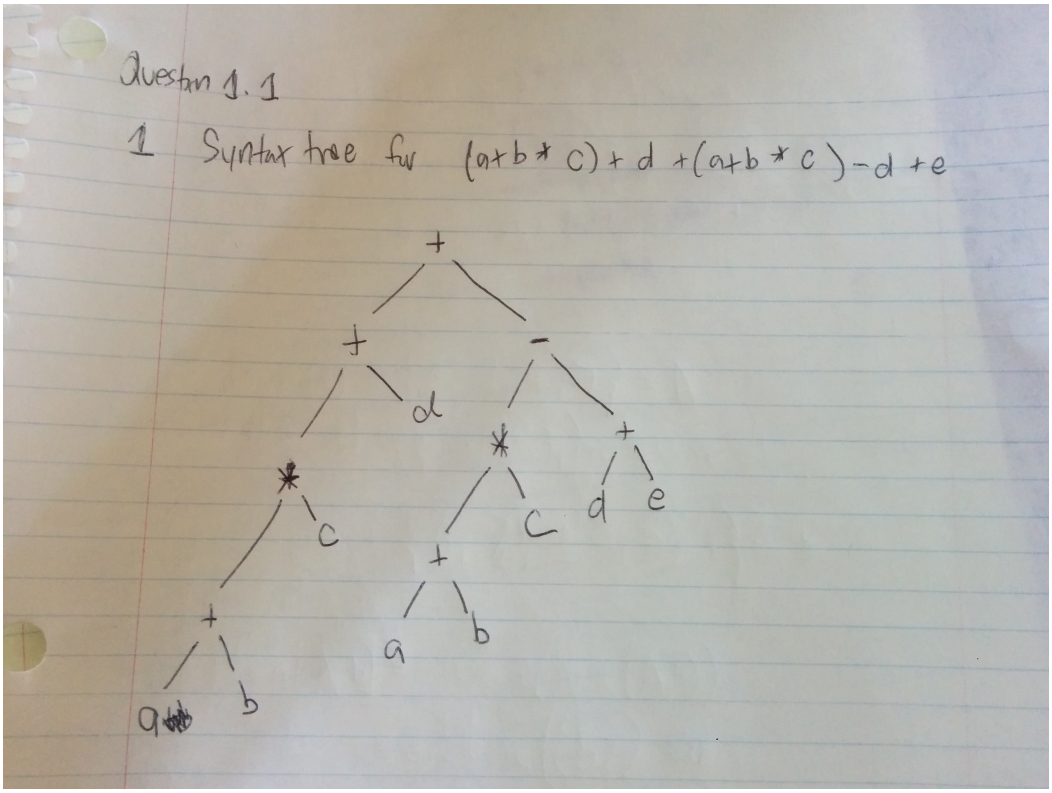
Homework 7

Abhi Agarwal

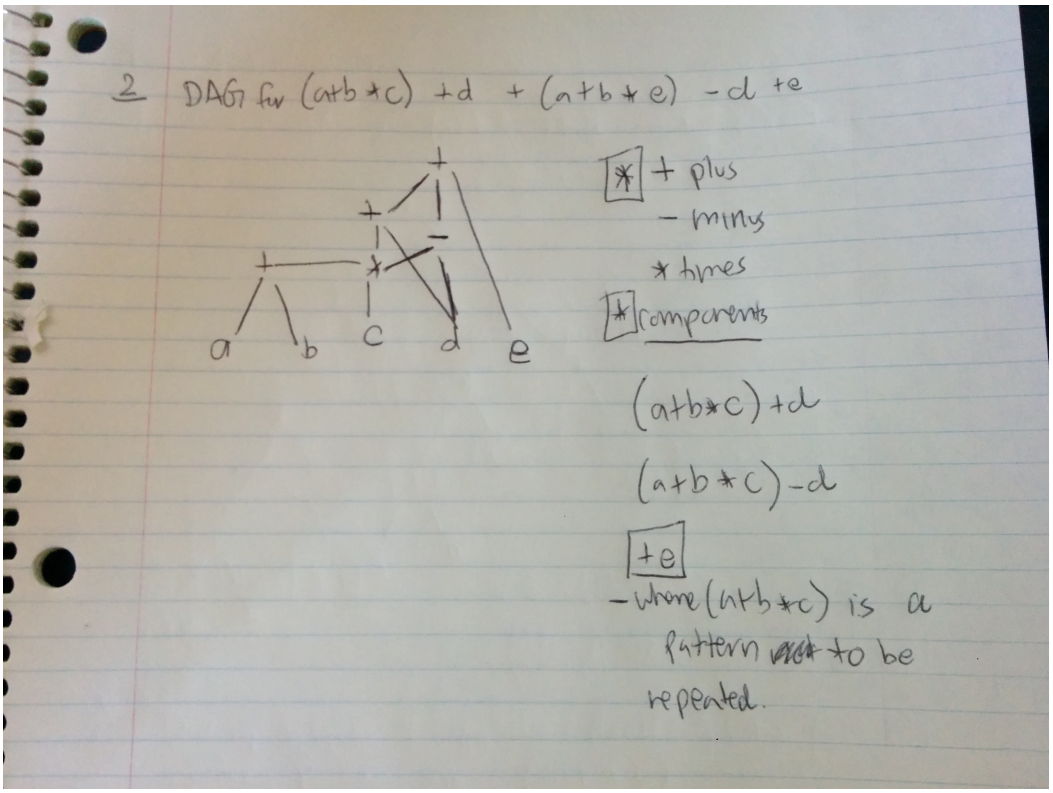
1 Representation

1.1 Question 1.1

