School of Software Engineering, USTC (Suzhou)

Final Term Exam Paper for Academic Year 2023-2024-2

Open or Close: Open

Course: Formal Methods _____ Time: July. 9th, 2024 _____

Student Name: _____ Student No._____

Class: _____ Score:_____

I: Logic

1. [5 points] Given propositions F_1

$$\neg(p \to q)$$

and F_2

$$p \wedge \neg q$$

Are these two propositions semantically equivalent? Prove your conclusion by using a truth table.

2. [10 points] Given the following inductively defined rules

$$\frac{}{even\ 0} \quad \text{(EVEN-0)}$$

$$\frac{even\ n}{even\ n+2} \quad \text{(EVEN-SS)}$$

Alan wants to use these rules to prove that **100** is an even number. This is the code he wrote to prove it using Z3:

```
1 isort = IntSort()
2 bsort = BoolSort()
```

```
even = Function("even", isort, bsort)
n = Int("n")
even_1 = even(1)
even_ss = ForAll(n, Implies(even(n), even(n+2)))

solver = Solver()
solver.add(even_1)
solver.add(even_ss)
solver.add(Not(Even(100)))

if solver.check() == unsat:
  print("The number 100 is even.")
else:
  print("The number 100 is not even.")
```

Is Alan's proof correct? If so, please give your reason. If not, please give the correct proof.

II: SAT

Here are the rules for eliminating implications:

$$\begin{aligned}
\mathcal{E}(\top) &= \top \\
\mathcal{E}(\bot) &= \bot \\
\mathcal{E}(p) &= p \\
\mathcal{E}(P \wedge Q) &= \mathcal{E}(P) \wedge \mathcal{E}(Q) \\
\mathcal{E}(P \vee Q) &= \mathcal{E}(P) \vee \mathcal{E}(Q) \\
\mathcal{E}(P \to Q) &= \mathcal{E}(\neg P) \vee \mathcal{E}(Q) \\
\mathcal{E}(\neg P) &= \neg \mathcal{E}(P)
\end{aligned}$$

Rules for conversion into NNF (Negation Normal Form):

$$\begin{aligned}
\mathcal{N}(\top) &= \top \\
\mathcal{N}(\bot) &= \bot \\
\mathcal{N}(p) &= p \\
\mathcal{N}(\neg P) &= \neg\mathcal{N}(P) \\
\mathcal{N}(\neg\neg P) &= \mathcal{N}(P) \\
\mathcal{N}(P \wedge Q) &= \mathcal{N}(P) \wedge \mathcal{N}(Q) \\
\mathcal{N}(P \vee Q) &= \mathcal{N}(P) \vee \mathcal{N}(Q) \\
\mathcal{N}(\neg(P \wedge Q)) &= \mathcal{N}(\neg P) \vee \mathcal{N}(\neg Q) \\
\mathcal{N}(\neg(P \vee Q)) &= \mathcal{N}(\neg P) \wedge \mathcal{N}(\neg Q)
\end{aligned}$$

Rules for converting into CNF (Conjunction Normal Form):

$$\begin{aligned}
\mathcal{C}(\top) &= \top \\
\mathcal{C}(\bot) &= \bot \\
\mathcal{C}(p) &= p \\
\mathcal{C}(\neg p) &= \neg\mathcal{C}(p) \\
\mathcal{C}(P \wedge Q) &= \mathcal{C}(P) \wedge \mathcal{C}(Q) \\
\mathcal{C}(P \vee Q) &= \mathcal{D}(\mathcal{C}(P), \mathcal{C}(Q)) \\
\mathcal{D}(P_1 \wedge P_2, Q) &= \mathcal{D}(P_1, Q) \wedge \mathcal{D}(P_2, Q) \\
\mathcal{D}(P, Q_1 \wedge Q_2) &= \mathcal{D}(P, Q_1) \wedge \mathcal{D}(P, Q_2) \\
\mathcal{D}(P, Q) &= P \vee Q
\end{aligned}$$

