Efficient reuse of learnt information for control-flow oriented IC3 algorithms

Master's Thesis

September 23, 2016

Thomas Mertens

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Outline

Outline

Preliminaries

Definition (Cube and clause)

A cube is defined as conjunction of theory atoms (literals) over a subset $X \subseteq Var$. A cube is characterized by its set of literals, i.e. $literals(x = 0 \wedge y = 1) = \{x = 0, y = 1\}$. A clause is defined as negation of a cube.

Preliminaries

Definition (Cube and clause)

A cube is defined as conjunction of theory atoms (literals) over a subset $X \subseteq Var$. A cube is characterized by its set of literals, i.e. $literals(x = 0 \wedge y = 1) = \{x = 0, y = 1\}$. A clause is defined as negation of a cube.

Definition (Operations)

The operations of a Guarded Command Language are defined by the grammar

$$
\begin{aligned}
basic\_op &= assume\ b \mid var := exp \\
exp &= a \mid b \\
a &= var \mid a_1 + a_2 \mid a_1 - a_2 \mid\ ? \\
b &= var \mid b \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid a_1 \circ a_2\ with \circ \in \{=, <, >, \leq, \geq\} \\
op &= basic\_op \mid op_1; op_2 \mid op_1 \mid\mid op_2
\end{aligned}
$$

Preliminaries

Definition (Control Flow Automaton (CFA))

A CFA $\mathcal{A} = (L, G, l_0, l_E)$ consists of a set of locations $L = \{0, \ldots, n\}$ and edges in $G \subseteq L \times op \times L$ labeled with operations of a Guarded Command Language, an initial location l_0, and an error location l_E.

Preliminaries

Definition (Control Flow Automaton (CFA))

A CFA $\mathcal{A} = (L, G, l_0, l_E)$ consists of a set of locations $L = \{0, \ldots, n\}$ and edges in $G \subseteq L \times op \times L$ labeled with operations of a Guarded Command Language, an initial location l_0, and an error location l_E.

Definition (Transition formula [LNN15])

The transition formula between two locations l_1 and l_2 is defined as:

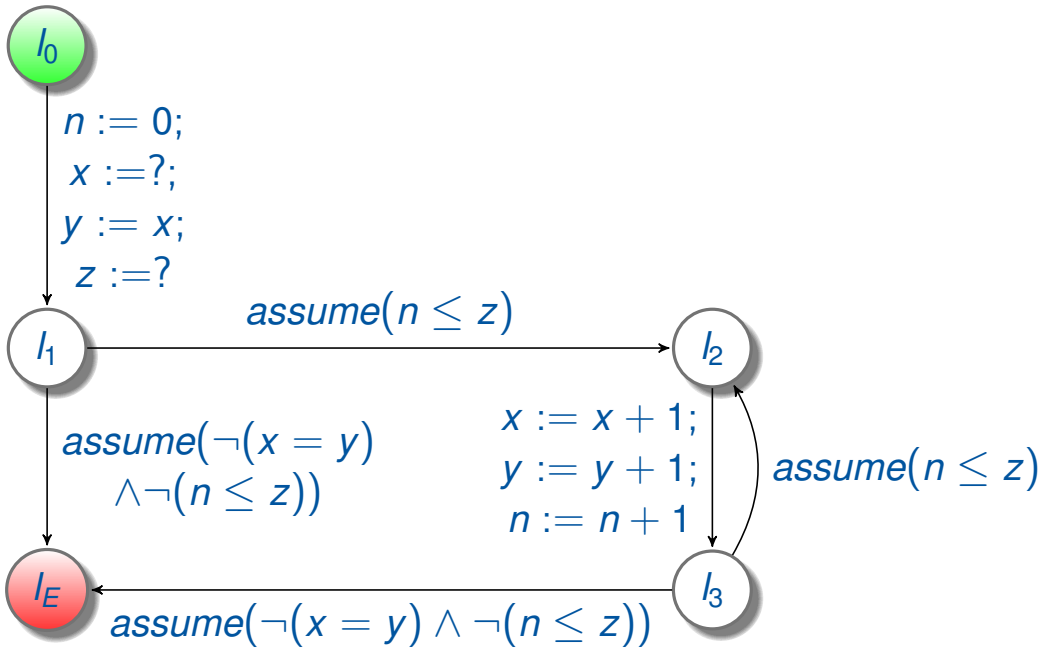
$$T_{l_1 \to l_2} = \begin{cases} Term(op) & , if (l_1, op, l_2) \in G \\ \texttt{false} & , otherwise \end{cases}$$

Preliminaries

```
int x, y, z;
int main(){
    x = ?;
    y = x;
    z = ?;
    n = 0;
    while(n <= z){
        n++;
        x++;
        y++;
    }
    assert(x == y);
    return 0;
}
```

Preliminaries

```
int x, y, z;
int main(){
    x = ?;
    y = x;
    z = ?;
    n = 0;
    while(n <= z){
        n++;
        x++;
        y++;
    }
    assert(x == y);
    return 0;
}
```



IC3CFA

Definition (Obligation)

An obligation is represented as three tuple (i, l, c) where i indicates the level of the obligation, $l \in L$ defines the location and c is a cube describing a data region.

IC3CFA

Definition (Obligation)

An obligation is represented as three tuple (i, l, c) where i indicates the level of the obligation, $l \in L$ defines the location and c is a cube describing a data region.

Definition (Result of transition functions)

The resulting variable evaluation after taking a CFA edge (l, op, l_s) is defined as

$$\begin{aligned} var' = &\bigwedge(x' = x \mid x \in \{Var \setminus assigned(op)\}) \\ &\wedge (\bigwedge(x' = exp \mid (x := exp) \in op)) \\ with\ assigned(op) = &\{x \mid (x := exp) \in op\} \end{aligned}$$

Outline

Information reuse

Motivation

- Incremental approach of IC3CFA
- Re-computations in further iterations

Information reuse

Motivation

- Incremental approach of IC3CFA
- Re-computations in further iterations

Four approaches

- Obligation reuse
- Improve termination
- Obligation skipping
- Pushing

Information reuse

Computed obligations

Information reuse

Computed obligations

iteration	$k = 1$	$k = 2$
Queue	$(1, l_1, c_1)$	$(1, l_2, c_2)$ $(2, l_1, c_1)$

Information reuse

Computed obligations

Queue			
			$(1, l_3, c_3)$
		$(1, l_2, c_2)$	$(2, l_2, c_2)$
	$(1, l_1, c_1)$	$(2, l_1, c_1)$	$(3, l_1, c_1)$
iteration	$k = 1$	$k = 2$	$k = 3$

Information reuse

Computed obligations

iteration	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
Queue	$(1, l_1, c_1)$	$(1, l_2, c_2)$ $(2, l_1, c_1)$	$(1, l_3, c_3)$ $(2, l_2, c_2)$ $(3, l_1, c_1)$	$(1, l_4, c_4)$ $(2, l_3, c_3)$ $(3, l_2, c_2)$ $(4, l_1, c_1)$	$(1, l_5, c_5)$ $(2, l_4, c_4)$ $(3, l_3, c_3)$ $(4, l_2, c_2)$ $(5, l_1, c_1)$

Information reuse

Computed obligations

Queue					
					$(1, l_5, c_5)$
				$(1, l_4, c_4)$	$(2, l_4, c_4)$
			$(1, l_3, c_3)$	$(2, l_3, c_3)$	$(3, l_3, c_3)$
		$(1, l_2, c_2)$	$(2, l_2, c_2)$	$(3, l_2, c_2)$	$(4, l_2, c_2)$
	$(1, l_1, c_1)$	$(2, l_1, c_1)$	$(3, l_1, c_1)$	$(4, l_1, c_1)$	$(5, l_1, c_1)$
iteration	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$

Definition (CTI)

$$wp(T_{l_p \to l_E}, true) \text{ s.t. } l_p \in pre(l_E)$$

Obligation reuse

Definition

The initial ObligationQueue for each inner loop is of the form:

$$Q_{init(k)} = \{(k, l_p, wp(T_{l_p \to l_E}, \texttt{true})) \mid (l_p, op, l_E) \in G\}$$

Obligation reuse

Definition

The initial ObligationQueue for each inner loop is of the form:

$$Q_{init(k)} = \{(k, l_p, wp(T_{l_p \to l_E}, \texttt{true})) \mid (l_p, op, l_E) \in G\}$$

- Extending reuse of CTI obligation

Obligation reuse

Definition

The initial ObligationQueue for each inner loop is of the form:

$$Q_{init(k)} = \{(k, l_p, wp(T_{l_p \to l_E}, \texttt{true})) \mid (l_p, op, l_E) \in G\}$$

- Extending reuse of CTI obligation
- Each obligation can be reused

Obligation reuse

Definition

The initial ObligationQueue for each inner loop is of the form:

$$Q_{init(k)} = \{(k, l_p, wp(T_{l_p \to l_E}, \texttt{true})) \mid (l_p, op, l_E) \in G\}$$

- Extending reuse of CTI obligation
- Each obligation can be reused
- Use computation of previous iteration

Obligation reuse

Definition

The initial ObligationQueue for each inner loop is of the form:

$$Q_{init(k)} = \{(k, l_p, wp(T_{l_p \to l_E}, \texttt{true})) \mid (l_p, op, l_E) \in G\}$$

- Extending reuse of CTI obligation
- Each obligation can be reused
- Use computation of previous iteration
- Start at lowest index

Obligation reuse

Definition

The initial ObligationQueue for each inner loop is of the form:

$$Q_{init(k)} = \{(k, l_p, wp(T_{l_p \to l_E}, \texttt{true})) \mid (l_p, op, l_E) \in G\}$$

- Extending reuse of CTI obligation
- Each obligation can be reused
- Use computation of previous iteration
- Start at lowest index
- Reduces predecessor computation

Obligation reuse

Definition

The initial ObligationQueue for each inner loop is of the form:

$$Q_{init(k)} = \{(k, l_p, wp(T_{l_p \to l_E}, \texttt{true})) \mid (l_p, op, l_E) \in G\}$$

- Extending reuse of CTI obligation
- Each obligation can be reused
- Use computation of previous iteration
- Start at lowest index
- Reduces predecessor computation

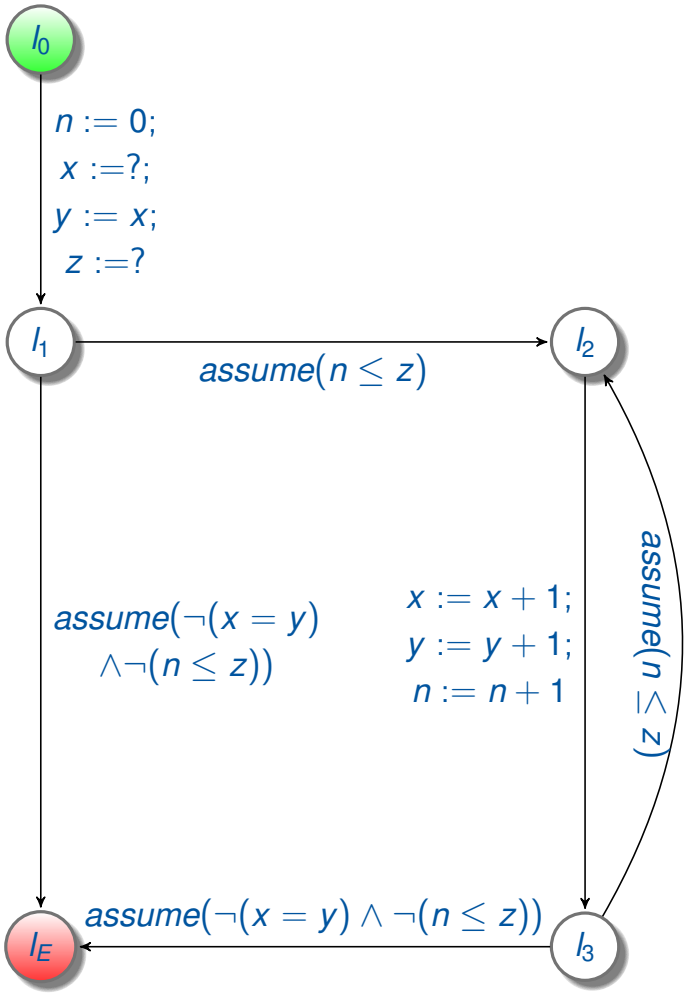
Lemma

Each $q \in Q_k$ would be computed in iteration k when starting the inner loop with $Q_{init(k)}$, i.e. $Q_k \subseteq seen(Q_{init(k)})$.