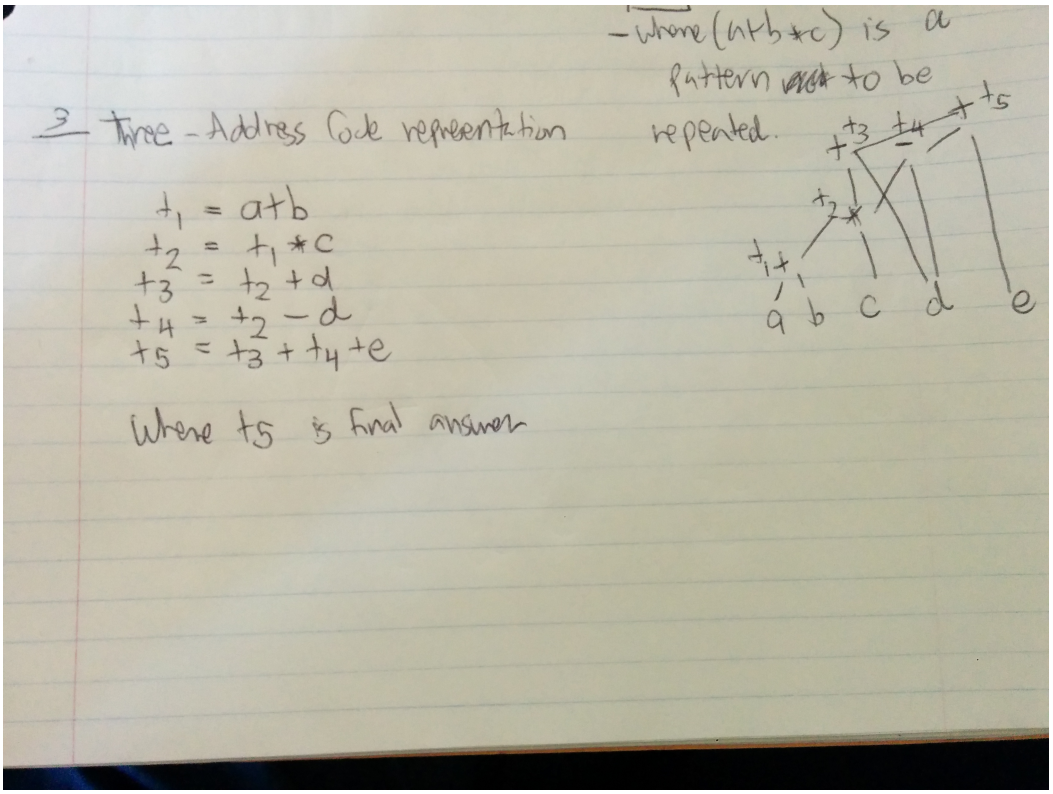
1.1.1 Syntax Tree



1.1.2 DAG (Directed Acyclic Graph) representation



