CIS 301: Logical Foundations of Programming

Spring 2023

Exam 3 – 100 points

**This test is closed-notes and closed-computers.**

There are 8 questions worth 10-15 points each.

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Score: \_\_\_\_\_\_\_\_\_\_\_\_

1. (10 pts) Fill in the missing claims and justifications for the natural deduction proof below, where each **???** is written.

∀ x ∀ y (P(y) → Q(x)) ⊢ (∃ x P(x)) → (∀ y Q(y))

{

1. ∀ x ∀ y (P(y) → Q(x)) premise

2. {

3. ∃ x P(x) assume

4. {

5. a

6. {

7. **b P(b)** assume

8. ∀ y (P(y) → Q(a)) ∀e 1 a

9. P(b) → Q(a) **Ae 8 b**

10. Q(a) **->e 9 7**

}

11. Q(a) **Ee 3 6**

}

12. ∀ y Q(y) ∀i 4

}

13. (∃ x P(x)) → (∀ y Q(y)) **->I 2**

}

1. (15 pts) Use mathematical induction to prove that if n is a positive integer, then:

21 + 22 + ... + 2n = 2n+1 - 2

Let P(n) be the claim that 21 + 22 + ... + 2n = 2n+1 – 2. Let LHS(n) = 21 + 22 + ... + 2n and RHS(n) = 2n+1 – 2. To prove that P(n)holds for some positive integer n, we must show that LHS(n) = RHS(n).

Base case. We must prove that P(n) holds for the smallest positive integer n – that is, n = 1. We have that LHS(1) = 2^1 = 2, and that RHS(1) = 2^(1+1) – 2 =4 – 2 = 2. We have that LHS(1) = RHS(1), so the base case holds.

Inductive step. We assume the inductive hypothesis – that P(k) holds for some fixed positive integer k. We must show that P(k+1) holds – that is, that LHS(k+1) = RHS(k+1). Since P(k) holds, we can assume that LHS(k)= RHS(k). We have that:

LHS(k+1) = 21 + 22 + ... + 2k + 2k+1

= LHS(k) + 2k+1

= RHS(k) + 2k+1

= 2k+1 - 2 + 2k+1

= 2\*(2k+1) -2

= 2^((k+1)+1) – 2

= RHS(k+1)

Thus LHS(k+1) = RHS(k+1), and so the inductive step holds.

1. (13 pts) Use a direct proof to prove that if x is an odd integer, then 6x+4 is even.
2. (10 pts) Consider the following questions about recursive definitions.
   1. (5 pts) Consider the pattern 3,8,13,18,23,... Give a recursive definition of the function *f(n)*, where n ∈ ℕ, that defines the pattern.

f(1) = 3

f(n) = f(n-1) + 5, n > 1

* 1. (5 pts) Give a recursive definition of the function *SumOdds(n)*, where *SumOdds(n)* is the sum of the first n positive odd integers.

SumOdds(1) = 1

SumOdds(n) = SumOdds(n-1) + (2n-1), n > 1

1. (13 pts) Consider the following program. Add appropriate logic blocks so that the assertion at the end would hold in Logika.

import org.sireum.logika.\_  
  
var x: Z = readInt(“Enter a positive number: “)  
val input: Z = x

assume(x > 0)

x = x + 2

x = x \* 2

//when you finish, the assertion below should hold

assert(x == input\*2 + 4 & x > 5)

1. (14 pts) Consider the following partial Logika program (the missing parts before and after the given loop are not relevant). Complete logic blocks ***PART 1*** and ***PART 2*** to finish the correctness proof for the loop invariants. You ONLY need to prove that the loop invariants still hold at the end of each iteration.

...

//assume x is some previously defined variable

var ans: Z = 2\*x

var k: Z = 0

l"""{

**//assume loop invariants were already shown to hold before loop began**

}"""

while (k < x) {

l"""{

invariant 2\*x == ans + k

k >= 0

modifies k, ans

}"""

ans = ans - 1

l"""{

**[PART 1]**

}"""

k = k + 1

l"""{

**[PART 2]**

}"""

}

...

1. (13 pts) Consider the following function, which computes and returns the absolute difference between two numbers. (Here, *absDiff(7,3)* returns 4 and *absDiff(3,7)* also returns 4, as |7 – 3| = |3 – 7| = 4, where | | denotes absolute value.) **Add appropriate logic blocks** so that the function would be verified in Logika. To save space, you do not need to put each l“””{ and }”””.

import org.sireum.logika.\_

def absDiff(a: Z, b: Z): Z = {

l"""{

ensures result >= 0

result == a-b V result == b-a

}"""

var diff: Z = 0

var eval: Z = a-b

if (eval >= 0) {

diff = eval

l”””{

1. eval >= 0 premise
2. diff == eval premise
3. eval == a-b premise
4. diff >= 0 algebra 1 2
5. diff == a-b algebra 2 3

}”””

} else {

diff = -1\*eval

l”””{

1. !(eval >= 0) premise
2. diff == -1\*eval premise
3. eval == a-b premise
4. diff >= 0 algebra 1 2
5. diff == b-a algebra 2 3

}”””

}

l”””{

1. diff >= 0 premise
2. diff == a-b V diff == b-a premise

}”””

return diff

}

1. (12 pts) Consider the following shell of a Logika program. Here, much of the code and reasoning is missing. Assume that *sampleFn* correctly verifies its function contract. **Add appropriate logic blocks under “*Calling code*”** so that the assert at the end would hold in Logika. (Again, assuming that the function was correctly verified.)

import org.sireum.logika.\_  
  
def sampleFn(x:Z, y:Z): Z = {

l"""{

requires x+y > 0

ensures result == 2\*x - y

}"""

...(code/verification for function)

}

//////////// Calling code ///////////

var a: Z = 4

var b: Z = 2

var back: Z = sampleFn(a-1, b)

assert(back == 4)