

# On Compositional safety verification with Max-SMT

Fabian Böller with David Korzeniewski

RWTH Aachen

*fabian.boeller@rwth-aachen.de*

SS 2017

# Overview

- 1 Introduction
- 2 Program graphs
- 3 Algorithm overview
- 4 Example execution
- 5 Conclusion

## Safety verification

Prove that an assertion is *always* true at a location

## Safety verification

Prove that an assertion is *always* true at a location

## Non-compositional safety verification

Safety verification where the whole program is analyzed in one step

## Safety verification

Prove that an assertion is *always* true at a location

## Non-compositional safety verification

Safety verification where the whole program is analyzed in one step

## Compositional safety verification

Safety verification where program parts are analyzed semi-independently and composed

Scalability  $\leftrightarrow$  Loss in precision

# Programs

Program

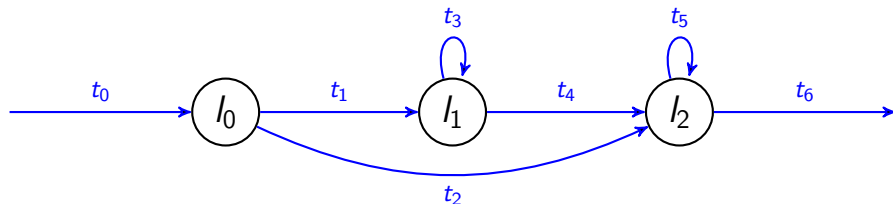
$$\mathcal{L} = \{\ell_0, \ell_1, \ell_2\}$$



# Programs

## Program

$\mathcal{L} = \{l_0, l_1, l_2\}$  ,  $\mathcal{T} = \{t_i \mid i \in \{1, \dots, 6\}\}$

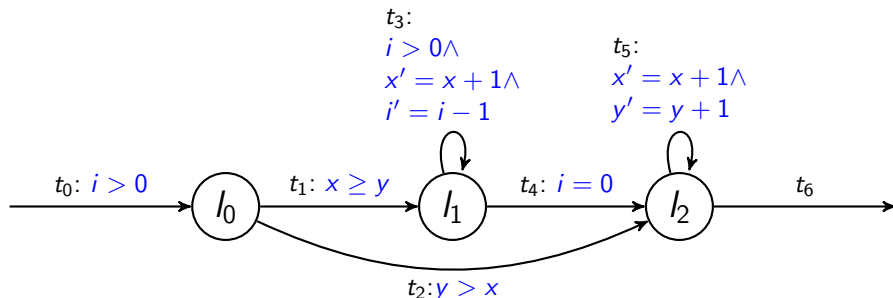




# Programs

## Program

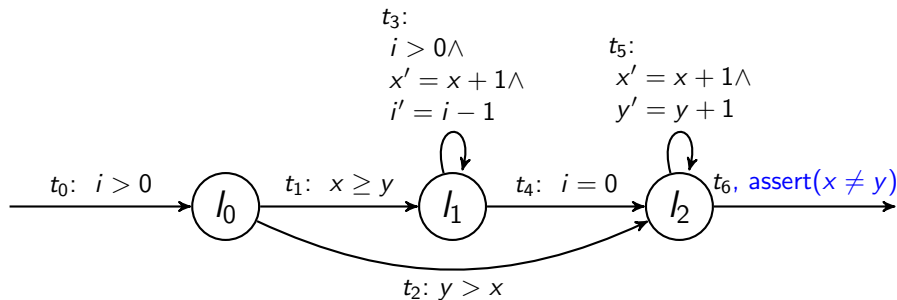
$\mathcal{L} = \{\ell_0, \ell_1, \ell_2\}$  ,  $\mathcal{T} = \{t_i \mid i \in \{1, \dots, 6\}\}$  ,  $\mathcal{V} = \{x, y, i\}$  ,  $\mathcal{V}' = \{x', y', i'\}$



# Programs

## Program

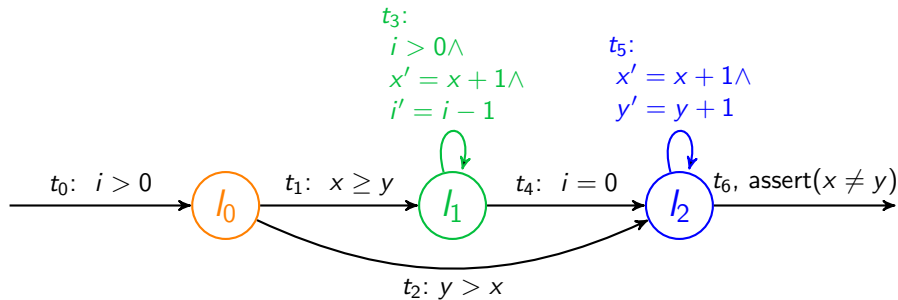
$\mathcal{L} = \{\ell_0, \ell_1, \ell_2\}$  ,  $\mathcal{T} = \{t_i \mid i \in \{1, \dots, 6\}\}$  ,  $\mathcal{V} = \{x, y, i\}$ ,  $\mathcal{V}' = \{x', y', i'\}$