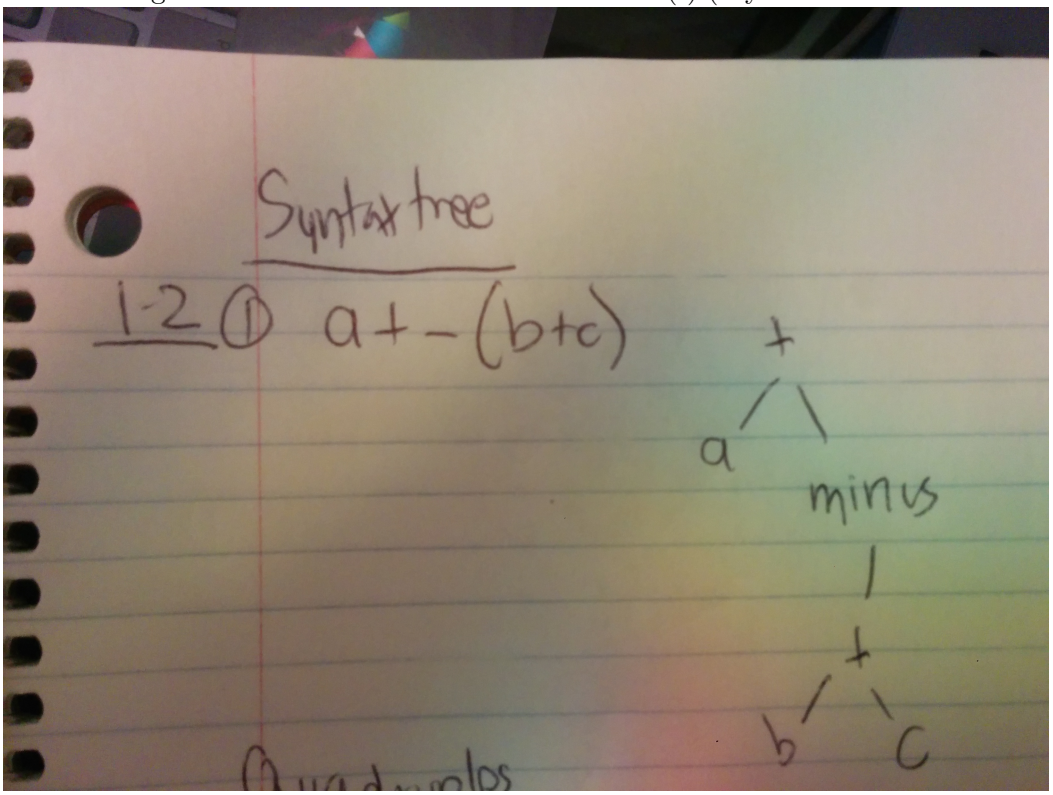
1.1.3 Three-Address Code representation



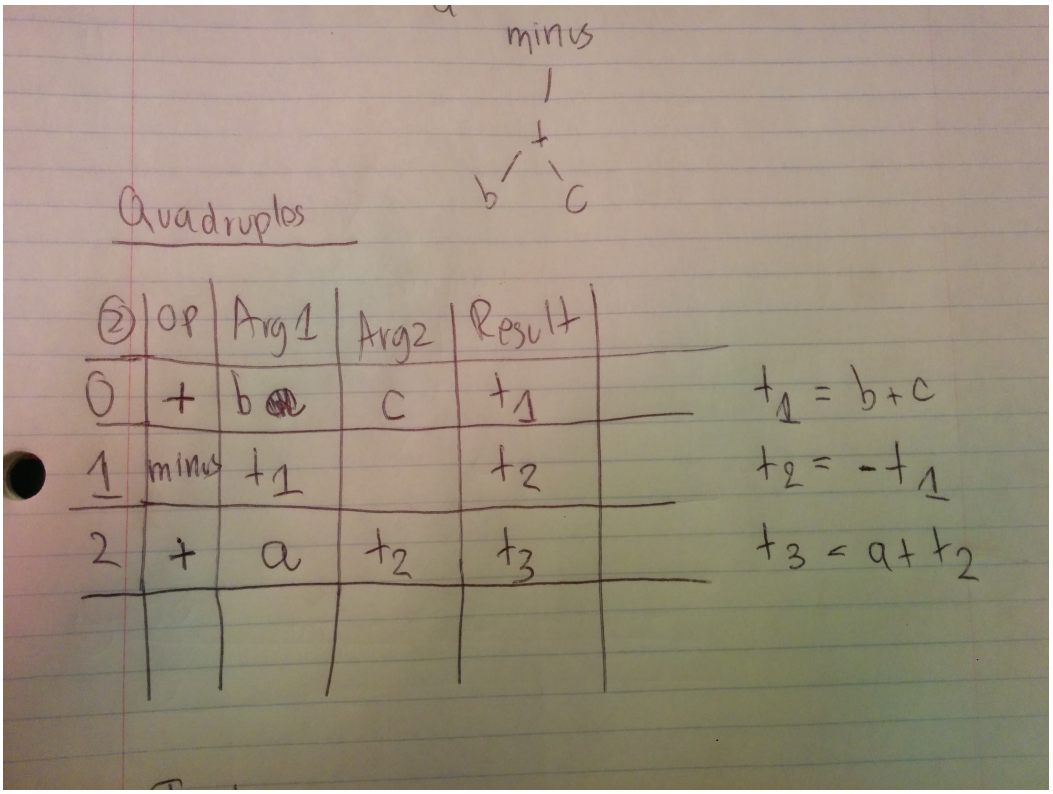
1.2 Question 1.2

1.2.1 Syntax Tree

