HERTENTAMEN COMPILERCONSTRUCTIE
Dinsdag 26 maart 2013, 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. Beschouw de context-vrije grammatica G met startvariabele S en de volgende producties:

$$\begin{aligned} S &\rightarrow \textbf{if } B\ S \mid \textbf{id} = \textbf{id} \\ B &\rightarrow B\ \textbf{boolop}\ E \mid E \\ E &\rightarrow \textbf{id relop id} \mid \textbf{id} \end{aligned}$$

   G is dus een eenvoudige grammatica voor een instructie in een programmeertaal. S, B, E zijn de variabelen en $\textbf{if}, \textbf{id}, =, \textbf{boolop}, \textbf{relop}$ zijn de terminalen in G.

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

   (b) Bepaal voor elke rechterkant α van een productie $A \rightarrow \alpha$ in G de verzameling FIRST(α).

   (c) Is G een LL(1) grammatica? Motiveer je antwoord.

   (d) Construeer vanuit G een nieuwe context-vrije grammatica G' met $L(G') = L(G)$, door

   - (indien van toepassing) links-recusie te elimineren, en
   - (indien van toepassing) links-factorisatie toe te passen.

   Leg uit hoe je G' uit G verkrijgt.

   (e) Bepaal voor elke variabele in de nieuwe grammatica G' zowel de FIRST- als de FOLLOW-verzameling.

   (f) Construeer de top-down *parsing table* bij de nieuwe grammatica G'.

   (g) Is de nieuwe grammatica G' een LL(1) grammatica? Motiveer je antwoord.

Beschouw nu de context-vrije grammatica H met startvariabele S en de volgende producties:

$$\begin{array}{lcl} S & \rightarrow & BA \\ A & \rightarrow & -BA \mid \epsilon \\ B & \rightarrow & DC \\ C & \rightarrow & DC \mid \epsilon \\ D & \rightarrow & pE \mid qE \\ E & \rightarrow & +E \mid \epsilon \end{array}$$

(h) Geef (ad hoc) een afleidingsboom in H voor de string $q + p$.

(i) Gegeven is dat de top-down parsing table bij grammatica H er als volgt uit ziet:

Non-terminal	Input Symbol				
	$-$	$+$	p	q	\$
S			$S \rightarrow BA$	$S \rightarrow BA$	
A	$A \rightarrow -BA$				$A \rightarrow \epsilon$
B			$B \rightarrow DC$	$B \rightarrow DC$	
C	$C \rightarrow \epsilon$		$C \rightarrow DC$	$C \rightarrow DC$	$C \rightarrow \epsilon$
D			$D \rightarrow pE$	$D \rightarrow qE$	
E	$E \rightarrow \epsilon$	$E \rightarrow +E$	$E \rightarrow \epsilon$	$E \rightarrow \epsilon$	$E \rightarrow \epsilon$

Parse de string $q+p$ met deze tabel. Laat bij iedere stap duidelijk zien wat je doet, bijvoorbeeld met behulp van een tabel van de volgende vorm:

Matched	Stack	Input	Action
	$S\$$	$q + p\$$	output $S \rightarrow BA$
$\ldots$	$\ldots$	$\ldots$	$\ldots$

---

3. Het *type* van een variabele of expressie in een programmeertaal kan door de compiler voor verschillende doelen gebruikt worden, bijvoorbeeld voor *type checking*. Type checking is een vorm van static checking.

   (a) Wat verstaan we onder type checking?

   (b) Noem twee andere doelen waarvoor de compiler het type van een variabele of expressie kan gebruiken.

   (c) Geef een voorbeeld van een andere vorm van static checking.

---

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.

   (a) Wanneer noemen we een attribuut *inherited* en wanneer *synthesized*?

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

Production	Semantic Rules
$S \rightarrow \textbf{if}\ (B)\ S_1$	$B.true = newlabel()$
	$B.false = S_1.next = S.next$
	$S.code = B.code \ \ \|\| \ \ label(B.true) \ \ \|\| \ \ S_1.code$

   Hierin komt de variabele S (en ook S_1) overeen met een instructie, en de variabele B met een boolese expressie. Zowel B als S heeft een attribuut *code*, S heeft daarnaast een attribuut *next*, en B heeft attributen *true* en *false*.

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

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

   (d) Leg uit wat er gebeurt (de semantic rules) bij de productie $S \rightarrow$ **if** $(B)\ S_1$. Leg ook uit waarom dat gebeurt.

---

5. (a) Bij het opsplitsen van drie-adres code in *basic blocks* maken we gebruik van *leaders*. In welke gevallen is (in het algemeen) een instructie in de drie-adres code een leader?

   We bekijken nu een algoritme om het minimale element in een array met integers te verplaatsen naar positie 0 in het array[1]:

```
i = 0;
min = a[i];
while (i < n-1)
{ i = i+1;
  if (a[i] < min)
  { a[0] = a[i];
    a[i] = min;
    min = a[0];
  }
}
```

   Als we aannemen dat een integer vier bytes in beslag neemt, kan een rechttoe-rechtaan vertaling van dit algoritme in drie-adres code het volgende opleveren (zie volgende pagina):

[1] Dit algoritme is bijvoorbeeld te gebruiken voor SelectionSort.

```
(1)  i = 0
(2)  t1 = 4*i
(3)  min = a[t1]
(4)  t2 = n-1
(5)  if i < n-1 goto 7
(6)  goto 21
(7)  i = i+1
(8)  t3 = 4*i
(9)  t4 = a[t3]
(10) if t4 < min goto 12
(11) goto 20
(12) t5 = 4*0
(13) t6 = 4*i
(14) t7 = a[t6]
(15) a[t5] = t7
(16) t8 = 4*i
(17) a[t8] = min
(18) t9 = 4*0
(19) min = a[t9]
(20) goto 4
(21) ...
```

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

(c) Teken de *flow graph* met de basic blocks bij regels 1–20 van bovenstaande drie-adres code. Nummer de basic blocks $B_1, B_2, \ldots$.

(d) Je moet de drie-adres code in regels 1–20 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.

(e) Zijn er voor de drie-adres code uit deze opgave nog andere soorten optimalisaties mogelijk dan degene die je bij het vorige onderdeel mocht gebruiken? Zo ja, welke en waar in de code.