

# On Compositional safety verification with Max-SMT

Author: Fabian Böller  
Supervision: David Korzeniewski

RWTH Aachen  
*fabian.boeller@rwth-aachen.de*

SS 2017

## 1 Example execution

- 1: Input: The program, an SCC, the entry transitions of the SCC, an exit transition with an assertion

# CheckSafe

- 1: Input: The program, an SCC, the entry transitions of the SCC, an exit transition with an assertion
- 2: **if** the exit transition already implies the assertion **then**
- 3:     **return** Safe

# CheckSafe

- 1: Input: The program, an SCC, the entry transitions of the SCC, an exit transition with an assertion
- 2: **if** the exit transition already implies the assertion **then**
- 3:     **return** Safe
- 4: **else if** the exit transition is an initial transition **then**
- 5:     **return** Maybe

# CheckSafe

- 1: Input: The program, an SCC, the entry transitions of the SCC, an exit transition with an assertion
- 2: **if** the exit transition already implies the assertion **then**
- 3:     **return** Safe
- 4: **else if** the exit transition is an initial transition **then**
- 5:     **return** Maybe
- 6: Call CondSafe for the SCC with the given assertion

# CheckSafe

- 1: Input: The program, an SCC, the entry transitions of the SCC, an exit transition with an assertion
- 2: **if** the exit transition already implies the assertion **then**
- 3:     **return** Safe
- 4: **else if** the exit transition is an initial transition **then**
- 5:     **return** Maybe
- 6: Call CondSafe for the SCC with the given assertion
- 7: **if** no CII could be found **then**
- 8:     **return** Maybe

# CheckSafe

- 1: Input: The program, an SCC, the entry transitions of the SCC, an exit transition with an assertion
- 2: **if** the exit transition already implies the assertion **then**
- 3:     **return** Safe
- 4: **else if** the exit transition is an initial transition **then**
- 5:     **return** Maybe
- 6: Call CondSafe for the SCC with the given assertion
- 7: **if** no CII could be found **then**
- 8:     **return** Maybe
- 9: **for all** entry SCCs **do**
- 10:     **for all** literals of the according condition of the CII **do**
- 11:         Call CheckSafe for the entry SCC and with the literal as assertion



# CheckSafe

- 1: Input: The program, an SCC, the entry transitions of the SCC, an exit transition with an assertion
- 2: **if** the exit transition already implies the assertion **then**
- 3:     **return** Safe
- 4: **else if** the exit transition is an initial transition **then**
- 5:     **return** Maybe
- 6: Call CondSafe for the SCC with the given assertion
- 7: **if** no CII could be found **then**
- 8:     **return** Maybe
- 9: **for all** entry SCCs **do**
- 10:     **for all** literals of the according condition of the CII **do**
- 11:         Call CheckSafe for the entry SCC and with the literal as assertion
- 12: **if** all calls returned Safe **then**
- 13:     **return** Safe

# CheckSafe

- 1: Input: The program, an SCC, the entry transitions of the SCC, an exit transition with an assertion
- 2: **if** the exit transition already implies the assertion **then**
- 3:     **return** Safe
- 4: **else if** the exit transition is an initial transition **then**
- 5:     **return** Maybe
- 6: Call CondSafe for the SCC with the given assertion
- 7: **if** no CII could be found **then**
- 8:     **return** Maybe
- 9: **for all** entry SCCs **do**
- 10:     **for all** literals of the according condition of the CII **do**
- 11:         Call CheckSafe for the entry SCC and with the literal as assertion
- 12: **if** all calls returned Safe **then**
- 13:     **return** Safe
- 14: **return** the result of a call to CheckSafe with a narrowed version

- 1: Input: An SCC, the entry transitions of the SCC, an exit transition with an assertion

- 1: Input: An SCC, the entry transitions of the SCC, an exit transition with an assertion
- 2:  $k \leftarrow 1$

