School of Software Engineering, USTC (Suzhou)
Exam Paper for Academic Year 2019-2020-2
Open or Close: Close

Course: Formal Methods	Time: July 14, 2020
Student Name: ________	Student No.________
Class: ________	Score:________

I: Propositional Logic

Given the following inference rules for propositional logic:

$$\frac{}{\Gamma, P \vdash P}\ (Var) \qquad \frac{}{\Gamma \vdash T}\ (T1)$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash P}\ (\bot E) \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q}\ (\wedge I)$$

$$\frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash P}\ (\wedge E1) \qquad \frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash Q}\ (\wedge E2)$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash P \vee Q}\ (\vee I1) \qquad \frac{\Gamma \vdash Q}{\Gamma \vdash P \vee Q}\ (\vee I2)$$

$$\frac{\Gamma, P \vdash Q}{\Gamma \vdash P \to Q}\ (\to I) \qquad \frac{\Gamma \vdash P \to Q \quad \Gamma \vdash P}{\Gamma \vdash Q}\ (\to E)$$

$$\frac{\Gamma, P \vdash \bot}{\Gamma \vdash \neg P}\ (\neg I) \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash \neg P}{\Gamma \vdash \bot}\ (\to E)$$

$$\frac{\Gamma \vdash P \vee Q \quad \Gamma, P \vdash R \quad \Gamma, Q \vdash R}{\Gamma \vdash R}\ (\vee E) \qquad \frac{\Gamma \vdash \neg\neg P}{\Gamma \vdash P}\ (\neg\neg E)$$

1. (5 points) Draw the proof tree for proposition:

$$(P \vee Q) \to (Q \vee P)$$

using the above inference rules.

II: Constructive Logic

2. (5 points) Given the following exclusive middle law (EM):

$$\vdash P \ \vee \neg P$$

Does this rule hold in constructive logic? Explain your conclusion. (Only write down your idea, no need to write down strict proof.)

III: SAT

3. (5 points) Converting the following proposition into CNF:

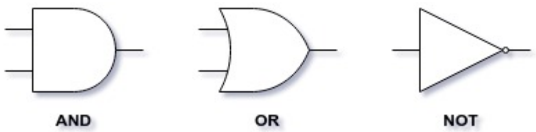
$$p_1 \wedge \neg(p_2 \vee p_3) \vee \neg p_4$$

4. (5 points) Try to check the satisfiability of

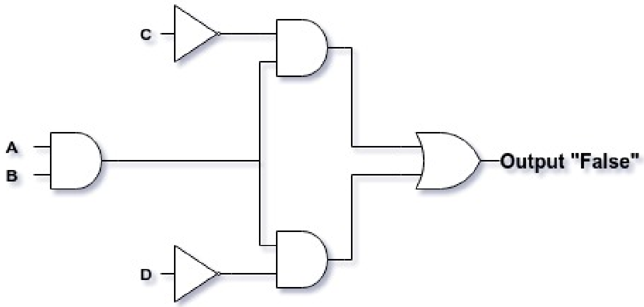
$$(\neg p_1 \vee \neg p_3) \wedge (p_2 \vee p_4) \wedge (p_2 \vee \neg p_3)$$

via DPLL. (Draw up the detail DPLL steps.) To speed up the DPLL algorithm, we mentioned a concurrent version of DPLL, briefly explain why DPLL can utilize concurrency to speed up its calculation.

5. (5 points) Given the following three basic logic gates, try to write down the logic proposition for circuit



layout shown below.



IV: Predicate Logic

6. (5 points) While learning predicate logic, Bob found that Z3 supports a theory T which is undecidable. Bob also found that when he send some T formulae to Z3, Z3 will reply various result: `SAT`, `UNSAT` or `UNKNOW`. Try to explain why $Z3$ may generate these results.

V: Theory for EUF

7. (10 points) One important application of the EUF theory is proving program equivalence. In the following, we present two implementations of the same algorithm, one is:

```
int power3(int in){
  int i, out_a;
  out_a = in;
  for(i = 0; i < 2; i++)
    out_a = out_a * in;
  return out_a;
}
```

and the other one is:

```
int power3_new(int in){
  int out_b;
  out_b = (in*in)*in;
  return out_b;
}
```

