HERTENTAMEN COMPILERCONSTRUCTIE
Dinsdag 8 maart 2016, 14:00 – 17:00 uur

Dit tentamen bestaat uit 5 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. [7 pt] In een programmeertaal als C++ kan dezelfde variabele `x` meerdere keren gedeclareerd worden, ook binnen geneste *blocks*.

   (a) Beschouw het volgende (eenvoudige) stukje code:

```
{ int x = 7;
  int z = 3;
  { int x,y;
    y = 14;
    x = y+z;
    cout << x << endl;
  }
  x = x+z;
  cout << x << endl;
}
```

   Wat wordt de uitvoer bij dit stukje code?

   (b) Leg uit hoe je de Symbol Table zo kunt inrichten, dat de compiler overweg kan met variabelen die meerdere keren gedeclareerd worden. Hoe bepaalt de compiler dan welke declaratie van toepassing is, als hij een voorkomen van zo'n variabele tegenkomt in een instructie?

---

2. [31 pt] Beschouw de context-vrije grammatica G met startvariabele B en de volgende producties:

$$\begin{aligned} B &\rightarrow B \textbf{ boolop } E \mid E \\ E &\rightarrow \textbf{id } E' \\ E' &\rightarrow \textbf{relop id} \mid \epsilon \end{aligned}$$

G is een eenvoudige grammatica voor een boolese expressie in een programmeertaal. B, E, E' zijn de variabelen en **id**, **boolop**, **relop** zijn de terminalen in G.

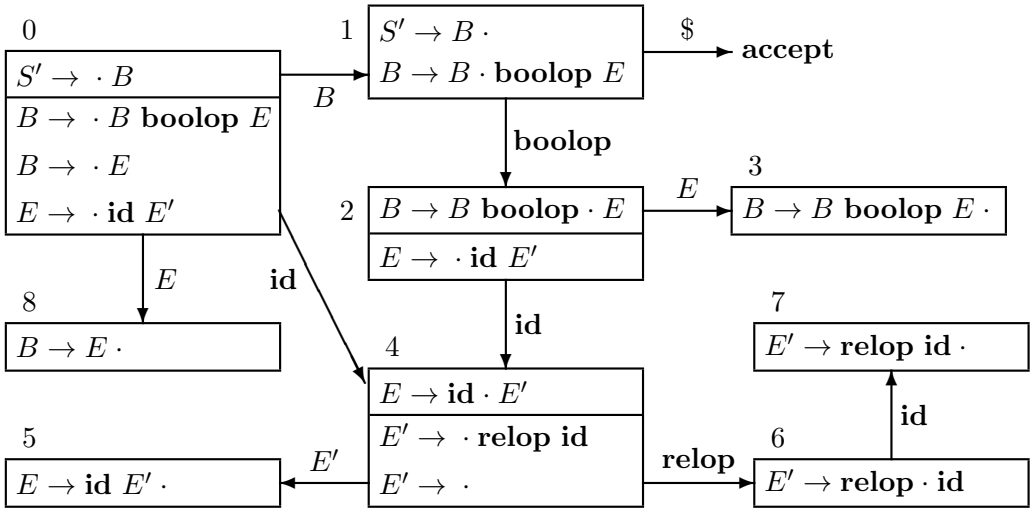
(a) Geef (ad hoc) een afleidingsboom (*parse tree*) in G voor de string **id boolop id relop id** (bijvoorbeeld overeenkomend met de expressie `OK && i<j`).

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

(c) Construeer de top-down *parsing table* bij grammatica G. Wellicht ten overvloede: dit is dus iets anders dan de SLR parsing table.

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

(e) Gegeven is dat de LR(0)-automaat bij grammatica G er als volgt uitziet:



Construeer de SLR *parsing table* bij grammatica G.

(f) Parse de string **id boolop id relop id** met de SLR parsing table van onderdeel (e). 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
...	...	...	...