# Programs

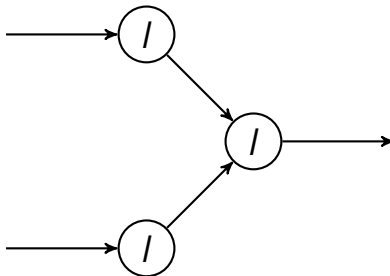
## Program

$\mathcal{L} = \{\ell_0, \ell_1, \ell_2\}$  ,  $\mathcal{T} = \{t_i \mid i \in \{1, \dots, 6\}\}$  ,  $\mathcal{V} = \{x, y, i\}$ ,  $\mathcal{V}' = \{x', y', i'\}$



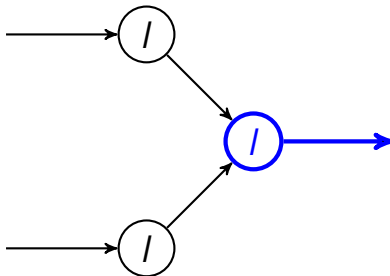
**Idea**

Prove that an assertion is satisfied by recursively checking all entry components



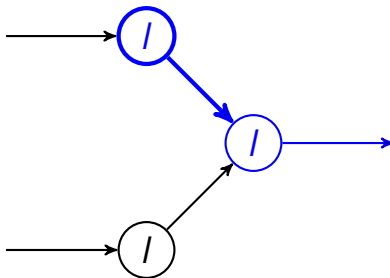
**Idea**

Prove that an assertion is satisfied by recursively checking all entry components



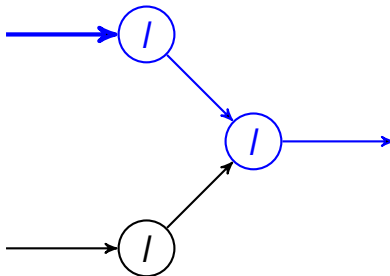
## Idea

Prove that an assertion is satisfied by recursively checking all entry components



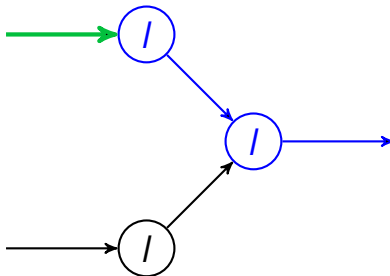
**Idea**

Prove that an assertion is satisfied by recursively checking all entry components



**Idea**

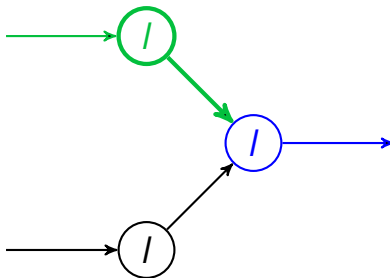
Prove that an assertion is satisfied by recursively checking all entry components





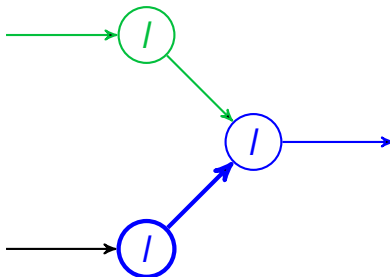
## Idea

Prove that an assertion is satisfied by recursively checking all entry components



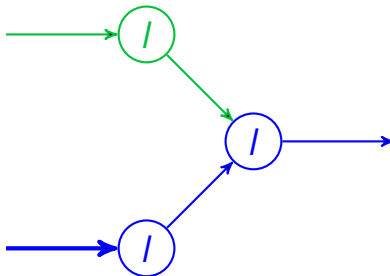
**Idea**

Prove that an assertion is satisfied by recursively checking all entry components



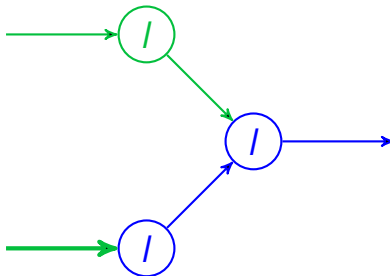
**Idea**

Prove that an assertion is satisfied by recursively checking all entry components



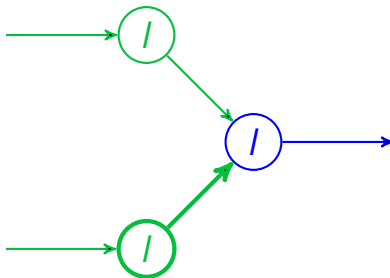
**Idea**

Prove that an assertion is satisfied by recursively checking all entry components



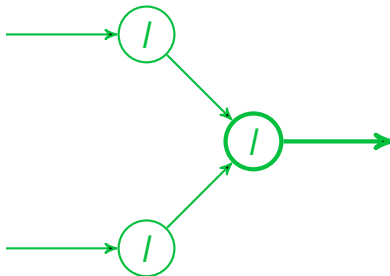
**Idea**

Prove that an assertion is satisfied by recursively checking all entry components