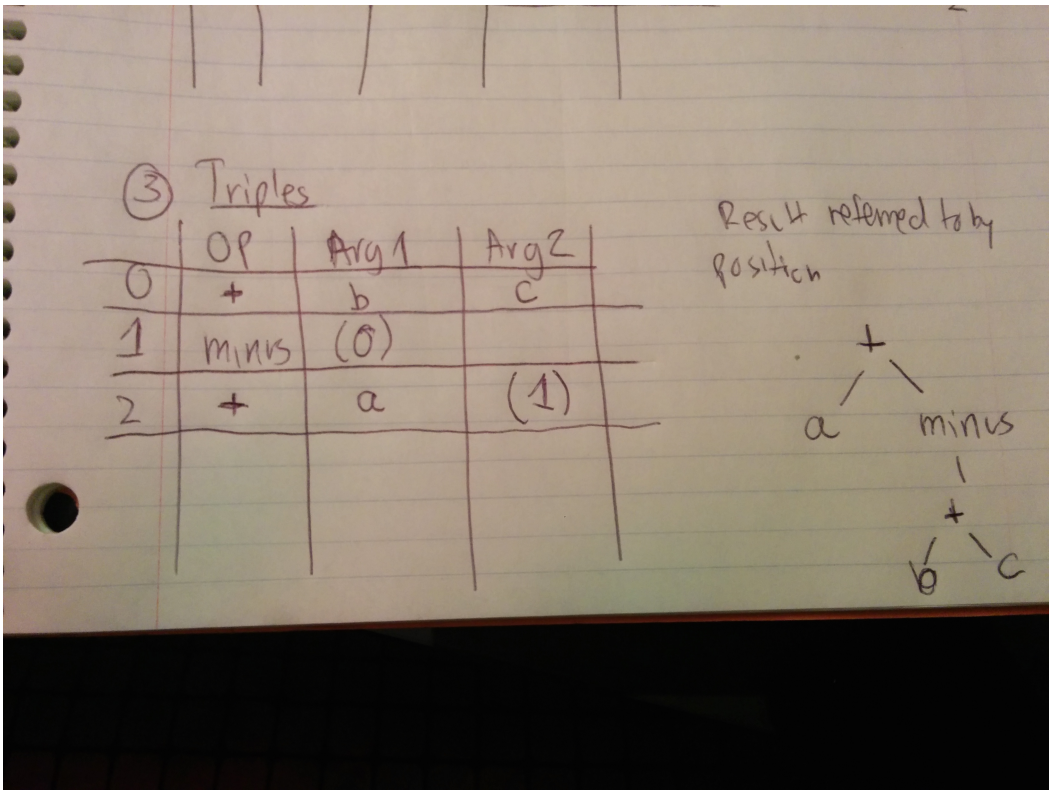
Please change the word "minus" to an actual minus (-) (My mistake I looked at this later)



