CS402 Introduction to Logic, Fall 2007

Homework #1 Solution

2007년 10월 2일

1.

(a) p v~ (q^ (r → q))

p v~ (q^ (r → q))

↳ p (open)

↳ ~(q^(r→q))

↳ q, ~(~r v q)

↳ q, ~q (close)

↳ q, r (open)

~(p v~ (q^ (r → q)))

≡~p ^ ~(q^(~r v q))

≡~p ^ (~q v ~(r v q))

↳ ~p, ~q (open)

↳ ~p, ~r ^ ~q (open)

↳ ~p, ~r (open)

↳ ~p, ~q (open)

⇨ SATISFIABLE but not VALID

p	q	r	p v~ (q^ (r → q))
T	T	T	T
T	T	F	T
T	F	T	T
T	F	F	T
F	T	T	F
F	T	F	F
F	F	T	T
F	F	F	T

(b) p^q→ p v q

p^q→ p v q

≡ (~p^q) v (p v q)

↳ p (open)

↳ (~p^q) v q

↳ q (open)

↳ ~p, q (open)

~(p^q → p v q)

≡~(~(p^q) v (p v q))

≡ (p^q) ^ ~(p v q)

≡ (p^q)^(~p^~q)

↳ p, q, ~p, ~q (close)

⇨ VALID

p	q	p^q-->p v q
T	T	T
T	F	T
F	T	T
F	F	T

(c) ((p → ~q) → ~p) → q

p	q	((p--> ~q)--> ~p)-->q
T	T	T
T	F	T
F	T	T
F	F	F

((p → ~q) → ~p) → q

((~p v ~q) ^ p) v q

↳ q (open)

↳ ((~p v ~q) ^ p)

↳ p, (~p v ~q)

↳ p, ~p (close)

↳ p, ~q (open)

~(((p → ~q) → ~p) → q)

~((p → ~q) → ~q) v q

↳ ~(~(p → ~q) v ~q)

≡ (~p v q) ^ q

↳ ~p, q (open)

↳ q (open)

↳ q (open)

⇨ SATISFIABLE but not VALID

(d) (p → q) v (p → ~q)

p	q	(p-->q)v (p--> ~q)
T	T	T
T	F	T
F	T	T
F	F	T

(p → q) v (p → ~q)

↳ p → q

≡~p v q

↳ ~p (open)

↳ q (open)

↳ p → ~q

≡~p v ~q

↳ ~p (open)

↳ ~q (open)

~((p → q) v (p → ~q))

≡~(~p v q) ^ ~(~p v ~q)

≡ (p ^ ~q) ^ (p ^ q)

↳ p, ~q, p, q (closed)

⇨ VALID

(e) ((p → q) → p) → p

p	q	((p-->q)-->p)-->p
T	T	T
T	F	T
F	T	T
F	F	T

((p → q) → p) → p
≡~(~(~p v q) v p) v p
≡ ((~p v q) ^ ~p) v p
↳ p (open)
↳ (~p v q) ^ ~p
↳ ~p (open)
↳ ~p ^ q (open)

~(((p → q) → p) → p)
≡~(~((p → q) → p) v p)
≡ ((p → q) → p) ^ ~p
≡ (~(~p v q) v p) ^ ~p
↳ ~(~p v q), ~p
≡ (p ^ ~q), ~p
↳ p, ~q, ~p (closed)
↳ p, ~p (closed)

⇨ VALID

(f) (p v q →r) → (p → r) v (q → r)

p	q	r	(p v q → r) → (p → r) v (q → r)
T	T	T	T
T	T	F	T
T	F	T	T
T	F	F	T
F	T	T	T
F	T	F	T
F	F	T	T
F	F	F	T

(p v q → r) → (p → r) v (q → r)
≡~(p v q → r) v ((p → r) v (q → r))
≡~(~(p v q) v r) v (~p v r) v (~q v r)
↳ ~(~(p v q) v r)
↳ (p v q) , ~r
↳ ~r, p (open)
↳ ~r, q (open)
↳ (~p v r) v (~q v r)
↳ ~p v r
↳ ~p (open)
↳ r (open)
↳ ~ q v r
↳ ~q (open)
↳ r (open)

