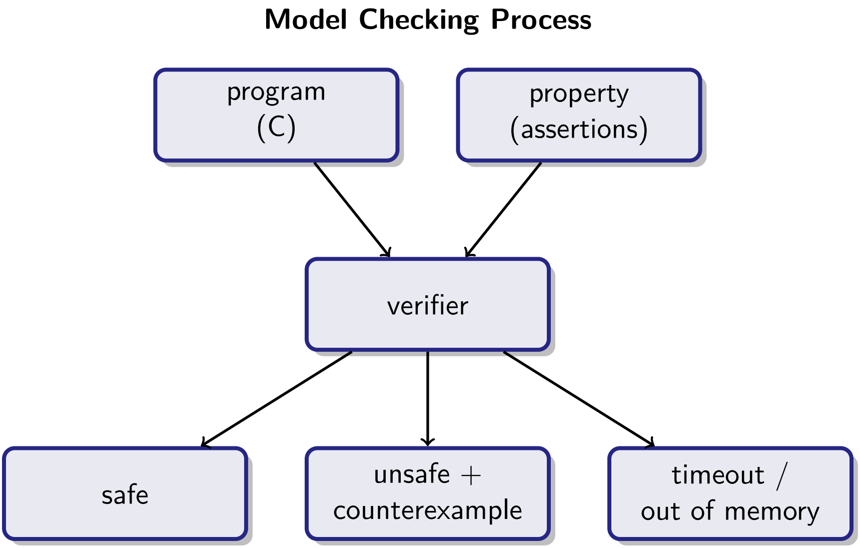
Generalisation Methods for Control-Flow Oriented IC3 Algorithms

Frederick Prinz

RWTH Aachen University

September 23, 2016 / Master's Thesis Presentation


Model Checking Process
program
(C)
property
(assertions)
verifier
safe
unsafe +
counterexample
timeout /
out of memory

Outline

1. Preliminaries
2. IC3CFA
3. Experimental Results
4. Conclusion

Outline

Literals / Cubes / CFA

Definitions

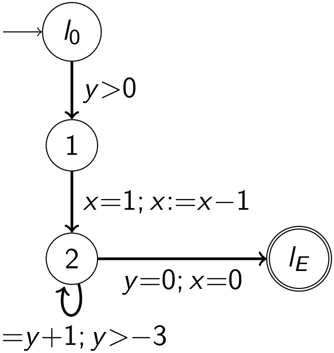
- a literal p is an atomic first-order formula
- a cube c is a conjunction of literals, i.e. $c = \bigwedge\{p_1, ..., p_n\}$

Literals / Cubes / CFA

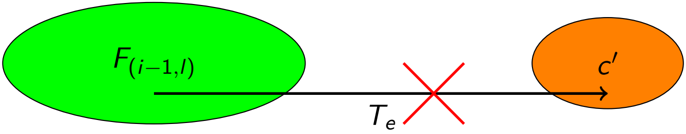
Definitions

- a literal p is an atomic first-order formula
- a cube c is a conjunction of literals, i.e. $c = \bigwedge\{p_1, ..., p_n\}$
- a control-flow automaton is a tuple (L, G, l_0, l_E)

```
procedure MAIN(x, y)
    assume(y > 0)
    assume(x = 1)
    x ← x − 1
    while true do
        assert(¬(y = 0 ∧ x = 0))
        y ← y + 1
        assume(y > −3)
    end while
end procedure
```



IC3CFA Generalisation I



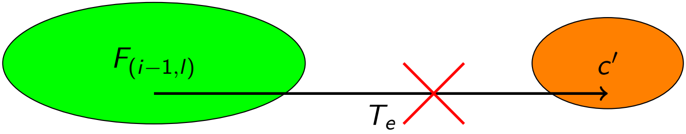
Relative Inductiveness

- block cube c with respect to edge e,
  if c is relative inductive with respect to $F_{(i-1,l)}$ and e, i.e.

$$(l \neq l') \qquad relInd(F_{(i-1,l)}, e, c) \Leftrightarrow UNSAT(F_{(i-1,l)} \wedge T_e \wedge c')$$

$$(l = l') \qquad relInd(F_{(i-1,l)}, e, c) \Leftrightarrow UNSAT(F_{(i-1,l)} \wedge \neg c \wedge T_e \wedge c')$$

IC3CFA Generalisation I



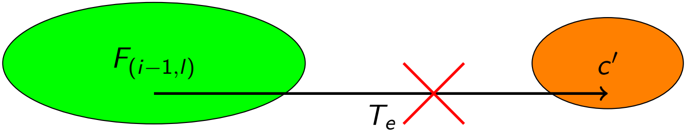
Relative Inductiveness

- block cube c with respect to edge e,
  if c is relative inductive with respect to $F_{(i-1,l)}$ and e, i.e.

$$(l \neq l') \qquad relInd(F_{(i-1,l)}, e, c) \Leftrightarrow UNSAT(F_{(i-1,l)} \wedge T_e \wedge c')$$
$$(l = l') \qquad relInd(F_{(i-1,l)}, e, c) \Leftrightarrow UNSAT(F_{(i-1,l)} \wedge \neg c \wedge T_e \wedge c')$$

- compute generalisation $gen_{(i,l')}(c, e)$ of cube c

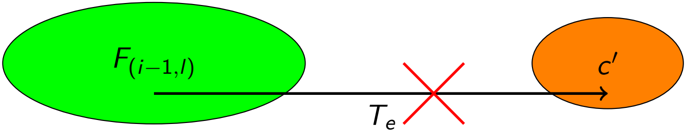
IC3CFA Generalisation I



Generalisation

- block cube c with respect to edge e
- compute generalisation $gen_{(i,l')}(c,e)$ of cube c

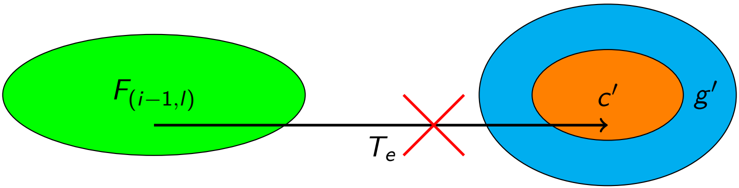
IC3CFA Generalisation I



Generalisation

- block cube c with respect to edge e
- compute generalisation $gen_{(i,l')}(c, e)$ of cube c
- $gen_{(i,l')}(c, e)$ is deterministic (not necessary)

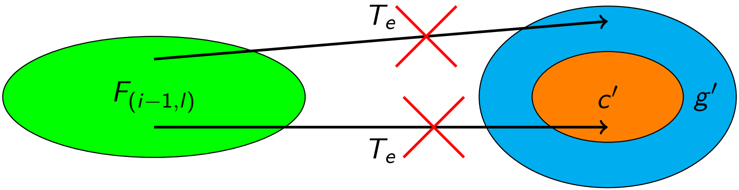
IC3CFA Generalisation I



Generalisation

- block cube c with respect to edge e
- compute generalisation $gen_{(i,l')}(c,e)$ of cube c
- $gen_{(i,l')}(c,e)$ is deterministic (not necessary)
- $gen_{(i,l')}(c,e)$ is syntactical subset of cube c, i.e. $gen_{(i,l')}(c,e) \subseteq c$

IC3CFA Generalisation I

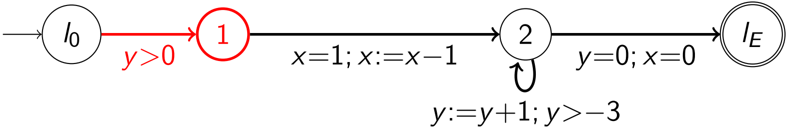


Generalisation

- block cube c with respect to edge e
- compute generalisation $gen_{(i,l')}(c,e)$ of cube c
- $gen_{(i,l')}(c,e)$ is deterministic (not necessary)
- $gen_{(i,l')}(c,e)$ is syntactical subset of cube c, i.e. $gen_{(i,l')}(c,e) \subseteq c$
- $gen_{(i,l')}(c,e)$ is relative inductive with respect to edge e, i.e.