1.2.2 Quadruples

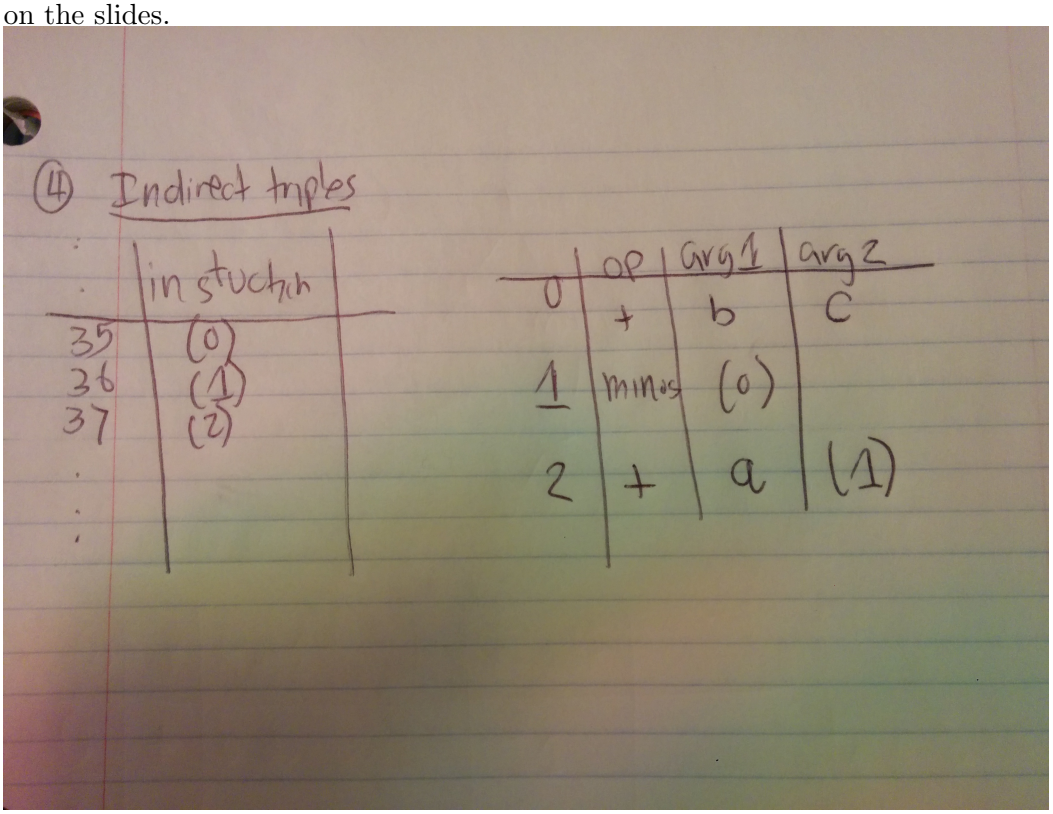


1.2.3 Triples


3 Triples
OP
Arg1
Arg2
0
+
b
c
1
minus
(0)
2
+
a
(1)
Result referred to by position
+
a
minus
+
b