Obligation reuse - Example

$k = 2$
$(2, l_1, x \neq y)$
$(2, l_3, x \neq y \wedge z \leq n)$

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`	`false`	
1		$x = y$		`false`
2				

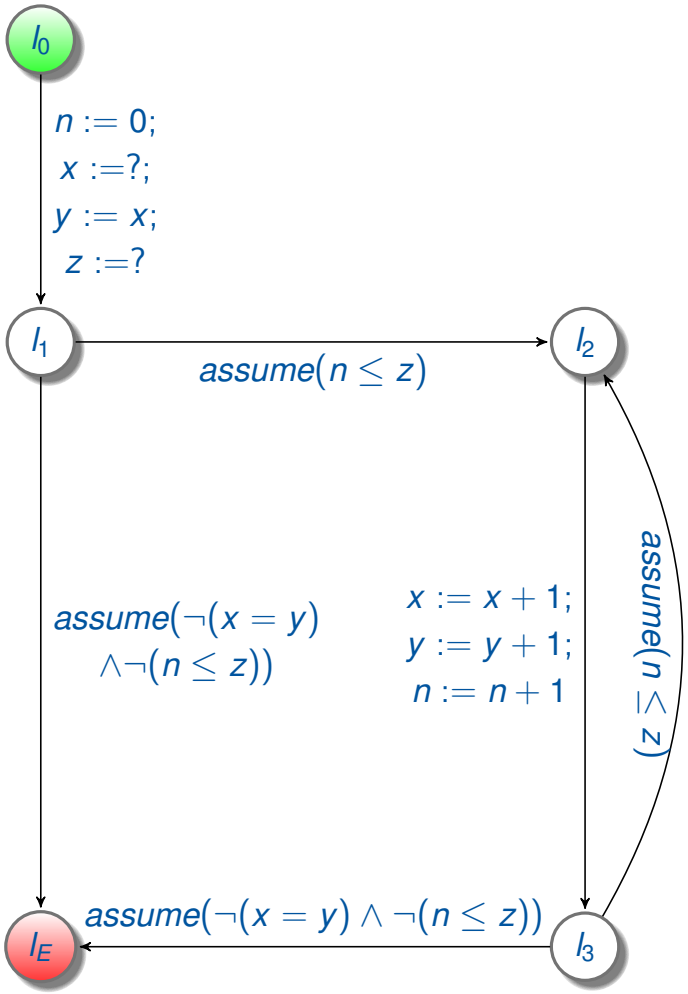


Obligation reuse - Example

$k = 2$
$(2, l_1, x \neq y)$
$(2, l_3, x \neq y \wedge z \leq n)$

$\text{SAT}(F_{(1,l_0)} \wedge T_{l_0 \to l_1} \wedge x' \neq y')$

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`	`false`	
1		$x = y$		`false`
2				

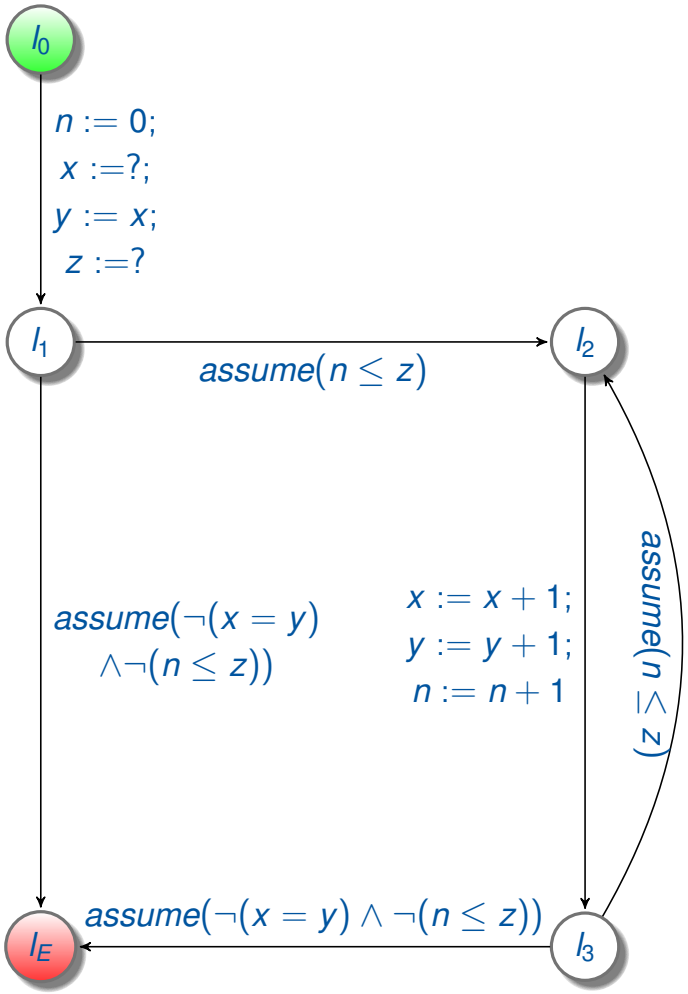


Obligation reuse - Example

$k = 2$
$(2, l_1, x \neq y)$
$(2, l_3, x \neq y \wedge z \leq n)$

SAT$(F_{(1,l_0)} \wedge T_{l_0 \to l_1} \wedge x' \neq y')$ ✗

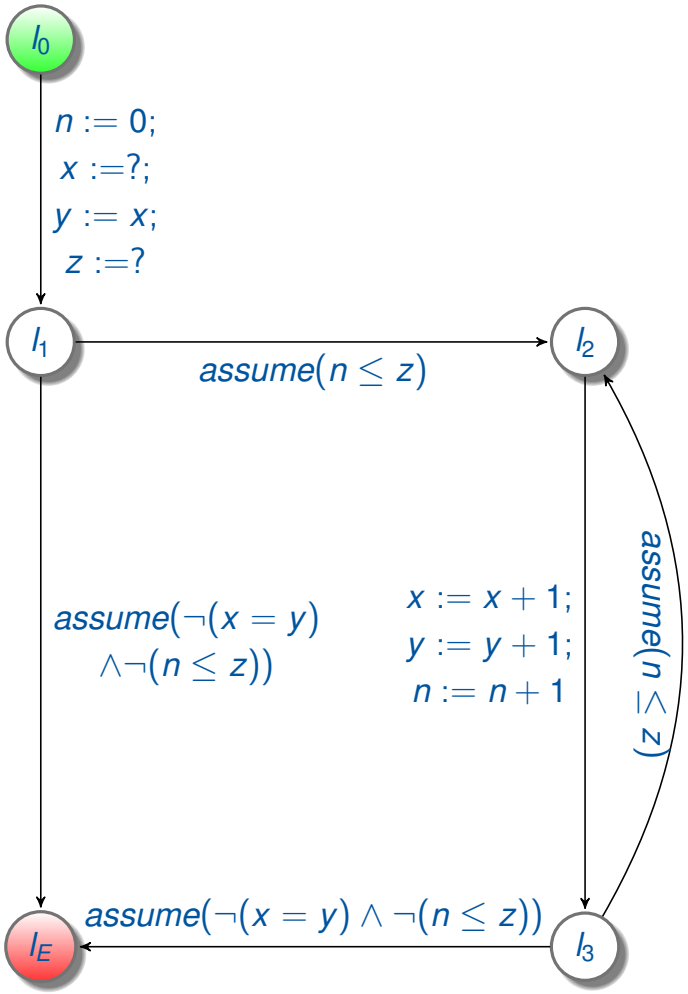
i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`	`false`	
1				`false`
2		$x = y$		



Obligation reuse - Example

$k = 2$
$(2, l_3, x \neq y \wedge z \leq n)$
$(2, l_1, x \neq y)$

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`	`false`	
1				`false`
2		$x = y$		

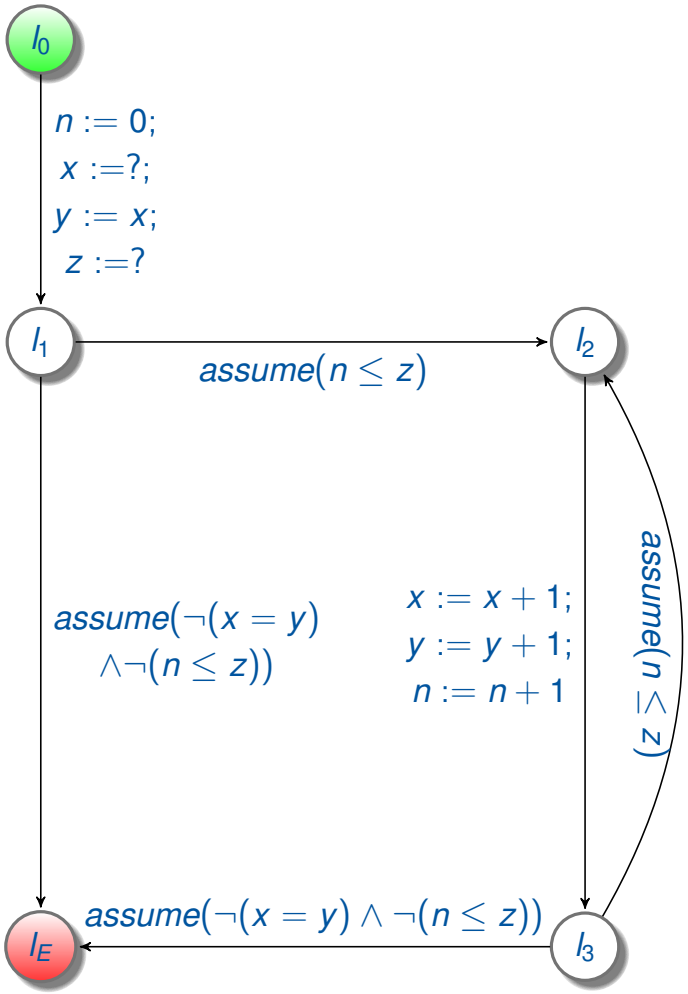


Obligation reuse - Example

$k = 2$
$(2, l_3, x \neq y \wedge z \leq n)$
$(2, l_1, x \neq y)$

$\text{SAT}(F_{(1,l_2)} \wedge T_{l_2 \to l_3} \wedge x' \neq y' \wedge z \leq n')$

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`	`false`	
1				`false`
2		$x = y$		