$$relInd(F_{(i-1,l)}, e, c) \Rightarrow relInd(F_{(i-1,l)}, e, g)$$

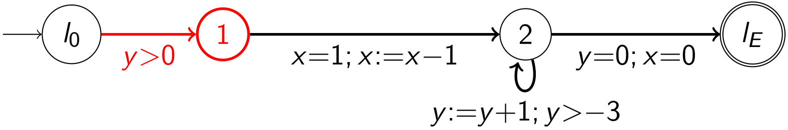
IC3CFA Generalisation II



Example

- block cube $c = \{y=0, x=1\}$ at location 1 with respect to edge $e_{l_0 \to 1}$

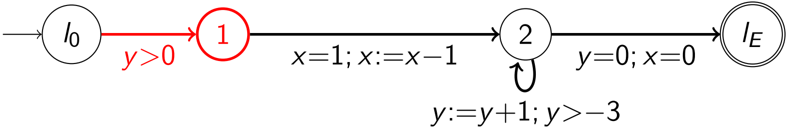
IC3CFA Generalisation II



Example

- block cube $c = \{y=0, x=1\}$ at location 1 with respect to edge $e_{l_0 \to 1}$
- compute generalisation $gen_{(i,1)}(\{y=0, x=1\}, e_{l_0 \to 1}) \subseteq \{y = 0, x = 1\}$

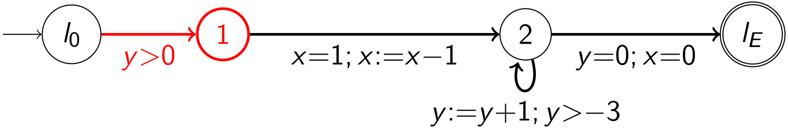
IC3CFA Generalisation II



Example

- block cube $c = \{y=0, x=1\}$ at location 1 with respect to edge $e_{l_0 \to 1}$
- compute generalisation $gen_{(i,1)}(\{y=0, x=1\}, e_{l_0 \to 1}) \subseteq \{y = 0, x = 1\}$
- drop literal $p \in c$, if $relInd(F_{(i-1,l)}, e_{l_0 \to 1}, c \setminus \{p\}) = true$ (IC3)

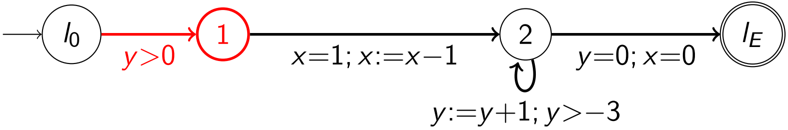
IC3CFA Generalisation II



Example

- block cube $c = \{y=0, x=1\}$ at location 1 with respect to edge $e_{l_0 \to 1}$
- compute generalisation $gen_{(i,1)}(\{y=0, x=1\}, e_{l_0 \to 1}) \subseteq \{y = 0, x = 1\}$
- drop literal $p \in c$, if $relInd(F_{(i-1,l)}, e_{l_0 \to 1}, c \setminus \{p\}) = true$ (IC3)
- e.g. drop literal $x=1$, since $relInd(true, e_{l_0 \to 1}, \{y=0\}) = true$, cannot drop remaining literal $y=0$, since $relInd(true, e_{l_0 \to 1}, \emptyset) = false$

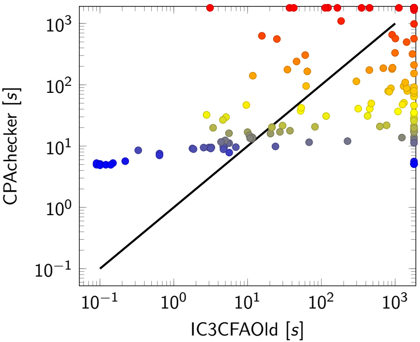
IC3CFA Generalisation II



Example

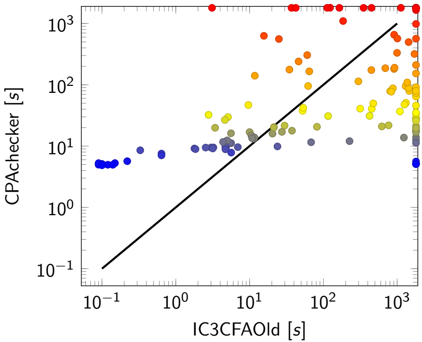
- block cube $c = \{y=0, x=1\}$ at location 1 with respect to edge $e_{l_0\to 1}$
- compute generalisation $gen_{(i,1)}(\{y=0, x=1\}, e_{l_0\to 1}) \subseteq \{y = 0, x = 1\}$
- drop literal $p \in c$, if $relInd(F_{(i-1,l)}, e_{l_0\to 1}, c \setminus \{p\}) = true$ (IC3)
- e.g. drop literal $x=1$, since $relInd(true, e_{l_0\to 1}, \{y=0\}) = true$, cannot drop remaining literal $y=0$, since $relInd(true, e_{l_0\to 1}, \emptyset) = false$
- in consequence, $gen_{(i,1)}(\{y=0, x=1\}, e_{l_0\to 1}) = \{y=0\}$

Motivation II



Algorithm	# solved	t solved	score	memory
IC3CFAOld	101 / 150	29,200 s	165	30,820 MB
CPAchecker	120 / 150	12,680 s	193	44,000 MB

Motivation II

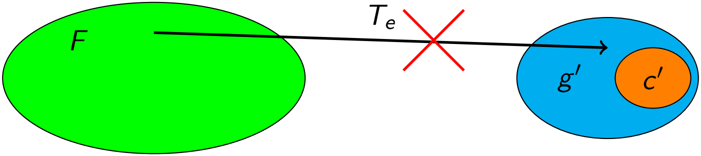


Idea

- IC3CFAOld spends 61% of time in SMT solver
- replace SMT calls by syntactical checks

Outline

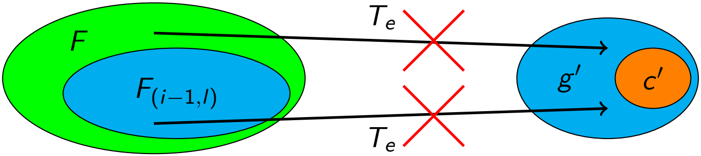
New Generalisation Context I



Generalisation Context

- let $g \subseteq c$ be a syntactic generalisation of cube c with respect to frame F and edge e
- cache generalisation g as generalisation context (c, F, e, g)

New Generalisation Context I

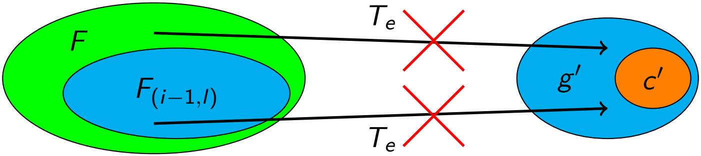


Generalisation Context

- let $g \subseteq c$ be a syntactic generalisation of cube c with respect to frame F and edge e
- cache generalisation g as generalisation context (c, F, e, g)
- frame relation $F_1 \sqsubseteq F_2 \Leftrightarrow F_1 \supseteq F_2$
- if we encounter cube c at frame $F_{(i-1,l)}$ and edge e again, then

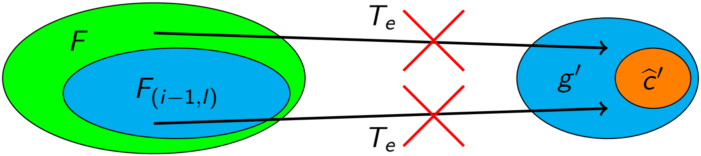
$$F_{(i-1,l)} \sqsubseteq F \quad \Rightarrow \quad gen_{(i,l')}(c, e) = g$$

New Generalisation Context I



New Generalisation Context

- cache generalisation g as generalisation context (c, F, e, g)
- so far, we have to encounter exactly the same cube c again

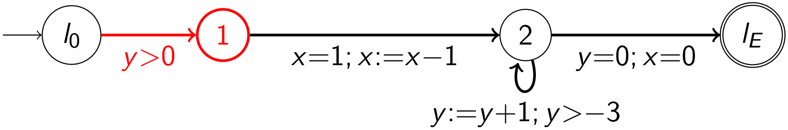


New Generalisation Context

- cache generalisation g as generalisation context (c, F, e, g)
- so far, we have to encounter exactly the same cube c again
- if we encounter another cube $\widehat{c}$ at frame $F_{(i-1,l)}$ and edge e, then

$$F_{(i-1,l)} \sqsubseteq F \wedge g \subseteq \widehat{c} \quad \Rightarrow \quad gen_{(i,l')}(\widehat{c}, e) = g$$

