag 15 maart 2018, 14:00 – 17:00 uur

Dit tentamen bestaat uit vijf opgaven. waarbij steeds tussen [en] staat hoeveel punten er ongeveer mee te verdienen zijn. In totaal zijn er 100 punten te verdienen.

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. [8 pt] Leg uit wat het verschil is tusen de *parse tree* en een *syntax tree* bij een bepaald stukje code.

   Geef, ter illustratie van je antwoord, beide bomen voor de expressie $a + b * (c + d)$ en de grammatica met startvariabele E en de volgende producties:

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \mathbf{id} \end{aligned}$$

---

2. [23 pt] Beschouw de context-vrije grammatica G met startvariabele S en de volgende producties:

$$\begin{aligned} S &\rightarrow SaT \mid aTa \\ T &\rightarrow bSb \mid ba \end{aligned}$$

   (a) Bepaal voor elke variabele in de grammatica 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.

---

3. [12 pt]

   (a) De compiler kan informatie over het type van een variabele of expressie voor verschillende doeleinden gebruiken. Noem drie van zulke doeleinden.

   (b) Behalve basis types kun je ook samengestelde types hebben, zoals `int[2][3]` . Geef de type expressie bij `int[2][3]` .

   (c) Behalve de *array* operator kun je ook andere operatoren (*type constructors*) gebruiken om samengestelde types te construeren. Noem nog twee van zulke operatoren.

---

4. [31 pt] 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
$P \rightarrow S$	$S.next = newlabel()$
	$P.code = S.code \;\; \|\| \;\; label(S.next)$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$
	$B.false = S_1.next = S.next$
	$S.code = B.code \;\; \|\| \;\; label(B.true) \;\; \|\| \;\; S_1.code$
$B \rightarrow B_1 \&\& B_2$	$B_1.true = newlabel()$
	$B_1.false = B.false$
	$B_2.true = B.true$
	$B_2.false = B.false$
	$B.code = B_1.code \;\; \|\| \;\; label(B_1.true) \;\; \|\| \;\; B_2.code$
$B_1 \rightarrow$ **id**$_1$ **rel** **id**$_2$	$B_1.code = gen('\texttt{if}'\ \textbf{id}_1.addr\ \textbf{rel}.op\ \textbf{id}_2.addr\ '\texttt{goto}'\ B_1.true)$
	$\quad \|\| \;\; gen('\texttt{goto}'\ B_1.false)$
$S_1 \rightarrow$ **id**$_1$ = **id**$_2$ **binop** **num**;	$S_1.code = gen(\textbf{id}_1.addr = \textbf{id}_2.addr\ \textbf{binop}.op\ \textbf{num}.val)$

   Hierin is P de startvariabele, en 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 *parse tree* bij bovenstaande grammatica voor het volgende 'programma':

```
if (x<a && x<b)
  x = x+1;
```

   (d) Bepaal voor elke variabele in de parse tree van het vorige onderdeel de waarde van zijn attributen *code*, *next*, *true* en *false* volgens de semantische regels hierboven (uiteraard alleen de attributen die voor een variabele van toepassing zijn). Vermeld de attributen in de volgorde waarin ze hun waarde krijgen. Nummer de gebruikte labels $L1, L2, \ldots$

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

$$B \rightarrow B_1 \&\& B_2$$

   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. [26 pt]

   (a) Wat zijn *available expressions* op een bepaald punt in een programma?

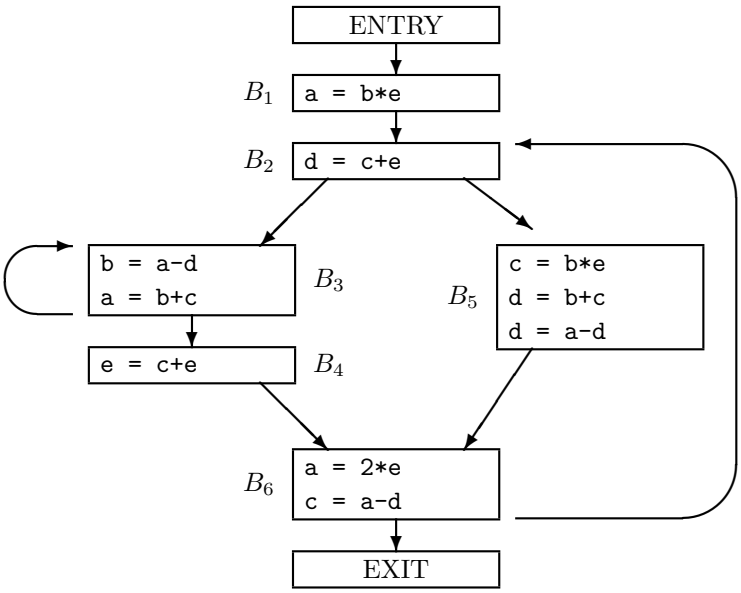
De algemene opzet van een iteratief algoritme voor voorwaartse *dataflow* analyse is als volgt (waarbij de knopen de *basic blocks* zijn):

```
OUT[ENTRY] = ...
for each node B other than ENTRY
    OUT[B] = ...

while (changes to any OUT occur)
    for each node B other than ENTRY
    {   IN[B] = ...predecessors P of B OUT[P]
        (i.e., combine the OUT-sets of the predecessors in some way)

        OUT[B] = ... (some function of IN[B])
    }
```

   (b) Vul de vier '...' in de algemene opzet van het iteratieve algoritme in voor het berekenen van available expressions. Voor iedere knoop B moeten IN[B] en OUT[B] aan het eind van het algoritme alle available expressions aan het begin, respectievelijk einde van B bevatten.

   Wees met name ook precies in je beschrijving van 'some function'.

Beschouw nu het volgende stroomdiagram:



   (c) Wanneer we het iteratieve algoritme willen gebruiken om de available expressions in een stroomdiagram te berekenen, moeten we een volgorde kiezen waarin we de basic blocks aflopen in de binnenste for-lus (dwz: de for-lus binnen de while-lus in het algoritme).

   Wat is een gunstige volgorde van de basic blocks bij bovenstaand stroomdiagram? Motiveer je antwoord.

Z.O.Z.

(d) Pas nu het iteratieve algoritme om de available expressions te berekenen toe op bovenstaand stroomdiagram. Laat met een overzichtelijke tabel zien wat de IN en OUT-verzameling van elk basic block (op ENTRY na) is na de initialisatie en na iedere iteratie van de while-lus. De laatste iteratie, waarin je zou constateren dat er toch niets veranderd is, hoef je niet uit te voeren.
