1. Provide cx for the statement:

inv2(x,y,a,b) = inv(x,y)[5000->a, 10000->b]

In [ ]:
```
import sys
sys.path.insert(1, '/Users/ekvashyn/Code/spacer-on-jupyter/src/')
from spacer_tutorial import *
import z3
z3.set_param(proof=True)
z3.set_param(model=True)
z3.set_html_mode(True)
```

In [ ]:
```
def mk_ts0():
    T = Ts('Ts0')
    x, x_out = T.add_var(z3.IntSort(), name='x')
    y, y_out = T.add_var(z3.IntSort(), name='y')
    T.Init = z3.And(x == 0, y == 5000)
    T.Tr = z3.And(x_out == x + 1, y_out == z3.If(x >= 5000, y+1, y))
    T.Bad = z3.And(x == 10000, x != y)
    return T

ts0 = mk_ts0()
HtmlStr(ts0)
```

Out[ ]: Transition System: Ts0

Init: $x = 0 \wedge y = 5000$

Bad: $x = 10000 \wedge x \neq y$

Tr: $x' = x + 1 \wedge y' = \text{If}(x \geq 5000, y + 1, y)$

In [ ]:
```
def mk_ts1():
    T = Ts('Ts1')
    x, x_out = T.add_var(z3.IntSort(), name='x')
    y, y_out = T.add_var(z3.IntSort(), name='y')
    a, a_out = T.add_var(z3.IntSort(), name='a')
    b, b_out = T.add_var(z3.IntSort(), name='b')
    T.Init = z3.And(a == 5000, b == 10000, x == 0, y == a)
    T.Tr = z3.And(x_out == x + 1, y_out == z3.If(x >= a, y+1, y), a_out == a
    T.Bad = z3.And(x == b, x != y)
    return T

ts1 = mk_ts1()
HtmlStr(ts1)
```

Out[ ]: Transition System: Ts1

Init: $a = 5000 \wedge b = 10000 \wedge x = 0 \wedge y = a$

Bad: $x = b \wedge x \neq y$

Tr: $x' = x + 1 \wedge y' = \text{If}(x \geq a, y + 1, y) \wedge a' = a \wedge b' = b$

In [ ]:
```
def vc_gen(T):
    '''Verification Condition (VC) for an Inductive Invariant'''
```

```
    Inv = z3.Function('Inv', *(T.sig() + [z3.BoolSort()]))

    InvPre = Inv(*T.pre_vars())
    InvPost = Inv(*T.post_vars())

    all_vars = T.all()
    vc_init = z3.ForAll(all_vars, z3.Implies(T.Init, InvPre))
    vc_ind = z3.ForAll(all_vars, z3.Implies(z3.And(InvPre, T.Tr), InvPost))
    vc_bad = z3.ForAll(all_vars, z3.Implies(z3.And(InvPre, T.Bad), z3.BoolVa
    return [vc_init, vc_ind, vc_bad], InvPre
```

In [ ]:

```
vc0, inv0 = vc_gen(ts0)
vc1, inv1 = vc_gen(ts1)
```

In [ ]:

```
chc_to_str(vc0)
```

Out[ ]: ∀x, y, x', y' : x = 0 ∧ y = 5000 ⇒ Inv(x, y)

∀x, y, x', y' :

Inv(x, y) ∧ x' = x + 1 ∧ y' = If(x ≥ 5000, y + 1, y) ⇒

Inv(x', y')

∀x, y, x', y' : Inv(x, y) ∧ x = 10000 ∧ x ≠ y ⇒ False

In [ ]:

```
chc_to_str(vc1)
```

Out[ ]: ∀x, y, a, b, x', y', a', b' :

a = 5000 ∧ b = 10000 ∧ x = 0 ∧ y = a ⇒ Inv(x, y, a, b)

∀x, y, a, b, x', y', a', b' :

Inv(x, y, a, b) ∧

x' = x + 1 ∧

y' = If(x ≥ a, y + 1, y) ∧

a' = a ∧

b' = b ⇒

Inv(x', y', a', b')

∀x, y, a, b, x', y', a', b' :

Inv(x, y, a, b) ∧ x = b ∧ x ≠ y ⇒ False

In [ ]:

```
HtmlStr(inv0)
```

Out[ ]: Inv(x, y)

In [ ]:

```
HtmlStr(inv1)
```

Out[ ]: Inv(x, y, a, b)

In [ ]:
```
res0, answer0 = solve_horn(vc0, max_unfold=100)
```

In [ ]:
```
res1, answer1 = solve_horn(vc1, max_unfold=100)
```

In [ ]:
```
res0
```

Out[ ]: **sat**

In [ ]:
```
res1
```

Out[ ]: **sat**

In [ ]:
```
answer0
```

Out[ ]: [Inv = [else → $(\neg(\nu_0 \leq 5000) \vee \neg(\nu_1 \geq 5001)) \wedge \neg(\nu_0 + -1\cdot\nu_1 \geq 1) \wedge (\neg(\nu_0 \geq 5000) \vee \neg(\nu_0 + -1\cdot\nu_1 \leq -1)) \wedge \neg(\nu_1 \leq 4999)$]]