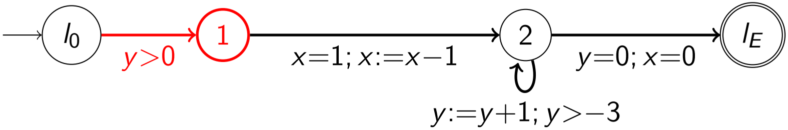
New Generalisation Context II



Example

- block cube $\widehat{c} = \{y=0, x=1\}$ at location 1 with respect to edge $e_{l_0 \to 1}$
- generalisation context $(c, \mathit{true}, e_{l_0 \to 1}, \{y=0\})$, where c is an arbitrary cube

New Generalisation Context II

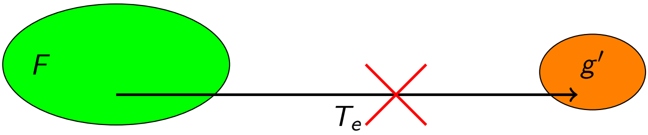


Example

- block cube $\widehat{c} = \{y=0, x=1\}$ at location 1 with respect to edge $e_{l_0 \to 1}$
- generalisation context $(c, true, e_{l_0 \to 1}, \{y=0\})$, where c is an arbitrary cube
- since $F_{(i-1,l_0)} \sqsubseteq true$ for every frame $F_{(i-1,l_0)}$, we get

$$gen_{(i,1)}(\{y=0, x=1\}, e_{l_0 \to 1}) = \{y=0\}$$

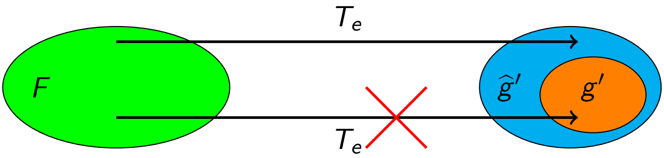
Minimal Generalisation I



Idea

- assume that syntactic generalisation $g \subseteq c$ is cached as generalisation context (c, F, e, g)

Minimal Generalisation I

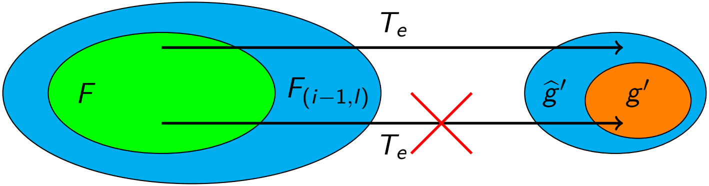


Idea

- assume that syntactic generalisation $g \subseteq c$ is cached as generalisation context (c, F, e, g)
- it holds that g is minimal, i.e.

$$\forall \widehat{g} \subset g.\ \widehat{g} \text{ is not relative inductive with respect to } F$$

Minimal Generalisation I



Idea

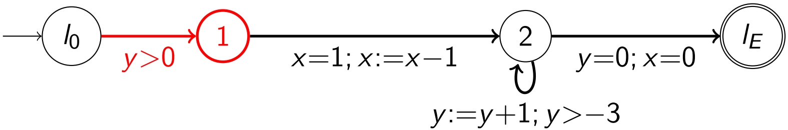
- assume that syntactic generalisation $g \subseteq c$ is cached as generalisation context (c, F, e, g)
- it holds that g is minimal, i.e.

$$\forall \widehat{g} \subset g.\ \widehat{g} \text{ is not relative inductive with respect to } F$$

- if we encounter cube c at frame $F_{(i-1,l)}$ and edge e again, then

$$F \sqsubseteq F_{(i-1,l)} \quad \Rightarrow \quad g \subseteq gen_{(i,l')}(c, e) \subseteq c$$

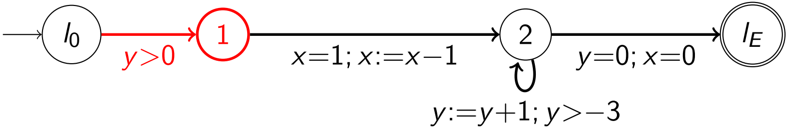
Minimal Generalisation II



Example

- block cube $c = \{y = 0, x = 1\}$ at location 1
  with respect to edge $e_{l_0 \to 1}$

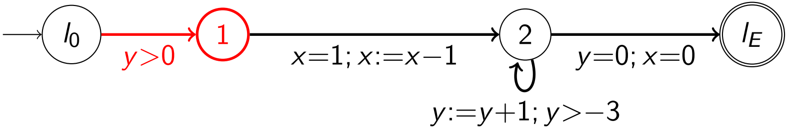
Minimal Generalisation II



Example

- block cube $c = \{y = 0, x = 1\}$ at location 1 with respect to edge $e_{l_0 \to 1}$
- generalisation context $(c, \mathit{true}, e_{l_0 \to 1}, \{y=0\})$
- assume that $F_{(i-1,l_0)} = \mathit{true}$, such that $\mathit{true} \sqsubseteq F_{(i-1,l_0)}$

Minimal Generalisation II



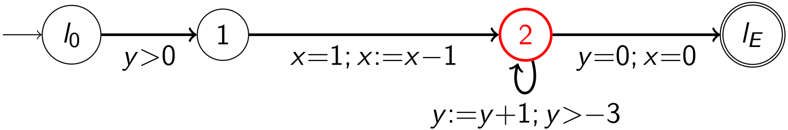
Example

- block cube $c = \{y = 0, x = 1\}$ at location 1 with respect to edge $e_{l_0 \to 1}$
- generalisation context $(c, \mathit{true}, e_{l_0 \to 1}, \{y=0\})$
- assume that $F_{(i-1,l_0)} = \mathit{true}$, such that $\mathit{true} \sqsubseteq F_{(i-1,l_0)}$
- we get

$$\{y = 0\} \subseteq gen_{(i,1)}(\{y=0, x=1\}, e_{l_0 \to 1}) \subseteq \{y = 0, x = 1\}$$

i.e. $gen_{(i,1)}(\{y=0, x=1\}, e_{l_0 \to 1}) = \emptyset \equiv \mathit{true}$ is not possible

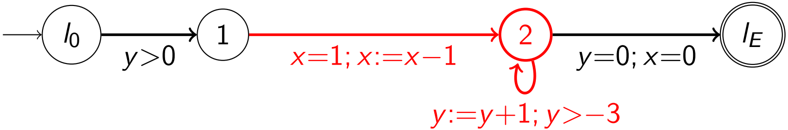
Multiple Predecessors I



Example

- generalise cube $c = \{y{=}0, x{=}0\}$ at location 2

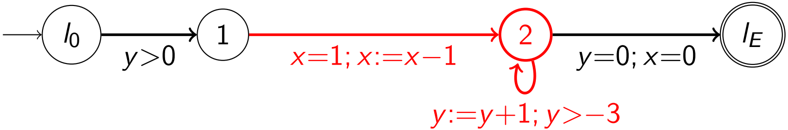
Multiple Predecessors I



Example

- generalise cube $c = \{y=0, x=0\}$ at location 2
- compute edge-based generalisations with respect to $e_{1\to2}$ and $e_{2\to2}$

Multiple Predecessors I



Example

- generalise cube $c = \{y=0, x=0\}$ at location 2
- compute edge-based generalisations with respect to $e_{1\to 2}$ and $e_{2\to 2}$