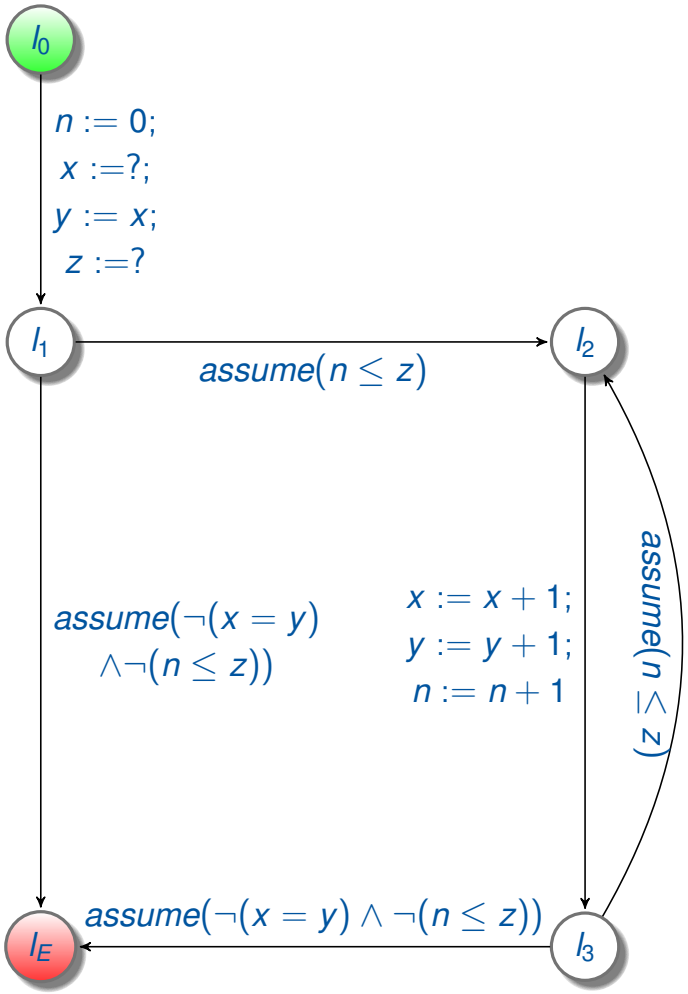


Obligation reuse - Example

$k = 2$
$(1, l_2, (z \leq i + 1 \wedge x + 1 \neq y + 1))$
$(2, l_3, x \neq y \wedge z \leq n)$
$(2, l_1, x \neq y)$

SAT$(F_{(1,l_2)} \wedge T_{l_2 \to l_3} \wedge x' \neq y' \wedge z \leq n')$ ✓

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`	`false`	
1				`false`
2		$x = y$		

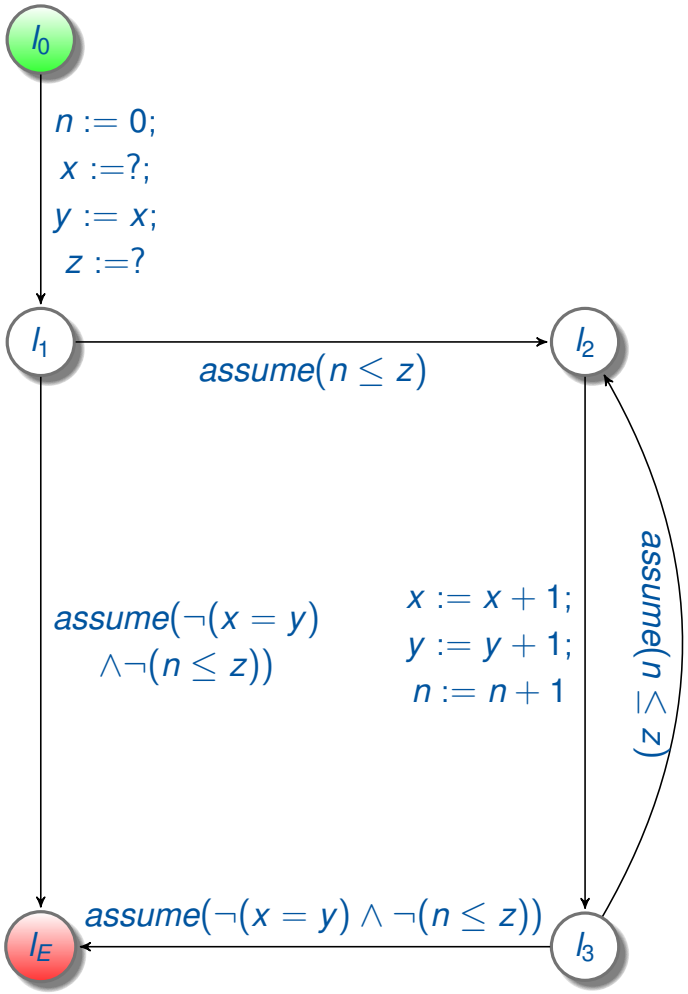


Obligation reuse - Example

$k = 2$
$(1, l_2, (z \leq n + 1 \wedge x + 1 \neq y + 1))$
$(2, l_3, x \neq y \wedge z \leq n)$
$(2, l_1, x \neq y)$

$\text{SAT}(F_{(0,l_1)} \wedge T_{l_1 \to l_2} \wedge z \leq n + 1 \wedge x + 1 \neq y + 1)$

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`	`false`	
1				`false`
2		$x = y$		

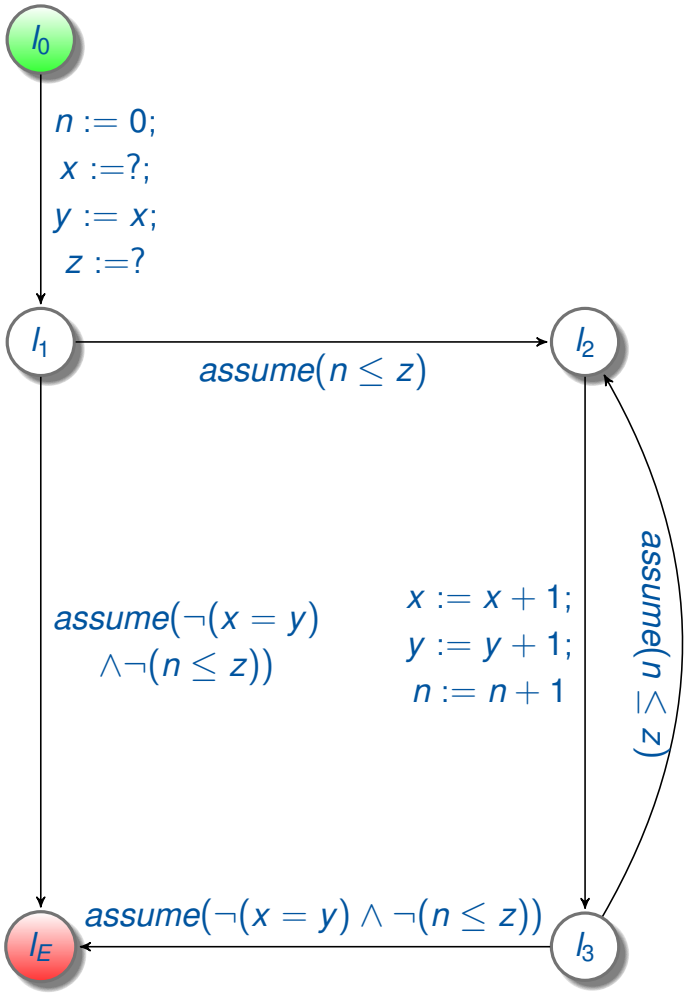


Obligation reuse - Example

$k = 2$
$(1, l_2, (z \leq n + 1 \wedge x + 1 \neq y + 1))$
$(2, l_3, x \neq y \wedge z \leq n)$
$(2, l_1, x \neq y)$

$\text{SAT}(F_{(0,l_1)} \wedge T_{l_1 \to l_2} \wedge z \leq n + 1 \wedge x + 1 \neq y + 1)$ ✗

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	`false`
2		$x = y$		

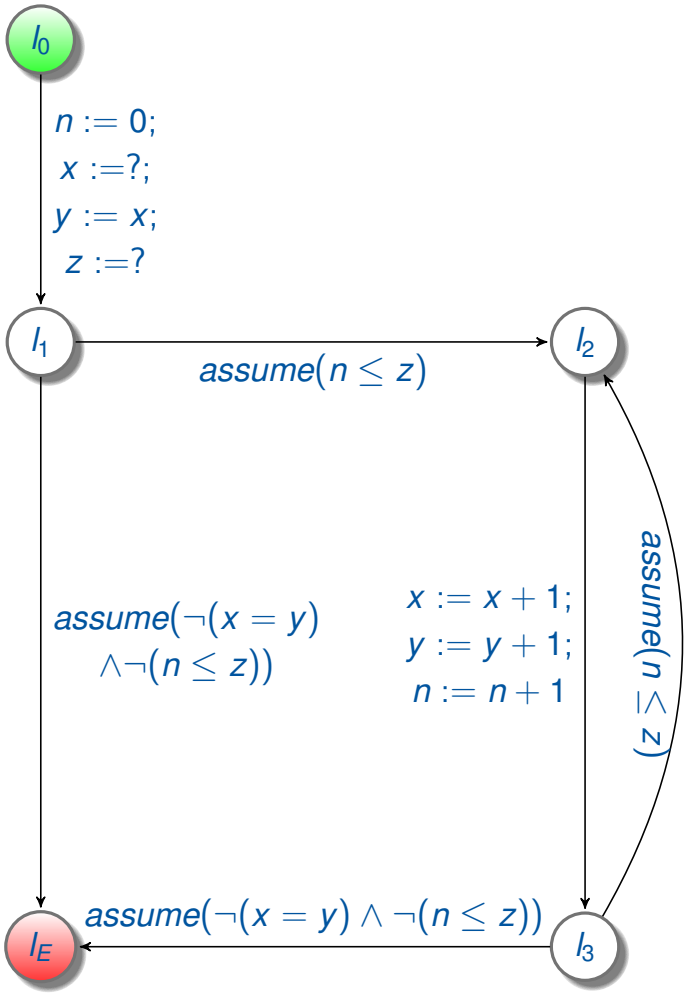


Obligation reuse - Example

$k = 2$
$(2, l_3, x \neq y \wedge z \leq n)$
$(1, l_2, (z \leq n + 1 \wedge x + 1 \neq y + 1))$
$(2, l_1, x \neq y)$

SAT$(F_{(1,l_2)} \wedge T_{l_2 \to l_3} \wedge x' \neq y' \wedge z \leq n')$

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	`false`
2		$x = y$		

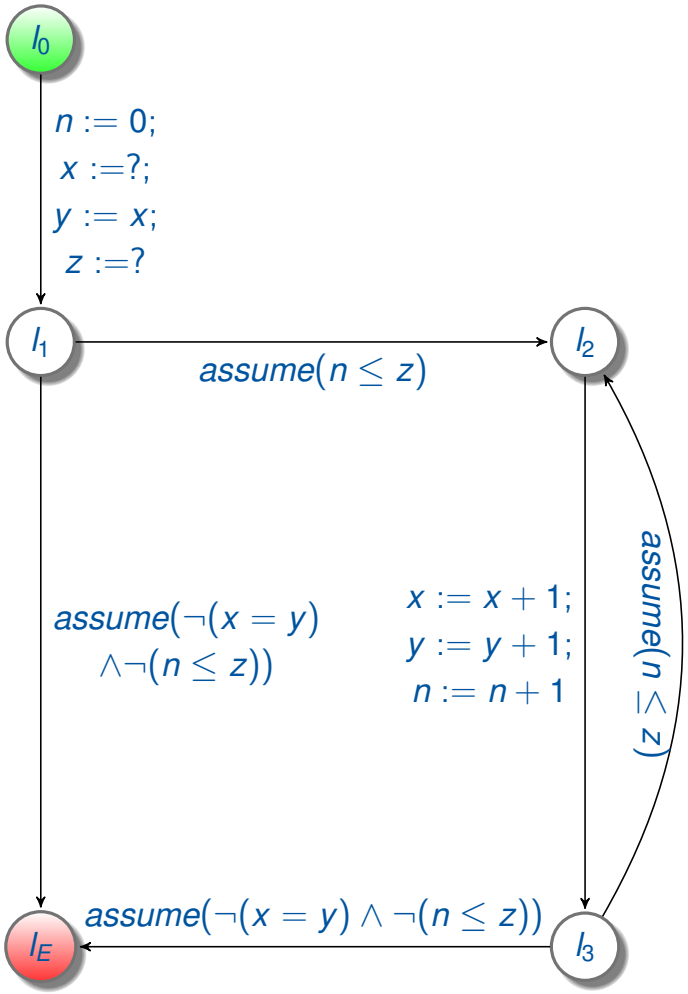


Obligation reuse - Example

$k = 2$
$(2, l_3, x \neq y \wedge z \leq n)$
$(1, l_2, (z \leq n + 1 \wedge x + 1 \neq y + 1))$
$(2, l_1, x \neq y)$

