

CS 6110 – Formal Methods – Spring 2011

February 12, 2011

Assignment 3, Handed out: Feb 3rd, Due Feb 11th Friday midnight

1. (20%) Problem 5.3

Answer: For each implication, see if it is true for all assignments to free variables. The symbols $=$, $<$, etc. are supposed to have their standard meaning.

(a) $(1 = 2) \Rightarrow (2 = 3)$.

This formula has truth value *true*.

(b) $(gcd(x, y) = z) \Rightarrow (gcd(x + y, y) = z)$. Here, *gcd* stands for the greatest common divisor of its arguments.

This formula has truth value *true*.

(c) $(k > 1) \wedge hasclique(G, k) \Rightarrow hasclique(G, k - 1)$. Here, *hasclique*(G, k) means that graph G has a clique of size k .

This formula has truth value *true*.

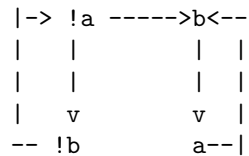
(d) $(k > 1) \wedge hasclique(G, k) \Rightarrow hasclique(G, k + 1)$.

This formula has a truth value of neither *true* nor *false*. It is satisfiable for some graphs G and falsifiable for others.

(e) $((a \Rightarrow (b \Rightarrow c)) \Rightarrow ((a \Rightarrow b) \Rightarrow (a \Rightarrow c)))$.

This formula has truth value *true*.

(f) (20%) Read Section 18.3.8 and apply this polynomial time algorithm to show that $(a \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee b)$ is satisfiable.



This implication graph results from the above 2CNF clauses. It has no cycles that include both x and $!x$, hence the formula is satisfiable. The assignment is obtained from the paths

$!a \rightarrow a$ and $!b \rightarrow b$ that exist. These paths give us the truth value assignment $a=1$ and $b=1$

- (g) (10%) Problem 18.3 Prove the following using a similar approach as illustrated above (Sperschneider's book, p. 107): *Every barber shaves everyone who does not shave himself. No barber shaves someone who shaves himself. Prove that there exists no barber.*

We obtain the following FOL assertions after a few trials:

- i. "If x is a barber, then for all y that don't shave themselves, x will shave them."

$$\forall x.(barb(x) \Rightarrow \forall y.(\neg shav(y, y) \Rightarrow shav(x, y)))$$

- ii. "If y is a self shaver, then for all x that are barbers, x will keep off y."

$$\forall y.(shav(y, y) \Rightarrow \forall x.(barb(x) \Rightarrow \neg shav(x, y)))$$

- iii. To prove there is no barber, suppose there is one, say b_0 . Then Step 1(g)i yields

$$\forall y.(\neg shav(y, y) \Rightarrow shav(b_0, y))$$

- iv. Specializing $y = b_0$, we get

$$\neg shav(b_0, b_0) \Rightarrow shav(b_0, b_0)$$

which is

$$shav(b_0, b_0).$$

- v. Specializing $y = b_0$ in Step 1(g)ii and doing modus ponens with $shav(b_0, b_0)$ yields

$$\forall x.(barb(x) \Rightarrow \neg shav(x, b_0))$$

- vi. Specializing $x = b_0$, we get $\neg shav(b_0, b_0)$ which contradicts $shav(b_0, b_0)$.

- vii. Hence there is no such b_0 .

25% of your points come from this problem: Using weakest preconditions, find two inputs that can make the Binary search loop print HL* . Apply these inputs and run the program to show that this printing happens.

```
#include <stdio.h>
#include <string.h>
#define STRLEN 16
int main(){
    char item; char str[STRLEN]; int lo=0; int hi; int mid, found=0;
    printf("Please give char to be searched: \n"); scanf("%c", &item);
    printf("Please give string within which to search: \n"); scanf("%s", str);
    hi=strlen(str)-1; // strlen ignores null at the end of string
    // hi points to last char in a 0 based array
    while(!found && lo <= hi) {
        mid = (lo+hi)/2;
        if (item == str[mid]) { printf("*"); found = 1; }
        else if (item < str[mid]) { hi = mid-1; printf("L"); }
        else { lo = mid+1; printf("H"); }
    }
    if (found) printf("\nitem %c found at posn %d\n", item, mid);
    else printf("\nitem %c not in str %s\n", item, str);
    return found;
}
```

Here is the sequence of path predicates

1: negate exit condition

$!(f \ \&\& \ l \leq h)$

2: set f to true, to get

$!(f \ \&\& \ l \leq h)$ i.e. True

3: Then we get $i = s[m]$

4: Then $i = s[(l+h)/2]$

5: Then $(f \ \&\& \ l \leq h) \ \&\& \ (i = s[(l+h)/2])$

6: Then $(f \ \&\& \ l \leq m-1) \ \&\& \ (i = s[(l+m-1)/2])$

7: Then $(i < s[m]) \ \&\& \ (f \ \&\& \ l \leq m-1) \ \&\& \ (i = s[(l+m-1)/2])$

8: Then $(i \neq s[m]) \ \&\& \ (i < s[m]) \ \&\& \ (f \ \&\& \ l \leq m-1) \ \&\& \ (i = s[(l+m-1)/2])$

which is

$(i < s[m]) \ \&\& \ (f \ \&\& \ l \leq m-1) \ \&\& \ (i = s[(l+m-1)/2])$

9: Then $(i < s[(l+h)/2]) \ \&\& \ (f \ \&\& \ l \leq (l+h)/2-1) \ \&\& \ (i = s[(l+(l+h)/2-1)/2])$

10: Then

$(f \ \&\& \ l \leq h) \ \&\& \ (i < s[(l+h)/2]) \ \&\& \ (f \ \&\& \ l \leq (l+h)/2-1) \ \&\& \ (i = s[(l+(l+h)/2-1)/2])$

which is

$(l \leq h) \ \&\& \ (i < s[(l+h)/2]) \ \&\& \ !f \ \&\& \ (l \leq (l+h)/2-1) \ \&\& \ (i = s[(l+(l+h)/2-1)/2])$

i.e.