Observation

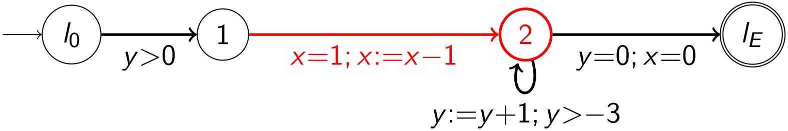
It holds that

$$gen_{(i,2)}(c) = \bigcup_{e\in G} gen_{(i,2)}(c, e)$$

such that

$$\forall e \in G.\ gen_{(i,2)}(c, e) \subseteq gen_{(i,2)}(c) \subseteq c$$

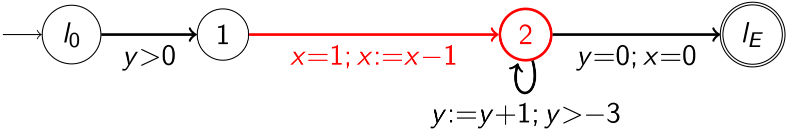
Multiple Predecessors II



First Edge

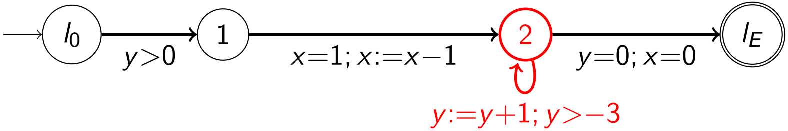
- assume that $gen_{(i,2)}(\{y=0, x=0\}, e_{1\to 2}) = \{y=0\}$

Multiple Predecessors II



First Edge

- assume that $gen_{(i,2)}(\{y=0, x=0\}, e_{1\to 2}) = \{y=0\}$



Second Edge

- in consequence, we get

$$\{y=0\} \subseteq gen_{(i,2)}(\{y=0, x=0\}, e_{2\to 2}) \subseteq \{y=0, x=0\}$$

Ordering

Observation

- given three cubes to be blocked, e.g.
  $c_1 = \{p_1, p_2\}$, $c_2 = \{p_1, p_3, p_4\}$, $c_3 = \{p_1, p_3, p_5\}$
- assume that there exists generalisation $g = \{p_1\}$
  blocking all cubes c_1, c_2, c_3

Ordering

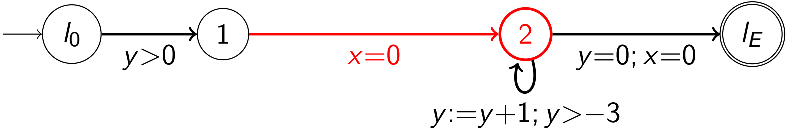
Observation

- given three cubes to be blocked, e.g.
  $c_1 = \{p_1, p_2\}$, $c_2 = \{p_1, p_3, p_4\}$, $c_3 = \{p_1, p_3, p_5\}$
- assume that there exists generalisation $g = \{p_1\}$ blocking all cubes c_1, c_2, c_3
- generalisation of c_1 requires only 2 SMT calls (2 literals), while the one of c_2, c_3 requires 3 SMT calls (3 literals)

Ordering

experimental results: ordering in ascending order according to cube size

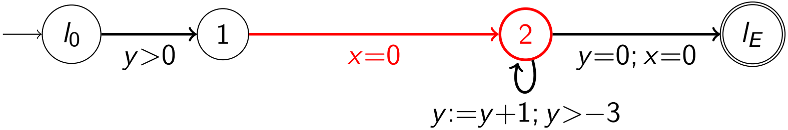
Guaranteed Literals



Observation

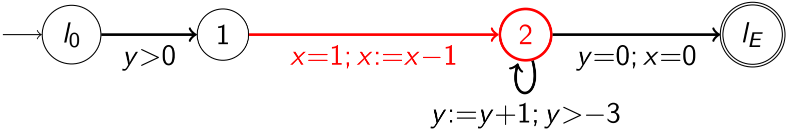
- so far, drop literals syntactically guaranteed by transition T_e
- e.g. $gen_{(i,2)}(\{y = 0, x = 0\}, e_{1\to 2}) = \{y = 0\}$

Guaranteed Literals



Observation

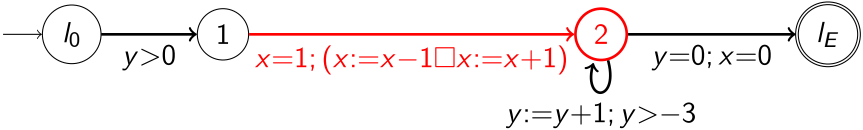
- so far, drop literals syntactically guaranteed by transition T_e
- e.g. $gen_{(i,2)}(\{y=0, x=0\}, e_{1\to 2}) = \{y=0\}$



Idea

- drop literals semantically guaranteed by transition T_e
- e.g. $gen_{(i,2)}(\{y=0, x=0\}, e_{1\to 2}) = \{y=0\}$

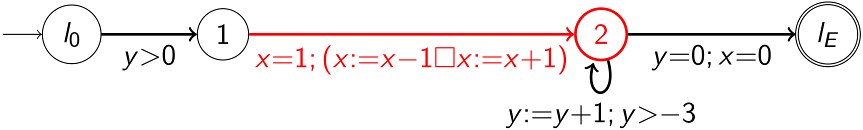
Predecessor Computation I



Observation

- consider cube $c = \{y=0, x=0\}$ at location 2
  and block predecessor cubes with respect to edge $e_{1\to2}$

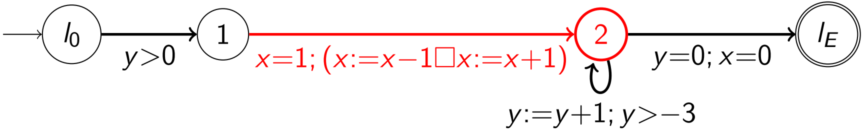
Predecessor Computation I



Observation

- consider cube $c = \{y=0, x=0\}$ at location 2 and block predecessor cubes with respect to edge $e_{1\to 2}$
- $wep(e_{1\to 2}, c) = (x=1 \wedge ((y=0 \wedge x-1=0) \vee (y=0 \wedge x+1=0)))$

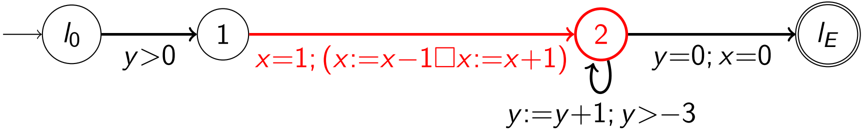
Predecessor Computation I



Observation

- consider cube $c = \{y=0, x=0\}$ at location 2
  and block predecessor cubes with respect to edge $e_{1\to2}$
- $wep(e_{1\to2}, c) = (x=1 \wedge ((y=0 \wedge x-1=0) \vee (y=0 \wedge x+1=0)))$
- so far, compute DNF to get predecessor cubes
  $dnf(wep(e_{1\to2}, c)) = \{\{x=1, y=0, x-1=0\}, \{x=1, y=0, x+1=0\}\}$

Predecessor Computation I



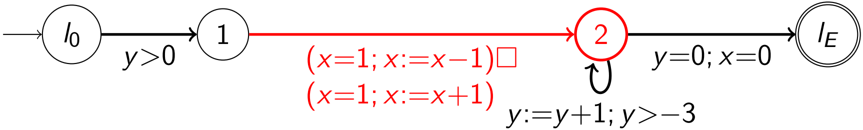
Observation

- consider cube $c = \{y=0, x=0\}$ at location 2
  and block predecessor cubes with respect to edge $e_{1\to2}$
- $wep(e_{1\to2}, c) = (x=1 \wedge ((y=0 \wedge x-1=0) \vee (y=0 \wedge x+1=0)))$
- so far, compute DNF to get predecessor cubes
  $dnf(wep(e_{1\to2}, c)) = \{\{x=1, y=0, x-1=0\}, \{x=1, y=0, x+1=0\}\}$

Idea

- avoid expensive DNF computation, derive DNF by command structure
- split command into sequential parts (constant for edge, cached)

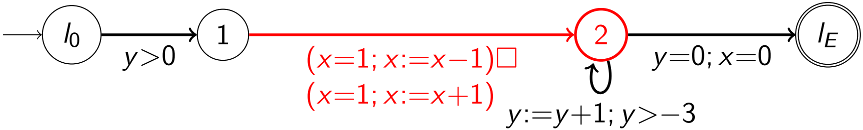
Predecessor Computation II



Idea

- avoid expensive DNF computation, derive DNF by command structure
- divide command into sequential parts (constant for edge, cached)