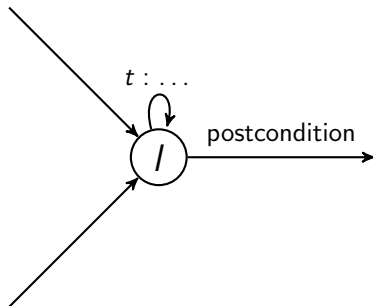
**Idea**

Prove that an assertion is satisfied by recursively checking all entry components



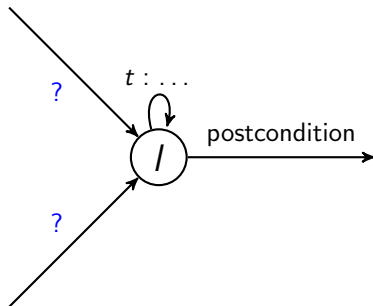
## Idea

Find a precondition for a component such that all runs satisfying the precondition always imply the postcondition



## Idea

Find a precondition for a component such that all runs satisfying the precondition always imply the postcondition

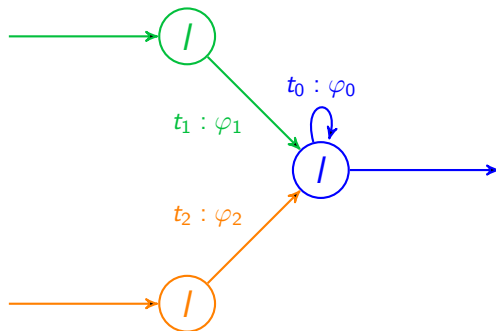




# Narrowing

## Idea

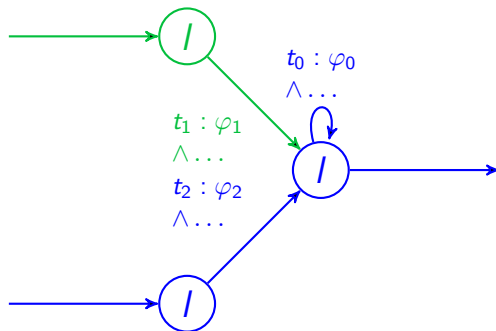
Manipulate the program such that new preconditions can be found



# Narrowing

## Idea

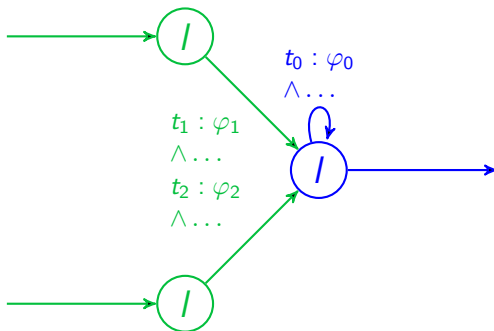
Manipulate the program such that new preconditions can be found



# Narrowing

## Idea

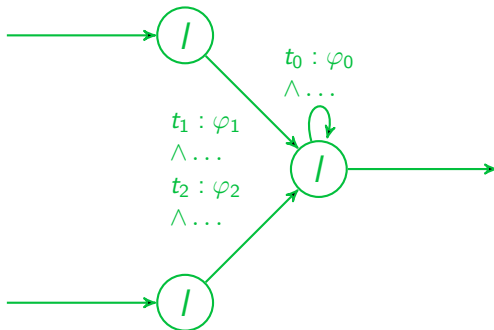
Manipulate the program such that new preconditions can be found



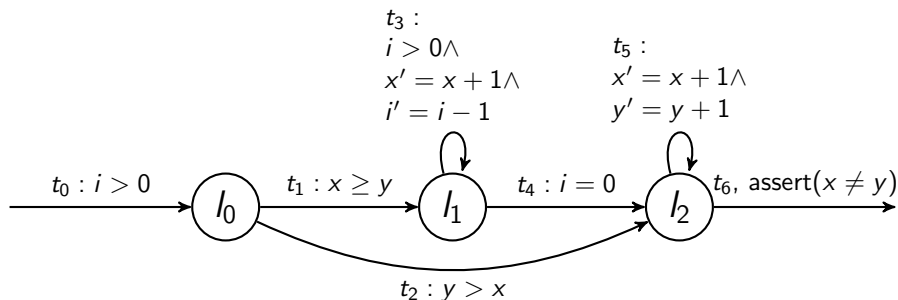
# Narrowing

## Idea

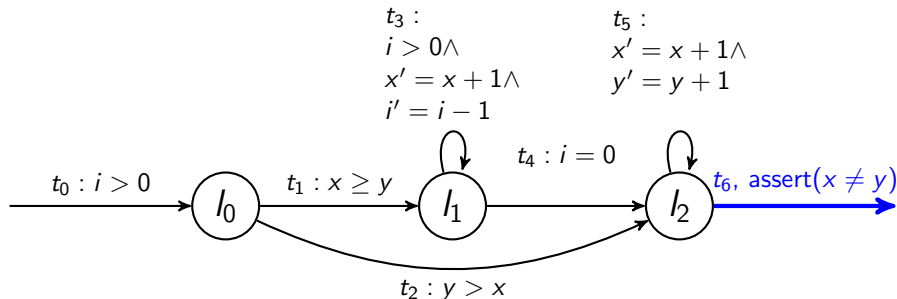
Manipulate the program such that new preconditions can be found