~((p v q → r) → (p → r) v (q → r))

≡~(~(p v q → r) v (p → r) v (q → r))

≡ (p v q → r) ^ ~(p → r) ^ ~(q → r)

≡ (~(p v q) v r) ^ (p ^ ~r) ^ (q ^ ~r)

↳ (~p ^ q) v r , p, ~r, q

↳ ~p, p, ~r, q (closed)

↳ r, p, ~r, q (closed)

⇨ VALID

(g) (p → q) → ~p → ~q

p	q	(p-->q)--> ~p--> ~q
T	T	T
T	F	T
F	T	F
F	F	T

(p → q) → ~p → ~q

≡~(~p v q) v (p v ~q)

≡ (p ^ ~q) v (p v ~q)

↳ p, (p v ~q)

↳ p (open)

↳ p, ~q (open)

↳ ~q, p, ~q

↳ ~q, p (open)

↳ ~q (open)

~((p → q) → ~p → ~q)

≡~(~(~p v q) v (~~P v ~q))

≡ ((~p v q) ^ (~p ^ q))

≡ (~p v q), ~p, q

↳ ~p, ~p, q (open)

↳ q, ~p, q (open)

⇨ SATISFIABLE but not VALID

2.

The proof is in pp.33-38 of the textbook.

3.

- Original code.

```
void f(unsigned int y) {
        unsigned int x = 1 ;
        x = x + y ;
        if (x == 2)
                x += 1 ;
        else
                x = 2 ;
        assert(x == 2) ;
```

- Translate the program into SSA form

$x_0 = 1$

$x_1 = x_0 + y$;

if (x_1 == 2)

$x_2 = x_1 + 1$

else

$x_2 = 2$

assert(x_2 ==2)

- Create a Boolean formula from the SSA representation

C : $(x_0==1)$ and $(x_1=x_0+y_0)$ and $(x_1==2$ imp $x_2==x_1+1)$ and $(x_1!=2$ imp $x_2==x_1+2)$

$x_{00} \wedge \neg x_{01} \wedge (x_{10} \leftrightarrow x_{00} \oplus y_{00}) \wedge (x_{11} \leftrightarrow x_{01} \oplus y_{01} \oplus (x_{00} \wedge y_{00}))$

$\wedge ((\neg x_{10} \wedge x_{11}) \rightarrow x_{20} \leftrightarrow x_{10} \oplus 1 \wedge x_{21} \leftrightarrow x_{11} \oplus 0 \oplus (x_{10} \wedge 1))$

$\wedge (\neg(x_{10} \wedge x_{11}) \rightarrow x_{20} \leftrightarrow x_{10} \oplus 0 \wedge x_{21} \leftrightarrow x_{11} \oplus 1 \oplus (x_{10} \wedge 0))$

P : $(x_2 == 2)$

$\neg x_{20} \wedge x_{21}$

F : C $\wedge$ $\neg$ P

- Translate the formula F into CNF form

$(x_0 == 1)$

$x_{00} \wedge \neg x_{01}$

$(x_1 = x_0 + y_0)$

$(\neg x_{11} \vee \neg x_{00} \vee \neg y_{00} \vee x_{01} \vee \neg y_{01}) \wedge (\neg x_{11} \vee \neg x_{00} \vee \neg y_{00} \vee \neg x_{01} \vee y_{01}) \wedge (\neg x_{11} \vee x_{00} \vee x_{01} \vee y_{01}) \wedge (\neg x_{11} \vee x_{00} \vee \neg x_{01} \vee \neg y_{01}) \wedge (\neg x_{11} \vee y_{00} \vee x_{01} \vee y_{01}) \wedge (\neg x_{11} \vee y_{00} \vee \neg x_{01} \vee \neg y_{01}) \wedge (x_{11} \vee x_{00} \vee x_{01} \vee \neg y_{01}) \wedge (x_{11} \vee x_{00} \vee \neg x_{01} \vee y_{01}) \wedge (x_{11} \vee y_{00} \vee x_{01} \vee \neg y_{01}) \wedge (x_{11} \vee y_{00} \vee \neg x_{01} \vee y_{01}) \wedge (x_{01} \vee y_{01} \vee \neg x_{00} \vee \neg y_{00} \vee x_{11}) \wedge (\neg x_{01} \vee \neg y_{01} \vee \neg x_{00} \vee \neg y_{00} \vee x_{11})$