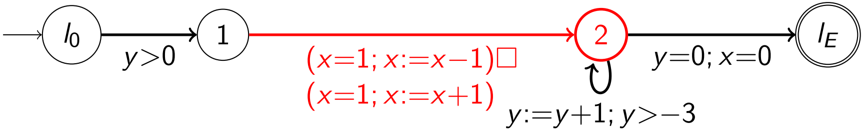
Predecessor Computation II



Idea

- avoid expensive DNF computation, derive DNF by command structure
- divide command into sequential parts (constant for edge, cached)
- compute WEP for every sequential part
  $wep(e'_{1\to 2}, c) = \{x{=}1, y{=}0, x{-}1{=}0\}$
  $wep(e''_{1\to 2}, c) = \{x{=}1, y{=}0, x{+}1{=}0\}$

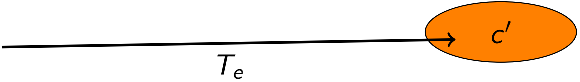
Predecessor Computation II



Idea

- avoid expensive DNF computation, derive DNF by command structure
- divide command into sequential parts (constant for edge, cached)
- compute WEP for every sequential part
  $wep(e'_{1\to 2}, c) = \{x=1, y=0, x-1=0\}$
  $wep(e''_{1\to 2}, c) = \{x=1, y=0, x+1=0\}$
- each computed WEP yields exactly one predecessor cube
- DNF computation not necessary any more

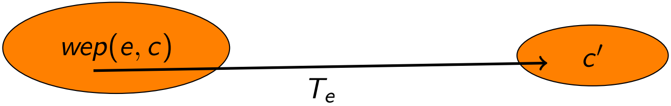
Predecessor Cubes I



Idea

- generalise cube c with respect to edge e,
  derive generalisation $gen_{(i,l')}(e, c)$ from predecessor cubes

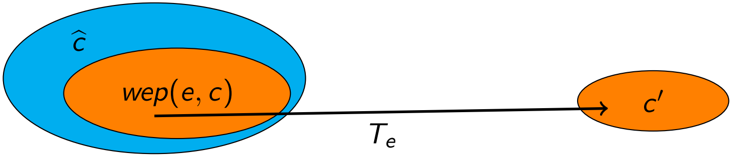
Predecessor Cubes I



Idea

- generalise cube c with respect to edge e,
  derive generalisation $gen_{(i,l')}(e, c)$ from predecessor cubes
- compute predecessor $wep(e, c)$ of cube c

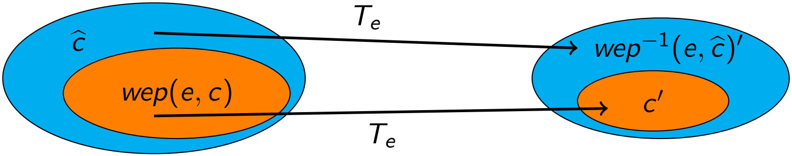
Predecessor Cubes I



Idea

- generalise cube c with respect to edge e,
  derive generalisation $gen_{(i,l')}(e, c)$ from predecessor cubes
- compute predecessor $wep(e, c)$ of cube c
- find superset $\widehat{c}$ of $wep(e, c)$ in predecessor frame $F_{(i-1,l)}$,
  i.e. $\widehat{c} \supseteq wep(e, c)$

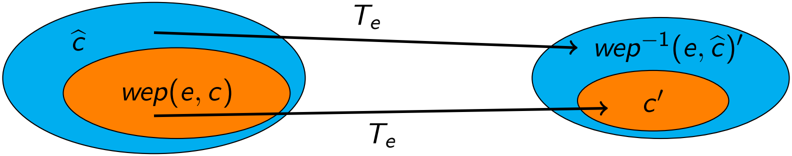
Predecessor Cubes I



Idea

- generalise cube c with respect to edge e,
  derive generalisation $gen_{(i,l')}(e, c)$ from predecessor cubes
- compute predecessor $wep(e, c)$ of cube c
- find superset $\widehat{c}$ of $wep(e, c)$ in predecessor frame $F_{(i-1,l)}$,
  i.e. $\widehat{c} \supseteq wep(e, c)$
- compute successor $wep^{-1}(e, \widehat{c})$ to get generalisation of cube c,
  i.e. $gen_{(i,l')}(e, c) = wep^{-1}(e, \widehat{c})$

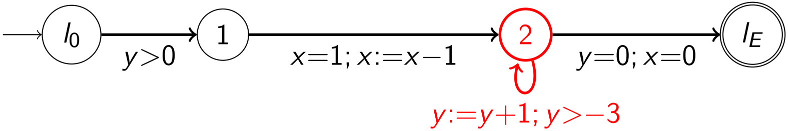
Predecessor Cubes I



Idea

- generalise cube c with respect to edge e,
  derive generalisation $gen_{(i,l')}(e, c)$ from predecessor cubes
- compute predecessor $wep(e, c)$ of cube c
- find superset $\widehat{c}$ of $wep(e, c)$ in predecessor frame $F_{(i-1,l)}$,
  i.e. $\widehat{c} \supseteq wep(e, c)$
- compute successor $wep^{-1}(e, \widehat{c})$ to get generalisation of cube c,
  i.e. $gen_{(i,l')}(e, c) = wep^{-1}(e, \widehat{c})$
- in fact, create mapping to compute $wep^{-1}(e, \widehat{c})$

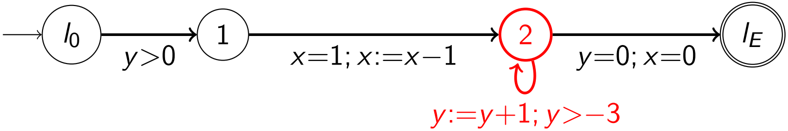
Predecessor Cubes II



Example

- generalise cube $c = \{y=0, x=0\}$ with respect to edge $e_{2\to 2}$

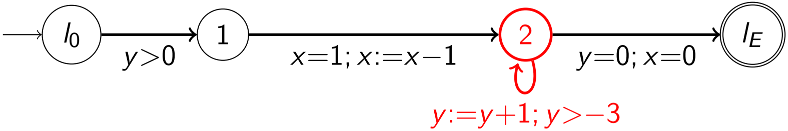
Predecessor Cubes II



Example

- generalise cube $c = \{y=0, x=0\}$ with respect to edge $e_{2\to2}$
- compute predecessor $wep(\{y=0, x=0\}, e_{2\to2}) = \{y=-1, x=0\}$

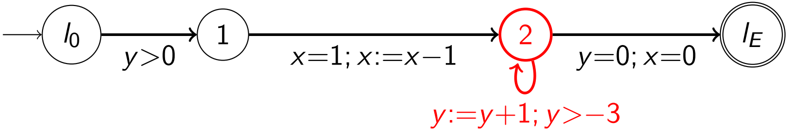
Predecessor Cubes II



Example

- generalise cube $c = \{y=0, x=0\}$ with respect to edge $e_{2\to2}$
- compute predecessor $wep(\{y=0, x=0\}, e_{2\to2}) = \{y=-1, x=0\}$
- assume that there exists $\{y=-1\} \subseteq \{y=-1, x=0\}$ in predecessor frame $F_{(i-1,2)}$

Predecessor Cubes II



Example

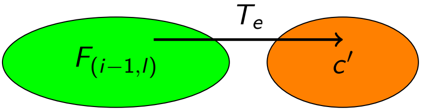
- generalise cube $c = \{y{=}0, x{=}0\}$ with respect to edge $e_{2\to2}$
- compute predecessor $wep(\{y{=}0, x{=}0\}, e_{2\to2}) = \{y{=}{-}1, x{=}0\}$
- assume that there exists $\{y{=}{-}1\} \subseteq \{y{=}{-}1, x{=}0\}$ in predecessor frame $F_{(i-1,2)}$
- compute successor $wep^{-1}(e_{2\to2}, \{y{=}{-}1\}) = \{y{=}0\}$ such that

$$gen_{(i,2)}(\{y{=}0, x{=}0\}, e_{2\to2}) = \{y{=}0\}$$

Alternative Relative Inductiveness

Relative Inductiveness

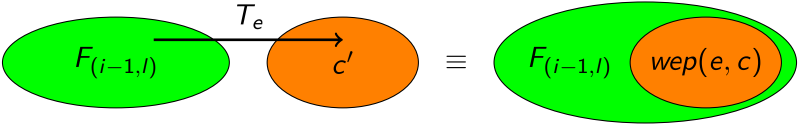
- so far, check $SAT(F_{(i-1,l)} \wedge T_e \wedge c')$



