Department of Computer Science Columbia University

Solutions to Final – COMS W4115

Programming Languages and Translators

Monday, May 4, 2009 4:10-5:25pm, 309 Havemeyer

Closed book, no aids. Do questions 1–5. Each question is worth 20 points. Question 6 is optional, extra credit, 10 points.

1. Here is a fragment of C code:

```
struct student {
   int id;
   char student[30];
} student;
```

a) Explain the roles of the three uses of the identifier `student`.

```
struct student {          // here student is a structure tag
   int id;
   char student[30];      // here student is a structure member
} student;                // here student is a variable
```

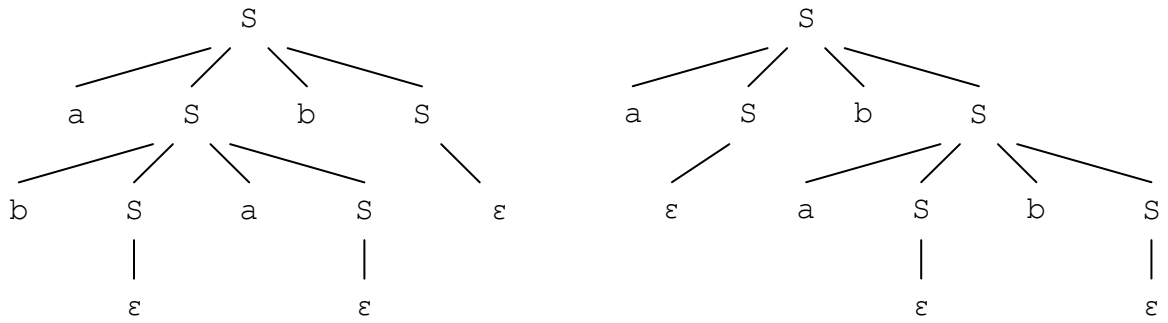
b) Are these three uses in the same scope? Explain.

The three uses are in the same scope but in different name spaces.

2. Consider the context-free grammar *G*: $S \rightarrow aSbS \mid bSaS \mid \varepsilon$

a) Describe *L(G)*. Show two parse trees for the sentence *abab* in *L(G)*.

L(*G*) is the set of all strings of *a*'s and *b*'s with the same number of *a*'s as *b*'s.



b) Construct the SLR(1) parsing action and goto tables for *L(G)*. Show the behavior of an LR(1) parser using these tables on the input *abab*.

The sets of items for the augmented grammar are:

```
I0: S'→ ·S          I1: S'→ S·          I2: S → a·SbS
    S → ·aSbS                               S → ·aSbS
    S → ·bSaS                               S → ·bSaS
    S → ·                                   S → ·

I3: S → b·SaS       I4: S → aS·bS       I5: S → bS·aS
    S → ·aSbS
    S → ·bSaS
    S → ·

I6: S → aSb·S       I7: S → bSa·S       I8: S → aSbS·
    S → ·aSbS           S → ·aSbS
    S → ·bSaS           S → ·bSaS
    S → ·               S → ·

I9: S → aSbS·
```

The parsing action and goto tables constructed from these sets of items are:

State	Action			Goto
	a	b	$	S
0	s2/r (3)	s3/r (3)	r(3)	1
1			accept	
2	s2/r (3)	s3/r (3)	r(3)	4
3	s2/r (3)	s3/r (3)	r(3)	5
4		s6		
5	s7			
6	s2/r (3)	s3/r (3)	r(3)	8
7	s2/r (3)	s3/r (3)	r(3)	9
8	r(1)	r(1)	r(1)	
9	r(2)	r(2)	r(2)	

Note the multiple shift-reduce conflicts that arise from the ambiguity in the grammar.

On the input abab, here is one sequence of moves an LR(1) parser using these tables can make:

Stack	Symbols	Input	Action
0		abab$	shift 2
02	a	bab$	shift 3
023	ab	ab$	reduce by $S \rightarrow \varepsilon$
0235	abS	ab$	shift 7
02357	abSa	b$	reduce by $S \rightarrow \varepsilon$
023579	abSaS	b$	reduce by $S \rightarrow bSaS$
024	aS	b$	shift 6
0246	aSb	$	reduce by $S \rightarrow \varepsilon$
02468	aSbS	$	reduce by $S \rightarrow aSbS$
01	S	$	accept

This sequence of moves corresponds to the first parse tree above. There is another sequence of moves corresponding to the second parse tree.

3. Consider the following partial syntax-directed definition for translating if-statements into three address code:

Production	Semantic Rules
$P \rightarrow S$	*S.next* = *newlabel*() *P.code* = *S.code* || *label*(*S.next*)
$S \rightarrow$ assign	*S.code* = assign.*code*
$S \rightarrow$ if (B) S_1	*B.true* = *newlabel*() *B.false* = *S.next* S_1.*next* = *S.next* *S.code* = *B.code* || *label*(*B.true*) || S_1.*code*
$B \rightarrow B_1$ && B_2	?
$B \rightarrow$ not B_1	?
$B \rightarrow$ true	?
$B \rightarrow$ false	?