$x_1 == 2$ imp $x_2 == x_1 + 1$

$(x_{10} \vee \neg x_{11} \vee x_{10} \vee x_{20}) \wedge (\neg x_{11} \vee x_{10} \vee x_{21})$

$x_1 != 2$ imp $x_2 == 2$

$(\neg x_{10} \vee \neg x_{20}) \wedge (\neg x_{10} \vee x_{21}) \wedge (x_{11} \vee \neg x_{20}) \wedge (x_{11} \vee x_{21})$

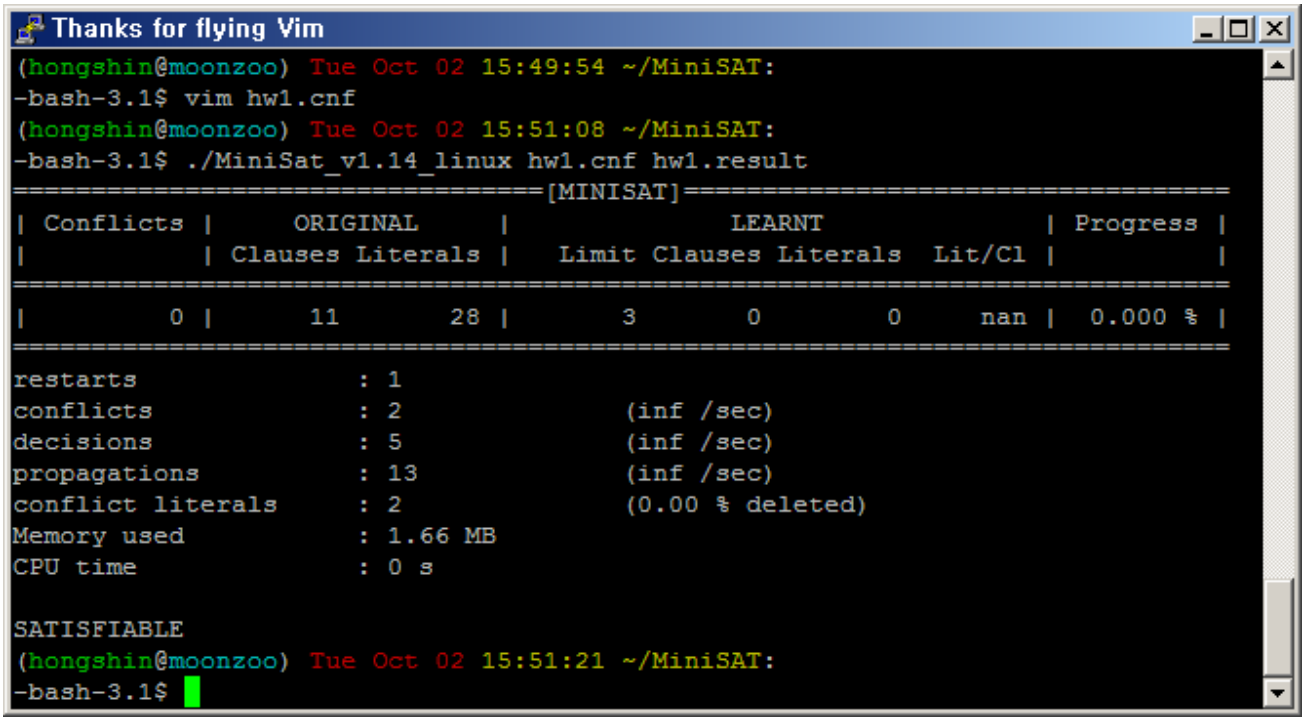
$\neg P (x_2 == 2)$

$x_{20} \vee \neg x_{21}$

- Run MiniSat and capture the result and interpret the result

Above CNF formula was translated into MiniSat input file format as below. The final formula in CNF form has 21 clauses composed with 8 boolean variables.

```
p cnf 8 21
1 0
-2 0
-4 -1 -7 2 -8 0
-4 -1 -7 -2 -8 0
-4 1 2 8 0
-4 1 -2 -8 0
-4 7 2 8 0
-4 7 -2 -8 0
4 1 2 -8 0
4 1 -2 8 0
4 7 2 -8 0
4 7 -2 8 0
2 8 -1 -7 4 0
-2 -8 -1 -7 4 0
3 -4 3 5 0
-4 3 6 0
-3 -5 0
-3 6 0
4 -5
4 6 0
5 -6 0
```