Alternative Relative Inductiveness

Relative Inductiveness

- so far, check $SAT(F_{(i-1,l)} \wedge T_e \wedge c')$



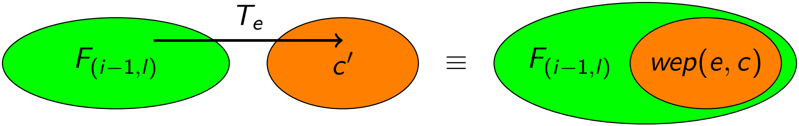
Alternative Relative Inductiveness

- compute exact preimage of c with respect to edge e, i.e. $wep(e, c)$
- check $SAT(F_{(i-1,l)} \wedge wep(e, c)) \equiv SAT(F_{(i-1,l)} \wedge T_e \wedge c')$

Alternative Relative Inductiveness

Relative Inductiveness

- so far, check $SAT(F_{(i-1,l)} \wedge T_e \wedge c')$



Alternative Relative Inductiveness

- compute exact preimage of c with respect to edge e, i.e. $wep(e, c)$
- check $SAT(F_{(i-1,l)} \wedge wep(e, c)) \equiv SAT(F_{(i-1,l)} \wedge T_e \wedge c')$

Experimental Results

- alternative relative inductiveness check performs significantly better
- $wep(e, c)$ already cached, smaller formula in SMT request, less variables, SMT solver with caching

Interpolation I

Idea

- so far, compute syntactic generalisation $g \subseteq c$
- instead, compute semantic generalisation X, such that $c \Rightarrow X$

Interpolation I

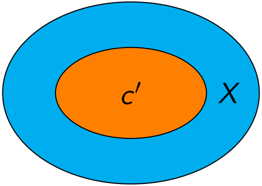
Idea

- so far, compute syntactic generalisation $g \subseteq c$
- instead, compute semantic generalisation X, such that $c \Rightarrow X$

Interpolant

Let $(A \wedge B)$ be unsatisfiable. An interpolant X for A and B satisfies

- $A \Rightarrow X$, $(A = c')$



Interpolation I

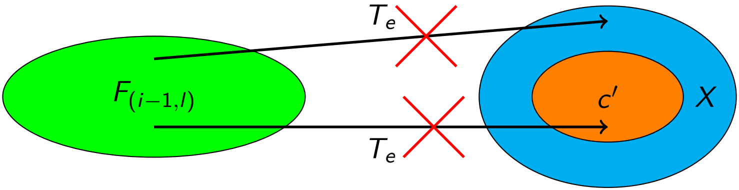
Idea

- so far, compute syntactic generalisation $g \subseteq c$
- instead, compute semantic generalisation X, such that $c \Rightarrow X$

Interpolant

Let $(A \wedge B)$ be unsatisfiable. An interpolant X for A and B satisfies

- $A \Rightarrow X$, $(A = c')$
- $X \Rightarrow \neg B$, $(B = F_{(i-1,l)} \wedge T_e)$



Interpolation I

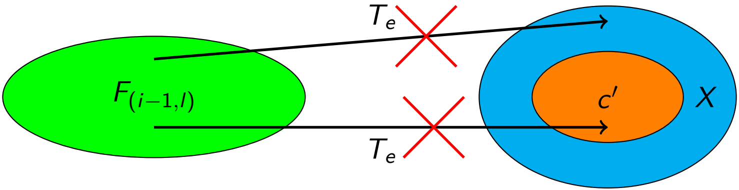
Idea

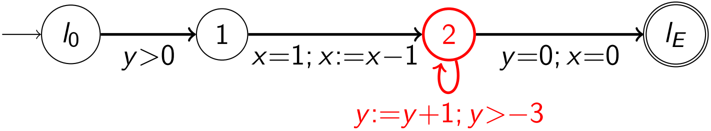
- so far, compute syntactic generalisation $g \subseteq c$
- instead, compute semantic generalisation X, such that $c \Rightarrow X$

Interpolant

Let $(A \wedge B)$ be unsatisfiable. An interpolant X for A and B satisfies

- $A \Rightarrow X$, $\quad (A = c')$
- $X \Rightarrow \neg B$, $\quad (B = F_{(i-1,l)} \wedge T_e)$
- $Var(X) \subseteq Var(A) \cap Var(B)$

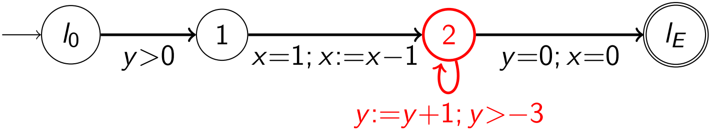




Observation

- consider location 2 with self loop $e_{2\to2}$
- IC3CFA derives proof obligations for $(y = \widehat{z} \wedge x = 0)$, where $\widehat{z} \in \{-3, -2, -1, 0\}$

Interpolation II



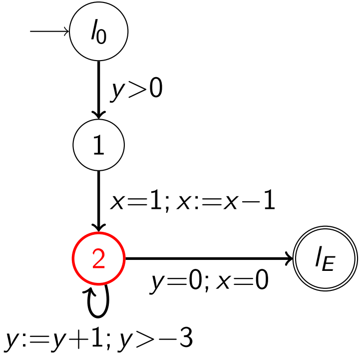
Observation

- consider location 2 with self loop $e_{2\to 2}$
- IC3CFA derives proof obligations for $(y = \hat{z} \wedge x = 0)$, where $\hat{z} \in \{-3, -2, -1, 0\}$

Interpolation

- apply IC3CFA with interpolation
- block CTI $\{y = 0, x = 0\}$ at location 2
- computed interpolant $X = \{y \leq 0\}$
- X is semantic generalisation and blocks all possible values of y

Improved Initialisation

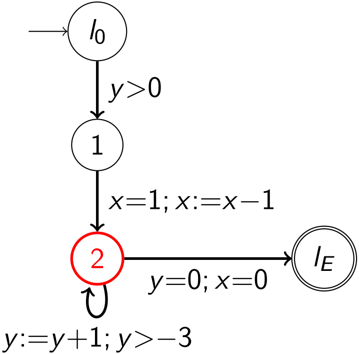


l: / i:	l_0	1	2
0	*true*	*false*	*false*
1	...	...	...
2	...	...	...

Idea

- location 2 only reachable with at least 2 steps
- *false* is safe overapproximation of reachable states $R_{(1,2)}$

Improved Initialisation



i: \ l:	l_0	1	2
0	*true*	*false*	*false*
1	...	...	*false*
2	...	...	...

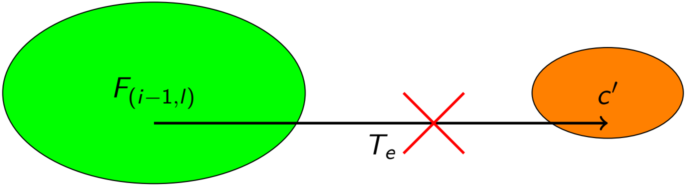
Idea

- location 2 only reachable with at least 2 steps
- *false* is safe overapproximation of reachable states $R_{(1,2)}$
- compute minimal distances by graph analysis
- initialise frame $F_{(1,2)}$ with *false* to avoid computations

CTGs

Idea

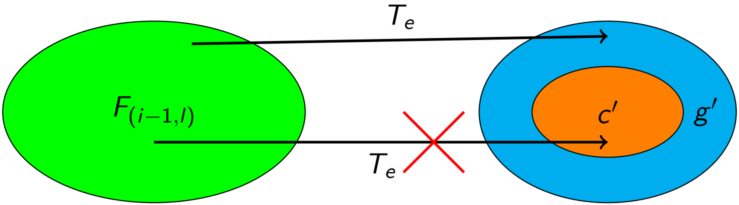
- try to generalise cube c



CTGs

Idea

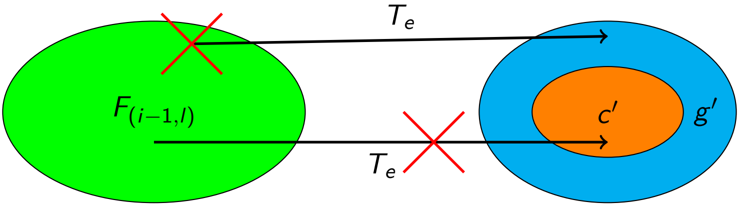
- try to generalise cube c
- possible generalisation $g \subset c$ is not relative inductive