3. [9 points] Suppose we have proposition F as following:

$$\neg(q \to r) \wedge p \wedge (q \vee \neg(p \wedge r))$$

Questions:

1. Please eliminate implications in the proposition F, by using the above rules.

2. Please convert your answer in question 1 to NNF, by using the above rules.

3. Please convert your answer in question 2 to CNF, by using the above rules.

4. [12 points] Given the following curriculum of the school of software engineering in the spring semester of 2024:

No.	Course Name	Category	Prerequisite
1	Formal Methods	Math	
2	Combinatorics	Math	
3	Discrete Mathematics	Math	
4	Software Engineering	CS	No.1
5	Algorithm Theory	CS	No.3
6	Compiler Engineering	CS	No.2
7	Computer Organization Principle	CS	No.2
8	Advanced Software Engineering	SE	No.4, No.7
9	Analysis of Linux System	SE	No.6, No.7
10	Advanced Algorithm	SE	No.5

Questions:

1. Please write down propositions in propositional logic to represent relationships of course prerequisite. For example, the prerequisite course for "Algorithm Theory" (No. 5) is "Discrete Mathematics" (No. 3).

2. Suppose that each student must take just two Math courses (i.e., No. 1 to No. 3). Please write down propositions to represent this constraint.

3. Suppose that a student Alan needs to take one course in Math, one from SE, and two from CS. Additionally, Alan needs to take the Advanced Algorithm course. Please write down the constraints using propositional logic.

III: SMT

5. [10 points] One important application of the EUF theory is proving program equivalence. Alan wants to use EUF theory to prove that the following two program segments are equivalent:

```
1 int power3(int in){
2   int i, out_a;
3   out_a = in;
4   for(i = 0; i < 2; i++)
5     out_a = out_a * in;
6   return out_a;
7 }
```

```
1 int power3_new(int in){
2   int out_b;
3   out_b = (in*in)*in;
4   return out_b;
5 }
```

The Z3 code that Alan used for the proof is as follows:

```
1  from z3 import *
2
3  arg_in = Const('arg_in', IntSort())
4  out_a_0 = Const('out_a_0', IntSort())
5  out_a_1 = Const('out_a_1', IntSort())
6  out_a_2 = Const('out_a_2', IntSort())
7  f_mul = Function('f_mul', IntSort(), IntSort(), IntSort())
8
9  P1 = And(arg_in == out_a_0, out_a_1 == f_mul(arg_in, out_a_0),
     out_a_2 == f_mul(arg_in, out_a_1))
10 P2 = (out_b == f_mul(arg_in, f_mul(arg_in, arg_in)))
11 solve(Implies(And(P1, P2), out_a_2 == out_b))
```

Alan ran the above code and obtained the following results:

```
1 [out_b = 1,
2  arg_in = 2,
```

```
3   out_a_0 = 3,
4   out_a_1 = 4,
5   out_a_2 = 0,
6   f_mul = [(2, 4) -> 6, (2, 2) -> 7, (2, 7) -> 8, else -> 5]]
```

Questions:

1. Please write down the constraints that Alan's proof code is trying to prove.
2. Please write down the meaning of the output.
3. Did Alan's proof code successfully prove the equivalence of the two program segments? If so, please give your reasons. If not, please correct potential errors in the proofs.

6. [10 points] Magic Squares have fascinated mathematicians for centuries. The pattern was a 3x3 grid of nine squares containing one of the numbers between 1 and 9 (each number can appear just once). The constraint is that the sub-totals of the 3 numbers in each row, each column, and both diagonals of the square are always equal. For example, the following is a sample solution for the 3×3 square:

2	7	6
9	5	1
4	3	8

that is, we have:

$$2+7+6=15$$
$$9+5+1=15$$
$$4+3+8=15$$
$$2+9+4=15$$
$$7+5+3=15$$
$$6+1+8=15$$
$$2+5+8=15$$
$$4+5+6=15$$

Questions:

1. please write down constraints using the linear arithmetic theory to find all the solutions for Magic Squares of 3×3.
2. please write down constraints using the linear arithmetic theory to find all the solutions for Magic Squares of $n \times n$, for any $n \in \mathbb{N}$.