$(i < s[(l+h)/2]) \ \&\& \ !f \ \&\& \ (l \leq (l+h)/2-1) \ \&\& \ (i = s[(l+(l+h)/2-1)/2])$

11: Then

$(i < s[(m+1+h)/2]) \ \&\& \ !f \ \&\& \ (m+1 \leq (m+1+h)/2-1) \ \&\& \ (i = s[((m+1)+(m+1+h)/2-1)/2])$

12: Then

$(i \geq s[m]) \ \&\& \ (i < s[(m+1+h)/2]) \ \&\& \ !f \ \&\& \ (m+1 \leq (m+1+h)/2-1) \ \&\& \ (i = s[((m+1)+(m+1+h)/2-1)/2])$

13: Then

$(i \neq s[m]) \ \&\& \ (i \geq s[m]) \ \&\& \ (i < s[(m+1+h)/2]) \ \&\& \ !f \ \&\& \ (m+1 \leq (m+1+h)/2-1) \ \&\& \ (i = s[((m+1)+(m+1+h)/2-1)/2])$

which is

```
(i >= s[m]) && (i < s[(m+1+h)/2]) && !f
    && (m+1 <= (m+1+h)/2-1) && (i = s[((m+1)+(m+1+h)/2-1)/2])
```

14: Then

```
(i >= s[(1+h)/2]) && (i < s[((1+h)/2+1+h)/2]) && !f
    && ((1+h)/2+1 <= ((1+h)/2+1+h)/2-1)
    && (i = s[((1+h)/2+1)+((1+h)/2+1+h)/2-1)/2])
```

15: Finally

```
!f && (l <= h) &&
(i >= s[(1+h)/2]) && (i < s[((1+h)/2+1+h)/2]) && !f
    && ((1+h)/2+1 <= ((1+h)/2+1+h)/2-1) && (i = s[((1+h)/2+1)+((1+h)/2+1+h)/2-1)/2])
```

which is

```
&& (l <= h)
&& (i >= s[(1+h)/2])
&& (i < s[((1+h)/2+1+h)/2])
&& !f
&& ((1+h)/2+1 <= ((1+h)/2+1+h)/2-1)
&& (i = s[((1+h)/2+1)+((1+h)/2+1+h)/2-1)/2])
```

Say $l = 1$, $h = 7$, then

```
&& (i >= s[4])
&& (i < s[6])
&& !f
&& (5 <= 5)
&& (i = s[5])
```

So pick $i = 3$ (for fun). Then,

```
&& (3 >= s[4])
&& (3 < s[6])
&& (3 = s[5])
```

Then pick $s[4,5,6] = 2,3,4$ and $l=1$, and $h=7$, and $i=3$.

This will make the HL* printout

```
The l pointer first goes up to 5, then the h pointer
comes down to 6, then finally  $l+h \text{ div } 2$  will set  $m$  to 5, to exit!
```

2. (5%) Show why Peterson's protocol can fail on a TSO memory model. Explain using an example.

Answer: It was clearly explained with respect to the proof of Peterson's algo, in class.

3. (10%) Show that the loop invariant guessed for Tournament min is one (a loop invariant) by walking back through the $A[2..y-1] \leq A[2..y]$ path.

Answer: Sweeping back, we have

```
LI:  $\min A[y:2y] = \min A[n:2n]$ 
```

```
Back through  $y := y-1$ 
```

$\min A[y-1 : 2y-2] = \min A[n:2n]$

Back through the $A[2.y-1] \leq A[2.y]$ path's assignment

$\min A[y-1 : 2y-2] = \min A[n:2n]$ -- where $A[y-1]$ is set to $A[2y-1]$

Back through conditional

$A[2y-1] \leq A[2y]$
 \implies
&& $\min A[y-1:2y-2] = \min(A[n:2n])$
 where $A[y-1] = A[2y-1]$

Now show

$(\min A[y:2y] = \min A[n:2n])$
 \implies
 $A[2y-1] \leq A[2y]$
 \implies
&& $\min A[y-1:2y-2] = \min(A[n:2n])$
 where $A[y-1] = A[2y-1]$

i.e.

$(\min A[y:2y] = \min A[n:2n]) \quad (A[2y-1] \leq A[2y])$

 $\min A[y-1:2y-2] = \min(A[n:2n]) \quad \text{where } A[y-1] = A[2y-1]$

This is true because

- since $(A[2y-1] \leq A[2y])$, the min element is $A[2y-1]$
- This element is guaranteed to be at $A[y-1]$

Thus the precondition denotes the same min as the post-condition.