MiniSat returns the result file which contains the situation where given formula is satisified as below.

```
SAT
1 -2 -3 4 5 6 7 -8 0
```

The result shows that the formula is satisfied when x_{00} is true, x_{01} is false, x_{10} is false, x_{11} is true, x_{20} is true, x_{21} is true, y_{00} is true, and y_{01} is false. The interpretation of the result is that the assertion is violated in the case of `y` being 1. In that case the value of `x` is 3 when the assertion is met.

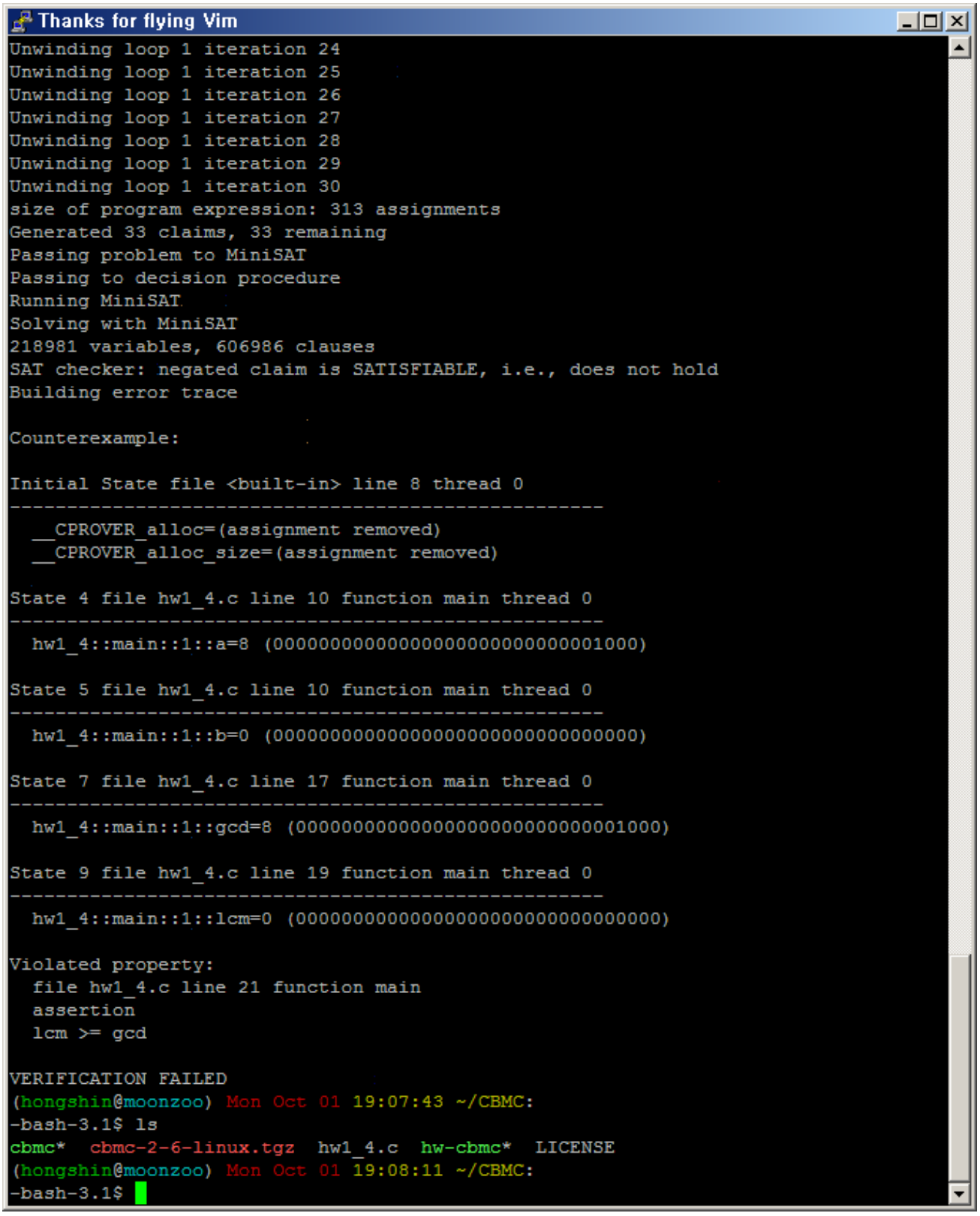
4.

