CSC410

EXAM 1

Date: October 20, 2023
weight: 20% of course mark, length: 110 minutes

First Name: —————————————–

Last Name: —————————————–

Student Number: ————————————

Read Before You Start

• Write your name and student number, and please do it so that it matches your record on Quercus.

• This exam is designed so that the answers require no English words. You are welcome to write English sentences, but be warned that they will not be graded.

• The exam may look long, but the questions have very short answers. The generous 110-minute time is there to remove any stress from this process. It is not an indication that this exam requires 2 hours of work!

• This exam contains 4 problems (some with multiple subproblems) for undergraduate students plus an extra problem for graduate students.
Problem 1: Simple Iterative Correctness

(15 points) Consider the Dafny method below that sorts a Boolean array in linear time:

```dafny
method BooleanSort(a:array<bool>)
  modifies a
  ensures forall i, j :: 0 <= i <= j < a.Length ==> (a[i] ==> a[j])
{
  if a.Length <= 1 {return;}
  var b := 0;
  var e := a.Length - 1;
  while b < e
    invariant 0 <= b <= a.Length
    invariant 0 <= e < a.Length
    invariant ?
    {
      if !a[b] {
        b := b + 1;
      } else if a[e] {
        e := e - 1;
      } else {
        a[b],a[e] := a[e], a[b];
        b := b + 1;
        e := e - 1;
      }
    }
}
```

The postcondition specifies sortedness of a boolean array. For the loop, two trivial required range invariants are already given. Specify the remaining (non-trivial) invariant that would prove that the postcondition holds as specified.
Problem 2: Termination

Consider the Dafny method below.

```daffny
method termination(a: nat)
{
    var x, y := a, a+1;
    while x != 0 && y != 0
        invariant ?
        decreases ?
    {
        if (x == y + 1) {
            x := y - 1;
        } else if (y == x + 1) {
            y := x - 1;
        }
    }
}
```

(a) (10 points) Provide the correct `decreases` clause (termination measure) that proves the loop terminating in Dafny syntax.

(b) (10 points) A very simple invariant is needed to prove the measure from part (a) as terminating. What is that invariant?
Dataflow Analysis Problems

Problem 3: Possibly Uninitialized Variables

(20 points) Our target is the dataflow analysis that determines if a program variable is possibly unitialized at a given program point. Here is a precise definition:

A variable $v$ is possibly unitialized at program location $\ell$ if there exists a path from the initial location of the program to location $\ell$ along which $v$ is never initialized.

By giving precise answers to the following questions, give a definition to this dataflow analysis. We use $PU$ (for possibly uninitialize) to denote our sets. If you are using generic sets (corresponding to information about the program) in your definitions, clearly state on the side what these sets are.

- Define the set of dataflow facts.
  \[ D = \text{Vars} : \text{the set of program variables} \]

- Is the analysis forward or backward? (circle one)
  \[ \text{Forward} \quad \text{Backward} \]

- Define the semi-lattice binary operation:
  \[ \forall x, y \in D : \ x \cap y = x \cup y \]

- Define the transfer functions for a given location $\ell$:
  \[ PU_{exit}(\ell) = PU_{entry}(\ell) - \text{writes}(\ell) \]
  \[ \text{writes}(\ell) : \text{(the variables assigned in } \ell) \]
  \[ PU_{entry}(\ell) = \bigcup_{\ell' \rightarrow \ell} PU_{exit}(\ell') \]

- Specify the initialization information by specifying:
  - Initial location (circle one):
    \[ \text{Entry} \quad \text{Exit} \]
  - Initial value at the location you selected above: $Vars$
  - Initial value for all other sets: $\emptyset$
Problem 4: Lattices

(20 points) Recall that
\[ \forall x, y : x \sqsubseteq y \iff x \sqcap y = x \]
and we proved in class that \( \sqcap \) denotes the greatest lower bound of \( x \) and \( y \). You can use these as assumptions. Show that the following two definitions of monotonicity of a given function \( f \) are equivalent by proving the implications in both directions:
\[ \forall x, y : x \sqsubseteq y \implies f(x) \sqsubseteq f(y) \iff \forall x, y : f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y) \]

Therefore, you will prove the following two statements. Feel free to continue writing on the back of this page if you run out of space here. But, you should not need it. The proofs are very short.

(a) \((\forall x, y : x \sqsubseteq y \implies f(x) \sqsubseteq f(y)) \implies (\forall x, y : f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y))\)

By definition of \( \sqcap \): \( x \sqcap y \sqsubseteq x \implies (\text{by above assumption}) f(x \sqcap y) \sqsubseteq f(x) \quad (1) \)
By definition of \( \sqcap \): \( x \sqcap y \sqsubseteq y \implies (\text{by above assumption}) f(x \sqcap y) \sqsubseteq f(y) \quad (2) \)

(1) and (2) and \( \sqcap \) being the greatest lower bound \( \implies f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y) \)

(b) \((\forall x, y : f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y)) \implies (\forall x, y : x \sqsubseteq y \implies f(x) \sqsubseteq f(y))\)

\[ x \sqsubseteq y \implies x = x \sqcap y \]
\[ f(x) = f(x \sqcap y) \quad (\text{by the line above}) \]
\[ \sqsubseteq f(x) \sqcap f(y) \quad (\text{by the left hand side of (b)'s assumption}) \]
\[ \sqsubseteq f(y) \quad (\text{by the definition of greatest lower bound } \sqcap) \]
The Graduate Problem

(25 points) Consider the recursive function `maxSeq` defined on sequences of natural numbers below. Provide the missing precondition such that the postcondition of the function (as specified) is correct for all inputs satisfying your precondition together with the ones already given. We expect this precondition to be as liberal as possible. Full marks will go to a precondition that will make this code work for as many input sequences as possible.

```plaintext
function maxSeq(s: seq<nat>): nat
    decreases |s|
    requires |s| > 0
    requires ?
    ensures forall k :: 0 <= k < |s| ==> s[k] <= maxSeq(s)
{
    if |s| == 1 then s[0]
    else
        var i :| 0 <= i < |s| - 1;
        if s[i] < s[i + 1] then maxSeq(s[i+..])
        else maxSeq(s[..i+1])
}
```