- 1: Input: An SCC, the entry transitions of the SCC, an exit transition with an assertion
- 2:  $k \leftarrow 1$
- 3: **repeat**
- 4:     Construct a formula  $\mathbb{F}_k$  for the SCC and the assertion

- 1: Input: An SCC, the entry transitions of the SCC, an exit transition with an assertion
- 2:  $k \leftarrow 1$
- 3: **repeat**
- 4:     Construct a formula  $\mathbb{F}_k$  for the SCC and the assertion
- 5:     Call the Max-SMT-Solver with  $\mathbb{F}_k$

- 1: Input: An SCC, the entry transitions of the SCC, an exit transition with an assertion
- 2:  $k \leftarrow 1$
- 3: **repeat**
- 4:     Construct a formula  $\mathbb{F}_k$  for the SCC and the assertion
- 5:     Call the Max-SMT-Solver with  $\mathbb{F}_k$
- 6:     **if** it returned a solution **then**
- 7:         **return** an invariant assigning each location the according condition from  $\mathbb{F}_k$

- 1: Input: An SCC, the entry transitions of the SCC, an exit transition with an assertion
- 2:  $k \leftarrow 1$
- 3: **repeat**
- 4:     Construct a formula  $\mathbb{F}_k$  for the SCC and the assertion
- 5:     Call the Max-SMT-Solver with  $\mathbb{F}_k$
- 6:     **if** it returned a solution **then**
- 7:         **return** an invariant assigning each location the according condition from  $\mathbb{F}_k$
- 8:      $k \leftarrow k + 1$
- 9: **until**  $k > \text{MAX\_CONJUNCTS}$



- 1: Input: An SCC, the entry transitions of the SCC, an exit transition with an assertion
- 2:  $k \leftarrow 1$
- 3: **repeat**
- 4:     Construct a formula  $\mathbb{F}_k$  for the SCC and the assertion
- 5:     Call the Max-SMT-Solver with  $\mathbb{F}_k$
- 6:     **if** it returned a solution **then**
- 7:         **return** an invariant assigning each location the according condition from  $\mathbb{F}_k$
- 8:      $k \leftarrow k + 1$
- 9: **until**  $k > \text{MAX\_CONJUNCTS}$
- 10: **return** None

1: Input: An SCC, the entry transitions of the SCC, a CII

# Narrowing

- 1: Input: An SCC, the entry transitions of the SCC, a CII
- 2: **for all** entry transitions **do**
- 3:     **for all** literals of the CII **do**

# Narrowing

- 1: Input: An SCC, the entry transitions of the SCC, a CII
- 2: **for all** entry transitions **do**
- 3:     **for all** literals of the CII **do**
- 4:         **if** literal could not be proved safe for this transition **then**
- 5:             Add a conjunct with the negated literal to the transition

- 1: Input: An SCC, the entry transitions of the SCC, a CII
- 2: **for all** entry transitions **do**
- 3:     **for all** literals of the CII **do**
- 4:         **if** literal could not be proved safe for this transition **then**
- 5:             Add a conjunct with the negated literal to the transition
- 6: **for all** transitions of the SCC **do**

# Narrowing

- 1: Input: An SCC, the entry transitions of the SCC, a CII
- 2: **for all** entry transitions **do**
- 3:     **for all** literals of the CII **do**
- 4:         **if** literal could not be proved safe for this transition **then**
- 5:             Add a conjunct with the negated literal to the transition
- 6: **for all** transitions of the SCC **do**
- 7:     Add a conjunct with the negated CII at the start location to the transition

