HERTENTAMEN COMPILERCONSTRUCTIE
Dinsdag 11 maart 2014, 14:00 – 17:00 uur

Dit tentamen bestaat uit 6 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. [6 pt] Tijdens de lexicale analyse kunnen verschillende soorten tokens herkend worden met verschillende transitiediagrammen. Stel dat je voor elke soort token een transitiediagram hebt.

   Noem drie manieren om al die verschillende transitiediagrammen te combineren tot één lexical analyser.

---

2. [7 pt] Leg duidelijk uit aan welke voorwaarden de FIRST- en FOLLOW-verzamelingen moeten voldoen om ervoor te zorgen dat een context-vrije grammatica LL(1) is.

---

3. [28 pt] Beschouw de context-vrije grammatica G met startvariabele S en de volgende producties:

$$\begin{array}{rcl} S & \rightarrow & M \mid O \\ M & \rightarrow & \textbf{if expr then } M \textbf{ else } M \mid \textbf{id} = \textbf{num} \\ O & \rightarrow & \textbf{if expr then } S \mid \textbf{if expr then } M \textbf{ else } O \end{array}$$

   G is een eenvoudige grammatica voor een instructie in een programmeertaal, waarin de *dangling-else*-dubbelzinnigheid is geëlimineerd. S, M, O zijn de variabelen (overeenkomend met *Statement*, *Matched statement* en *Open statement*) en $\textbf{if}, \textbf{expr}, \textbf{then}, \textbf{else}, \textbf{id}, =, \textbf{num}$ zijn de terminalen in G.

   (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. [10 pt]

   (a) Wat verstaan we onder een *activation record*?

   (b) Beschrijf vier soorten elementen van een activation record.

---

5. [25 pt]

   (a) Om op lokaal niveau efficiënt registers te alloceren, kunnen we gebruik maken van *register descriptors* en *address descriptors*. Wat verstaan we onder deze twee 'descriptors'?

   (b) We willen nu de volgende drie-adres code omzetten in assembly:

```
a=b+c
d=d-b
b=e
e=a+f
f=a-d
```

   Gegeven is dat we na afloop van dit stukje code als eerste de waardes van de variabelen d, c en f weer nodig hebben in een berekening (in die volgorde).

   Zet de gegeven drie-adres code instructie voor instructie om in assembly, waarbij je efficiënt met registers omgaat. Je mag hierbij gebruik maken van

   - drie registers R1, R2 en R3, die aanvankelijk leeg zijn
   - assembly instructies van de volgende vorm:

```
ADD Ri, Rj, Rk
SUB Ri, Rj, Rk
LD Ri, x
ST x, Ri
```

     Hierbij zijn Ri, Rj en Rk registers (eventueel dezelfde), en is x een variabele. Bij ADD en SUB komt het resultaat in Ri te staan.

   Aan het eind hoef je de inhoud van de registers niet met allemaal store instructies veilig op te slaan.

   Geef in een overzichtelijke tabel de inhoud van de register descriptors en de address descriptors

   - aan het begin
   - en na ieder stukje assembly dat met een drie-adres instructie overeenkomt.

   (dus in totaal zes keer).

   Geef bij iedere drie-adres instructie een korte motivatie van de keuze voor de gebruikte registers.

---

6. [24 pt]

   (a) Wat zijn *live* variabelen op een bepaald punt in een programma?

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

IN[EXIT] = ...
for each node B other than EXIT
    IN[B] = ...

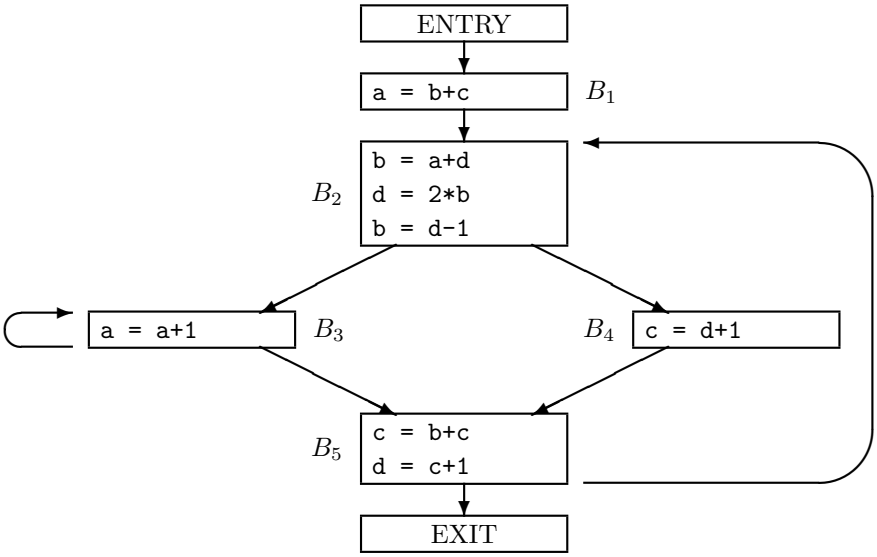
while (changes to any IN occur)
    for each node B other than EXIT
    { OUT[B] = $\ldots_{\text{successors } S \text{ of } B}$ IN[S]
        (i.e., combine the IN-sets of the successors in some way)

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

(b) Vul de vier '...' in de algemene opzet van het iteratieve algoritme in voor het berekenen van live variabelen. Voor iedere knoop B moeten IN[B] en OUT[B] aan het eind van het algoritme alle live variabelen 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 live variabelen 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.

(d) Pas nu het iteratieve algoritme om de live variabelen te berekenen toe op bovenstaand stroomdiagram. Laat met een overzichtelijke tabel zien wat de OUT en IN-verzameling van elk basic block (op EXIT 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.