Uitwerking tentamen Compilerconstructie, vrijdag 21 december 2012

10:51

1 (a)
* naam van de variabele (zijn 'lexeme'),
  dit kan al bij de lexical analysis, omdat je daar vaststelt dat het aan de syntax van een variabele naam voldoet
  Syntax analyse is ook goed (en misschien wel beter?), omdat je dan pas weet in welke symbol table de variabele moet komen
* type van de variabele
  dit kan bij de syntax analysis, omdat je daar vaststelt dat de variabele op de juiste (volgens de grammatica) manier wordt voorafgegaan of gevolgd door een type
* plaats (adres) in het geheugen van de variabele / offset
  dit kan pas bij de code generatie, omdat je daar b.v. weet voor welk soort machine je aan het compileren bent. Daar heb je dit adres ook pas nodig.

11:00
* soort symbool: variabele / functie / tijdelijke variabele
  dit kan bij de syntax analysis / het parsen, want daar bepaal je pas wat een lexeme voorstelt (voor variabele / functie). Tijdelijke variabelen komen pas bij intermediate code generation
* scope van symbool (impliciet via de symbol table waar hij in zit)
  dit kan bij de syntax analysis / het parsen, want daar ontdek je pas de verschillende scopes.
* regelnummer van declaratie
  dit kan bij lexical analyse al (daarna ben je regel overgangen kwijt?)

11:00

(b) De compiler heeft niet één algemene symbol table, maar een hierarchie van symbol tables, omdat variabelen een 'scope' hebben, waarbinnen ze gedeclareerd zijn. Daarbuiten mogen ze niet gebruikt worden. Sterker nog: daarbuiten kan er een andere variabele met dezelfde naam gedeclareerd zijn.

11:03

2 (a)

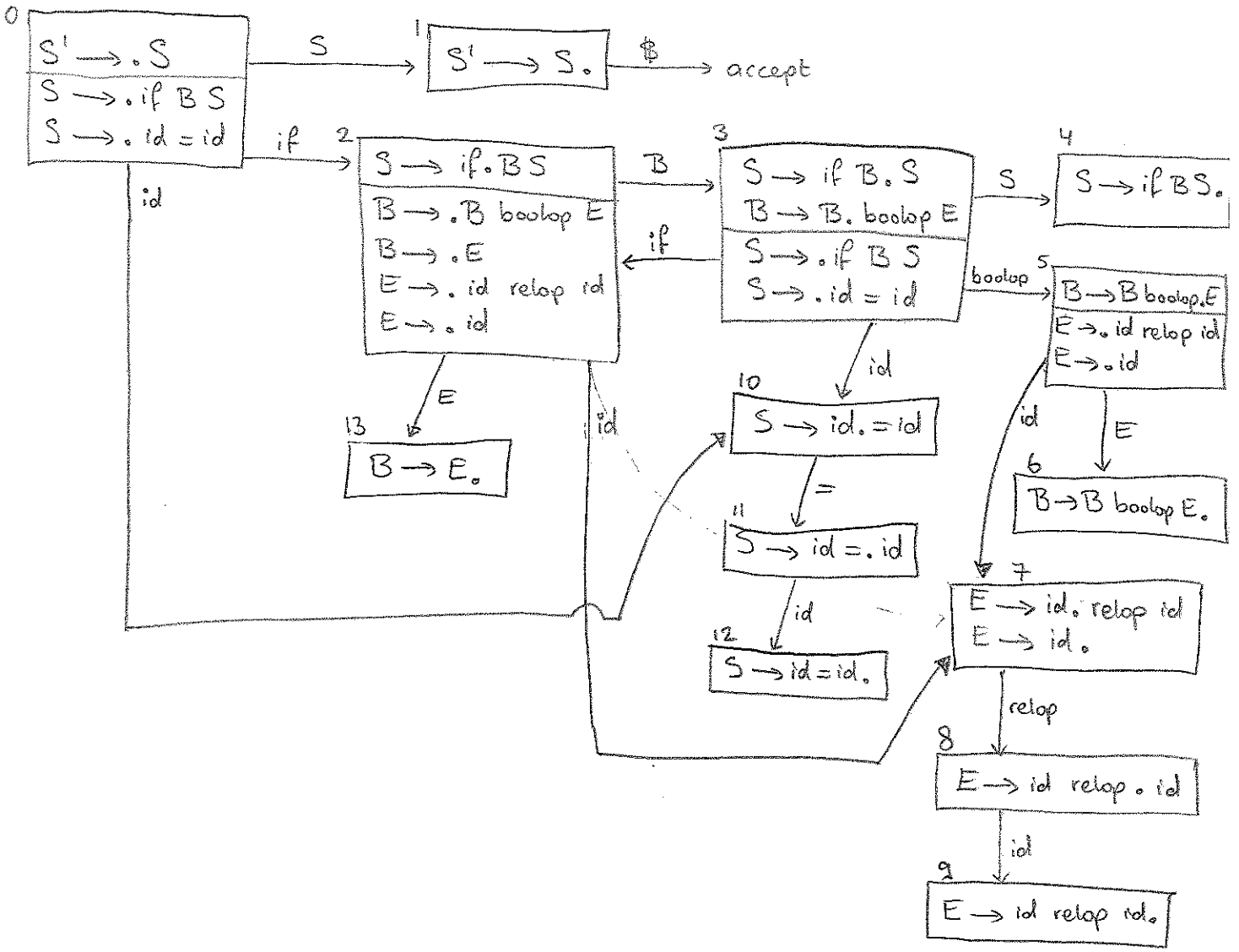
First (E) = {id}
First (B) = First (E) = {id}
First (S) = {if, id}
Follow (S) = {$}
(want Startsymbool)
Follow (B) = First (S) ∪ {boolop} = {if, id, boolop}
(vanwege S → if B S; vanwege B → B boolop E.)
Follow (E) = Follow (B) = {if, id, boolop}
(vanwege B → B boolop E en B → E.)