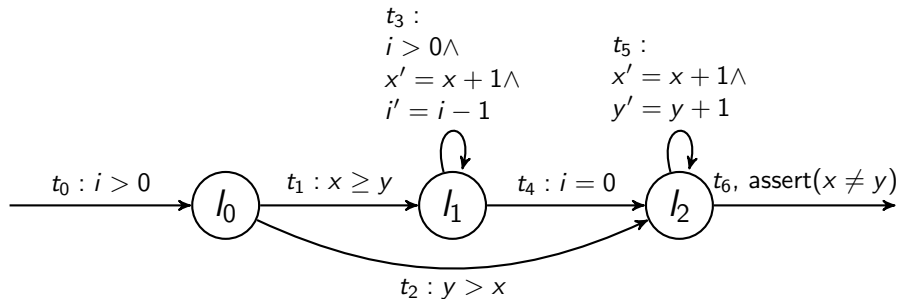
# Narrowing

- 1: Input: An SCC, the entry transitions of the SCC, a CII
- 2: **for all** entry transitions **do**
- 3:     **for all** literals of the CII **do**
- 4:         **if** literal could not be proved safe for this transition **then**
- 5:             Add a conjunct with the negated literal to the transition
- 6: **for all** transitions of the SCC **do**
- 7:     Add a conjunct with the negated CII at the start location to the transition
- 8:     Add a conjunct with the negated CII at the end location to the transition

# Example program

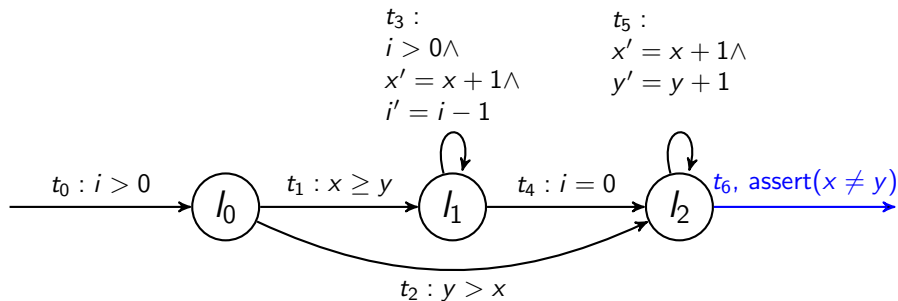
## Program

$\mathcal{V} = \{x, y, i\}$ ,  $\mathcal{L} = \{\ell_0, \ell_1, \ell_2\}$ ,  $\mathcal{T} = \{t_i \mid i \in \{1, \dots, 6\}\}$





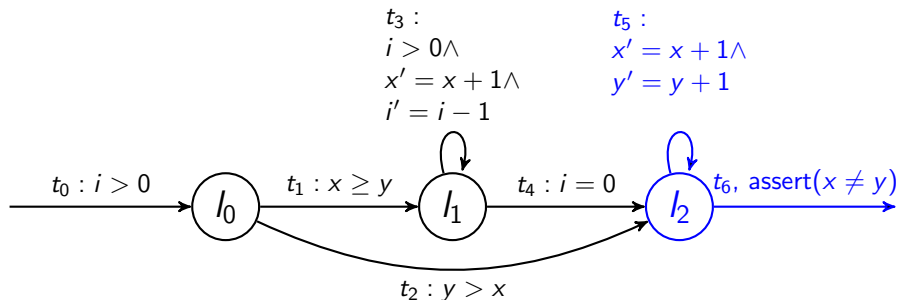
# Example execution



## Task

Prove that the program is safe for  $x \neq y$  at  $t_6$

# Example execution



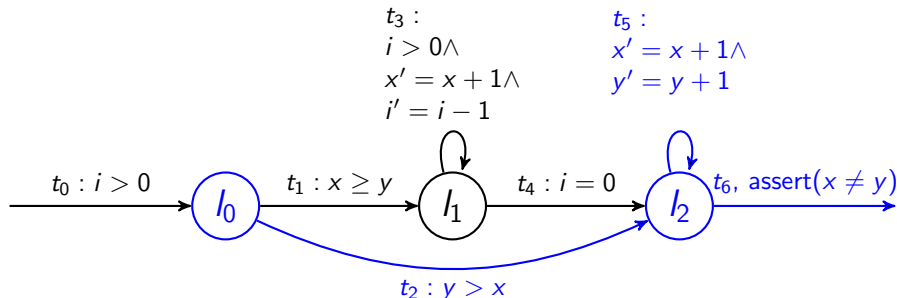
CheckSafe on  $\{l_2\}$  for  $x \neq y$

