HERTENTAMEN COMPILERCONSTRUCTIE
Donderdag 23 januari 2020, 14.15 – 17.15 uur

Dit tentamen bestaat uit zes 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. [29 pt] Beschouw de context-vrije grammatica G met startvariabele S en de volgende producties:

$$\begin{array}{lcl} S & \rightarrow & (T) \mid aS \mid b \\ T & \rightarrow & TS \mid \epsilon \end{array}$$

   De variabele S zou bijvoorbeeld kunnen staan voor een statement, en de variabele T voor een statementlist, met bijpassende interpretaties van de terminals (,), a en b.

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

   (b) Bij het SLR parsen maken we gebruik van een LR(0)-automaat. De toestanden in deze automaat bevatten items van de (algemene) vorm $A \rightarrow \beta_1 \cdot \beta_2$.
   Leg uit wat het betekent wanneer we tijdens het SLR parsen in een toestand met daarin het item $A \rightarrow \beta_1 \cdot \beta_2$ zijn aangekomen.

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

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

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

---

2. [8 pt] Deze opgave gaat over *recursive-descent parsing*?

   (a) Is recursive-descent parsing een vorm van top-down parsing of een vorm van bottom-up parsing?

   (b) Stel dat je bij recursive-descent parsing in een procedure/functie bezig bent met een productie $A \rightarrow X_1 X_2 \ldots X_k$. Leg uit hoe je dan de symbolen X_i in de rechterkant verwerkt. Maak hierbij onderscheid tussen terminals en nonterminals X_i. In plaats van een uitleg in woorden, mag je ook een (volledige) algoritmische beschrijving geven.

---

3. [14 pt]

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

   (b) Verschillende typen in een programmeertaal kunnen naar elkaar geconverteerd worden.

      i. Wanneer spreken we van een *widening conversion* en wanneer van een *narrowing conversion* ? Geef van elk van deze twee typen conversie een voorbeeld.

      ii. Wanneer spreken we van een *coercion* ?

---

4. [33 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
$P \rightarrow S$	$S.next = newlabel()$
	$P.code = S.code \;\; \| \| \;\; label(S.next)$
$S \rightarrow \textbf{if}\ (B)\ S_1$	$B.true = newlabel()$
	$B.false = S_1.next = S.next$
	$S.code = B.code \;\; \| \| \;\; label(B.true) \;\; \| \| \;\; S_1.code$
$B \rightarrow \textbf{id}_1\ \textbf{rel}\ \textbf{id}_2$	$B.code = gen('\texttt{if}'\ \ \textbf{id}_1.lexeme\ \ \textbf{rel}.op\ \ \textbf{id}_2.lexeme\ \ '\texttt{goto}'\ B.true)$
	$\| \| \;\; gen('\texttt{goto}'\ B.false)$
$S_1 \rightarrow \textbf{id} = \textbf{num};$	$S_1.code = gen(\textbf{id}.lexeme\ ' =' \ \textbf{num}.val);$
$S \rightarrow \textbf{if}\ (B)\ S_1\ \textbf{else}\ S_2$	$\ldots$

   Hierin is P de startvariabele, en komt de variabele S (en ook S_1) overeen met een instructie, en de variabele B met een boolese expressie. Het token **rel** staat voor een relationale 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 *code*, *next*, *true* en *false* is/zijn synthesized en welk(e) inherited?

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

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

   Leg ook uit waarom dat gebeurt.

   N.B.: het gaat er hier om wat er in het algemeen gebeurt bij deze productie.

   (d) Geef nu ook de semantische regels bij de productie $S \rightarrow \textbf{if}\ (B)\ S_1\ \textbf{else}\ S_2$

   (e) Teken de *parse tree* bij bovenstaande grammatica voor het volgende 'programma':

```
if (a<b)
  x=24;
else
  x=42;
```

   (f) 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 $L1, L2, \ldots$

5. [10 pt]

   (a) Voor de selectie van een register voor een variabele kunnen we gebruik maken van *register descriptors* en *address descriptors*. Wat verstaan we onder deze twee 'descriptors'?

   (b) Ga er nu vanuit dat de inhoud van alle descriptors klopt, en dat we vervolgens de volgende assembly instructie uitvoeren:

   $$\texttt{LD}\ R_i, x$$

   Hoe moeten de verschillende register descriptors en address descriptors worden aangepast, om het effect van deze assembly instructie te weerspiegelen? Geef de benodigde aanpassingen van de descriptors in een volgorde die efficiënt algoritmisch is uit te voeren.

---

6. [6 pt] Om de drie-adres code in een *basic block* te optimaliseren, kunnen we een compleet block transformeren in een gerichte acyclische graaf. Teken de gerichte acyclische graaf bij het basic block bestaande uit de volgende reeks instructies:

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

   Licht toe welke informatie je allemaal in/bij de knopen in de graaf opslaat.
