CSE P 501 21au Exam 12/2/21

Name _____ UW ID # _____

There are 7 questions worth a total of 125 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

The exam is closed books, closed notes, closed electronics. However, you may have two 5x8 notecards for reference with any hand-written information you wish on both sides. Please turn off all cell phones, personal electronics, alarm watches, and pagers, and return your tray tables and seat backs to their full upright, locked positions. Sound or video recording and the taking of photographs is prohibited.

If you have a question during the exam, please raise your hand and someone will come to help you.

There are two extra blank pages at the end of the exam you can use if your answer(s) do not fit in the space provided. Please indicate on the original page(s) if your answer(s) is(are) continued on these last pages.

Please wait to turn the page until everyone is told to begin.

Score _____

1 _____ / 16

2 _____ / 30

3 _____ / 8

4 _____ / 20

5 _____ / 20

6 _____ / 16

7 _____ / 15

Question 1. (16 points) Regular expressions. For this question, write a regular expression and construct a DFA for the language of all strings formed from only the letters `a` and `b`, with the properties that (i) the string contains '`aba`' as a substring, and (ii) the string ends in the letter `b`.

Examples in this set: `abab`, `aaaaaabaaaaab`, `bbbbaaabaaaab`, `abababab`

Examples that are not in the set: `aba` (doesn't end in `b`), `bbab` (doesn't contain `aba`), `bbbbaaabbbaaaab` (doesn't contain `aba`), `baba` (doesn't end in `b`), `ababcab` (contains a character `c` that is not `a` or `b`)

Fine print: You must restrict yourself to the basic regular expression operations covered in class and on homework assignments: *rs* , *r*`|`*s* , *r*`*` , *r*`+` , *r*`?`, character classes like `[a-cxy]` and `[^aeiou]`, abbreviations *name=regexp*, and parenthesized regular expressions. No additional operations that might be found in the "regexp" packages in various Unix programs, scanner generators like JFlex, or programming language libraries are allowed.

(a) (6 points) Give a regular expression that generates all strings in this language.

(b) (10 points) Draw a DFA that accepts all strings in this language.

Question 2. (30 points) The you're-probably-not-surprised-to-see-it LR parsing question. Here is a grammar for a language of arithmetic expressions involving subtraction, unary minus, and the terminal symbol `x`. The extra E' ::= E `$` rule needed to handle end-of-file in an LR parser has been added for you. As usual, whitespace in the grammar is only for readability and is not part of the grammar or the strings generated by it.

0. E' ::= E `$` (`$` is end-of-file)
1. E ::= E `-` T
2. E ::= T
3. T ::= `-` T
4. T ::= `x`

(a) (16 points) Draw the LR(0) state machine for this grammar. When you finish, you should number the states in the final diagram in whatever order you wish so you can use the state numbers to answer later parts of this question.

Question 2. (cont.) Grammar repeated from previous page for reference:

0. *E'* ::= *E* $ ($ is end-of-file)
1. *E* ::= *E* − *T*
2. *E* ::= *T*
3. *T* ::= − *T*
4. *T* ::= x

(b) (6 points) Compute *nullable* and the FIRST and FOLLOW sets for the nonterminals *E* and *T* in the above grammar:

Symbol	nullable	FIRST	FOLLOW
E			
T			

(c) (4 points) Is this grammar LR(0)? Explain why or why not. If it is not, your answer should describe **all** of the reasons why this is so by identifying the relevant state number(s) in your diagram in part (a) and the specific issues in those state(s) (i.e., something like "state 47 has a shift-reduce conflict if the next input is x", but with, of course, state numbers and correct details from your diagram). If the grammar is LR(0) you should explain why. You do not need to write out the full LR(0) parse tables as part of your answer, just explain whether the state machine is LR(0) or not, and why.