SAT$(F_{(1,l_2)} \wedge T_{l_2 \to l_3} \wedge x' \neq y' \wedge z \leq n')$ ✗

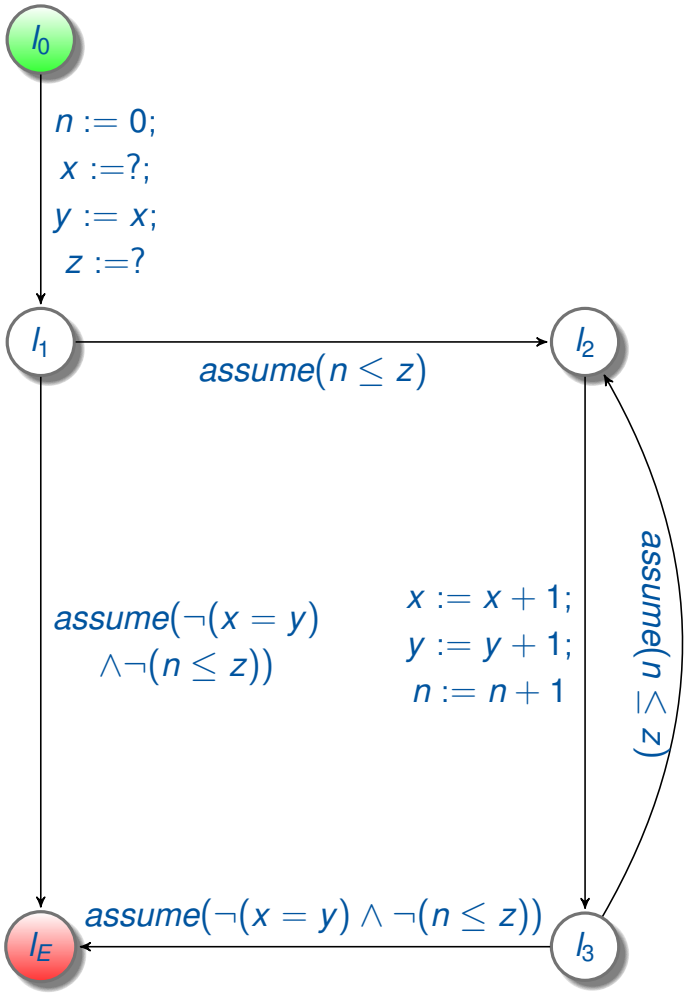
i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	
2		$x = y$		`false`



Obligation reuse - Example

$k = 2$
$(1, l_2, (z \leq n + 1 \wedge x + 1 \neq y + 1))$
$(2, l_1, x \neq y)$
$(2, l_3, x \neq y \wedge z \leq n)$

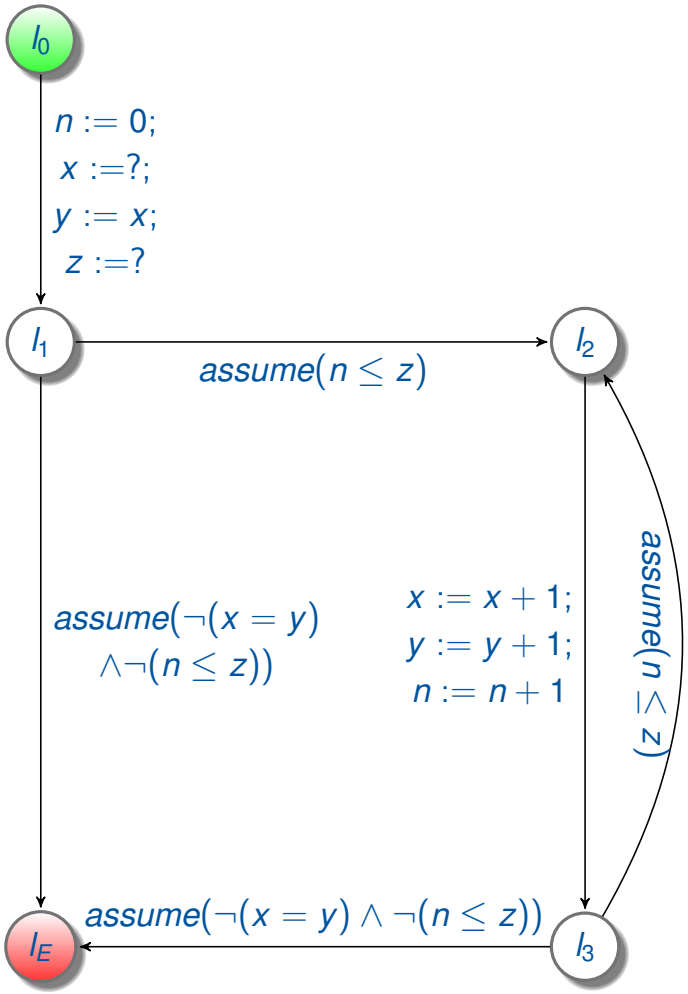
i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	
2		$x = y$		`false`



Obligation reuse - Example

$k = 3$
$(2, l_2, (z \leq n + 1 \wedge x + 1 \neq y + 1))$
$(3, l_1, x \neq y)$
$(3, l_3, x \neq y \wedge z \leq n)$

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	
2		$x = y$		`false`
3				

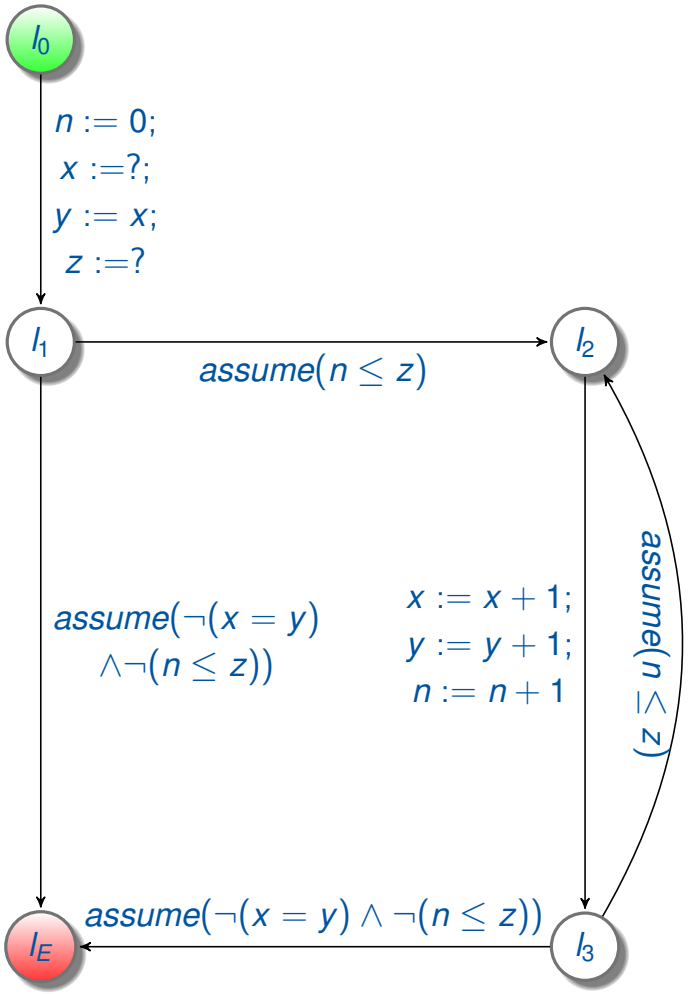


Obligation reuse - Example

$k = 3$
$(2, l_2, (z \leq n + 1 \wedge x + 1 \neq y + 1))$
$(3, l_1, x \neq y)$
$(3, l_3, x \neq y \wedge z \leq n)$

$\text{SAT}(F_{(1,l_1)} \wedge \wedge T_{l_1 \to l_2} \wedge z \leq n + 1 \wedge x + 1 \neq y + 1)$

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	
2		$x = y$		`false`
3				

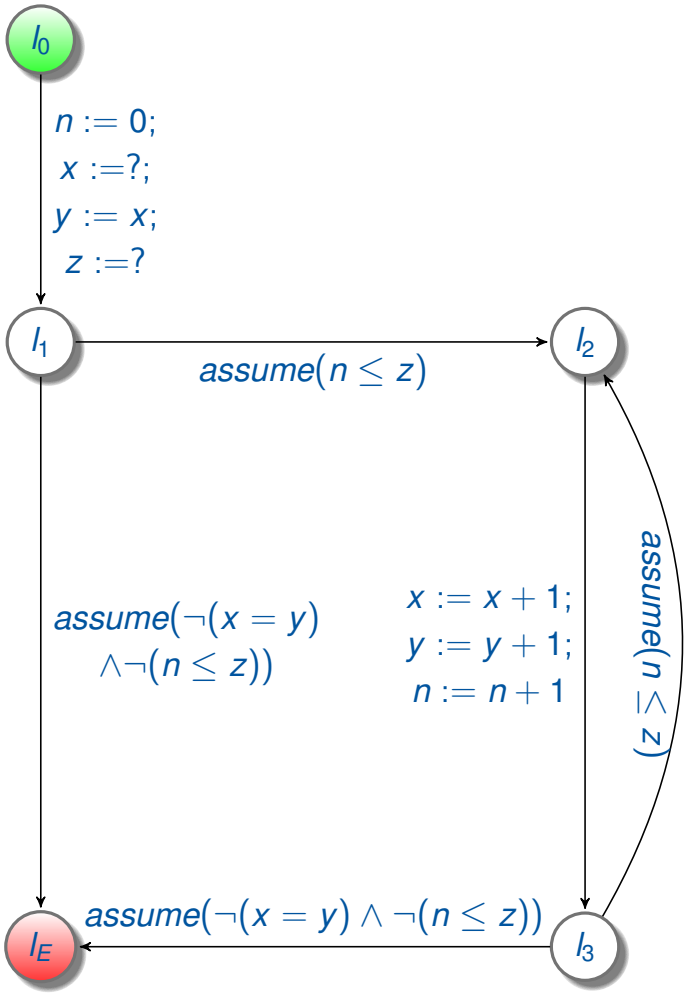


Obligation reuse - Example

$k = 3$
$(2, l_2, (z \leq n + 1 \wedge x + 1 \neq y + 1))$
$(3, l_1, x \neq y)$
$(3, l_3, x \neq y \wedge z \leq n)$

$\text{SAT}(F_{(1,l_1)} \wedge \wedge T_{l_1 \to l_2} \wedge z \leq n + 1 \wedge x + 1 \neq y + 1)$ ✗