In [ ]:
```
answer1
```

Out[ ]: [Inv = [else → $(\neg(\nu_1 + -1\cdot\nu_0 \leq -1) \vee \neg(\nu_0 + -1\cdot\nu_3 \geq -3)) \wedge \neg(\nu_2 + -1\cdot\nu_3 \geq -4999) \wedge (\neg(\nu_0 + -1\cdot\nu_2 \leq -1) \vee \neg(\nu_1 + -1\cdot\nu_2 \geq 1)) \wedge (\neg(\nu_1 + -1\cdot\nu_0 \leq -1) \vee \neg(\nu_0 + -1\cdot\nu_3 \leq -2) \vee \neg(\nu_0 + -1\cdot\nu_2 \geq 0)) \wedge \neg(\nu_1 + -1\cdot\nu_2 \leq -1) \wedge (\neg(\nu_0 + -1\cdot\nu_2 \geq 0) \vee \neg(\nu_1 + -1\cdot\nu_0 \geq 1))$]]

In [ ]:
```
answer0.eval(inv0)
```

Out[ ]: $(\neg(x \leq 5000) \vee \neg(y \geq 5001)) \wedge \neg(x + -1\cdot y \geq 1) \wedge (\neg(x \geq 5000) \vee \neg(x + -1\cdot y \leq -1)) \wedge \neg(y \leq 4999)$

In [ ]:
```
answer1.eval(inv1)
```

Out[ ]: $(\neg(y + -1\cdot x \leq -1) \vee \neg(x + -1\cdot b \geq -3)) \wedge \neg(a + -1\cdot b \geq -4999) \wedge (\neg(x + -1\cdot a \leq -1) \vee \neg(y + -1\cdot a \geq 1)) \wedge (\neg(y + -1\cdot x \leq -1) \vee \neg(x + -1\cdot a \geq 0) \vee \neg(x + -1\cdot b \leq -2)) \wedge \neg(y + -1\cdot a \leq -1) \wedge (\neg(x + -1\cdot a \geq 0) \vee \neg(y + -1\cdot x \geq 1))$

Invariant for the original benchmark is:

Inv1(x,y) =

$(\neg(y \geq 5001) \vee \neg(y + -1\cdot x \geq 1)) \wedge \neg(y + -1\cdot x \leq -1) \wedge \neg(y \leq 4999) =$

$(y \geq 5001) \Rightarrow (y \geq x + 1) \wedge y \geq x \wedge y > 4999$

Invariant for the transformed benchmark is:

Inv2(x,y,a,b) =

$(\neg(y \geq 5001) \vee x \geq y) \wedge y \geq a \wedge x \leq y =$

$(y > a \Rightarrow x \geq y) \wedge y \geq a \wedge x \leq y$

We need to prove that Inv1(x,y)[5000->a, 10000->b] != Inv2(x,y,a,b)

Let's rewrite Inv1(x,y) in such way:

Inv1(x,y) = ((y >= 5001 => x>=y) && y >= 5000 && x <= y)

Inv1(x,y)[5000->a, 10000->b] = ((y >= 5001 => x>=y) && y >= a && x <= y)

In [ ]:
```python
from z3 import *

# Create Z3 variables
a, b, x, y, x_prime, y_prime = Ints('a b x y x_prime y_prime')

# Create a Z3 solver
solver = Solver()

# Initial conditions
solver.add(x == 0)
solver.add(y == a)

# Transition relation constraints
transition_constraints = And(
    x_prime == x + 1,
    y_prime == If(x >= a, y + 1, y)
)

# Invariant constraints
invariant_constraints = And(
    Implies(y > 5001, x >= y),
    y >= a,
    x <= y
)

# Add transition relation and invariant constraints to the solver
solver.add(transition_constraints)
solver.add(Not(invariant_constraints))  # Adding negation of the invariant f

# Check for satisfiability
if solver.check() == unsat:
    print("Invariant holds for the transition system.")
else:
    print("Invariant does not hold for the transition system.")
    counterexample = solver.model()
    print(f"Counterexample found:")
    print("x =", counterexample[x])
    print("y =", counterexample[y])
    print("a =", counterexample[a])
```

```
Invariant does not hold for the transition system.
Counterexample found:
x = 0
y = 5002
a = 5002
```

2. Prove the statement:

inv2(x,y,a,b) = inv(x,y) ∧ a = 5000 ∧ b = 10000

In [ ]:
```
# vars for Ts0
x0, y0 = Ints('x0 y0')

# vars for Ts1
x1, y1, a1, b1 = Ints('x1 y1 a1 b1')

solver = Solver()

# Define the invariants for Ts0 and Ts1
I0 = And(
    Implies(y > 5000, x >= y),
    y >= 5000,
    x <= y
)
I1 = And( a1 == 5000, b1 == 10000, Implies(y1 >= a, x1 >= y1), x1 <= y1, y1

# Check if (I0 and (a = 5000) and (b = 10000)) is equivalent to I1
solver.add(Not(Implies(And(I0, a1 == 5000, b1 == 10000), I1)))
solver.add(Not(Implies(I1, And(I0, a1 == 5000, b1 == 10000))))

# Check for satisfiability (unsat implies equivalence)
if solver.check() == unsat:
    print("Invariant equivalency holds.")
else:
    print("Invariant equivalency does not hold.")
```

```
Invariant equivalency holds.
```