# Example program



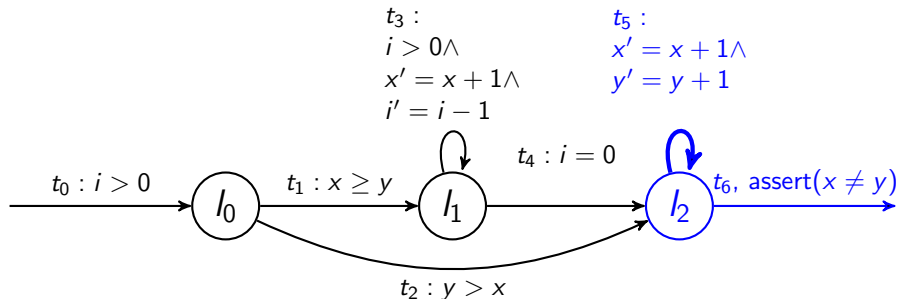
# Example execution



## Task

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

# Example execution



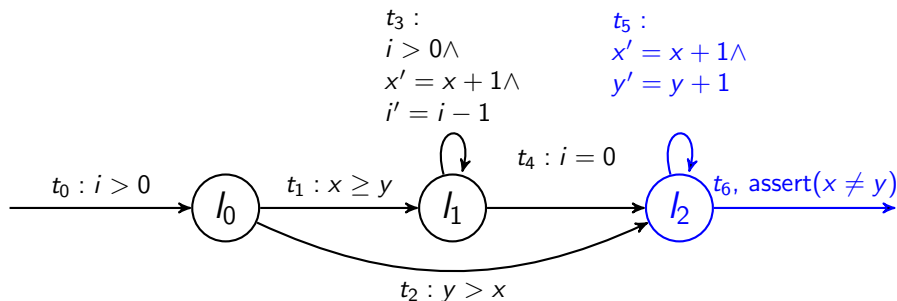
CheckSafe on  $\{l_2\}$  for  $x \neq y$

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

$t_6$  is not an initial transition

Call CondSafe

# Example execution

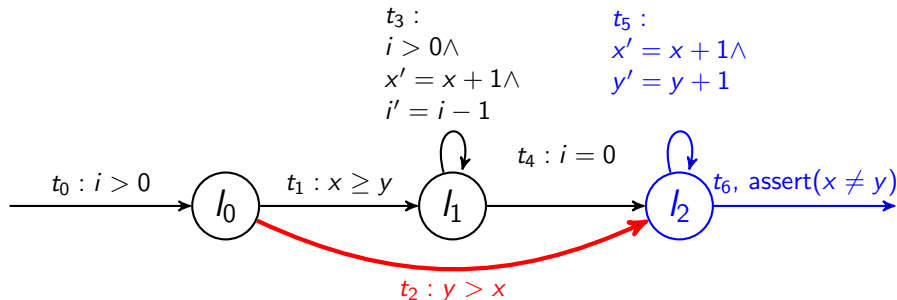


Max-SMT on  $\{\ell_2\}$  for  $x \neq y$

$$I_{\ell_2,1}(\{x, y, i\}) \equiv i_{\ell_2,1} + i_{\ell_2,1,x} * x + i_{\ell_2,1,y} * y + i_{\ell_2,1,i} * i \leq 0$$



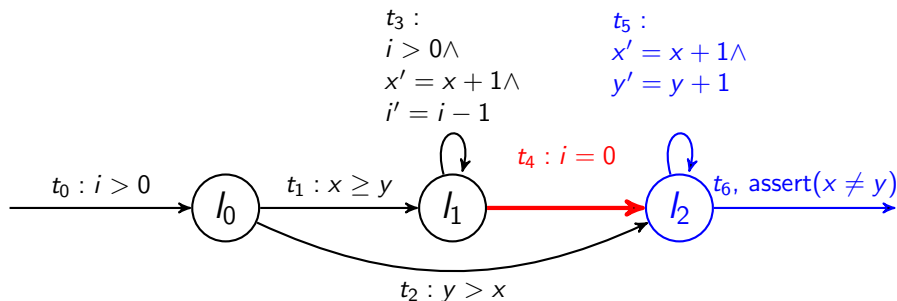
# Example execution



Max-SMT on  $\{l_2\}$  for  $x \neq y$

$$\mathbb{I}_{t_2,1,1} \equiv y > x \wedge i' = i \wedge x' = x \wedge y' = y \Rightarrow l'_{l_2,1,1}$$