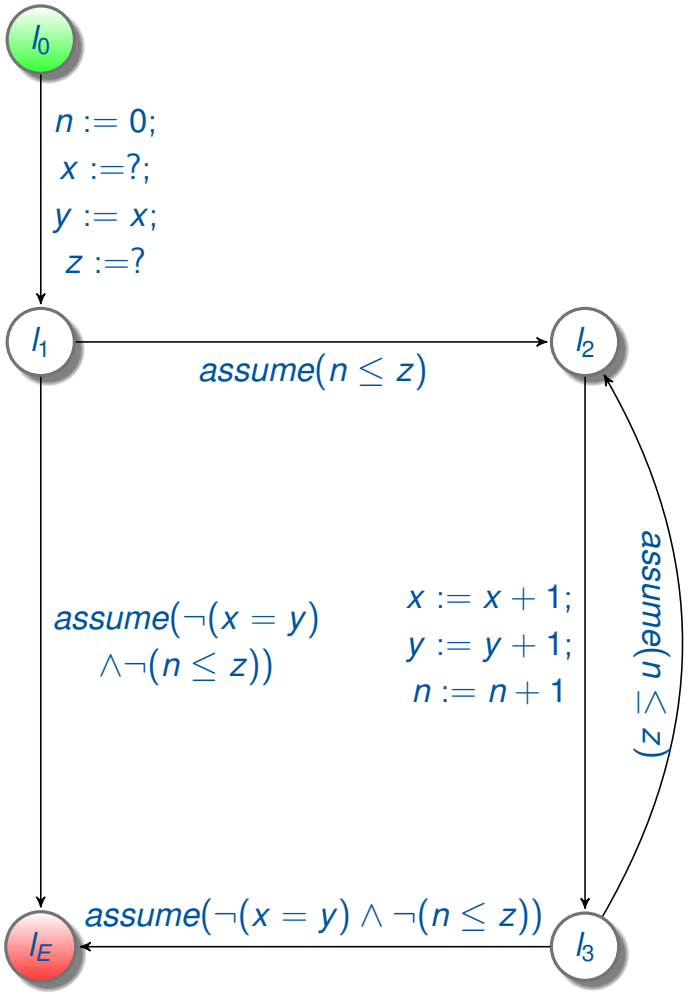
i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	
2		$x = y$	$x + 1 = y + 1$	`false`
3				



Obligation reuse - Example

$k = 3$
$(3, l_1, x \neq y)$
$(3, l_3, x \neq y \wedge z \leq n)$
$(2, l_2, (z \leq n + 1 \wedge x + 1 \neq y + 1))$

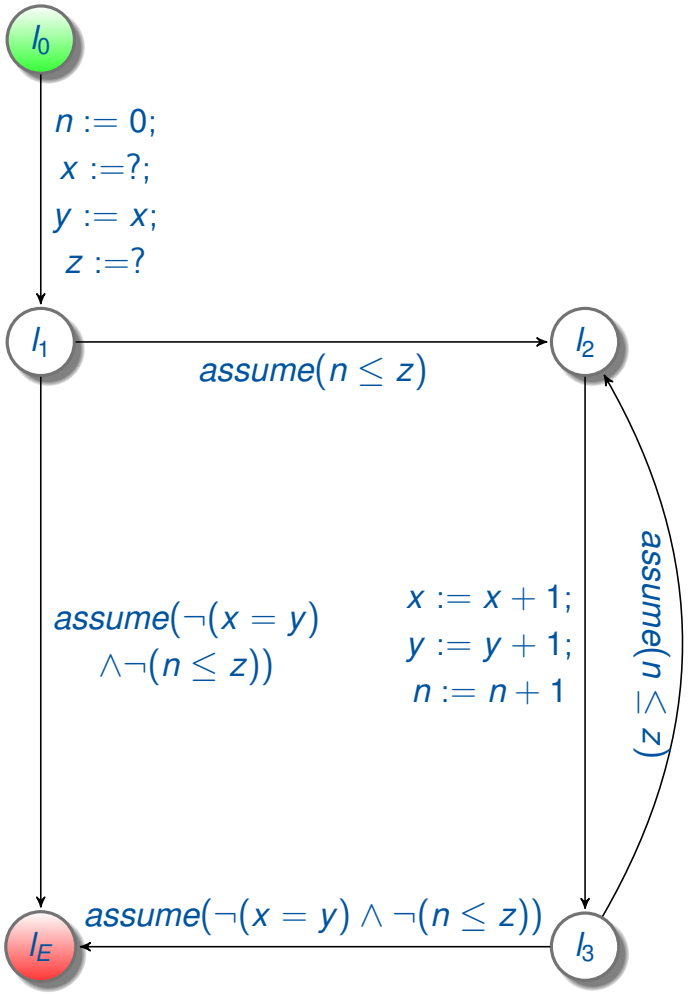
i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	
2		$x = y$	$x + 1 = y + 1$	`false`
3				



Obligation reuse - Example

$k = 3$
$(2, l_2, (z \leq n + 1 \wedge x + 1 \neq y + 1))$
$(3, l_1, x \neq y)$
$(3, l_3, x \neq y \wedge z \leq n)$

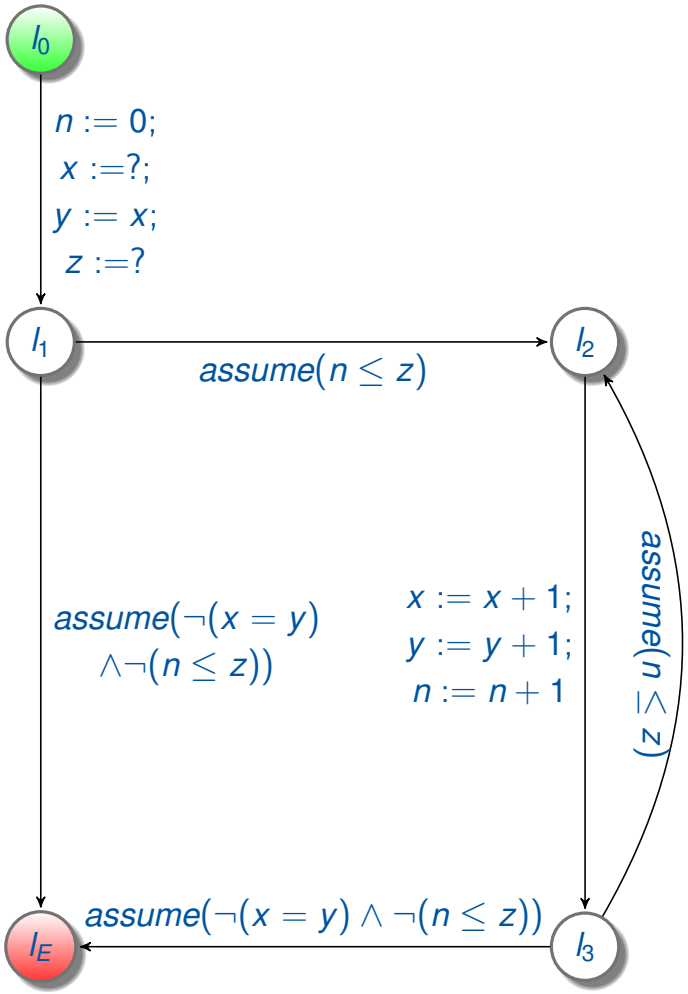
i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	
2			$x + 1 = y + 1$	`false`
3		$x = y$		$x = y$



Obligation reuse - Example

$k = 4$
$(3, l_2, (z \leq n + 1 \wedge x + 1 \neq y + 1))$
$(4, l_1, x \neq y)$
$(4, l_3, x \neq y \wedge z \leq n$

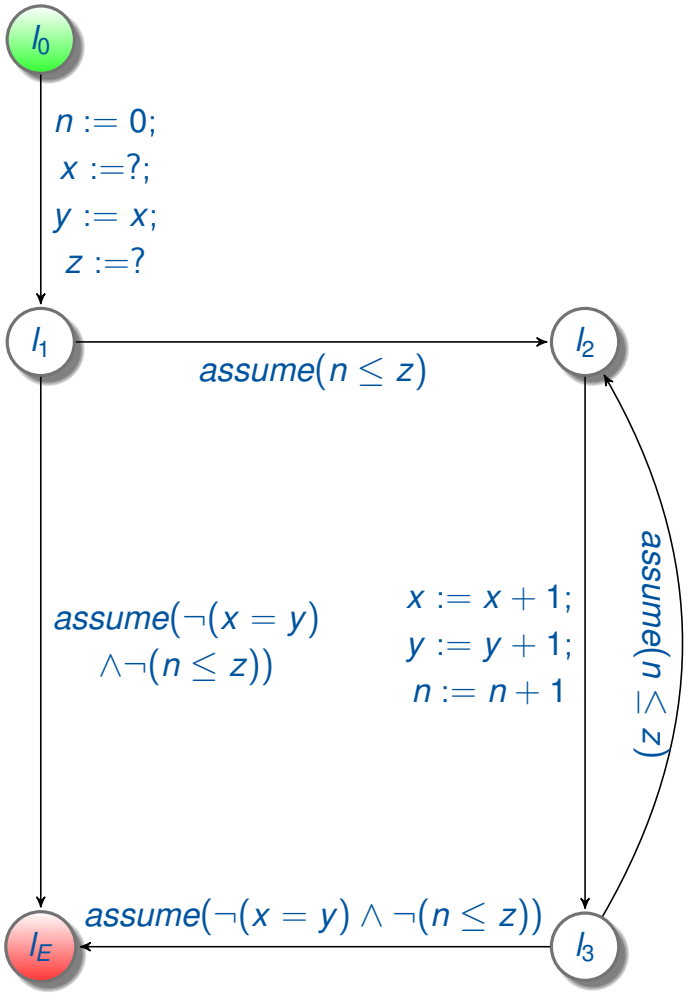
i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	
2				`false`
3			$x + 1 = y + 1$	
4		$x = y$		$x = y$



Obligation reuse - Example

$k = 5$
$(4, l_2, (z \leq n + 1 \wedge x + 1 \neq y + 1))$
$(5, l_1, x \neq y)$
$(5, l_3, x \neq y \wedge z \leq n$

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	
2				`false`
3				
4			$x + 1 = y + 1$	
5		$x = y$		$x = y$



Termination

Definition (Lifted ObligationQueue)

Let Q_k be an ObligationQueue at the beginning of iteration k with minimal element $q^k_{min} = (i, l, c)$. Q_k is lifted iff $i > 2$.

Termination

Definition (Lifted ObligationQueue)

Let Q_k be an ObligationQueue at the beginning of iteration k with minimal element $q^k_{min} = (i, l, c)$. Q_k is lifted iff $i > 2$.

Definition (Stable iteration)

Two consecutive iterations i and $i + 1$ are called stable, which is denoted by $Q_i \simeq Q_{i+1}$ iff

$$\{(j + 1, l, c) | (j, l, c) \in Q_i\} = Q_{i+1}$$

Elementary cycles

Definition (Elementary cycle [Joh75])

An elementary cycle is defined as cycle in which each location of the cycle is only visited once.

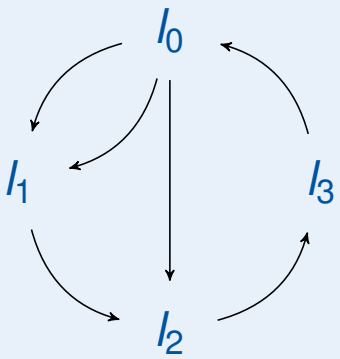
Elementary cycles

Definition (Elementary cycle [Joh75])

An elementary cycle is defined as cycle in which each location of the cycle is only visited once.