$t_6$  does not already imply  $x \neq y$

$t_6$  is not an initial transition

Call CondSafe, get  $x > y$  as precondition

# Example execution



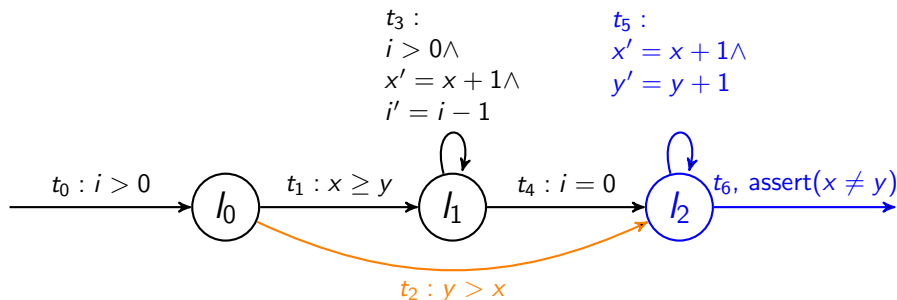
CheckSafe on  $\{l_0\}$  for  $x > y$

$t_2$  does not already imply  $x > y$

$t_2$  is not an initial transition

Call CondSafe

# Example execution

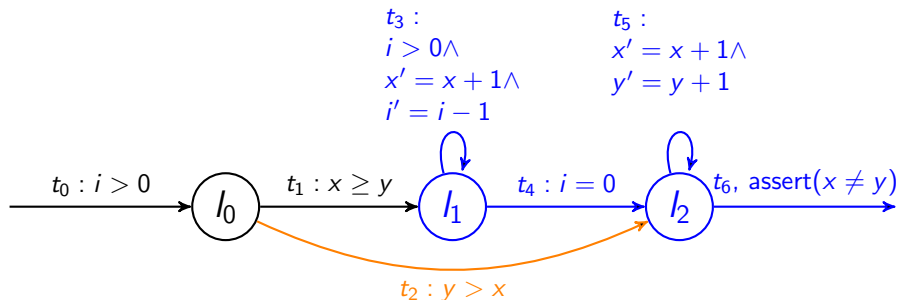


CheckSafe on  $\{\ell_0\}$  for  $x > y$

No precondition, since  $y > x$  contradicts  $x > y$

Path is maybe safe, but not for  $x > y$

# Example execution



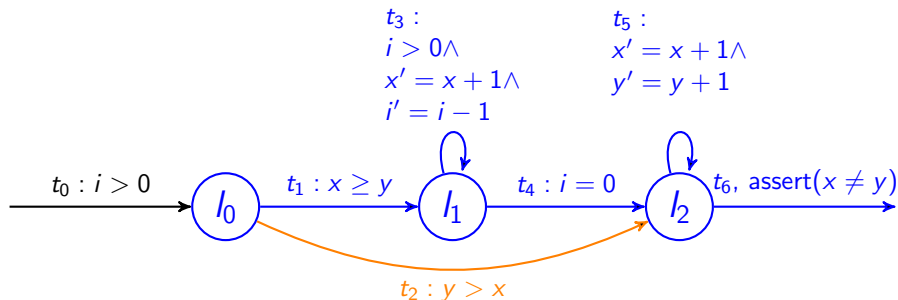
CheckSafe on  $\{l_1\}$  for  $x > y$

$t_4$  does not already imply  $x > y$