CTGs

Idea

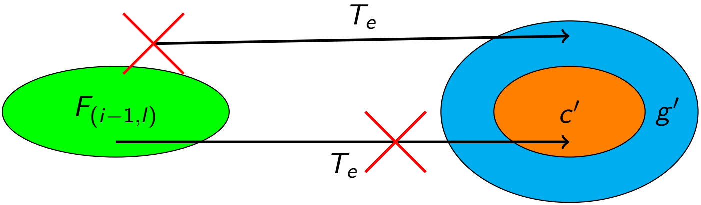
- try to generalise cube c
- possible generalisation $g \subset c$ is not relative inductive
- add proof obligations with counterexamples to generalisation (CTGs)



CTGs

Idea

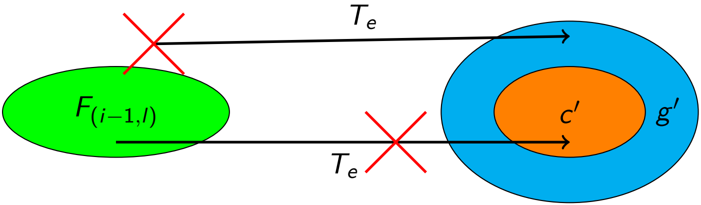
- try to generalise cube c
- possible generalisation $g \subset c$ is not relative inductive
- add proof obligations with counterexamples to generalisation (CTGs)
- block all CTGs, such that we can generalise c to g



CTGs

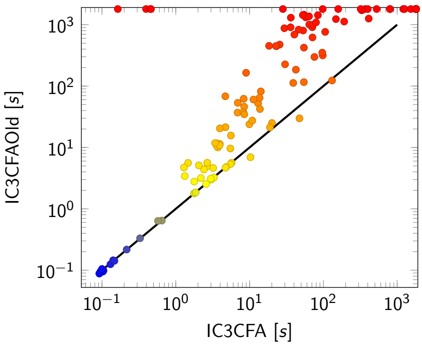
Idea

- try to generalise cube c
- possible generalisation $g \subset c$ is not relative inductive
- add proof obligations with counterexamples to generalisation (CTGs)
- block all CTGs, such that we can generalise c to g
- however, additional computations are too expensive



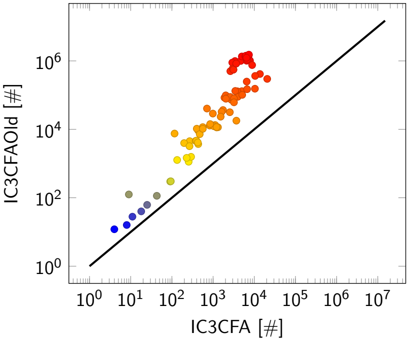
Outline

Evaluation I



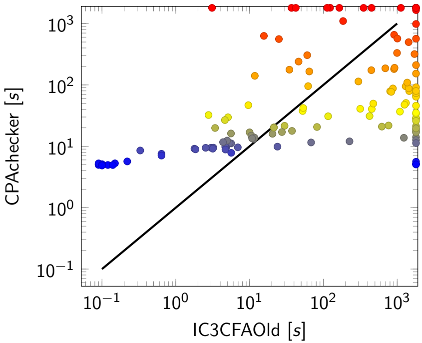
Algorithm	# solved	t solved	score	memory
IC3CFA	117 / 150	10,700 s	194	14,230 MB
IC3CFAOld	101 / 150	29,200 s	165	30,820 MB

Evaluation II



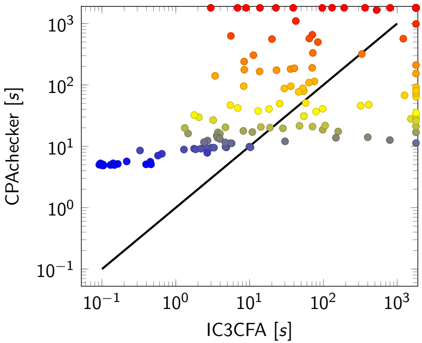
Reduction of SMT calls

- best case: 895,207 $\rightarrow$ 2,924 (factor 306)
- overall: 24,700k $\rightarrow$ 249k (average factor 67)



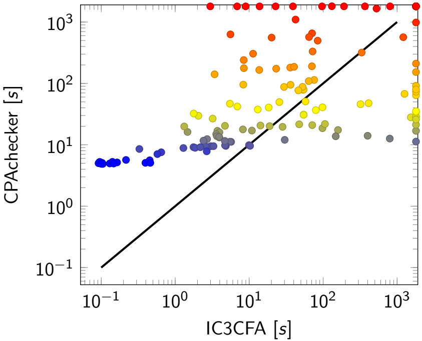
Algorithm	# solved	t solved	score	memory
IC3CFAOld	101 / 150	29,200 s	165	30,820 MB
CPAchecker	120 / 150	12,680 s	193	44,000 MB

Evaluation III



Algorithm	# solved	t solved	score	memory
IC3CFA	117 / 150	10,700 s	194	14,230 MB
CPAchecker	120 / 150	12,680 s	193	44,000 MB

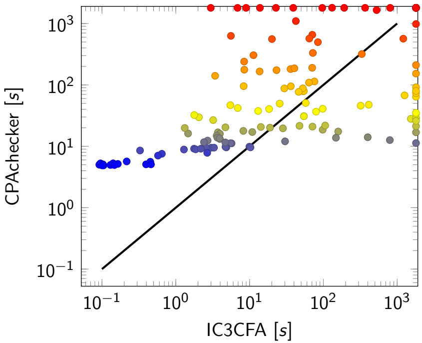
Evaluation III



Observation

- IC3CFA solves 11 programs exclusively (CPAchecker: 14)
- IC3CFA solves 83 programs faster (CPAchecker: 23)
- IC3CFA and CPAchecker are orthogonal approaches

Evaluation III



Future work

- IC3CFA and CPAchecker are orthogonal approaches
- only 18 programs are not solvable at all
- combine both approaches, i.e. integrate IC3CFA into CPA framework

Outline

Summary I

Contributions

- new generalisation context
- (concept of) minimal generalisation
- (heuristic for) multiple predecessors
- ordering (of multiple cubes)
- guaranteed literals
- predecessor computation (without DNF)
- (generalisation based on) predecessor cubes
- alternative relative inductiveness
- interpolation
- improved initialisation
- CTGs

Summary II

Contributions

- ...

Implementation

- new IC3CFA generalisation algorithm
- prototypical implementation into existing framework
- about 800 lines of OCaml code

Summary II

Contributions

- ...

Implementation

- new IC3CFA generalisation algorithm
- prototypical implementation into existing framework
- about 800 lines of OCaml code

Evaluation

- new IC3CFA generalisation outperforms old one
- significant reduction of SMT calls
- competitive to other state-of-the-art model checkers
- IC3CFA and CPAChecker are orthogonal approaches

Appendix

IC3 I

Frames

- frame F_i overapproximates i-step reachable program states R_i

IC3 I

Frames

- frame F_i overapproximates i-step reachable program states R_i
- IC3 iteratively derives sequence of frames $F_0, ..., F_k$

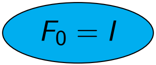
IC3 I

Frames

- frame F_i overapproximates i-step reachable program states R_i
- IC3 iteratively derives sequence of frames $F_0, ..., F_k$

IC3 Invariants

- $I \Rightarrow F_0$, (initial states I)



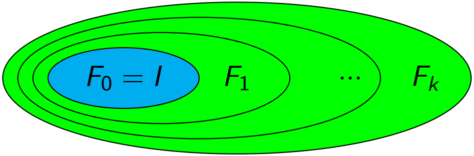
IC3 I

Frames

- frame F_i overapproximates i-step reachable program states R_i
- IC3 iteratively derives sequence of frames $F_0, ..., F_k$

IC3 Invariants

- $I \Rightarrow F_0$, (initial states I)
- $\forall 0 \leq i < k.\ F_i \Rightarrow F_{i+1}$,