Example

Elementary cycles:

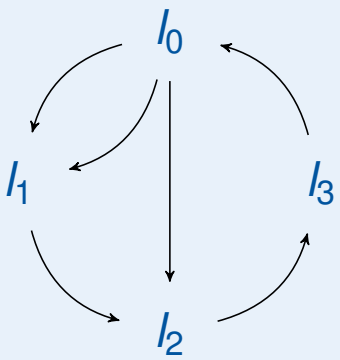


Elementary cycles

Definition (Elementary cycle [Joh75])

An elementary cycle is defined as cycle in which each location of the cycle is only visited once.

Example



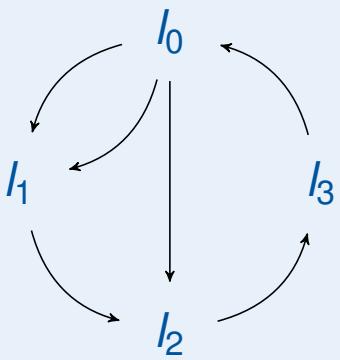
Elementary cycles:
$\{l_0, l_2, l_3\}$, $\{l_0, l_1, l_2, l_3\}$ *and* $\{l_0, l_1, l_2, l_3\}$

Elementary cycles

Definition (Elementary cycle [Joh75])

An elementary cycle is defined as cycle in which each location of the cycle is only visited once.

Example



Elementary cycles:
$\{l_0, l_2, l_3\}$, $\{l_0, l_1, l_2, l_3\}$ *and* $\{l_0, l_1, l_2, l_3\}$

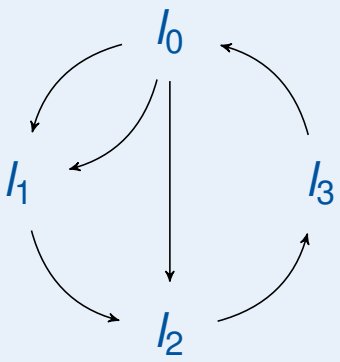
$sum(elementary\_cycles)$
$= 3 + 4 + 4 = 11$

Elementary cycles

Definition (Elementary cycle [Joh75])

An elementary cycle is defined as cycle in which each location of the cycle is only visited once.

Example



Elementary cycles:
$\{l_0, l_2, l_3\}$, $\{l_0, l_1, l_2, l_3\}$ *and* $\{l_0, l_1, l_2, l_3\}$

$sum(elementary\_cycles)$
$= 3 + 4 + 4 = 11$

$sum(unique\_elementary\_cycles)$
$= 3 + 4 = 7$

Termination

Lemma (New Termination Criterion)

IC3CFA can terminate and return `true` *if the number of stable iterations is greater than the sum of the length of all elementary cycles in the CFA.*

Termination

Lemma (New Termination Criterion)

IC3CFA can terminate and return `true` *if the number of stable iterations is greater than the sum of the length of all elementary cycles in the CFA.*

Lemma (Greedy Termination Criterion)

IC3CFA can terminate and return `true` *if the number of stable iterations is greater than the sum of the length of all unique elementary cycles in the CFA.*

Obligation skipping

Obligation skipping

- Caching too much is not possible due to limited cache size

Obligation skipping

- Caching too much is not possible due to limited cache size
- Cube generalization is complex

Obligation skipping

- Caching too much is not possible due to limited cache size
- Cube generalization is complex

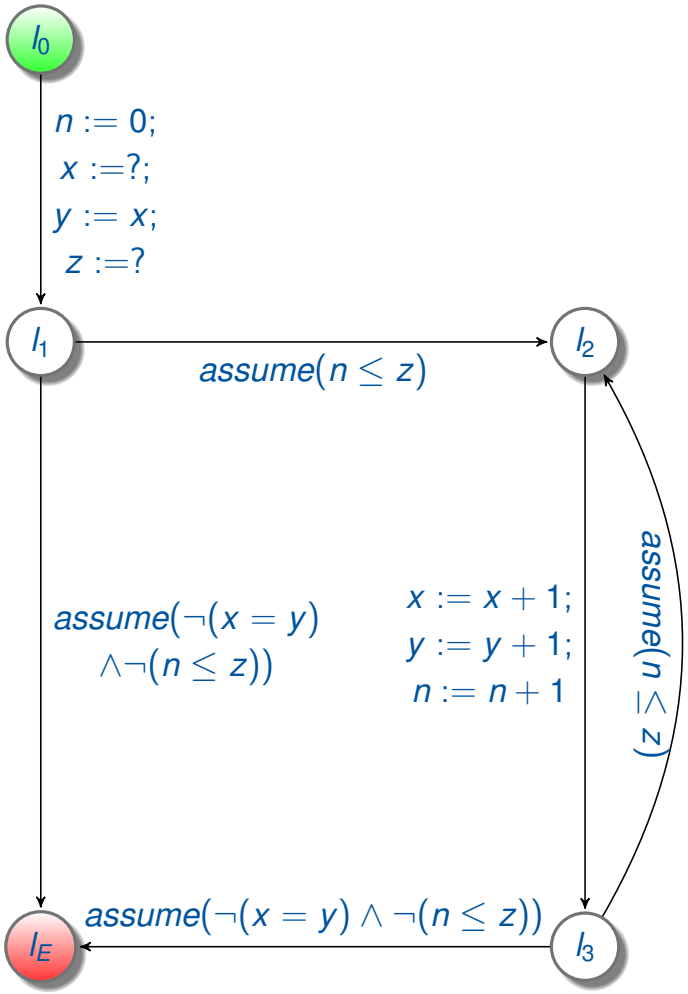
Lemma (Blocking without SAT-query)

For any $q^j \in Q_k$ with $q^j = (i, l, c)$, s.t. $(i-1, l, c) \in seen(Q_{init(k-1)})$, the generalization of c can be blocked at $F_{(i,l)}$ if $\forall l_p \in pre(l)$: $F_{\Delta(i-2,l_p)} = \emptyset$ without causing a SAT-query.

Obligation skipping

$k = 4$
$(3, l_2, (z \leq i + 1 \wedge x + 1 \neq y + 1))$
$(4, l_1, x \neq y)$
$(4, l_3, x \neq y \wedge z \leq i)$

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	
2			$x + 1 = y + 1$	`false`
3		$x = y$		$x = y$
4				

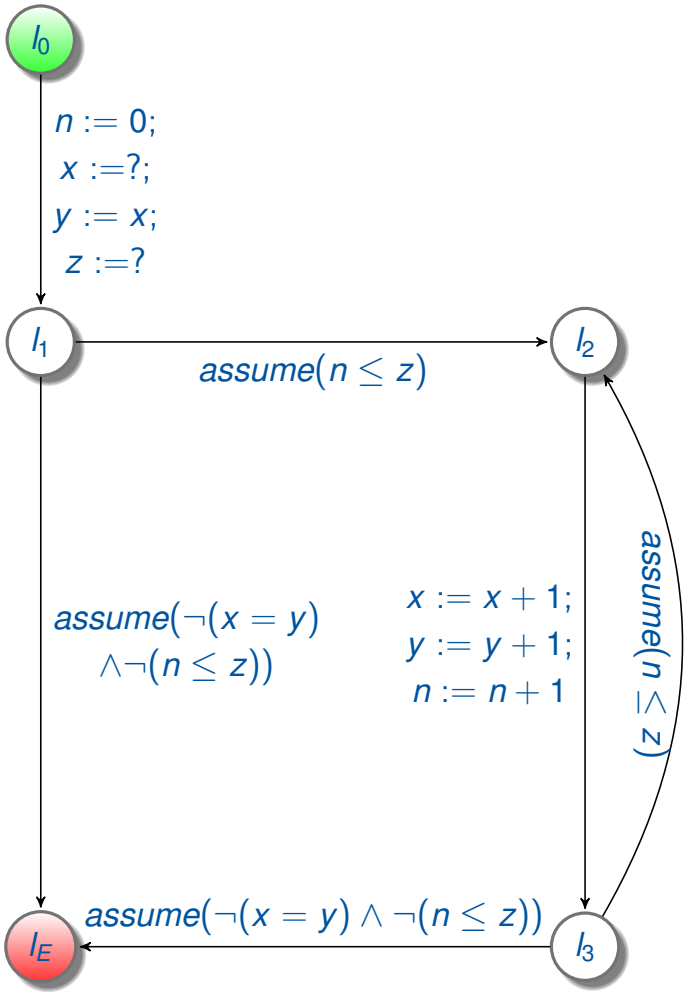


Obligation skipping

$k = 4$
$(3, l_2, (z \leq i + 1 \wedge x + 1 \neq y + 1))$
$(4, l_1, x \neq y)$
$(4, l_3, x \neq y \wedge z \leq i)$

$\forall l_p \in pre(l_2) : F_{\Delta(1,l_p)} = \emptyset$

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	
2			$x + 1 = y + 1$	`false`
3		$x = y$		$x = y$
4				

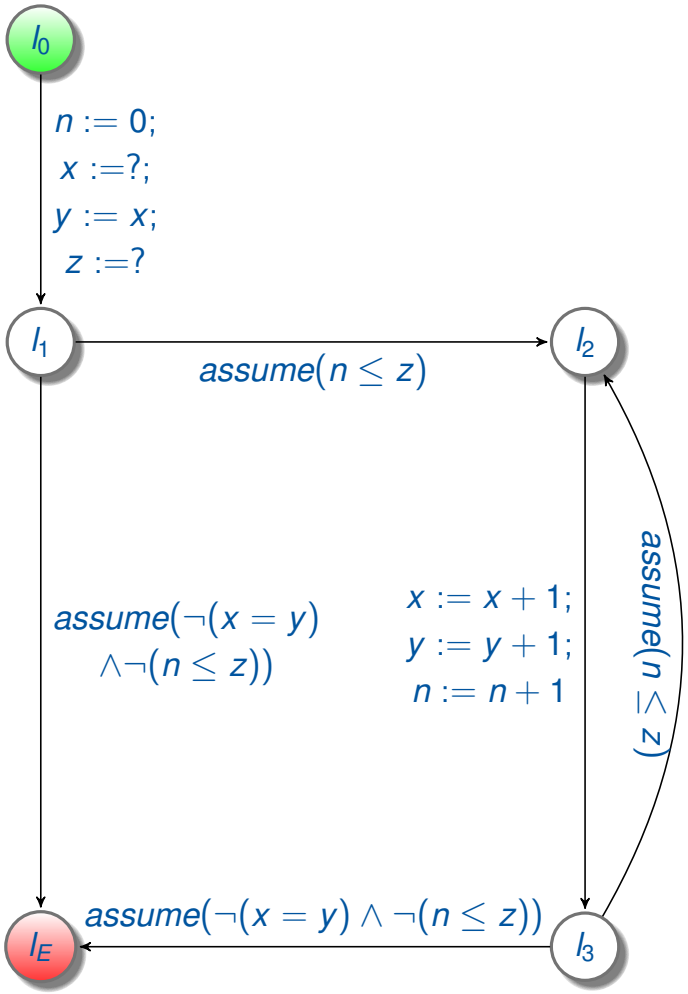


Obligation skipping

$k = 4$
$(3, l_2, (z \leq i + 1 \wedge x + 1 \neq y + 1))$
$(4, l_1, x \neq y)$
$(4, l_3, x \neq y \wedge z \leq i)$

$\forall l_p \in pre(l_2) : F_{\Delta(1,l_p)} = \emptyset \Leftrightarrow F_{\Delta(1,l_1)} = \emptyset$ and
$F_{\Delta(1,l_3)} = \emptyset \Rightarrow$ ✓

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	
2			$x + 1 = y + 1$	`false`
3		$x = y$		$x = y$
4				