Here the function *newlabel*() creates a new label each time it is called and *label*(*L*) attaches label *L* to the next three-address instruction to be generated.

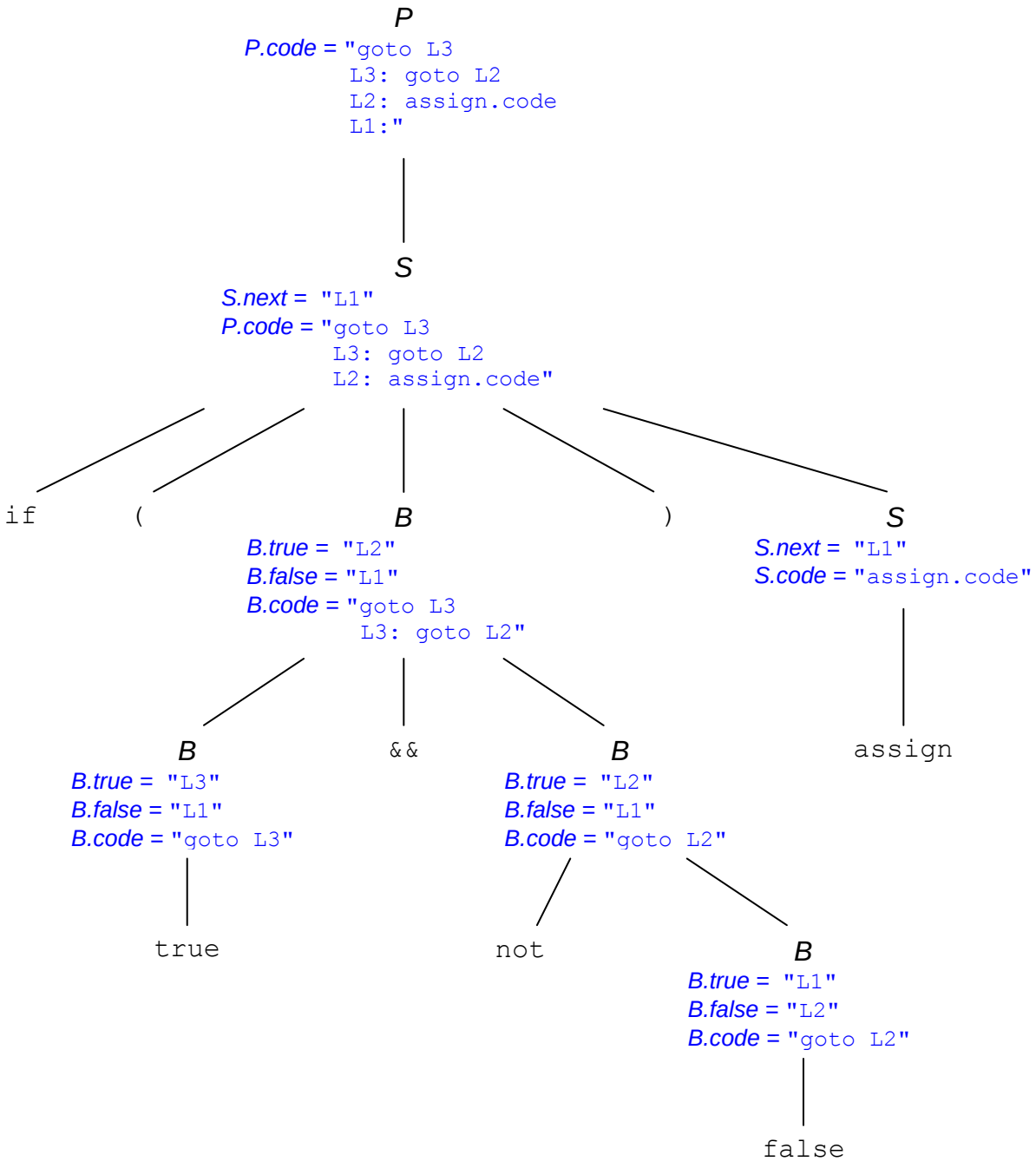
a) Fill in the semantic rules for the *B*-productions (they represent boolean expressions) using jumps to true and false labels.

See ALSU, p. 404.

b) Show how your SDD translates the if-statement

```
if (true && not false) assign
```

into three-address instructions by constructing an annotated parse tree for the if-statement.



4. Consider the arithmetic expression `a * b + c / (d – e)` and a register machine with instructions of the form

```
LD reg, src
ST dst, reg
OP reg1, reg2, reg3    // the registers need not be distinct
```

a) Draw a syntax tree for the expression and label the nodes with Ershov numbers.

The syntax tree of this expression is isomorphic to the one on p. 568, ALSU.