With EUF, we can prove these two algorithms are equivalent by proving this proposition is valid:

$$P_1 \wedge P_2 \rightarrow \texttt{out\_a} == \texttt{out\_b}$$

The proving code with Z3 looks like:

```
S = DeclareSort('S')
inp, out_a_0, out_a_1, out_a_2, out_b = Consts('inp out_a_0 out_a_1 out_a_2 out_b', S)
f = Function('f', S, S, S)

P1 = And(out_a_0 == inp, out_a_1 == f(out_a_0, inp), out_a_2 == f(out_a_1, inp))
P2 = And(out_b == f(f(inp, inp), inp))
solve(Implies(And(P1, P2), out_b == out_a_2))
```

and the ouput is:

```
[out_a_1 = S!val!2,
 out_a_2 = S!val!1,
 out_b = S!val!0,
 inp = S!val!3,
 out_a_0 = S!val!5,
 f = [(S!val!5, S!val!3) -> S!val!6,
      (S!val!3, S!val!3) -> S!val!7,
      (S!val!7, S!val!3) -> S!val!8,
      else -> S!val!4]]
```

Questions:

1. Why the `out_b` and `out_a_2` are not equal in the ouput?
2. Does the code prove the equivalence of two programs? If so, give the reason. If not, give the reason and the correct solution (just write down your idea, no need to write code).

VI: Linear Arithmetic

8. (4 points) The Fourier-Motzkin variable elimination algorithm is a popular algorithm to solve equalities and inequalities. Here are some linear inequalities already normalized, where $P_1(x), \ldots, P_s(x)$, $Q_1(x), \ldots, Q_t(x)$ and $R_1(x), \ldots, R_r(x)$ don't contain the variable x_1:

$$\begin{cases} x_1 + P_1(x) \geq 0 \\ \ldots \\ x_1 + P_s(x) \geq 0 \\ -x_1 + Q_1(x) \geq 0 \\ \ldots \\ -x_1 + Q_t(x) \geq 0 \\ R_1(x) \geq 0 \\ \ldots \\ R_r(x) \geq 0 \end{cases}$$

Question: how many inequalities are there after eliminating the variable x_1 by using Fourier-Motzkin variable elimination algorithm?

9. (8 points) Given the following linear inequalities with two variables x, y and three constraints:

$$\begin{cases} 2x - y \geq 0 \\ x + 2y \geq 1 \\ x + y \geq 2 \end{cases}$$

Question: calculate the solution of the inequalities above by using Simplex algorithm, please write down the table pivot procedures and result.

VII: Theories for Data Structures

10. (8 points) Logic with pointers can be converted into EUF problem, by eliminating pointers through encoding their semantics using the store function S and heap function H. To simplify things, we assume that the heap only contains values of integer type, and the address is also of integer type:

$$S : \texttt{int} \to \texttt{int}$$

$$H : \texttt{int} \to \texttt{int}$$

The rules to eliminate a pointer T are:

$$\begin{aligned} [\![x]\!] &= H(S(x)) \\ [\![T + E]\!] &= [\![T]\!] + [\![E]\!] \\ [\![\&x]\!] &= S(x) \\ [\![\& * T]\!] &= [\![T]\!] \\ [\![*T]\!] &= H([\![T]\!]) \\ [\![\texttt{NULL}]\!] &= 0 \end{aligned}$$

The rules to eliminate an expression E are:

$$
\begin{aligned}
[\![n]\!] &= n \\
[\![x]\!] &= H(S(x)) \\
[\![E + E]\!] &= [\![E]\!] + [\![E]\!] \\
[\![E - E]\!] &= [\![E]\!] - [\![E]\!] \\
[\![*T]\!] &= H([\![T]\!])
\end{aligned}
$$

The rules to eliminate a relation R are:

$$
\begin{aligned}
[\![E = E]\!] &= [\![E]\!] = [\![E]\!] \\
[\![E \neq E]\!] &= [\![E]\!] \neq [\![E]\!] \\
[\![E < E]\!] &= [\![E]\!] < [\![E]\!] \\
[\![T = T]\!] &= [\![T]\!] = [\![T]\!] \\
[\![T \neq T]\!] &= [\![T]\!] \neq [\![T]\!] \\
[\![T < T]\!] &= [\![T]\!] < [\![T]\!]
\end{aligned}
$$

The rules to eliminate a proposition P are:

$$
\begin{aligned}
[\![P \wedge Q]\!] &= [\![P]\!] \wedge [\![Q]\!] \\
[\![\neg R]\!] &= \neg [\![R]\!]
\end{aligned}
$$

Questions:

1. Translate the following proposition with pointers to EUF, by using the above rules:

$$*p = 1 \wedge **q = 1 \rightarrow p \neq q$$

2. Is this proposition valid in the memory model we used in the assignment? If yes, explain your conclusion. If not, give the reason and also your idea how to validate this proposition.