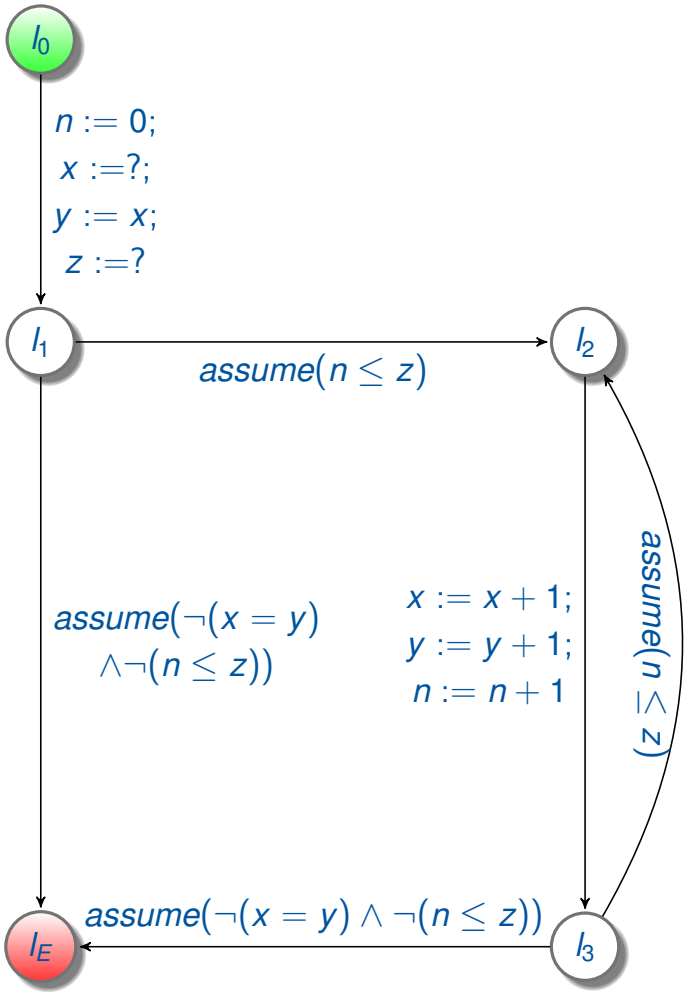


Obligation skipping

$k = 4$
$(3, l_2, (z \leq i + 1 \wedge x + 1 \neq y + 1))$
$(4, l_1, x \neq y)$
$(4, l_3, x \neq y \wedge z \leq i)$

$\forall l_p \in pre(l_2) : F_{\Delta(1,l_p)} = \emptyset \Leftrightarrow F_{\Delta(1,l_1)} = \emptyset$ and
$F_{\Delta(1,l_3)} = \emptyset \Rightarrow$ ✓

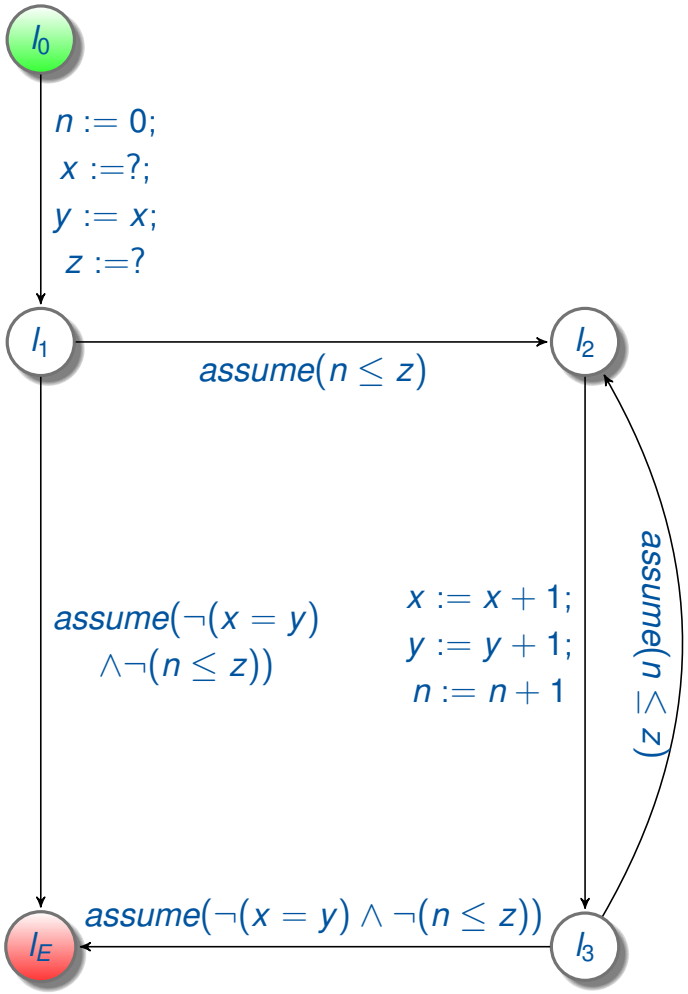
i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	
2				`false`
3		$x = y$	$x + 1 = y + 1$	$x = y$
4				



Obligation skipping

$k = 4$
$(4, l_1, x \neq y)$
$(4, l_3, x \neq y \wedge z \leq i)$
$(3, l_2, (z \leq i + 1 \wedge x + 1 \neq y + 1))$

i : \ l :	l_0	l_1	l_2	l_3
0	`true`	`false`		
1			`false`	
2				`false`
3		$x = y$	$x + 1 = y + 1$	$x = y$
4				



Pushing

- Use learnt information in further iteration
- Push cubes to next level
- Increase inductiveness

Pushing

- Use learnt information in further iteration
- Push cubes to next level
- Increase inductiveness

Definition (Pushing on edges)

c pushable to $F_{(i+1,l)}$ *on edge* $e = (l_p, op, l) \in G \Leftrightarrow F_{(i,l_p)} \wedge T_{l_p \to l} \wedge c'$ *is unsatisfiable*

Pushing

- Use learnt information in further iteration
- Push cubes to next level
- Increase inductiveness

Definition (Pushing on edges)

c pushable to $F_{(i+1,l)}$ *on edge* $e = (l_p, op, l) \in G \Leftrightarrow F_{(i,l_p)} \wedge T_{l_p \to l} \wedge c'$ *is unsatisfiable*

Lemma (Pushing in IC3CFA)

c is pushable to $F_{(i+1,l)} \Leftrightarrow \forall e \in pre\_edges(l)$ *c is pushable to* $F_{(i+1,l)}$ *on edge e*

Pushing

- Pushing causes additional SAT-queries

Pushing

- Pushing causes additional SAT-queries
- Some push request are complex and cause high solver times

Pushing

- Pushing causes additional SAT-queries
- Some push request are complex and cause high solver times
- Choice statements are expensive in queries checking unsatisfiability

Pushing

- Pushing causes additional SAT-queries
- Some push request are complex and cause high solver times
- Choice statements are expensive in queries checking unsatisfiability
- Introduced filters to reduce complexity of push requests

Pushing

- Pushing causes additional SAT-queries
- Some push request are complex and cause high solver times
- Choice statements are expensive in queries checking unsatisfiability
- Introduced filters to reduce complexity of push requests

Pushing

- Pushing causes additional SAT-queries
- Some push request are complex and cause high solver times
- Choice statements are expensive in queries checking unsatisfiability
- Introduced filters to reduce complexity of push requests

Lemma (Pushing without SAT-query)

A cube c can be pushed at $F_{i+1,l}$ without querying the SAT-solver if $pre\_edges(l) = \{(l_p, op, l)\}$ and $assigned(op) \cap variables(c) = \emptyset$.

Pushing

Proof (Proof Pushing without SAT-query)

Let $F_{(i,l_p)}$ be an arbitrary frame, c be a cube, $T_{l_p \to l}$ the translation relation with (1) $pre\_edges(l) = \{(l_p, op, l)\}$ and (2) $assigned(op) \cap variables(c) = \emptyset$.

$$c \text{ is pushable to } F_{(i+1,l)} \Leftrightarrow \forall l_p \in pre(l): \left(\bigvee (F_{(i,l_p)} \wedge T_{l_p \to l}) \wedge c'\right) \text{ is unsatisfiable}$$

Pushing

Proof (Proof Pushing without SAT-query)

Let $F_{(i,l_p)}$ be an arbitrary frame, c be a cube, $T_{l_p \to l}$ the translation relation with (1) $pre\_edges(l) = \{(l_p, op, l)\}$ and (2) $assigned(op) \cap variables(c) = \emptyset$.

$$c \text{ is pushable to } F_{(i+1,l)} \Leftrightarrow \forall l_p \in pre(l): \left(\bigvee (F_{(i,l_p)} \wedge T_{l_p \to l}) \wedge c'\right) \text{ is unsatisfiable}$$

$$\overset{(1)}{\Rightarrow} F_{i,l_p} \wedge T_{l_p \to l} \wedge c' \text{ is unsatisfiable}$$

Pushing

Proof (Proof Pushing without SAT-query)

Let $F_{(i,l_p)}$ be an arbitrary frame, c be a cube, $T_{l_p \to l}$ the translation relation with (1) $pre\_edges(l) = \{(l_p, op, l)\}$ and (2) $assigned(op) \cap variables(c) = \emptyset$.

$$c \text{ is pushable to } F_{(i+1,l)} \Leftrightarrow \forall l_p \in pre(l): \left(\bigvee (F_{(i,l_p)} \wedge T_{l_p \to l}) \wedge c'\right) \text{ is unsatisfiable}$$

$$\overset{(1)}{\Rightarrow} F_{i,l_p} \wedge T_{l_p \to l} \wedge c' \text{ is unsatisfiable}$$

$$\overset{(2)}{\Rightarrow} F_{i,l_p} \wedge T_{l_p \to l} \wedge c \text{ is unsatisfiable}$$

Pushing

Proof (Proof Pushing without SAT-query)

Let $F_{(i,l_p)}$ be an arbitrary frame, c be a cube, $T_{l_p \to l}$ the translation relation with (1) $pre\_edges(l) = \{(l_p, op, l)\}$ and (2) $assigned(op) \cap variables(c) = \emptyset$.

$$c \text{ is pushable to } F_{(i+1,l)} \Leftrightarrow \forall l_p \in pre(l): \left(\bigvee (F_{(i,l_p)} \wedge T_{l_p \to l}) \wedge c'\right) \text{ is unsatisfiable}$$

$$\overset{(1)}{\Rightarrow} F_{i,l_p} \wedge T_{l_p \to l} \wedge c' \text{ is unsatisfiable}$$

$$\overset{(2)}{\Rightarrow} F_{i,l_p} \wedge T_{l_p \to l} \wedge c \text{ is unsatisfiable}$$

$$\neg c \in F_{i,l_p} \text{ because } l \in succ(l_p)$$

Pushing

Proof (Proof Pushing without SAT-query)

Let $F_{(i,l_p)}$ be an arbitrary frame, c be a cube, $T_{l_p \to l}$ the translation relation with (1) $pre\_edges(l) = \{(l_p, op, l)\}$ and (2) $assigned(op) \cap variables(c) = \emptyset$.

$$c \text{ is pushable to } F_{(i+1,l)} \Leftrightarrow \forall l_p \in pre(l): \left(\bigvee (F_{(i,l_p)} \wedge T_{l_p \to l}) \wedge c'\right) \text{ is unsatisfiable}$$

$$\overset{(1)}{\Rightarrow} F_{i,l_p} \wedge T_{l_p \to l} \wedge c' \text{ is unsatisfiable}$$