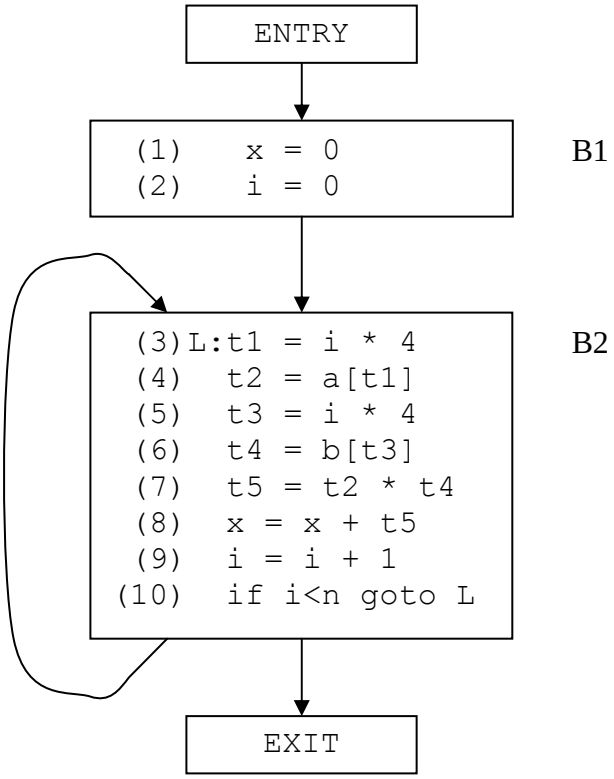
b) Generate machine code for the expression on a machine with two registers minimizing the number of spills.

This expression needs one spill on a two-register machine. See ALSU, p. 572, for an isomorphic code sequence.

5. Consider the following sequence of three-address code:

```
   x = 0
   i = 0
L: t1 = i * 4
   t2 = a[t1]
   t3 = i * 4
   t4 = b[t3]
   t5 = t2 * t4
   x = x + t5
   i = i + 1
   if i < n goto L
```

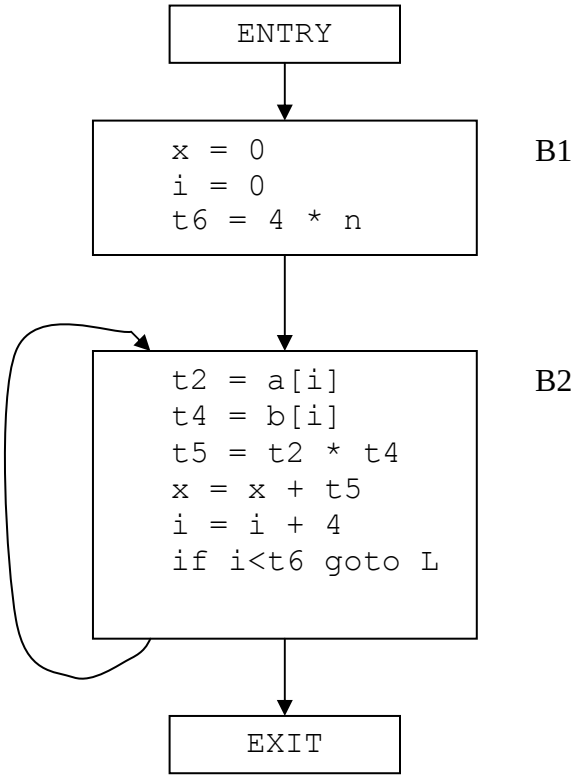
a) Draw a flow graph for this three-address code.



b) Optimize this code by eliminating common subexpressions, performing reduction in strength on induction variables, and eliminating all the induction variables that you can. State what transformations you are using at each optimization step.

First, we can eliminate the common subexpression `i*4` in lines (3) and (4) by using `t1` in place of `t3` in line (6) and eliminating line (5).

`Next, t1` and `i` are both induction variables in block B2. We can eliminate either one of these induction variables in the loop. We choose to eliminate `t1` by replacing line (9) by `i = i + 4`, adding the statement `t6 = 4 * n`, to the end of block B1, replacing the test `i<n` in line (10) by `i<t6`, and using `i` to index the arrays. The resulting optimized flow graph is:



6. Optional [extra credit, 10 points]. Consider again the context-free grammar *G* from question 2: $S \rightarrow aSbS \mid bSaS \mid \varepsilon$

a) How many parse trees are there for the sentence *ababab*?

There are 5 parse trees.

b) Write a recurrence relation for the number of parse trees for the sentence $(ab)^n$.

Let $T(n)$ be the number of parse trees for $(ab)^n$. By symmetry $T(n)$ is also the number of parse trees for $(ba)^n$. For the base cases, we have

$$T(0) = T(1) = 1$$

Since we can write $(ab)^n$ as $a(ba)^i\, b(ab)^{n-1-i}$, we have the recurrence

$$T(n) = \sum_{i=0}^{n-1} T(i)T(n-1-i)$$

c) What is the solution to your recurrence?

The solution is the Catalan numbers, $T(n) = \frac{1}{n+1}\binom{2n}{n}$.