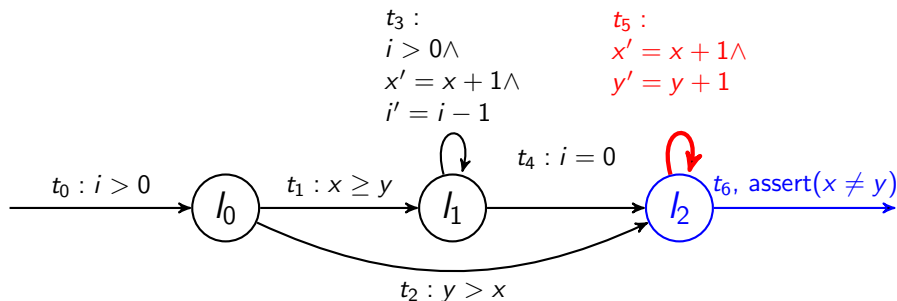
# Example execution



Max-SMT on  $\{\ell_2\}$  for  $x \neq y$

$$\mathbb{I}_{t_4,1,1} \equiv i = 0 \wedge i' = i \wedge x' = x \wedge y' = y \Rightarrow \mathbb{I}'_{\ell_2,1,1}$$

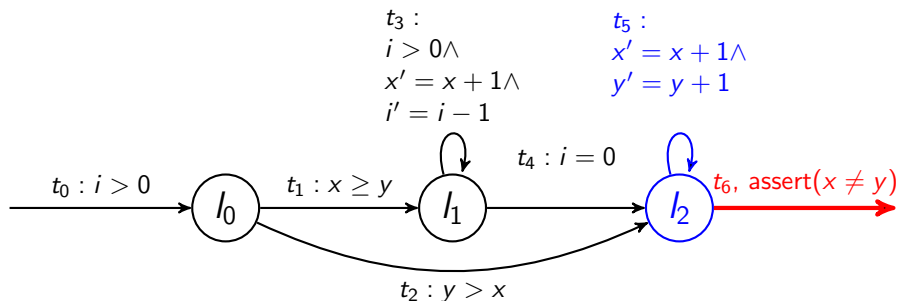
# Example execution



Max-SMT on  $\{\ell_2\}$  for  $x \neq y$

$$\mathbb{C}_{t_5,1} \equiv l_{\ell_2,1} \wedge x' = x + 1 \wedge y' = y + 1 \wedge i' = i \Rightarrow l'_{\ell_2,1}$$

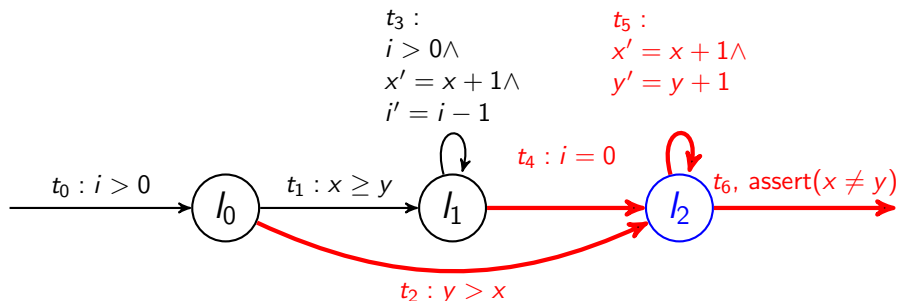
# Example execution



Max-SMT on  $\{\ell_2\}$  for  $x \neq y$

$$S_1 \equiv l_{\ell_2,1} \wedge i' = i \wedge x' = x \wedge y' = y \Rightarrow x' \neq y'$$

# Example execution

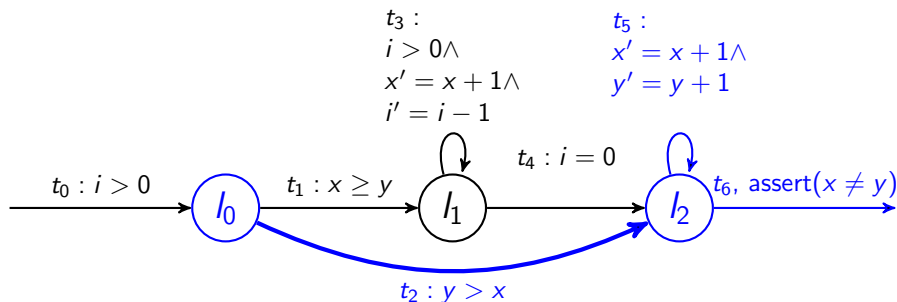


Max-SMT on  $\{\ell_2\}$  for  $x \neq y$

$\mathbb{F}_1 \equiv \mathbb{C}_{t_5,1} \wedge \mathbb{S}_1 \wedge (\mathbb{I}_{t_2,1,1} \vee \neg p_{t_2}) \wedge (\mathbb{I}_{t_4,1,1} \vee \neg p_{t_4}) \wedge [p_{t_2}, 1] \wedge [p_{t_4}, 1]$