$t_4$  is not an initial transition

Call CondSafe, get  $i > 0 \wedge x \geq y$  as precondition

# Example execution



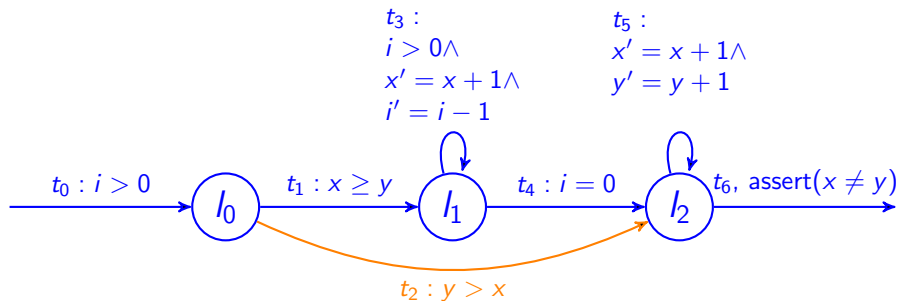
CheckSafe on  $\{l_0\}$  for  $i > 0$

$t_1$  does not already imply  $i > 0$

$t_1$  is not an initial transition

Call CondSafe, get  $i > 0$  as precondition

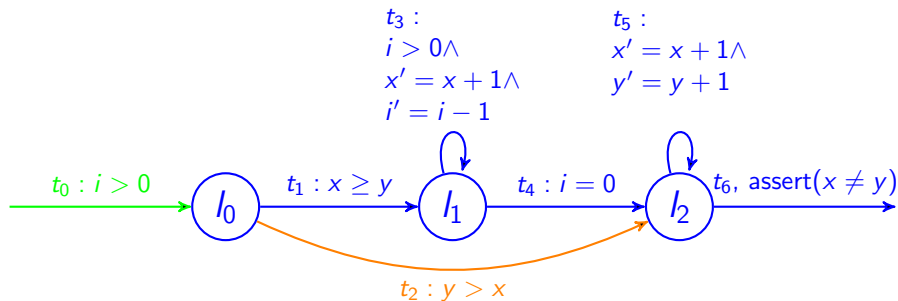
# Example execution



CheckSafe on initial SCC for  $i > 0$

$t_0$  does already imply  $i > 0$

# Example execution

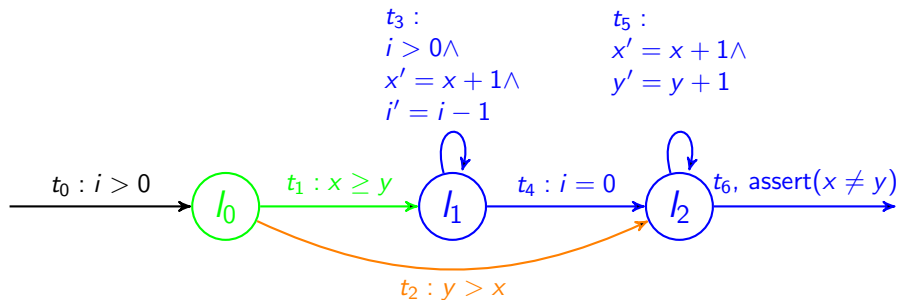


CheckSafe on initial SCC for  $i > 0$

Path is safe for  $i > 0$



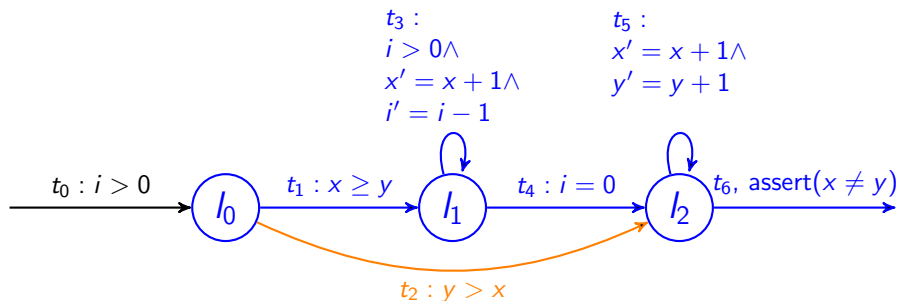
# Example execution



CheckSafe on  $\{l_0\}$  for  $i > 0$