3. [25 pt] Beschouw de volgende context-vrije grammatica (met startvariabele S) om toekenningen van expressies aan identifiers te genereren:

$$\begin{array}{lcl} S & \rightarrow & \textbf{id} = E \\ E & \rightarrow & E + E \mid -E \mid (E) \mid \textbf{id} \end{array}$$

   (a) Teken een afleidingsboom (*parse tree*) voor de instructie `a = b + (-c) + d`. Houd er rekening mee dat de operator + links-associatief is.

   (b) Teken ook een met deze instructie corresponderende syntax tree.

   Om bij dergelijke instructies drie-adres code te genereren, kunnen we gebruik maken van de volgende *syntax-directed definition*, waarbij de *semantic rule* voor de productie $E \rightarrow E_1 + E_2$ ontbreekt:

Production	Semantic Rules
$S \rightarrow \textbf{id} = E;$	$S.code = E.code \;\; \|\|$ $gen(top.get(\textbf{id}.lexeme)\ '='\ E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = \ldots$
	$E.code = \ldots$
$\mid -E_1$	$E.addr = \textbf{new}\ Temp()$
	$E.code = E_1.code \;\; \|\|$ $gen(E.addr\ '='\ '\textbf{minus}'\ E_1.addr)$
$\mid (E_1)$	$E.addr = E_1.addr$
	$E.code = E_1.code$
$\mid \textbf{id}$	$E.addr = top.get(\textbf{id}.lexeme)$
	$E.code = \ ''$

   De variabele S heeft dus een attribuut *code* en de variabele E heeft attributen *code* en *addr*.

   (c) Wat is de functie van de aanroep van $top.get()$ in de eerste en de laatste semantic rule?

   (d) Wat is de semantic rule voor de productie $E \rightarrow E_1 + E_2$?

   (e) Geef zowel voor het attribuut *code* als het attribuut *addr* aan of het een *synthesized* attribuut of een *inherited* attribuut is. Motiveer je antwoord.

   (f) Annoteer de afleidingsboom van onderdeel (a). Dat wil zeggen: vul voor elk (voorkomen van een) variabele in de boom de attributen *code* en (indien van toepassing) *addr* in, zoals gespecificeerd door de syntax-directed definition. Ga er hierbij vanuit dat benodigde nieuwe tijdelijke variabelen achtereenvolgens t1, t2, ... heten.

---

4. [12 pt]

   (a) Beschrijf vier soorten elementen van een *activation record*.

   (b) Het run-time geheugen valt onder te verdelen in de vier gebieden *Code*, *Heap*, *Stack* en *Static*.

   Leg globaal uit waar elk van deze vier gebieden toe dient (een zin per gebied is in principe voldoende). Welk(e) van de vier gebieden is/zijn beschikbaar voor dynamische geheugenallocatie? Waar worden de activation records opgeslagen?

---

5. [25 pt]

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

Z.O.Z.

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

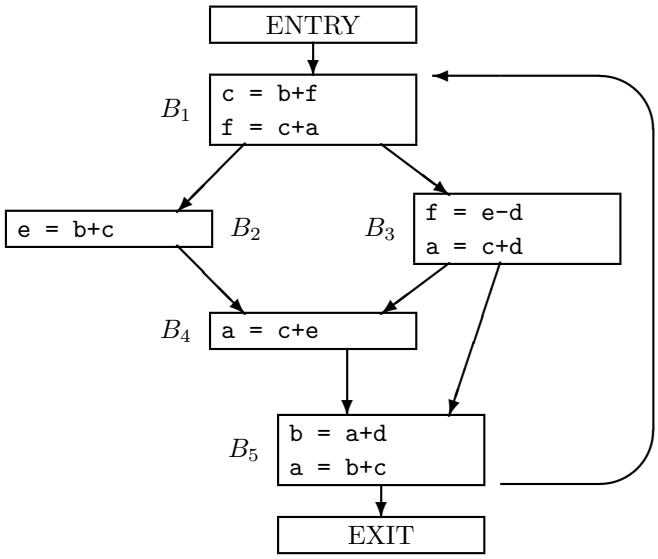
IN[EXIT] = ...
for each node B other than EXIT
    IN[B] = ...

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.