7. [12 points] Consider the following formula F which mixes linear arithmetic (over domain $\mathbb{Z}$) and uninterpreted functions (function f):

$$(1 \leq x \leq 2) \wedge (f(1) = a) \wedge (f(2) = f(1) + 3) \wedge (a = b + 2).$$

Questions:

1. Is the formula F convex or non-convex? Please write down your reasons.
2. Use the Nelson-Oppen cooperation procedure to decide the satifiability of the formula F, please write down the intermediate steps and final result.

8. [12 points] Integer overflows are a notorious source of bugs and are very difficult to debug. Consider the following C code:

```
1  int myfunction(int *array, int len){
2    int *arr, i;
3
4    arr = malloc(len * sizeof(int)); /* [1] */
5    if(arr == NULL){
6      return -1;
7    }
8
9    for(i = 0; i < len; i++){ /* [2] */
10     arr[i] = array[i];
11   }
12
13    return arr;
14 }
```

One student Alan believes there is an integer overflow bug at `[1]`, causing buffer overflow at `[2]`. So Alan wrote the following Z3 program to try to figure out the input that triggers the overflow:

```
1 from z3 import *
2
3 x, y = BitVecs('x y', 32)
4 solver = Solver()
5 solver.add(x >= 1, y == 4, x * y < 0)
6 res = solver.check()
7 if res == sat:
8   print('found integer overflow: ', solver.model())
```

The output of this program is as follows:

```
1 found integer overflow: [x = 536870912, y = 4]
```

Question: Does this output trigger an integer overflow? If so, please explain briefly how the overflow is triggered. If not, please modify the Z3 code to fix potential bugs.

IV: Symbolic Execution

9. [10 points] During symbolic execution, we can use the following memory model to store arguments, symbolic values, and path conditions:

```
1  @dataclass
2  class Memory:
3      args: List[str]
4      symbolic_memory: Dict[str, Expr]
5      path_condition: List[Expr]
6
7  @dataclass
8  class Expr:
9      pass
10
11 @dataclass
12 class ExprNum(Expr):
13     value: int| float
14
15 @dataclass
16 class ExprVar(Expr):
17     var: str
18
19 @dataclass
20 class ExprBop(Expr):
21     left: Expr
22     right: Expr
23     bop: Bop
```

The `symbolic_memory` is a dictionary that stores variable name as key and expression as value. We need `symbolic_expr()` function to replace the variables in expression according to the `symbolic_memory` when updating the `symbolic_memory` or appending condition to the `path_condition`. This ensures expressions in `symbolic_memory` and `path_condition` contain only argument variables and `ExpNum`.

The following is Alan's implementation of the `symbolic_expr()` function:

```
1  def symbolic_expr(memory: Memory, expr: Expr):
2      match(expr):
3          case ExprNum():
4              return expr
5          case ExprVar(var):
6              return memory.symbolic_memory[var]
7          case ExprBop(left, right, bop):
8              left = symbolic_expr(memory, left)
9              right = symbolic_expr(memory, right)
10             return ExprBop(left, right, bop)
```

Questions:

1. Is the Alan's `symbolic_exp()` function implementation correct?
2. If your answer is yes, please explain your reason. If not, please give your ideas to correct the implementation (you do not need to write code, just explain your ideas).

V: Verification

10. [10 points] Please prove the validity of the Hoare triple using the Hoare logic inference rules.

$$\vdash \{x \geq 10\}\textbf{while } \texttt{x<=5} \textbf{ do } \texttt{x=x+1}\{x \geq 7\}.$$

End of Test.