Path is safe for  $i > 0$

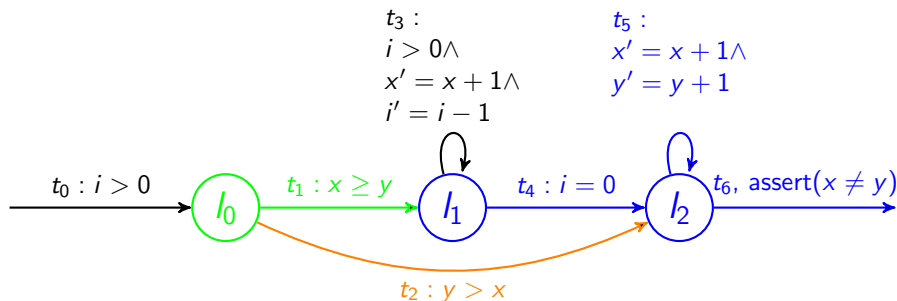
# Example execution



CheckSafe on  $\{l_0\}$  for  $x \geq y$

$t_1$  does already imply  $x \geq y$

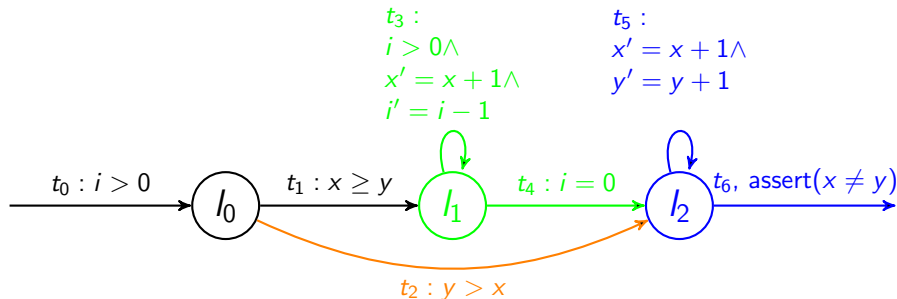
# Example execution



CheckSafe on  $\{l_0\}$  for  $x \geq y$

Path is safe for  $x \geq y$

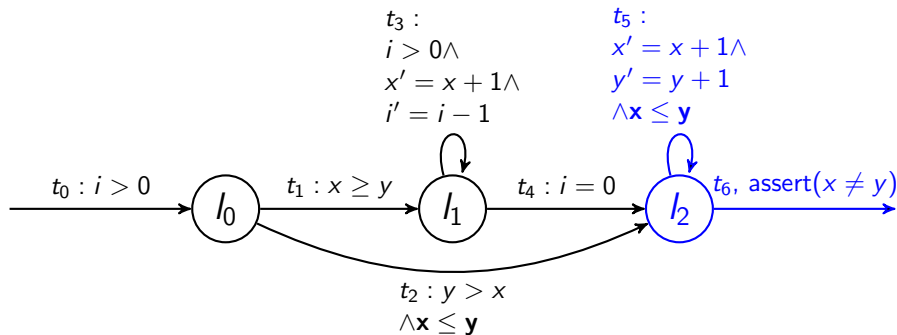
# Example execution



CheckSafe on  $\{l_1\}$  for  $x > y$

Path is safe for  $x > y$

# Example execution

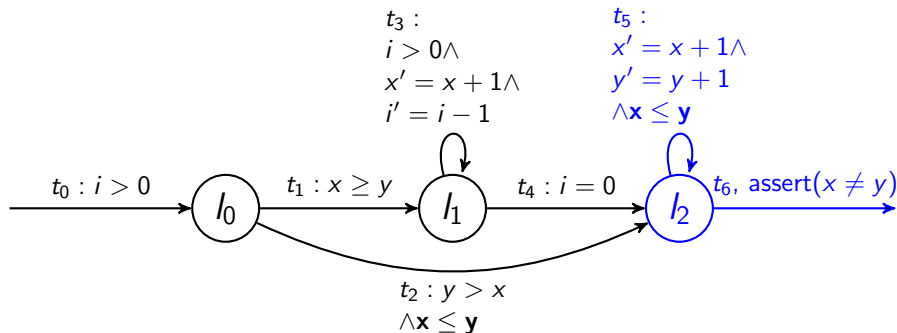


Narrow on  $\{l_2\}$

Add  $x \leq y$  to  $t_2$

Add  $x \leq y$  to  $t_5$

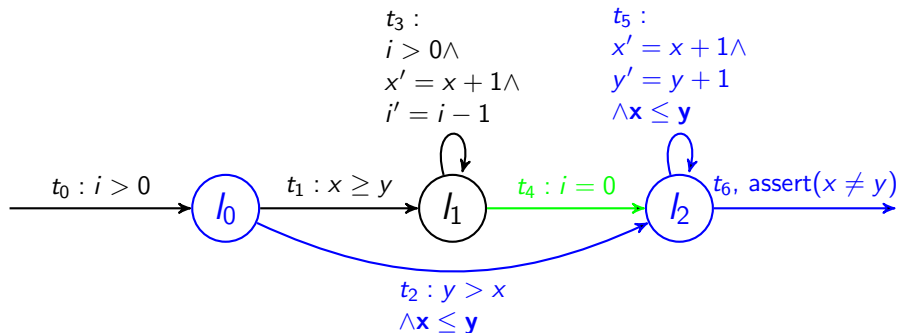