I made a program which computes GCD and LCM of two given integers. And the assertion of the program is located at the last line to check whether GCD is always no greater than LCM. The code of the program is below.

```
int main()
{
    int x, y ;
    int a, b, t ;
    int gcd, lcm ;

    scanf("%d", &x) ;
    scanf("%d", &y) ;

    a = x ; b = y ;
    while (b != 0) {
        t = b ;
        b = a % b ;
        a = t ;
```

```
    }
    gcd = a ;
    lcm = x * y / gcd ;

    assert(lcm >= gcd) ;
}
```

This program was verified by CBMC with 30 numbers of unwinding. The result is below.

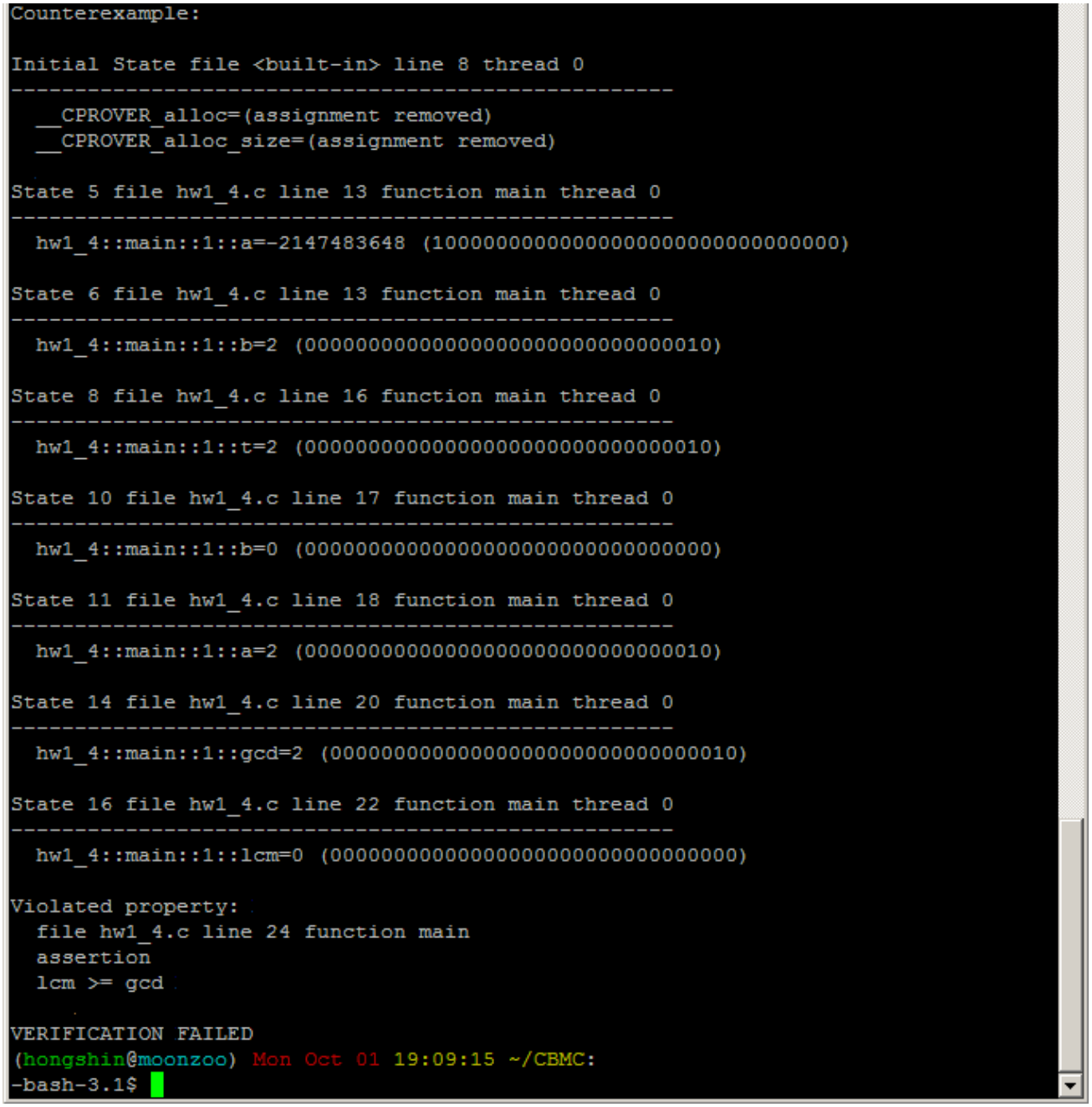


CMBC results that the verification has failed. The error trace showed that the program allows divide-by-zero case. To correct the program, I modified the program as below.

```
int main()
{
```

```
    int x, y ;
    int a, b, t ;
    int gcd, lcm ;

    scanf("%d", &x) ;
    scanf("%d", &y) ;
    if (x == 0 || y == 0) return 1 ;
    a = x ; b = y ;
    while (b != 0) {
        t = b ;
        b = a % b ;
        a = t ;
    }
    gcd = a ;
    lcm = x * y / gcd ;

    assert(lcm >= gcd) ;
}
```

The result of verification of modified program is below.



The verification result shows that overflow problem was not considered. So I modified the program to limit the range of input value as 0~100.

```
int main()
{
    int x, y ;
    int a, b, t ;
    int gcd, lcm ;

    scanf("%d", &x) ;
    scanf("%d", &y) ;
    if (x <= 0 || y <= 0) return 1 ;
    if (x > 100 || y > 100) return 1 ;
    a = x ; b = y ;
