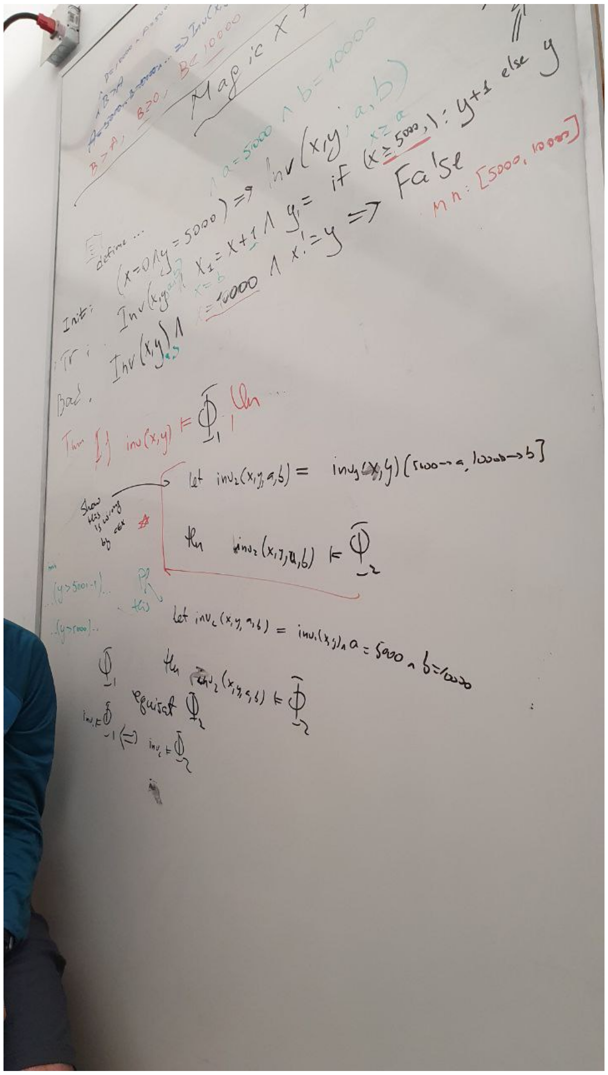
Prerequisite: During last weeks of summer I tried to show my technique to my advisor's collegue. During the conversation I got a task to prove 2 statement that will prove my technique. The picture of notes from this conversation I attach here. Everything on the blackboard was duplicated to this file. As an example was taken the file `s_split_01.smt2`



Proofs for equisat/non-equisat for variable substitution technique used in magicXform

In the file represented original transition system (ts0) and transformed transition system (ts1)

In [ ]:
```python
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

Invariants for those 2 systems locates below

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 → (¬(ν_0 ≤ 5000) ∨ ¬(ν_1 ≥ 5001)) ∧ ¬(ν_0 + -1·ν_1 ≥ 1) ∧ (¬(ν_0 ≥ 5000) ∨ ¬(ν_0 + -1·ν_1 ≤ -1)) ∧ ¬(ν_1 ≤ 4999)]]

In [ ]:
```
answer1
```

Out[ ]: [Inv = [else → (¬(ν_1 + -1·ν_0 ≤ -1) ∨ ¬(ν_0 + -1·ν_3 ≥ -3)) ∧ ¬(ν_2 + -1·ν_3 ≥ -4999) ∧ (¬(ν_0 + -1·ν_2 ≤ -1) ∨ ¬(ν_1 + -1·ν_2 ≥ 1)) ∧ (¬(ν_1 + -1·ν_0 ≤ -1) ∨ ¬(ν_0 + -1·ν_3 ≤ -2) ∨ ¬(ν_0 + -1·ν_2 ≥ 0)) ∧ ¬(ν_1 + -1·ν_2 ≤ -1) ∧ (¬(ν_0 + -1·ν_2 ≥ 0) ∨ ¬(ν_1 + -1·ν_0 ≥ 1))]]

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

Out[ ]: (¬(x ≤ 5000) ∨ ¬(y ≥ 5001)) ∧ ¬(x + -1·y ≥ 1) ∧ (¬(x ≥ 5000) ∨ ¬(x + -1·y ≤ -1)) ∧ ¬(y ≤ 4999)

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

Out[ ]: (¬(y + -1·x ≤ -1) ∨ ¬(x + -1·b ≥ -3)) ∧ ¬(a + -1·b ≥ -4999) ∧ (¬(x + -1·a ≤ -1) ∨ ¬(y + -1·a ≥ 1)) ∧ (¬(y + -1·x ≤ -1) ∨ ¬(x + -1·a ≥ 0) ∨ ¬(x + -1·b ≤ -2)) ∧ ¬(y + -1·a ≤ -1) ∧ (¬(x + -1·a ≥ 0) ∨ ¬(y + -1·x ≥ 1))

1. Provide cx for the statement: inv2(x,y,a,b) = inv(x,y)[5000->a, 10000->b]

- Invariant for the original benchmark is:
  Inv1(x,y) =

(¬(y ≥ 5001) ∨ ¬(y + -1·x ≥ 1)) ∧ ¬(y + -1·x ≤ -1) ∧ ¬(y ≤ 4999) =

(y ≥ 5001) => (y ≥ x + 1) ∧ y ≥ x ∧ y > 4999

- Invariant for the transformed benchmark is:
  Inv2(x,y,a,b) =

(¬(y ≥ 5001) ∨ x ≥ y) ∧ y ≥ a ∧ x ≤ y =

(y > a => x ≥ y) ∧ y ≥ a ∧ x ≤ y

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

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)

Let's to find a cx using z3:

In [ ]:
```
from z3 import *

a, b, x, y, x_prime, y_prime = Ints('a b x y x_prime y_prime')

solver = Solver()

solver.add(x == 0)
solver.add(y == a)

transition_constraints = And(
    x_prime == x + 1,
    y_prime == If(x >= a, y + 1, y)
)

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

solver.add(transition_constraints)
solver.add(Not(invariant_constraints))

# 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

- Ts0: `I0=Inv(x,y)`
- Ts1: `I2=Inv2(x,y,a,b)`

We aim to prove that (`I0` ∧(a=5000)∧(b=10000)) ≡ `I2`

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

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

```python
solver = Solver()
solver2 = Solver()

# Define the invariants for Ts0 and Ts1
I0 = And(Implies(y > 5000, x >= y), y >= 5000, x <= y)
I2 = And( a1 == 5000, b1 == 10000,
         Implies(y1 >= a1, x1 >= y1), x1 <= y1, y1 >= a1)


# Check if (I0 and (a = 5000) and (b = 10000)) is equivalent to I1
solver.add(Not(Implies(And(I0, a1 == 5000, b1 == 10000), I2)))
solver.add(Not(Implies(I2, And(I0, a1 == 5000, b1 == 10000))))

# NOTE: solver work above is the main here, but when I thought about
# the thing that we are trying to prove, I realized that adding constrains
# to I0 makes no sense, because I didn't pass a or b somewhere in the
# invariant. If I add `a` and `b` to I0 and substitute all 5000 and 1000 to
# the variables I'll get the I2 statement. So I think I can try to check
# equisat for
# inv2(x,y,a,b) = inv(x,y)
#
solver2.add(Not(Implies(I0, I2)))
solver2.add(Not(Implies(I2, I0)))


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

print(f"solver:")
check_is_sat(solver)
print()
print(f"solver2:")
check_is_sat(solver2)
```

```
solver:
Invariant equivalency holds.

solver2:
Invariant equivalency holds.
```