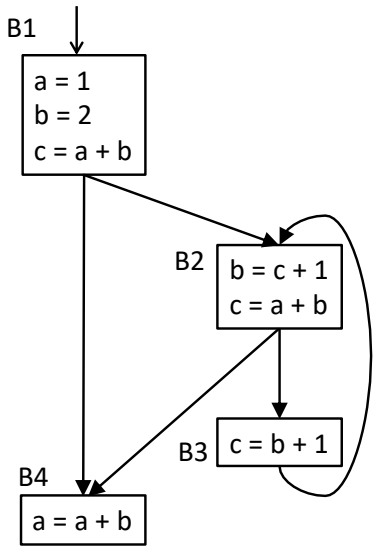
(d) (4 points) Is this grammar SLR? Explain why or why not. As with your answer to the previous part of the question, refer to states by number in the LR diagram in your answer to part (a) if needed and give specific explanations of why the grammar is SLR or why not.

Question 3. (8 points) LL parsing. Here is another look at the grammar from the previous question (the extra $E' ::= E \$$ production needed to handle end-of-file in the LR parser has been omitted since it is not needed here).

1. $E ::= E - T$
2. $E ::= T$
3. $T ::= - T$
4. $T ::= \texttt{x}$

Is this grammar, as written, suitable for constructing a top-down LL(1) predictive parser? If it is, your answer should give a technical explanation why it is. If not, your answer should give a technical explanation listing **all** of the problems with this particular grammar that prevent it from being suitable for a LL(1) predictive parser. You do not need to rewrite the grammar to fix the problems if there are any – just explain why it is or is not suitable for this use. (Hint: Explanations in terms of FIRST/FOLLOW and other properties needed by a LL(1) predictive parser may be helpful.)

The next two questions concern the following control flow graph:



Question 4. (20 points) Dataflow – available expressions. Recall from lecture that an expression *e* is *available* at a program point *p* if every path leading to point *p* contains a prior definition of expression *e* and *e* is not killed along a path from a prior definition by having one of its operands re-defined on that path. We would like to compute the set of available expressions at the beginning of each basic block in the flowgraph shown above. For each basic block *b* we define the following sets, as we've seen before:

AVAIL(*b*) = the set of expressions available on entry to block *b*
NKILL(*b*) = the set of expressions *not killed* in *b* (i.e., all expressions defined somewhere in any block in the flowgraph except for those killed in *b*)
DEF(*b*) = the set of all expressions defined in *b* and not subsequently killed in *b*

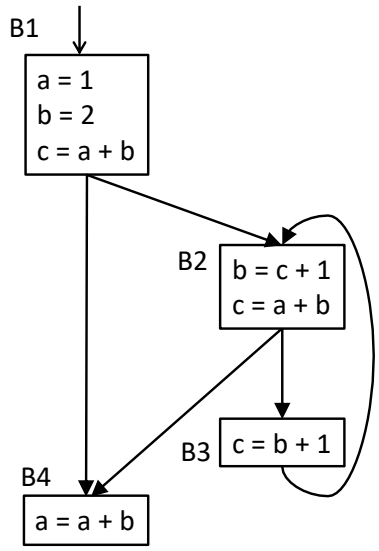
The dataflow equation relating these sets is

$$\text{AVAIL}(b) = \bigcap_{x \in \text{preds}(b)} (\text{DEF}(x) \cup (\text{AVAIL}(x) \cap \text{NKILL}(x)))$$

i.e., the expressions available on entry to block *b* are the intersection of the sets of expressions available on exit from all of its predecessor blocks *x* in the flow graph.

On the next page, calculate the DEF and NKILL sets for each block, then use that information to calculate the AVAIL sets for each block. You will only need to calculate the DEF and NKILL sets once for each block. You may need to re-calculate some of the AVAIL sets more than once as information about predecessor blocks change.

Hint: notice that there are only three expressions calculated in this flowgraph: a+b, b+1, and c+1. So all of the AVAIL, DEF, and NKILL sets for the different blocks will contain some, none, or all of those three expressions.