c

1.2.4 Indirect Triples

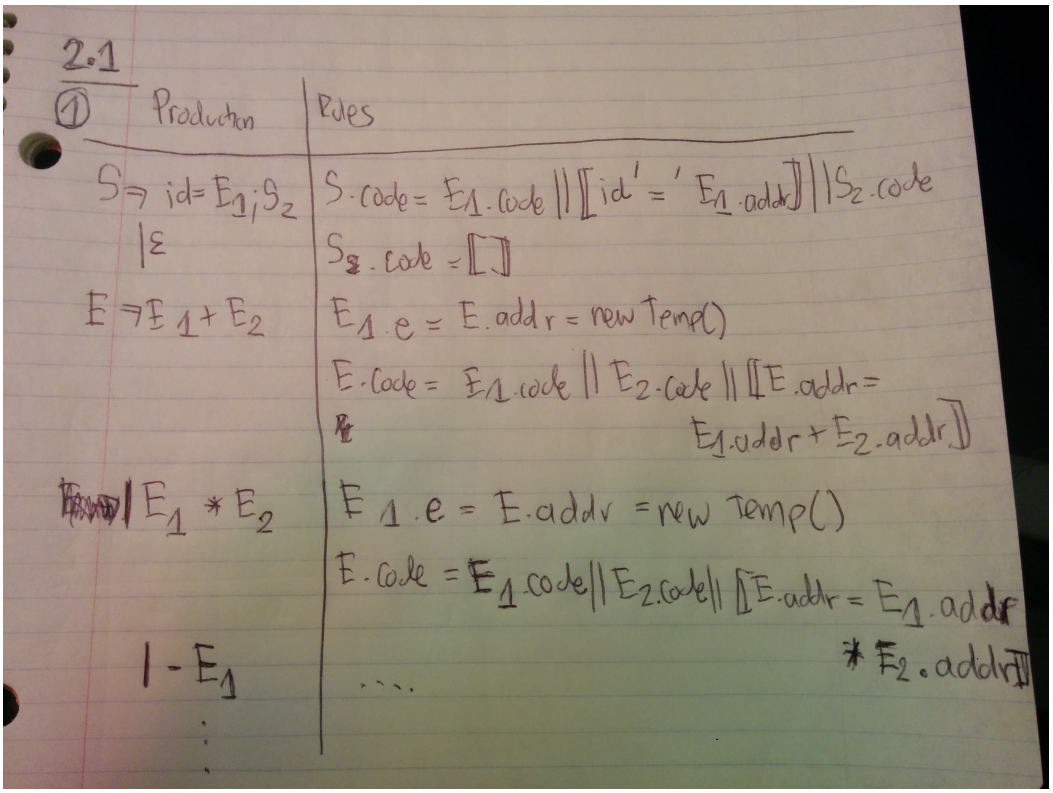
Please change the 35, 36, 37 to 0, 1, 2 instead as that's arbitrary and I misunderstood it on the slides.