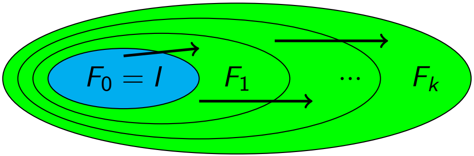
IC3 I

Frames

- frame F_i overapproximates i-step reachable program states R_i
- IC3 iteratively derives sequence of frames $F_0, \ldots, F_k$

IC3 Invariants

- $I \Rightarrow F_0,$ (initial states I)
- $\forall 0 \leq i < k.\ F_i \Rightarrow F_{i+1},$
- $\forall 0 \leq i < k.\ F_i \wedge T \Rightarrow F'_{i+1},$ (transition relation T)



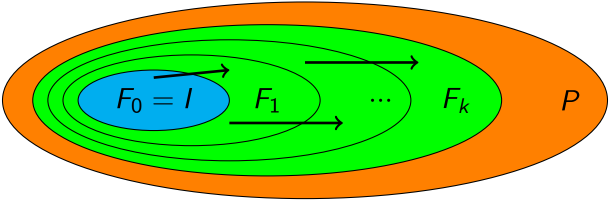
IC3 I

Frames

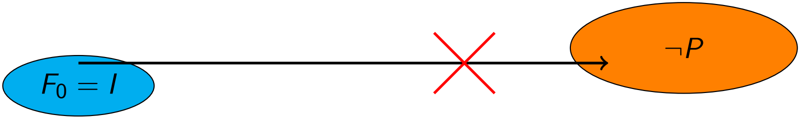
- frame F_i overapproximates i-step reachable program states R_i
- IC3 iteratively derives sequence of frames $F_0, \ldots, F_k$

IC3 Invariants

- $I \Rightarrow F_0$, (initial states I)
- $\forall 0 \leq i < k.\ F_i \Rightarrow F_{i+1}$,
- $\forall 0 \leq i < k.\ F_i \wedge T \Rightarrow F'_{i+1}$, (transition relation T)
- $\forall 0 \leq i \leq k.\ F_i \Rightarrow P$. (desired property P)



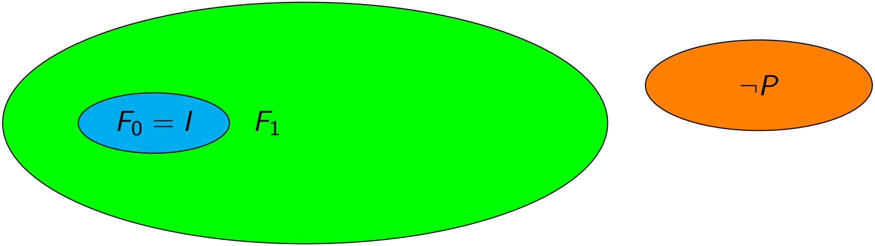
IC3 II



IC3 Algorithm

- initialise F_0 with I, check 0-/1-step counterexamples

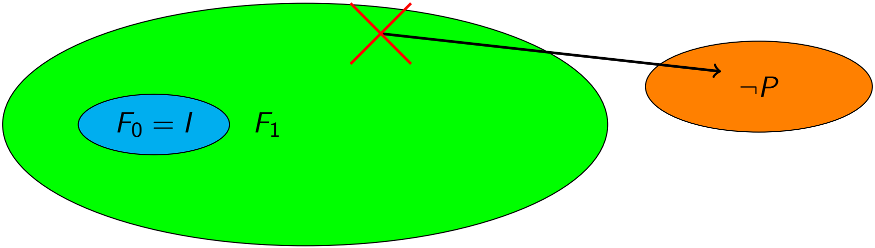
IC3 II



IC3 Algorithm

- initialise F_0 with I, check 0-/1-step counterexamples
- set frame F_1 to P

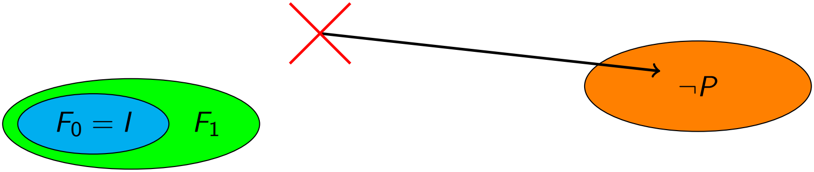
IC3 II



IC3 Algorithm

- initialise F_0 with I, check 0-/1-step counterexamples
- set frame F_1 to P
- add proof obligations with counterexamples to induction (CTIs)

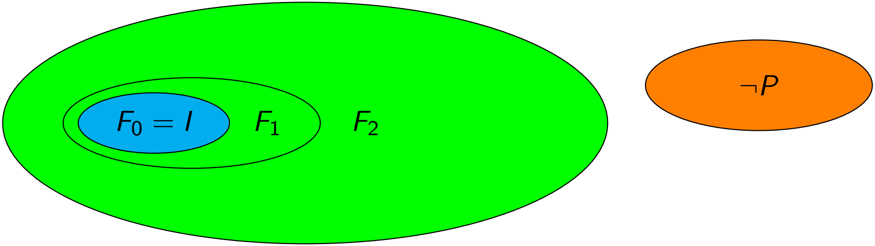
IC3 II



IC3 Algorithm

- initialise F_0 with I, check 0-/1-step counterexamples
- set frame F_1 to P
- add proof obligations with counterexamples to induction (CTIs)
- (recursively) block all CTIs in frame F_1

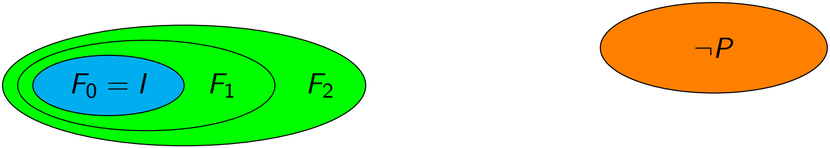
IC3 II



IC3 Algorithm

- initialise F_0 with I, check 0-/1-step counterexamples
- set frame F_1 to P
- add proof obligations with counterexamples to induction (CTIs)
- (recursively) block all CTIs in frame F_1
- continue with frame F_2

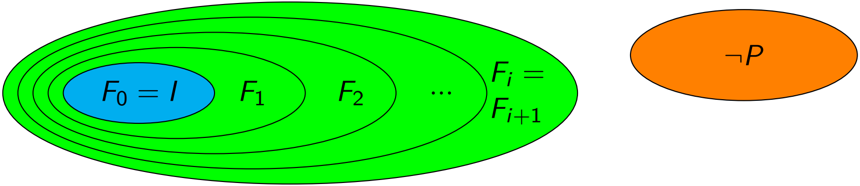
IC3 II



IC3 Algorithm

- initialise F_0 with I, check 0-/1-step counterexamples
- set frame F_1 to P
- add proof obligations with counterexamples to induction (CTIs)
- (recursively) block all CTIs in frame F_1
- continue with frame F_2

IC3 II



IC3 Algorithm

- initialise F_0 with I, check 0-/1-step counterexamples
- set frame F_1 to P
- add proof obligations with counterexamples to induction (CTIs)
- (recursively) block all CTIs in frame F_1
- continue with frame F_2
- fixpoint reached if $F_{i+1} = F_i$

IC3CFA

IC3CFA

- based on original IC3 algorithm
- lifted to software model checking (SMT instead of SAT solving)
- adapted to incorporate CFA information, i.e. $F_{(i,l)}$

IC3CFA

IC3CFA

- based on original IC3 algorithm
- lifted to software model checking (SMT instead of SAT solving)
- adapted to incorporate CFA information, i.e. $F_{(i,l)}$

IC3CFA Invariants

- $F_{(0,l_0)} = \mathit{true}$,
  $\forall l \neq l_0.\ F_{(0,l)} = \mathit{false}$, (initial location l_0)
- $\forall l \in L,\ 0 \leq i < k.\ F_{(i,l)} \Rightarrow F_{(i+1,l)}$, (CFA locations L)
- $\forall l' \in L \setminus \{l_E\},\ e_{l \rightarrow l'} \in G,\ 0 \leq i < k.$
  $F_{(i,l)} \wedge T_{e_{l \rightarrow l'}} \Rightarrow F'_{(i+1,l')}$, (CFA edges G)
- $\forall 0 \leq i \leq k.\ \neg\exists F_{(i,l_E)}$. (error location l_E)