11:09

1 (a) Vervolg
* address-descriptor: op welke locatie(s) bevindt zich huidige waarde van variabele
  dit wordt tijdens code-generatie bijgehouden, want daar gebruik je deze informatie om efficiënt met registers om te gaan

11:09 **Uitwerking tentamen Compilerconstructie, vrijdag 21 december 2012**

2 (b) Introduceer een start-productie: $S' \rightarrow S$



11:25 Controle

11:29 (c)

state	if	id	=	boolop	relop	$	S	B	E
	Actron						Goto		
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					

11:35 (onderbroken)

11:47

Nummer de producties:

1. $S \rightarrow$ if B S
2. $S \rightarrow$ id = id
3. $B \rightarrow B$ boolop E
4. $B \rightarrow E$
5. $E \rightarrow$ id relop id
6. $E \rightarrow$ id

11:56 aanvulle

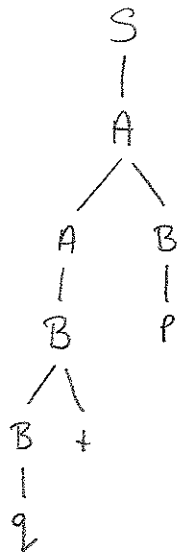
12:06

Uitwerking tentamen Compilerconstructie, vrijdag 21 december 2012

(d) Ja, G is een SLR-grammatica, want elk vakje in de SLR parsing table van het vorige onderdeel bevat hooguit één actie.

12:02

(e)



12:05

(f)

States on stack	Corresponding Symbols on stack	Input	Action
0	–	q+p $	Shift 7 (q)
0 7	q	+p $	Reduce by (7): B → q
0 8	B	+p $	Shift 5 (+)
0 8 5	B+	p $	Reduce by (5): B → B+
0 8	B	p $	Reduce by (4): A → B
0 9	A	p $	Shift 6 (p)
0 9 6	Ap	$	Reduce by (6): B → p
0 9 4	AB	$	Reduce by (3): A → AB
0 9	A	$	Reduce by (2): S → A
0 1	S	$	accept

De acties hierboven komen overeen met een LRW-wandeling door de afleidingsboom bij (e).

12:15

3 (a) We spreken van een widening conversion, als er bij de conversie geen informatie verloren kan gaan (zodat)
Vb: x = (float) y als x een float-variabele is en y een int-variabele.

(als we van een 'klein' type naar een 'groot' type gaan)

We spreken van een narrowing conversion, als bij de conversie wel (we van een 'groot' type naar een 'klein' of even groot type gaan, zodat er)
informatie verloren kan gaan
Vb: y = (int) x als x een float-variabele is en y een int-variabele

12:19

(b) We spreken van een coercion, als de conversie impliciet is, en automatisch door de compiler kan worden uitgevoerd

12:21

4 (a)

B.truelist bevat de adressen van Goto-instructies die (allemaal) naar het adres toe moeten, waar we heengaan als de boolese expressie B true is.

B.falselist bevat de adressen van Goto-instructies die (allemaal) naar het adres toe moeten waar we heengaan als de boolese expressie B false is

S.nextlist bevat de adressen van Goto-instructies die (allemaal) naar het adres toe moeten waar we heengaan als we klaar zijn met instructie S (~~de~~ het adres waar de instructie ná S begint)

12:27.
09:50

(b) De variabelen B_1 en B_2 hebben allebei hun eigen truelist en falselist

Om de waarde van B te bepalen, wordt eerst de waarde van B_1 geëvalueerd. Wanneer diet waarde false is, moet ook de waarde van B_2 geëvalueerd worden. Dus de goto's in B_1.falselist moeten naar het begin van B_2. Het adres waar B_2 begint, komt in M.instr, (want nextinstr = eerstvolgende adres). De instructie backpatch (B_1.falselst, M.instr) zorgt ervoor dat alle goto's (op de adressen) in B_1.falselist naar M.instr gaan.

