HERTENTAMEN COMPILERCONSTRUCTIE
Dinsdag 10 maart 2015, 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. In deze opgave bekijken we drie context-vrije grammatica's G_1, G_2 en G_3 om eenvoudige expressies met $+$ en $-$ te genereren. In alle drie grammatica's is *list* de startvariabele.

   G_1 kent de volgende producties:

   $$\begin{aligned} \mathit{list} &\rightarrow \mathit{list} + \mathit{list} \mid \mathit{list} - \mathit{list} \mid \mathit{digit} \\ \mathit{digit} &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{aligned}$$

   G_2 kent de volgende producties:

   $$\begin{aligned} \mathit{list} &\rightarrow \mathit{list} + \mathit{digit} \mid \mathit{list} - \mathit{digit} \mid \mathit{digit} \\ \mathit{digit} &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{aligned}$$

   G_3 kent de volgende producties:

   $$\begin{aligned} \mathit{list} &\rightarrow \mathit{digit} + \mathit{list} \mid \mathit{digit} - \mathit{list} \mid \mathit{digit} \\ \mathit{digit} &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{aligned}$$

   (a) Wat is een belangrijk nadeel (uit oogpunt van compilerconstructie) van G_1 ten opzichte van G_2 en G_3. Leg ook uit waarom het een nadeel is.

   (b) Wat is een nadeel (uit oogpunt van compilerconstructie) van G_2 ten opzichte van G_3? Leg ook uit waarom het een nadeel is.

   (c) Wat is een nadeel (uit oogpunt van compilerconstructie) van G_3 ten opzichte van G_2? Leg ook uit waarom het een nadeel is.

---

2. (a) Wat bedoelen we als we zeggen dat een context-vrije grammatica G links-recursie kent?

   In het boek wordt een algoritme beschreven om directe (*immediate*) en indirecte links-recursie uit een context-vrije grammatica te elimineren, zonder de gegenereerde taal te veranderen. Dit algoritme begint ermee dat de variabelen in de grammatica geordend worden als $A_1, A_2, \ldots, A_n$ (als er n variabelen zijn).

   (b) Beschrijf (in woorden of pseudo-code, maar in ieder geval duidelijk en volledig) hoe dit algoritme verder gaat.

   (c) Pas het algoritme uit onderdeel (b) toe op de grammatica G_1 met startvariabele A en de volgende producties:

$$\begin{aligned} A &\rightarrow Ba \mid ab \\ B &\rightarrow Cba \mid \epsilon \\ C &\rightarrow ABc \mid cc \end{aligned}$$

   Leg uit hoe je te werkt gaat en geef tussenresultaten.

   N.B.: hoewel G_1 een ϵ-productie bevat, werkt het algoritme uit het boek wel gewoon.

   (d) Het algoritme uit onderdeel (b) werkt niet *altijd* voor een grammatica G met ϵ-producties. Geef een voorbeeld van een grammatica G met een ϵ-productie, waarbij er complicaties optreden als het algoritme op G wordt toegepast. Licht ook toe *wat* er 'misgaat'.

   (e) Laat G_2 de context-vrije grammatica zijn die het resultaat is van onderdeel (c). Pas indien mogelijk linksfactorisatie toe op de producties in G_2.

   (f) Laat G_3 de context-vrije grammatica zijn die het resultaat is van het vorige onderdeel. Is deze grammatica LL(1)? Motiveer je antwoord.

---

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

(1) $S \to \textbf{if}\ B\ S$
(2) $S \to \textbf{id} = \textbf{id}$
(3) $B \to B\ \textbf{boolop}\ E$
(4) $B \to E$
(5) $E \to \textbf{id}\ \textbf{relop}\ \textbf{id}$
(6) $E \to \textbf{id}$

(a) Geef (ad hoc) een afleidingsboom (parse tree) in G voor de string

if id boolop id id = id

(bijvoorbeeld overeenkomend met **if** ok1 && ok2 $a = b$).

(b) Gegeven is dat de SLR parsing table bij grammatica G er als volgt uit ziet:

State	Action						Goto		
	if	**id**	=	**boolop**	**relop**	$	S	B	E
0	s2	s10					1		
1						acc			
2		s7						3	13
3	s2	s10		s5			4		
4						r1			
5		s7							6
6	r3	r3		r3					
7	r6	r6		r6	s8				
8		s9							
9	r5	r5		r5					
10			s11						
11		s12							
12						r2			
13	r4	r4		r4					

Parse de string **if id boolop id id = id** met deze tabel. Laat bij iedere stap duidelijk zien wat je doet, bijvoorbeeld met behulp van een tabel van de volgende vorm:

States on stack	Corresponding symbols on stack	Input	Action
0	$	...	...
...	...	...	...