Assume  $x > y$  does satisfy  $\mathbb{F}_1$

# Example execution



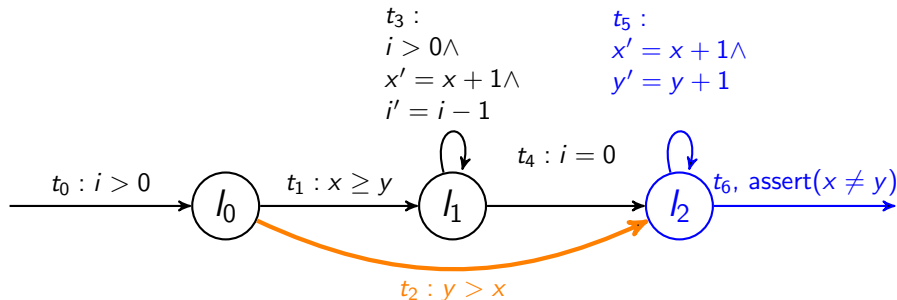
CheckSafe on  $\{l_0\}$  for  $x > y$

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

$t_2$  is not an initial transition

Call CondSafe

# Example execution

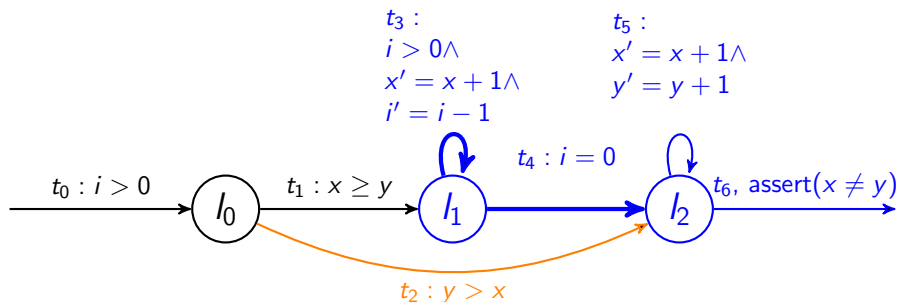


CheckSafe on  $\{\ell_0\}$  for  $x > y$

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

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

# Example execution



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

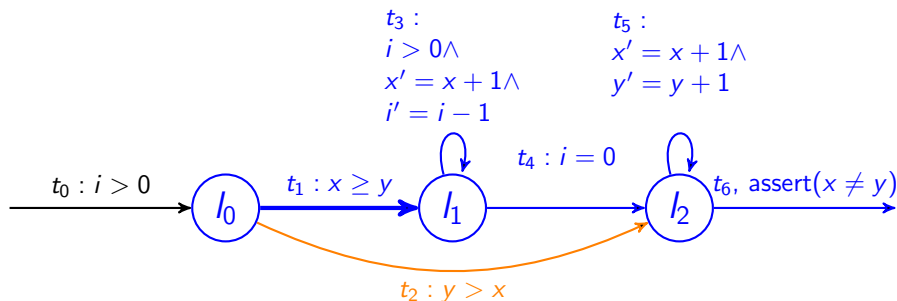
$t_4$  does not already imply  $x > y$

$t_4$  is not an initial transition

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



# Example execution



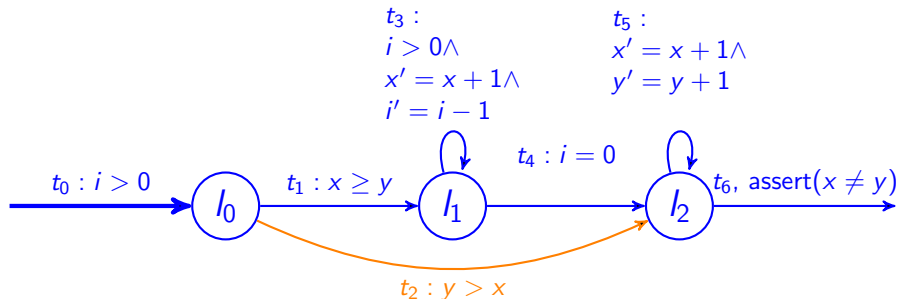
CheckSafe on  $\{l_0\}$  for  $i > 0$

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

$t_1$  is not an initial transition

Call CondSafe, get  $i > 0$  as precondition

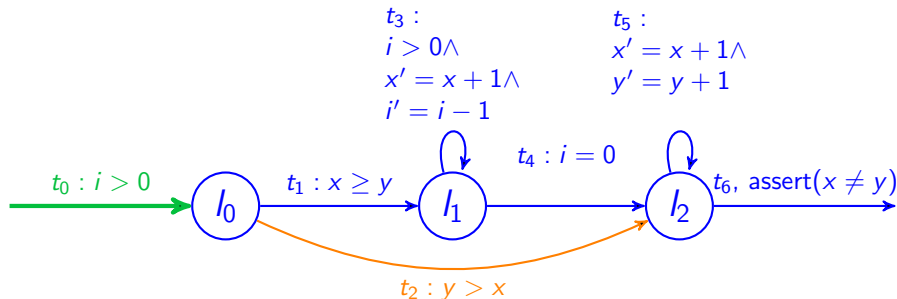