# Example execution



CheckSafe on  $\{l_2\}$  for  $x \neq y$

Call CondSafe, get  $y > x$  instead of  $x > y$  as precondition

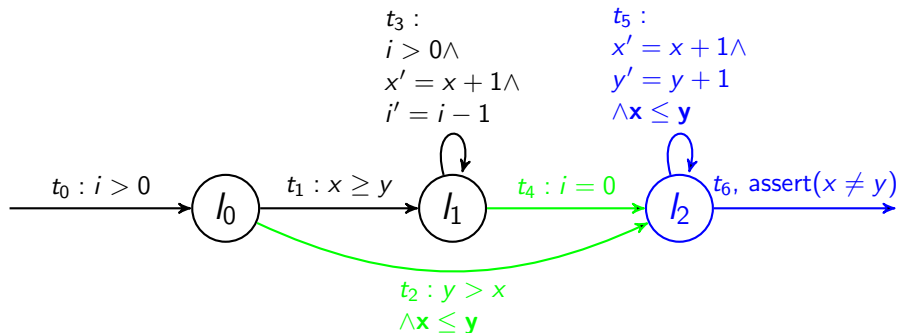
# Example execution



CheckSafe on  $\{l_0\}$  for  $y > x$

$t_2$  does already imply  $y > x$

# Example execution

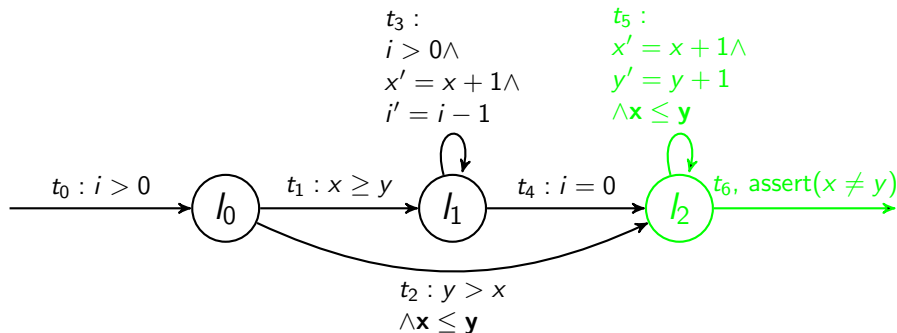


CheckSafe on  $\{\ell_0\}$  for  $y > x$

Path is safe for  $y > x$



# Example execution



CheckSafe on  $\{l_0\}$  for  $y > x$

Program is safe for  $x \neq y$



Brockschmidt, Marc and Larraz, Daniel and Oliveras, Albert and Rodriguez-Carbonell, Enric and Rubio, Albert (2015)

Compositional Safety Verification with Max-SMT

*Proceedings of FMCAD'15*

# The End