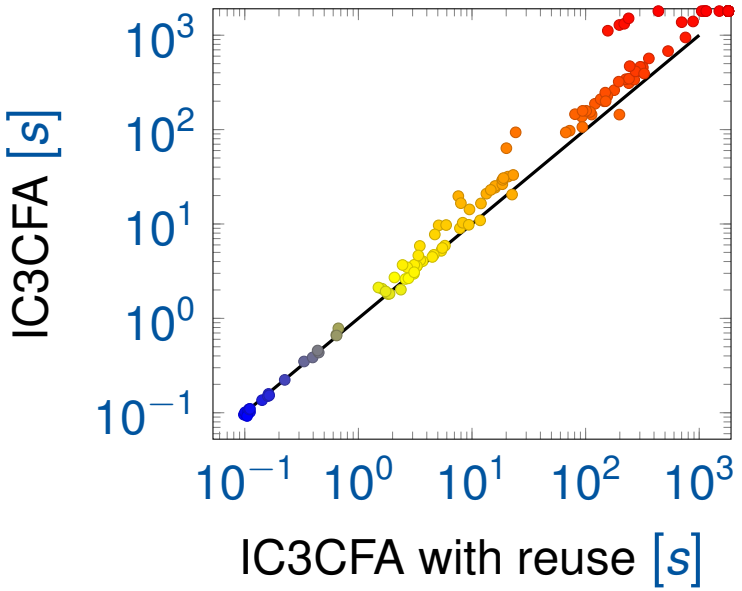
$$\overset{(2)}{\Rightarrow} F_{i,l_p} \wedge T_{l_p \to l} \wedge c \text{ is unsatisfiable}$$

$$\neg c \in F_{i,l_p} \text{ because } l \in succ(l_p)$$

$$\Rightarrow \text{ query is unsatisfiable}$$

Outline

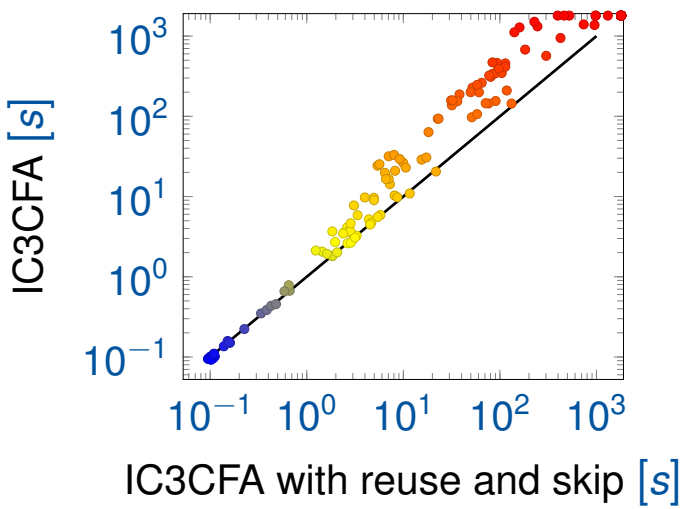
Obligation reuse

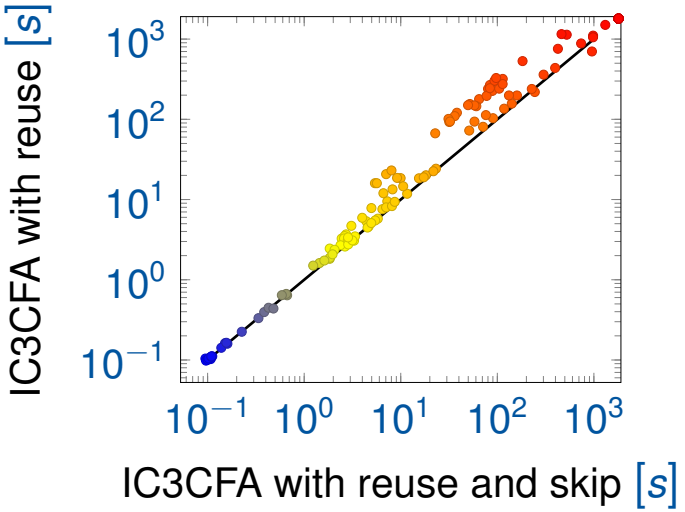


tool	score	solved	t solved	memory
IC3CFA	182	110/150	18,170	20,866
IC3CFA with reuse	193	116/150	16,132	16,264

Evaluation

Obligation skipping

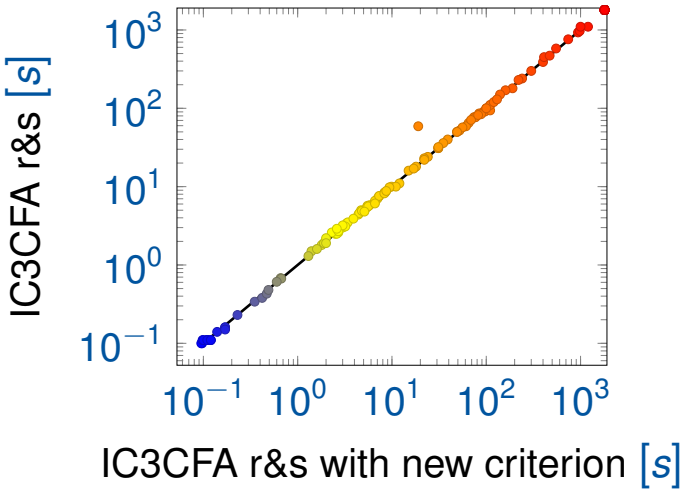




tool	score	solved	t solved	memory
IC3CFA	182	110/150	18,170	20,866
IC3CFA with reuse	193	116/150	16,132	16,264
IC3CFA with reuse and skip	193	116/150	10,407	14,986

Evaluation

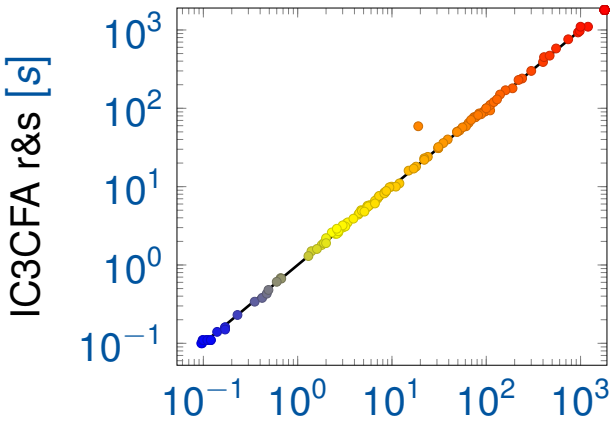
Termination

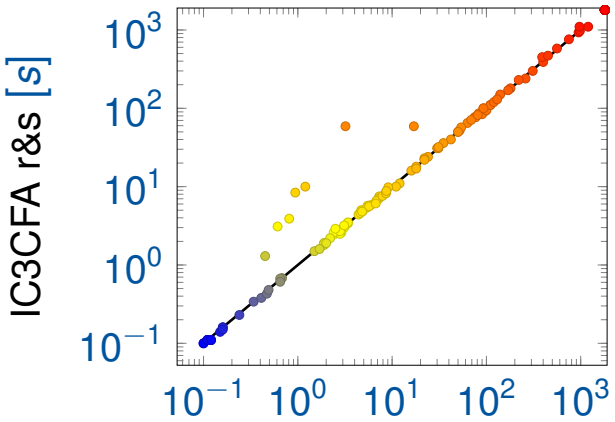


tool	score	solved	t solved	memory	# SAT	# k
IC3CFA r&s	193	116/150	10,407	14,986	408,194	3,887
IC3CFA r&s new	193	116/150	10,335	15,129	404,926	3,862
IC3CFA r&s greedy	193	116/150	10,176	14,953	394,872	3,727

Evaluation

Termination



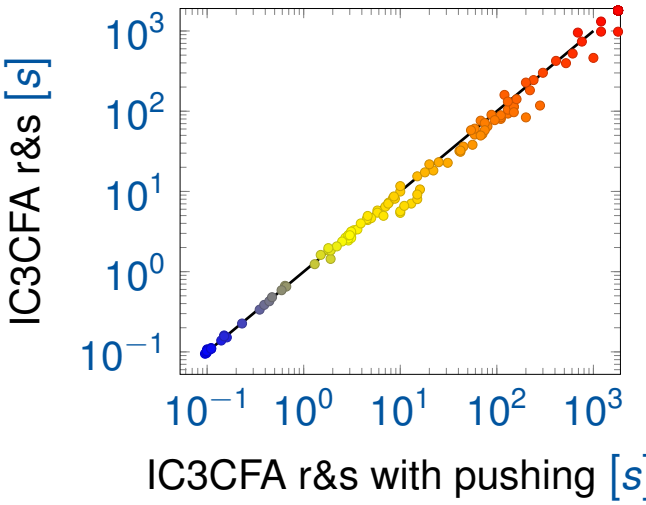


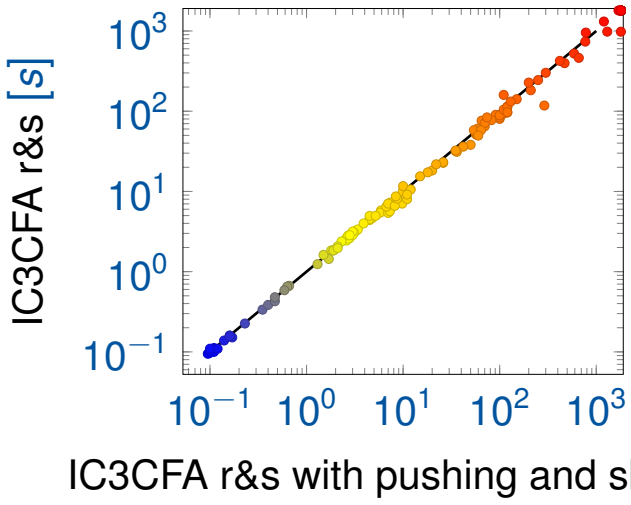
tool	score	solved	t solved	memory	# SAT	# k
IC3CFA r&s	193	116/150	10,407	14,986	408,194	3,887
IC3CFA r&s new	193	116/150	10,335	15,129	404,926	3,862
IC3CFA r&s greedy	193	116/150	10,176	14,953	394,872	3,727

Pushing

filter	checked	successful	rate
NONE	3.451.204	2.865.003	83%
ASSIGN	2.353.298	2.076.557	88%
PREDECESSOR	1.000.217	983.060	98%
RESTRICTIVE	799.974	799.974	100%

Evaluation





tool	score	solved	t solved	memory	# SAT
IC3CFA r&s	193	116/150	10,407	14,986	408,194
IC3CFA r&s with pushing	194	116/150	12,776	18,751	576,036
IC3CFA r&s with pushing & skip	194	116/150	11,916	15,733	396,158

Outline

Conclusion

Conclusion

- Implemented four approaches for information reuse

Conclusion

- Implemented four approaches for information reuse
- Pushing improves the original algorithm but not the lifted version

Conclusion

- Implemented four approaches for information reuse
- Pushing improves the original algorithm but not the lifted version
- New Termination criteria affect only some programs

Conclusion

- Implemented four approaches for information reuse
- Pushing improves the original algorithm but not the lifted version
- New Termination criteria affect only some programs
- Obligation skipping yields significant time reduction

Conclusion

- Implemented four approaches for information reuse
- Pushing improves the original algorithm but not the lifted version
- New Termination criteria affect only some programs
- Obligation skipping yields significant time reduction
- Best improvements achieved by obligation reuse

References I

- Donald B. Johnson, *Finding all the elementary circuits of a directed graph*, SIAM J. Comput. **4** (1975), no. 1, 77–84.
- Tim Lange, Martin R. Neuhäußer, and Thomas Noll, *IC3 software model checking on control flow automata*, Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, September 27-30, 2015., 2015, pp. 97–104.