VIII: Theory Combination

11. (8 points) Consider the following formulae which mixes linear arithmetic (over domain $\mathbb{R}$) and uninterpreted functions (function f):

$$(f(x_1, 0) \geq x_3) \wedge (f(x_2, 0) \leq x_3) \wedge (x_1 \geq x_2) \wedge (x_2 \geq x_1) \wedge (x_3 - f(x_1, 0) \geq 1)$$

Question: simplify this formula using Nelson-Oppen method, write down the steps and result.

IX: Symbolic Execution

12. (10 points) We can use the following memory model to store arguments, symbolic values and path conditions, during symbolic execution:

```
@dataclass
class Memory:
    args: List[str]
    symbolic_memory: Dict[str, Exp]
    path_condition: List[Exp]
```

The `symbolic_memory` is a dictionary stores variable name as key and expression as value. We need `symbolic_exp()` 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 process ensures expressions in `symbolic_memory` and `path_condition` contain only argument variables and `ExpNum`.

The following is Bob's implementation of the `symbolic_exp()` function:

```
def symbolic_exp(memory: Memory, exp: Exp):
    if isinstance(exp, ExpNum):
        return exp
    if isinstance(exp, ExpVar):
        symbolic_value = memory.symbolic_memory[exp.var]
        return symbolic_exp(memory, symbolic_value)
    if isinstance(exp, ExpBop):
        left = symbolic_exp(memory, exp.left)
        right = symbolic_exp(memory, exp.right)
        return ExpBop(left, right, exp.bop)
```

Questions:

1. Is the `symbolic_exp()` function implementation correct?
2. If your answer is yes, explain the reason. If not, give your reason and your idea to correct it (no need to write code).

13. (6 points) For the following function in `C--`:

```
f_loop(m,n){
  while(m < n){
    if(m > 0){
      m = m * 2;
    }
    else{
      m = m + 1;
    }
  }
  return m;
}
```

Question: write down the path conditions after concolic execution of function `f_loop()` with input: `m=0, n=4`.

X: Hoare Logic

14. (10 points) The backward verification condition generation algorithmic rules for `StmWhile` and `StmAssign` are:

$$
\begin{aligned}
VC(while_I(e; s), P) &= I \wedge (\forall \vec{x}.I \to (e \to VC(s, I) \wedge (\neg e \to P))) \\
VC(x = e, P) &= P[x \mapsto e]
\end{aligned}
$$

The rule for `StmWhile` generates a verification condition containing universal quantification proposition, which are modified variables `vars_set` inside `StmWhile` body:

```
class ExpUni(Exp):
    # forall(vars_set).exp
    def __init__(self, vars_set: Set[str], exp: Exp):
        self.vars_set = vars_set
        self.exp = exp
```

The `StmAssign` requires `var_substitution()` function to substitute variable x with expression e inside a proposition P, here is an implementation for substituting universal quantifier expression (`ExpUni`):

```
def var_substitute(var: str, exp: Exp, post_cond: Exp):
    ...

    if isinstance(post_cond, ExpUni):
        return ExpUni(post_cond.vars_set, var_substitute(var, exp, post_cond.exp))
```

Questions:

1. Is the above `var_substitute()` function correct?
2. If your answer is yes, explain briefly the reason. If not, give your reason and your idea for correct implementation (no need to write code).

15. (6 points) The Hoare logic inference rules are defined inductively on the statement S:

$$\frac{}{\{P\}skip\{P\}} \quad \text{(H-Skip)}$$

$$\frac{}{\{P[x \mapsto E]\}x = E\{P\}} \quad \text{(H-Assign)}$$

$$\frac{\{P\}S_1\{R\} \qquad \{R\}S_2\{Q\}}{\{P\}S_1; S_2\{Q\}} \quad \text{(H-Seq)}$$

$$\frac{\{P \wedge E\}S_1\{Q\} \qquad \{P \wedge \neg E\}S_2\{Q\}}{\{P\}if(E; S_1; S_2)\{Q\}} \quad \text{(H-If)}$$

$$\frac{\{I \wedge E\}S\{I\}}{\{I\}while(E;S)\{I \wedge \neg E\}} \tag{H-While}$$

$$\frac{P \rightarrow A \qquad \{A\}S\{B\} \qquad B \rightarrow Q}{\{P\}S\{Q\}} \tag{H-Conseq}$$

Question: for the following Hoare triple, draw its proof tree:

$$\{True\}a = x + 1; if(a - 1 == 0; y = 1; y = a)\{y = x + 1\}$$