# CS510 Final 2010 Fall

December 16, 2010

**Name:** ____________________

**Qual Exam: (Yes/No)**

## 1 Proof and Verification (20p)

(a) Prove the following

$$\vdash (p \to q) \ \vee \ (q \to r)$$

(b) Use Hoare Logic to prove

$\{n = n_0 \wedge sum = 0\}$ *Sum* $\{sum = (n_0 + 1)n_0/2\}$

,in which *Sum* is defined is as follows.

```
while (n!=0) {
  sum=sum+n;
  n=n-1;
}
```

## 2 Validity, 10 points

Decide validity of the following theorem USING SAT SOLVING.

$$(p \rightarrow q) \ \rightarrow ((\neg p \rightarrow q) \rightarrow q)$$

# 3 Slicing (10p)

```
1. input(a);
2. c=10;
3. d=2;
4. if (a<0) /* should be a<=0 */
5.   x=c+d;
6. else
7.   x=c-d;
8. print (x);
```

Given input a=0, a wrong output is observed. The dynamic backward slice of the failure contains statements 1 and 2 that are not relevant with the failure. The observation that the failure only occurs under certain input implies that it is input relevant. Devise an algorithm to preclude statements 1 and 2 in the slice, you can leverage other analysis you have learned in the class and use them as primitives.

## 4 Abstraction (10p)

Assume we perform abstraction on the statement "x=x+1" regarding predicates "$p$: x is an odd number" and "$q$: x==5".

Please identify if the following abstractions are over-approximation, under-approximation, both, or neither.

(a) $p$=*

(b) if ($q$) $p$=false

(c) $p$=$\neg p$

(d) $p$= $\neg q$

(e) $p$=$q$? false: *

# 5 Model Checking and Test Generation (15p)

```
input(a,b,c);
z=0;
while (a>0) {
   if (a%b==0) {
      c=c-1;
      if (c>a)
         z=z+2;
   }
   a--;
   z++;
}
```

(a) Consider the above program. Unroll the loop once and translate it to SSA form.

(b) Design a test generation algorithm that generates test cases to cover each possible path of the tranformed program. The algorithm should not rely on concrete execution.

## 6 Testing (10p)

Assume a program has 4 factors A, B, C and D. Each has two levels. How many test cases do you need to achieve pair-wise full coverage. Writing down those test cases is encouraged but not required.

## 7 Project (5p)

Consider the following simplied IR, present your instrumentation to detect data dependence.

```
t1 = GET(6)

t2 = LOAD (t1)

t2 = t2+1

ST(0x35086) = t2
```

## 8 Concurrency (20p)

Thread T1

```
1  x=0;
2  spawn(T2);
3  acquire (L);
4  x=input();
5  release (L)
6  join (T2);
7  y=x+1;
```

Thread T2

```
10  acquire (L);
11  x=x+1;
12  release (L)
13  spawn (T3)
14  join (T3)
```

Thread T3

```
20  x=x+15;
```

Design a STATIC hybrid data race detection algorithm that analyses the CFGs of the threads and determines if there are data races about variable $x$. Assume the program is loop free and there is only one lock $L$. You are provided with the following primitive functions. **path**($l_1$, $l_2$) returns the set of statements in between $l_1$ and $l_2$ (in the same thread). For example, **path**(1,3)={1,2,3}. You can use "foreach n in path($l_1$, $l_2$)" to traverse the nodes in a path. **isAcq**($l$), **isRel**($l$), **isSpawn**($l$), **isJoin**($l$) decide if $l$ is an acquisition, release, spawn or join, respectively. **currentThread**($l$) decides the current thread of a statement. **spawnedThread**($l$) decides the thread that is spawned at $l$. **joinedThread**($l$) decides the thread that gets joined at $l$, e.g. **joinedThread** (6)=T2. If you assume additional primitives, please state them.

(a) define a function that decides if $x$ is protected by lock $L$ (5p).

(b) define a function that determines if a statement can happen before another statement (5p).