    while (b != 0) {
        t = b ;
```

```
        b = a % b ;
        a = t ;
    }
    gcd = a ;
    lcm = x * y / gcd ;

    assert(lcm >= gcd) ;
}
```

```
Starting Bounded Model Checking
**** WARNING: no body for function c::scanf
Unwinding loop 1 iteration 1
Unwinding loop 1 iteration 2
Unwinding loop 1 iteration 3
Unwinding loop 1 iteration 4
Unwinding loop 1 iteration 5
Unwinding loop 1 iteration 6
Unwinding loop 1 iteration 7
Unwinding loop 1 iteration 8
Unwinding loop 1 iteration 9
Unwinding loop 1 iteration 10
Unwinding loop 1 iteration 11
Unwinding loop 1 iteration 12
Unwinding loop 1 iteration 13
Unwinding loop 1 iteration 14
Unwinding loop 1 iteration 15
Unwinding loop 1 iteration 16
Unwinding loop 1 iteration 17
Unwinding loop 1 iteration 18
Unwinding loop 1 iteration 19
Unwinding loop 1 iteration 20
Unwinding loop 1 iteration 21
Unwinding loop 1 iteration 22
Unwinding loop 1 iteration 23
Unwinding loop 1 iteration 24
Unwinding loop 1 iteration 25
Unwinding loop 1 iteration 26
Unwinding loop 1 iteration 27
Unwinding loop 1 iteration 28
Unwinding loop 1 iteration 29
Unwinding loop 1 iteration 30
size of program expression: 329 assignments
Generated 33 claims, 33 remaining
Passing problem to MiniSAT
Passing to decision procedure
Running MiniSAT
Solving with MiniSAT
220255 variables, 610457 clauses
SAT checker: negated claim is UNSATISFIABLE, i.e., holds
VERIFICATION SUCCESSFUL
(hongshin@moonzoo) Mon Oct 01 19:11:00 ~/CBMC:
-bash-3.1$
```

Then, CBMC results that the program was verified successfully.