De waarde van B is true als òf de waarde van B_1 òf de waarde van B_2 true is. Of andersom: als de waarde van B_1 (of B_2) true is, is ook de waarde van B true, en moet je naar de instructie toespringen die daarby hoort. Daarom worden met de instructie
B.truelist = merge (B_1.truelist, B_2.truelist) de truelists van B_1 en B_2 samengevoegd tot B.truelist. Als je straks weet naar welk adres je moet springen als B true is, kun je ook de goto's in B_1.truelist en B_2.truelist naar dat adres sturen.
Uiteraard is B.truelist niet groter dan de combinatie van B_1.truelist en B_2.truelist, want B bestaat alleen maar uit B_1 en B_2.

De waarde van B is false als zowel de waarde van B_1 als de waarde van B_2 false is. We gaan B_2 alleen evalueren wanneer B_1 false is (want als B_1 true is, wordt B sowieso true). Kortom, als je ontdekt dat B_2 false is, moet je eerder al ontdekt hebben / weet je zeker dat ook B_1.false is, En dat geval weet je dus dat ook B.false is.
Als je straks weet naar welk adres je moet springen als B false is, kun je de goto's in B_2.falselist naar dat adres sturen. Vandaar de instructie
B.falselist = B_2.falselist

Omdat je pas kunt concluderen dat B false is, wanneer B_2 false is (op het moment dat en niet eerder), is B.falselist ook niet groter dan B_2.falselist.

10:16

(c) Dit gaat symmetrisch aan de productie $B \rightarrow B_1 \,||\, M\, B_2$.

Bij $B \rightarrow B_1 \,||\, M B_2$ moet je B_2 evalueren als B_1 false is
Bij $B \rightarrow B_1 \,\&\&\, M B_2$ " " " " " B_1 true is, want alleen dan kan B true worden

(1) $\Rightarrow$ backpatch (B_1.truelist, M.instr);

Bij $B \rightarrow B_1 \,||\, M B_2$ is B true als B_1 of B_2 true is
Bij $B \rightarrow B_1 \,\&\&\, M B_2$ is B false " " " " false is

(2) $\Rightarrow$ B.falselist = merge (B_1.falselist, B_2.falselist);

Bij $B \rightarrow B_1 \,||\, M B_2$ is (door de volgorde van evalueren) B false als je ontdekt dat B_2 false is

Bij $B \rightarrow B_1 \,\&\&\, M B_2$ " " " " " " B true als je ontdekt dat B_2 true is

(3) $\Rightarrow$ B.truelist = B_2.truelist;

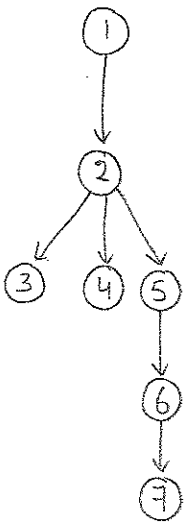
De instructies (1), (2) en (3) bij elkaar vormen dus de semantic action bij de productie $B \rightarrow B_1 \,\&\&\, M B_2$

10:24.

5 (a) Knoop d in een flow graph met beginknoop 'entry' is een dominator van knoop n, wanneer ieder pad in de flow graph van entry naar n ook knoop d bevat. Ofwel: om in n te komen, moet je via d gaan.

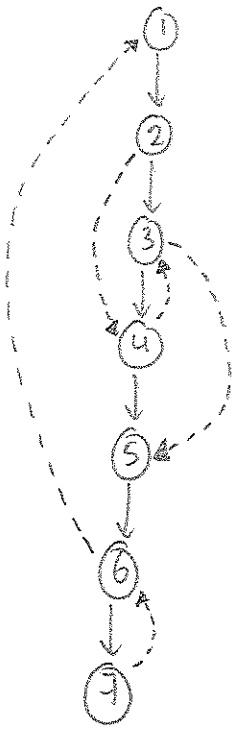
10:28

(b)



10:30

(c) We tekenen de complete flow graph G in in T:



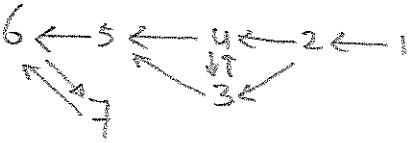
De retreating edges van flow graph G bij boom T zijn alle takken van een knoop naar een voorouder in T:
$4 \rightarrow 3$, $6 \rightarrow 1$ en $7 \rightarrow 6$

De back edges van G zijn alleen die retreating edges $n \rightarrow d$ waarvoor d een dominator van n is: $6 \rightarrow 1$ en $7 \rightarrow 6$

10:35

(d) We vinden de natural loop bij een back edge $n \rightarrow d$ door vanaf n terug te lopen in graaf G, totdat we tegen d aanlopen. Alle knopen die we zo tegenkomen, horen bij de natural loop

Bij $6 \rightarrow 1$



Resultaat: $\{1, 2, 3, 4, 5, 6, 7\}$, ofwel alle knopen

Bij $7 \rightarrow 6$

Resultaat: $\{6, 7\}$

10:40