Question 4. (cont) Graph and definitions repeated from previous page:

AVAIL(b) = expressions available on entry to block b
NKILL(b) = expressions *not killed* in b
DEF(b) = expressions defined in b and not subsequently killed in b
AVAIL(b) = $\bigcap_{x \in \text{preds}(b)}$ (DEF(x) $\cup$ (AVAIL(x) $\cap$ NKILL(x)))

Hint: remember the only expressions in this flowgraph are a+b, b+1, c+1

(a) (8 points) For each of the blocks B1, B2, B3, and B4, write their DEF and NKILL sets in the table below.

Block	DEF	NKILL
B1		
B2		
B3		
B4		

(b) (12 points) Now, give the AVAIL sets showing in the table below the expressions available on entry to each block. If you need to update this information as you calculate the sets, be sure to cross out previous information so it is clear what your final answer is.

Block	AVAIL
B1	
B2	
B3	
B4	

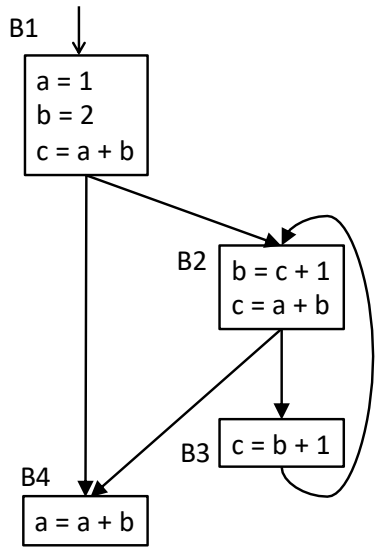
Question 5. (20 points) Dominators and SSA. Here are the basic definitions of dominators and related concepts we have seen previously in class:

- Every control flow graph has a unique **start node** s0.
- Node *x* **dominates** node *y* if every path from s0 to *y* must go through x.
  - A node *x* dominates itself.
- A node *x* **strictly dominates** node *y* if *x* dominates *y* and $x \neq y$.
- The **dominator set** of a node *y* is the set of all nodes *x* that dominate *y*.
- An **immediate dominator** of a node *y*, idom(*y*), has the following properties:
  - idom(*y*) strictly dominates *y* (i.e., dominates *y* but is different from *y*)
  - idom(*y*) does not dominate any other strict dominator of *y*

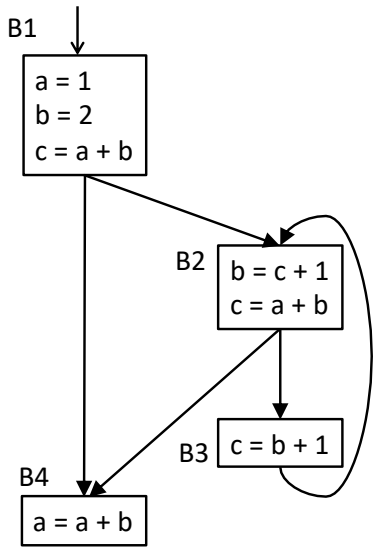
  A node might not have an immediate dominator. A node has at most one immediate dominator.
- The **dominator tree** of a control flow graph is a tree where there is an edge from every node *x* to its immediate dominator idom(*x*).
- The **dominance frontier** of a node *x* is the set of all nodes *w* such that
  - *x* dominates a predecessor of *w*, but
  - *x* does not strictly dominate *w*

(a) (10 points) Using the same control flow graph from the previous problem, complete the following table. List for each node: the nodes that dominate it, the node that is its immediate dominator (if any), and the nodes that are in its dominance frontier (if any):

Node	Nodes that dominate this node	IDOM	Dominance Frontier
B1			
B2			
B3			
B4			



Question 5. (cont.) (b) (10 points) Now redraw the flowgraph in SSA (static single-assignment) form. Your answer should include all of the Φ-functions required by the Dominance Frontier Criteria (or alternatively the path convergence criteria, which places the same set of Φ-functions), but no additional ones. It should include all Φ-functions that satisfy the Dominance Frontier Criteria even if some of those are assignments to variables that are never used (i.e., dead assignments). Answers that have a couple of extraneous Φ-functions will receive appropriate partial credit, but answers that, for example, use a maximal-SSA strategy of placing Φ-functions for all variables at the beginning of every block will not be looked on with favor.