4. Tijdens het genereren van drie-adres code voor boolese expressies en *flow-of-control* instructies weten we bij goto-instructies vaak niet onmiddellijk waar we naartoe moeten springen. We kunnen *backpatching* gebruiken om dit achteraf op te lossen. Hierbij krijgt de variabele B in de grammatica (overeenkomend met een boolese expressie) attributen *truelist* en *falselist*. De variabele S (overeenkomend met een instructie) krijgt een attribuut *nextlist*.

   (a) Leg voor elk van deze drie lijsten uit wat de lijst bevat.

   Naast deze variabelen gebruiken we hieronder een hulpvariabele M (met een attribuut *instr*), en een variabele L (met een attribuut *nextlist*) voor een reeks instructies.

   Beschouw het volgende *translation scheme* voor het genereren van de genoemde drie-adres code:

$S \rightarrow \textbf{if}\ (B)\ MS_1$	{	$backpatch(B.truelist, M.instr);$
		$S.nextlist = merge(B.falselist, S_1.nextlist);\}$
$S \rightarrow \textbf{id}_1 = \textbf{id}_2;$	{	$S.nextlist = \textbf{null};$
		$gen(\textbf{id}_1.addr\ '='\ \textbf{id}_2.addr);\}$
$S \rightarrow \{L\}$	{	$S.nextlist = L.nextlist;\}$
$L \rightarrow L_1 MS$	{	$backpatch(L_1.nextlist, M.instr);$
		$L.nextlist = S.nextlist;\}$
$L \rightarrow S$	{	$L.nextlist = S.nextlist;\}$
$B \rightarrow B_1\ \&\&\ MB_2$	{	$backpatch(B_1.truelist, M.instr);$
		$B.truelist = B_2.truelist;$
		$B.falselist = merge(B_1.falselist, B_2.falselist);\}$
$B \rightarrow \textbf{id}_1\ \textbf{rel}\ \textbf{id}_2$	{	$B.truelist = makelist(nextinstr);$
		$B.falselist = makelist(nextinstr + 1);$
		$gen('\texttt{if}'\ \textbf{id}_1.addr\ \textbf{rel}.op\ \textbf{id}_2.addr\ '\texttt{goto}\ \_');$
		$gen('\texttt{goto}\ \_');\}$
$M \rightarrow \epsilon$	{	$M.instr = nextinstr;\}$

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

```
{ if (a==b && c<d)
    x=y;
  y=z;
}
```

   (c) Pas bij de afleidingsboom van het vorige onderdeel de semantische acties toe zoals beschreven in het translation scheme. Geef bij elke variabele in de boom aan wat de attributen (*truelist*, *falselist*, *nextlist* en/of *instr*) worden. Geef ook de resulterende drie-adres code. Ga ervanuit dat deze drie-adres code begint op instructienummer 100.

   (d) Leg uit wat er volgens het translation scheme gebeurt (de semantische actie) bij de productie

$$S \rightarrow \textbf{if}\ (B)\ MS_1$$

   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. (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] = $\ldots_{\text{predecessors } P \text{ of } B}$ OUT[P]
           (i.e., combine the OUT-sets of the predecessors in some way)

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

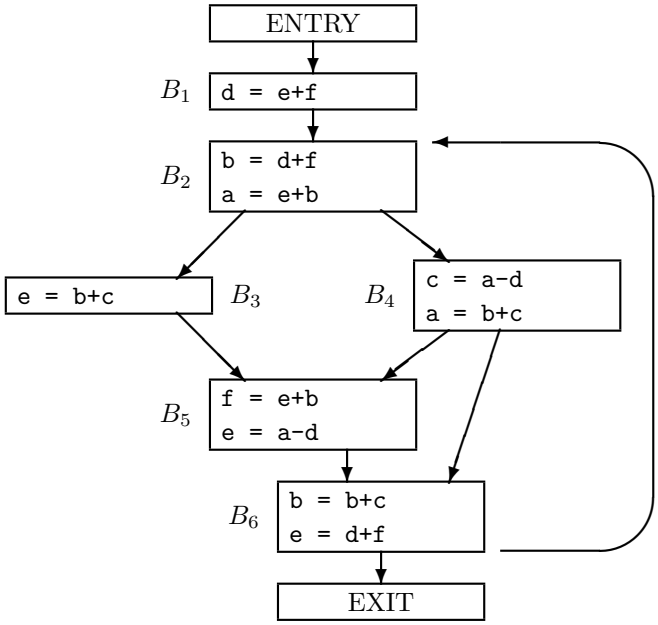
   (b) Leg uit waarom we een *voorwaartse* (dus niet een achterwaartse) dataflow analyse gebruiken voor het berekenen van available expressions in een stroomdiagram (*flow graph*).

   (c) 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'.

Z.O.Z.

Beschouw nu het volgende stroomdiagram:



(d) 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.

(e) 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.