# Example execution



CheckSafe on initial SCC for  $i > 0$

$t_0$  does already imply  $i > 0$

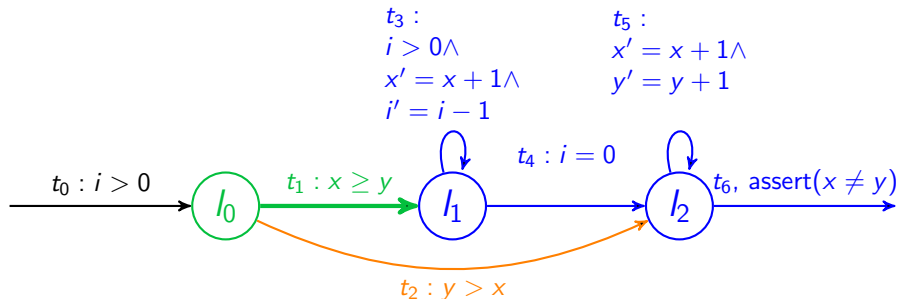
# Example execution



CheckSafe on initial SCC for  $i > 0$

Path is safe for  $i > 0$

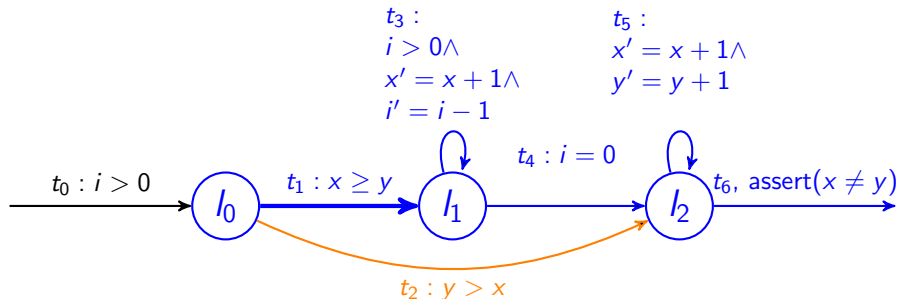
# Example execution



CheckSafe on  $\{\ell_0\}$  for  $i > 0$

Path is safe for  $i > 0$

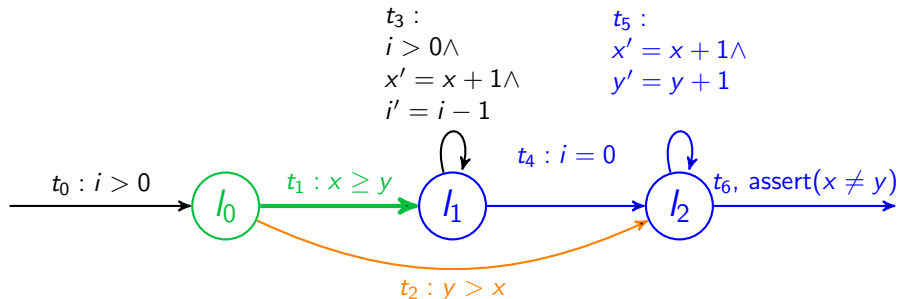
# Example execution



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

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

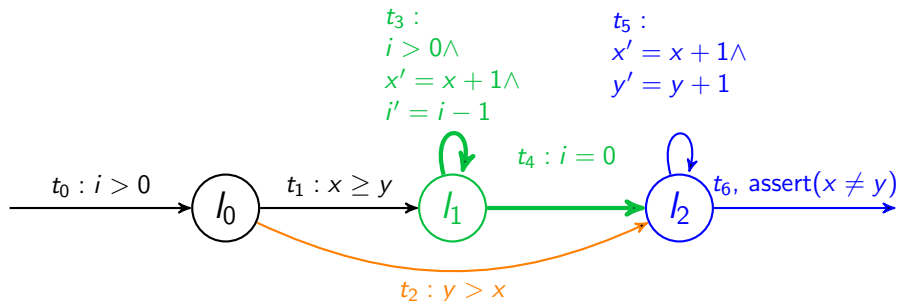
# Example execution



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

Path is safe for  $x \geq y$

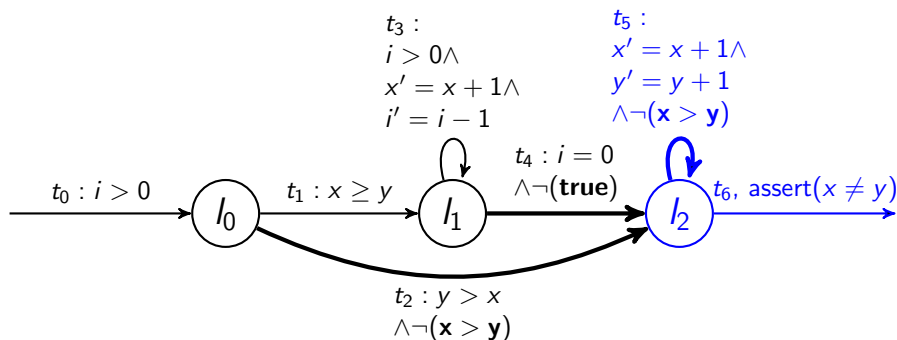
# Example execution



CheckSafe on  $\{\ell_1\}$  for  $x > y$

Path is safe for  $x > y$

# Example execution



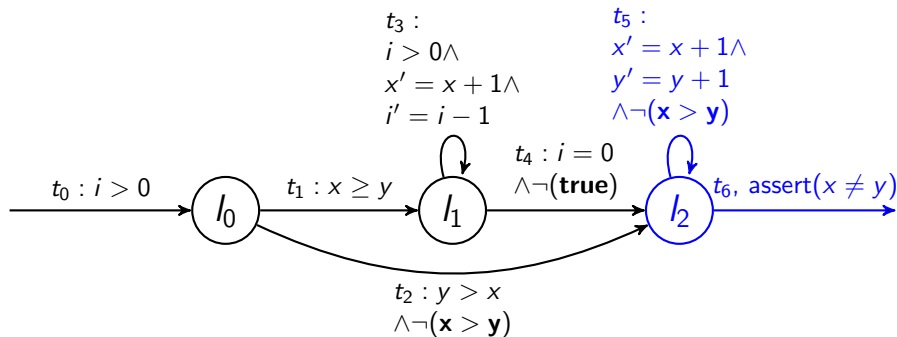
Narrow on  $\{l_2\}$

Add  $\neg(x > y)$  to  $t_2$

Add  $\neg(x > y)$  to  $t_5$