2 Translation

2.1 Question 2.1

2.1.1 Add a translation rule for the following expression production: E → E1 * E2



2.2 Question 2.2

2.2.1 Explain how the code is expected to evaluate, assuming parameters are passed call-by-value.

Taken from the slides: If parameters are passed call-by-value then the contract is:

E_1.code, ..., E_n.code are evaluated before results placed into temporary

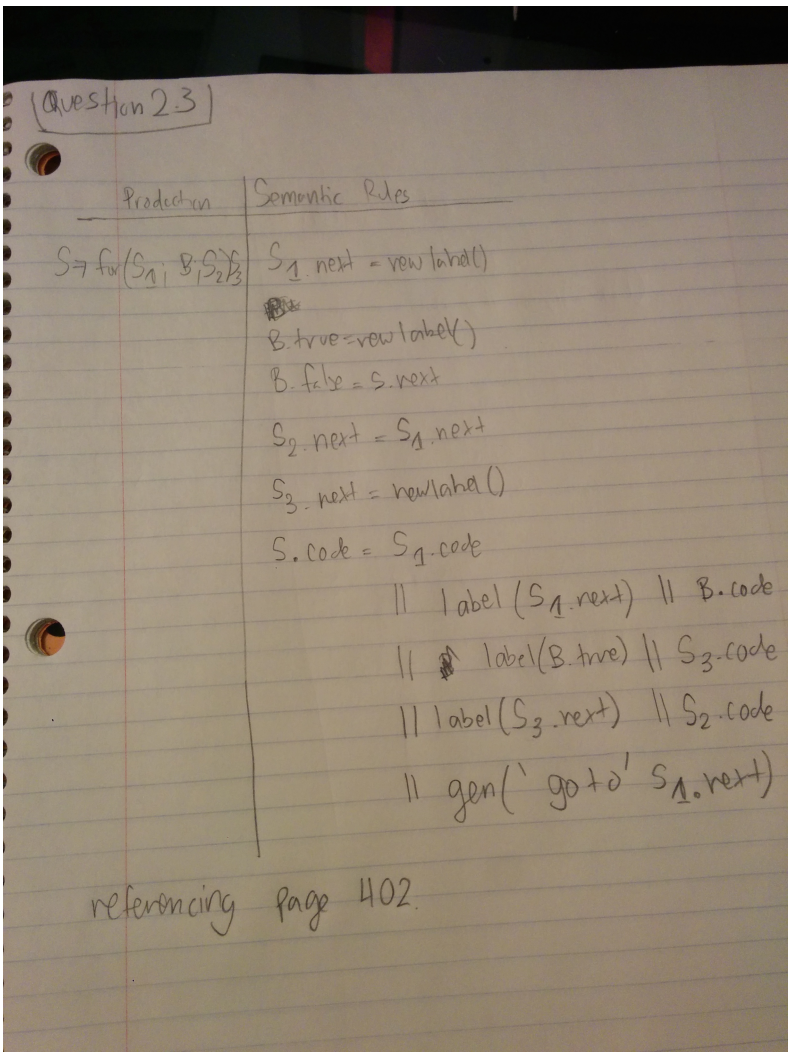
E_1.addr, ..., E_n.addr

The values are taken in as parameters, and the code is executed before you change or put the values into the add temporarily. For example, if the param you take in is 2, 2 and your code's execution is $param_1 + param_2$, then you would place the 4 into the E_1.addr after you've completed execution for E_1.code.

2.3 Question 2.3

2.3.1 S → for(S1; B; S2) S3

Extending the code from the book



In some cases, the instruc... label L. A jump to a jump to L from within ... produces three-address code

The syntax-directed definition in Fig. 6.36-6.37 produces three-address code for boolean expressions in the context of if-, if-else-, and while-statements.

PRODUCTION	SEMANTIC RULES
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$
$S \rightarrow$ **assign**	$S.code =$ **assign**$.code$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$
$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ S.next)$ $\mid\mid label(B.false) \mid\mid S_2.code$
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.co$

Figure 6.36: Syntax-directed definition for flow-of-control statements

We assume that *newlabel*() creates a new label each time it is called, and *label*(*L*) attaches label *L* to the ...