Question 6. (16 points) x86-64 code. This question concerns these C functions and the translation of one of them to x86-64 code. Function `flip` returns a 0 or 1 randomly. Function `pick` returns either its first or its second argument depending on the result of a call to `flip`.

```
// return 0 or 1 randomly
int flip() {
  …
}

// flip a coin and return x if result is 0 or
// y if result is 1.
int pick(int x, int y) {
  if (flip() == 0){
    return x;
  } else {
    return y;
  }
}
```

For this problem, translate function `pick` to x86-64 assembly language code and write your answer on the next page. You must use the Linux/gcc x86-64 assembly language, and must follow the x86-64 function call, register, and stack frame conventions:

- Argument registers: %rdi, %rsi, %rdx, %rcx, %r8, %r9 in that order
- Called function must save and restore %rbx, %rbp, and %r12-%r15 if these are used in the function; all other registers may be changed by a function call
- Function result returned in %rax
- %rsp must be aligned on a 16-byte boundary when a call instruction is executed
- %rbp must be used as the base pointer (frame pointer) register for this exam, even though this is not strictly required by the x86-64 specification.
- Pointers, Booleans, and ints are 64 bits (8 bytes) each, as in MiniJava.
- Your x86-64 code must be a straightforward translation of the given code. You may not rewrite or rearrange the code even if it produces equivalent results. However, you can use any reasonable x86-64 code that follows the standard function call and register conventions – you do not need to mimic the code produced by a MiniJava compiler. In particular, these are C functions, not Java methods. Ordinary C calling conventions without object vtables should be used.
- Please include *brief* comments in your code to help us understand what the code is supposed to be doing (which will help us assign partial credit if it doesn't do exactly what you intended.)

(continued on next page)

Queston 6. (cont.) Write your x86-64 version of function `pick` below. Code repeated below for convenience. If you don't remember the exact assembly code for something, write down your best attempt and add a comment if needed to explain it. We will take that into account when grading.

Hint: be careful about what might happen to registers when function `flip` is called.

```
int pick(int x, int y) {
  if (flip() == 0){
    return x;
  } else {
    return y;
  }
}
```

Question 7. (15 points) AST/semantics. Many familiar languages including C, C++, and Java include a ternary (3-operand) `?:` operator. The meaning of *e1* `?` *e2* `:` *e3* is: evaluate *e1*. If *e1* is true, then the value of the expression is the value of *e2*, otherwise it is the value of *e3*. So, for example, the statement `max=x>y?x+1:y+1;` will store the larger value of `x+1` or `y+1` in `max`. (Note that `?:` has low precedence, so a fully parenthesized version of this statement would be `max=(x>y)?(x+1):(y+1);`) Suppose we add this operation to MiniJava and we then have the following statement in a program:

```
ans = a > b ? x : y;
```

(a) (6 points) Draw an abstract syntax tree (AST) for this statement at the bottom of this page. Use appropriate names for AST nodes and have an appropriate level of abstraction and structural detail similar to the AST nodes in the MiniJava project, but don't worry about matching the exact names of classes or nodes found in the MiniJava source code.

(b) (9 points) Annotate your AST by writing next to the appropriate nodes the checks or tests that should be done in the static semantics/type-checking phase of the compiler to ensure that this assignment statement does not contain errors. You do not need to specify or use an attribute grammar – just indicate the necessary tests. If a particular test applies to multiple nodes, you can write it once and indicate which nodes it applies to, as long as your meaning is clear and readable.

Extra space for answers, if needed. Please be sure to label which question(s) are answered here, and be sure to put a note on the question page so the grader will know to look here.

More extra space for answers, if needed. Please be sure to label which question(s) are answered here, and be sure to put a note on the question page so the grader will know to look here.