(c) present your algorithm. You can make use of the functions defined in first two steps (5p).

(d) present the result of applying it to the above program (5p).

The basic rules of natural deduction:

| | *introduction* | *elimination* |
|---|---|---|
| $\wedge$ | $\dfrac{\phi \quad \psi}{\phi \wedge \psi}\ \wedge\mathrm{i}$ | $\dfrac{\phi \wedge \psi}{\phi}\ \wedge\mathrm{e}_1 \qquad \dfrac{\phi \wedge \psi}{\psi}\ \wedge\mathrm{e}_2$ |
| $\vee$ | $\dfrac{\phi}{\phi \vee \psi}\ \vee\mathrm{i}_1 \qquad \dfrac{\psi}{\phi \vee \psi}\ \vee\mathrm{i}_2$ | $\dfrac{\phi \vee \psi \quad \boxed{\begin{array}{c}\phi\\ \vdots\\ \chi\end{array}} \quad \boxed{\begin{array}{c}\psi\\ \vdots\\ \chi\end{array}}}{\chi}\ \vee\mathrm{e}$ |
| $\rightarrow$ | $\dfrac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \psi\end{array}}}{\phi \rightarrow \psi}\ \rightarrow\mathrm{i}$ | $\dfrac{\phi \quad \phi \rightarrow \psi}{\psi}\ \rightarrow\mathrm{e}$ |
| $\neg$ | $\dfrac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \bot\end{array}}}{\neg\phi}\ \neg\mathrm{i}$ | $\dfrac{\phi \quad \neg\phi}{\bot}\ \neg\mathrm{e}$ |
| $\bot$ | (no introduction rule for $\bot$) | $\dfrac{\bot}{\phi}\ \bot\mathrm{e}$ |
| $\neg\neg$ | | $\dfrac{\neg\neg\phi}{\phi}\ \neg\neg\mathrm{e}$ |

Some useful derived rules:

$$\frac{\phi \rightarrow \psi \quad \neg\psi}{\neg\phi}\ \mathrm{MT} \qquad\qquad \frac{\phi}{\neg\neg\phi}\ \neg\neg\mathrm{i}$$

$$\frac{\boxed{\begin{array}{c}\neg\phi\\ \vdots\\ \bot\end{array}}}{\phi}\ \mathrm{PBC} \qquad\qquad \frac{}{\phi \vee \neg\phi}\ \mathrm{LEM}$$

**Figure 1.2.** Natural deduction rules for propositional logic.

$$\frac{(\!|\phi|\!)\ C_1\ (\!|\eta|\!) \qquad (\!|\eta|\!)\ C_2\ (\!|\psi|\!)}{(\!|\phi|\!)\ \ C_1; C_2\ \ (\!|\psi|\!)}\ \text{Composition}$$

$$\frac{}{(\!|\psi[E/x]|\!)\ x = E\ (\!|\psi|\!)}\ \text{Assignment}$$

$$\frac{(\!|\phi \wedge B|\!)\ C_1\ (\!|\psi|\!) \qquad (\!|\phi \wedge \neg B|\!)\ C_2\ (\!|\psi|\!)}{(\!|\phi|\!)\ \texttt{if}\ B\ \{C_1\}\ \texttt{else}\ \{C_2\}\ (\!|\psi|\!)}\ \text{If-statement}$$

$$\frac{(\!|\psi \wedge B|\!)\ C\ (\!|\psi|\!)}{(\!|\psi|\!)\ \texttt{while}\ B\ \{C\}\ (\!|\psi \wedge \neg B|\!)}\ \text{Partial-while}$$

$$\frac{\vdash_{\mathrm{AR}} \phi' \rightarrow \phi \qquad (\!|\phi|\!)\ C\ (\!|\psi|\!) \qquad \vdash_{\mathrm{AR}} \psi \rightarrow \psi'}{(\!|\phi'|\!)\ C\ (\!|\psi'|\!)}\ \text{Implied}$$

**Figure 4.1.** Proof rules for partial correctness of Hoare triples.