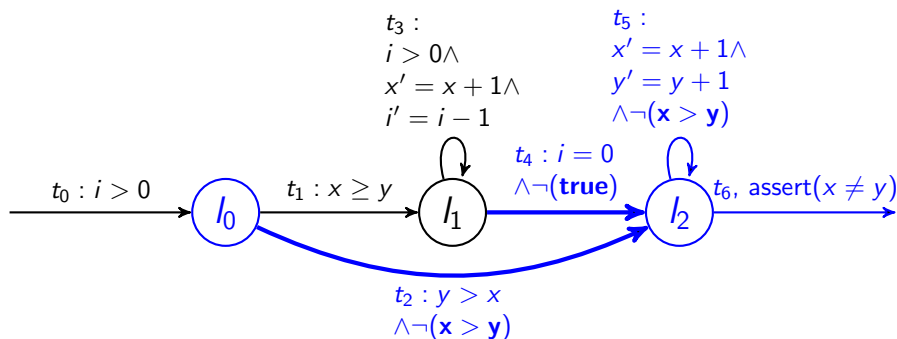
# Example execution



CheckSafe on  $\{\ell_2\}$  for  $x \neq y$

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

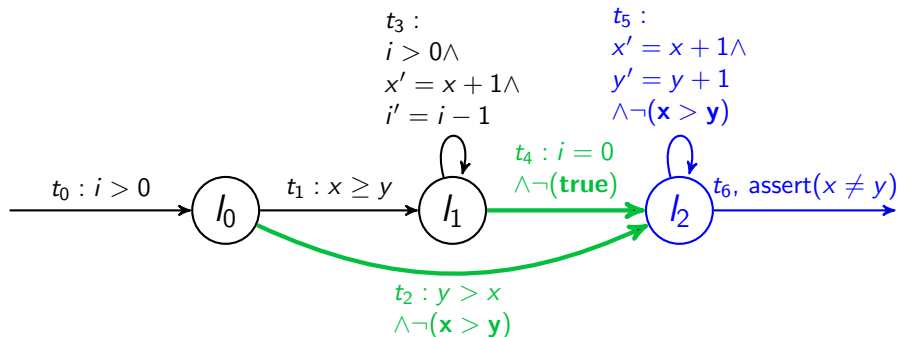
# Example execution



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

$t_2$  does already imply  $y > x$   $t_4$  does already imply  $y > x$

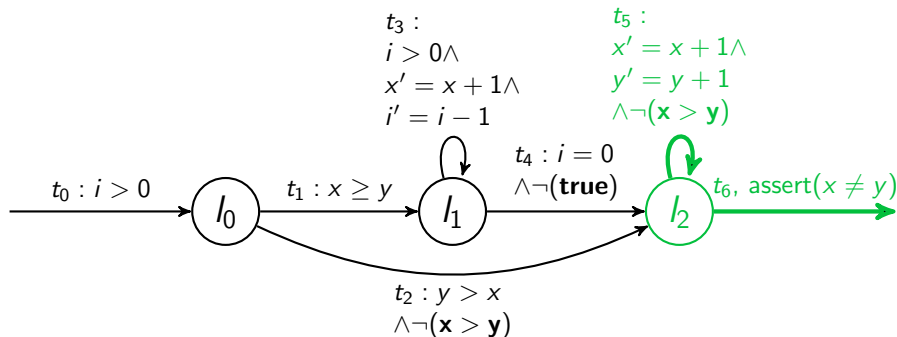
# Example execution



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

Paths are safe for  $y > x$

# Example execution



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

Program is safe for  $x \neq y$

# Conclusion

## We saw:

- 1 The exploration of multiple entry SCCs by CheckSafe
- 2 The effect of narrowing if a path to the SCC is proved safe and another could not be proved safe
- 3 The behavior of CheckSafe if a precondition with multiple conjunctions is found

## We didn't saw:

- 1 SCCs with multiple transitions (effects consecution conditions in Max-SMT and entry locations in Narrowing)