Sample

Final Exam
Open book/notes

1. [35 points]

(a) The BNF definition in Figure 1 below (start = A, terminals = {a,b,c}) is ambiguous. Describe the language L(A) that it defines, and show why the BNF is ambiguous.

(b) Using these same productions, provide an attribute grammar that disqualifies all but one derivation tree when there are several. Specifically add a Boolean attribute 'valid' (and others if you wish) so that for every string in L(A) there is exactly one derivation tree with attribute valid=true at the root node. Justify your attribute rules.

```
A ::= BC
B ::= aBb | Bb | □
C ::= bCc | bC | □
```

Figure 1.

2. [35 points]

Consider a new command to be added to the Wren language, an "increment all" command. It has the syntax

<command> **::= incAll** <variable list>

added to the Wren BNF (p. 11), introduces the new reserved word 'incAll', and adds the context sensitive constraint that all the variables in the list must be of type **integer**. It is permitted to repeat a variable in the list. The semantics of this new statement is that each variable in the list is to have its value incremented by 1, and variables that are repeated are incremented the number of times they appear in the list. Extend the denotational semantics of Wren (p. 291) to include this new command.

3. [30 points]

Describe the least fixed point and one other fixed point for each of the following functionals (in the domain **N** = {0, 1, 2, … }) — justify that your answers are fixed points.

(a) A(n) = **if** n=0 **then** 1 **else** A(n+1)

(b) B(n) = **if** n=0 **then** 5
else if n=1 **then** B(n+2) **else** B(n–2)

4. [35 points]
The Wren program fragment (with array) in Figure 2 below, sorts an array A[1..N] in increasing order. Write a loop invariant sufficient for proving the pre/post-conditions given, and informally relate it to the program's operation (i.e., why is it an invariant?). Note that a program proof is *not* required.

```
{N≥2}
I := 1;
{loop invariant?}
while I < N do
  if A[I] <= A[I+1]
    then I:= I+1
    else T:= A[I];
         A[I]:= A[I+1];
         A[I+1]:= T;
          if I > 1 then I:= I-1 end if
  end if
end while
{1≤k<N   A[k]≤A[k+1]}
```

Figure 2.

5. [30 points]
Are there any initial values for the program variables A, B, and C for which the Wren program fragment in Figure 3 below fails to terminate? Justify your answer, either with initial values which you show causes an infinite loop (according to denotational semantics), or with a termination measure which you show decreases with every loop iteration.

```
while A<B or B<C do
D:= C;  C:= A;  A:= B;  B:= D
end while
```

Figure 3.

6. [35 points]
Consider a new command to be added to the Wren language, the “undo” command. It has the syntax

<command> ::= **undo**

added to the Wren BNF (p. 11), and introduces the new reserved word 'undo'. The semantics of this command is that it resets the last variable to which an assignment has been made (if any) to its value prior to that assignment (multiple undos allowed)

For this problem, provide an algebraic definition of this Wren extension. Specifically,
(a) Add the appropriate operation signature for 'astUndo' for this command to the WrenASTs module (p. 489), and
(b) its semantics will be defined by adding the equation

execute(astUndo, sto, input, output) = <undo(sto), input, output>

to the WrenEvaluator module (p. 495), plus adding a description of this new store operation. Extend the WrenStore module provided in Figure 4 below by adding an 'undo' operation.

module WrenStores
imports WrenValues, WrenASTs
exports
sorts WrenStore
operations
emptySto: WrenStore
errorSto: WrenStore
updateSto(_, _, _): WrenStore, Ident, WrenValue → WrenStore
applySto(_, _): WrenStore, Ident → WrenValue
end exports
variables
id, id1, id2 : Ident
sto: Store
v: Value
equations (assuming that “errors propagate”)
[St1] applySto(emptySto, id) = errorVal
[St2] applySto(updateSto(sto, id,v), id) = v
[St3] applySto(updateSto(sto, id1, v), id2) = applySto(sto,id2) **if** id1 ≠ id2
end WrenStores

Figure 4.