Generalisation Methods for Control-Flow Oriented IC3 Algorithms

Frederick Prinz

RWTH Aachen University

September 23, 2016 / Master’s Thesis Presentation
Motivation I

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
1 Preliminaries

2 IC3CFA

3 Experimental Results

4 Conclusion
Definitions

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

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

1: procedure MAIN($x, y$)
2:   assume($y > 0$)
3:   assume($x = 1$)
4:   $x \leftarrow x - 1$
5:   while true do
6:     assert($\neg(y = 0 \land x = 0)$)
7:     $y \leftarrow y + 1$
8:     assume($y > -3$)
9:   end while
10: end procedure

Diagram:

- $l_0 \rightarrow l_0$
- $l_0 \rightarrow 1$ (y>0)
- $1 \rightarrow 2$ (x=1; x:=x-1)
- $2 \rightarrow l_E$ (y=0; x=0)
- $2 \rightarrow 2$ (y:=y+1; y>−3)
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.

\[
(l \neq l') \quad \text{relInd}(F_{(i-1,l)}, e, c) \Leftrightarrow \quad \text{UNSAT}(F_{(i-1,l)} \land T_e \land c')
\]

\[
(l = l') \quad \text{relInd}(F_{(i-1,l)}, e, c) \Leftrightarrow \quad \text{UNSAT}(F_{(i-1,l)} \land \neg c \land T_e \land c')
\]
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.
  
  \[
  (l \neq l') \quad \text{relInd}(F_{(i-1,l)}, e, c) \iff \text{UNSAT}(F_{(i-1,l)} \land T_e \land c')
  \]
  
  \[
  (l = l') \quad \text{relInd}(F_{(i-1,l)}, e, c) \iff \text{UNSAT}(F_{(i-1,l)} \land \neg c \land T_e \land c')
  \]

- compute generalisation $\text{gen}_{(i,l')}(c, e)$ of cube $c$
Generalisation

- block cube $c$ with respect to edge $e$
- compute generalisation $\text{gen}_{(i,l')}((c, e)$ of cube $c$
Generalisation

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

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

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

\[
\text{rellnd}(F_{(i-1, l)}, e, c) \Rightarrow \text{rellnd}(F_{(i-1, l)}, e, g)
\]
Example

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

- block cube \( c = \{y=0, x=1\} \) at location 1 with respect to edge \( e_{l_0 \rightarrow 1} \)
- compute **generalisation** \( \text{gen}_{(i,1)}(\{y=0, x=1\}, e_{l_0 \rightarrow 1}) \subseteq \{y = 0, x = 1\} \)
Example

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

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

\[ y>0 \quad x=1; x:=x-1 \quad y=0; x=0 \quad y:=y+1; y>-3 \]
Example

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th># solved</th>
<th>t solved</th>
<th>score</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC3CFAOld</td>
<td>101 / 150</td>
<td>29,200 s</td>
<td>165</td>
<td>30,820 MB</td>
</tr>
<tr>
<td>CPAchecker</td>
<td>120 / 150</td>
<td>12,680 s</td>
<td>193</td>
<td>44,000 MB</td>
</tr>
</tbody>
</table>

![Graph](image)
Motivation II

Idea

- IC3CFAOld spends 61% of time in SMT solver
- replace SMT calls by syntactical checks
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 $F_1 \sqsubseteq F_2 \iff F_1 \supseteq F_2$
- if we encounter cube $c$ at frame $F_{(i-1,l)}$ and edge $e$ again, then
  $$F_{(i-1,l)} \sqsubseteq F \implies gen_{(i,l')} (c, e) = g$$
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 \( \hat{c} \) at frame \( F_{(i-1,l)} \) and edge \( e \), then

\[
F_{(i-1,l)} \subseteq F \land g \subseteq \hat{c} \quad \Rightarrow \quad \text{gen}_{(i,l')}(\hat{c}, e) = g
\]
New Generalisation Context II

Example

- block cube $\hat{c} = \{y=0, x=1\}$ at location 1 with respect to edge $e_{l_0 \rightarrow 1}$
- generalisation context $(c, true, e_{l_0 \rightarrow 1}, \{y=0\})$, where $c$ is an arbitrary cube
New Generalisation Context II

\[
\begin{align*}
 l_0 &\rightarrow 1 & y > 0 \\
 1 &\rightarrow 2 & x = 1; x := x - 1 \\
 2 &\rightarrow l_E & y = 0; x = 0
\end{align*}
\]
\[y := y + 1; y > -3\]

Example

- block cube \(\hat{c} = \{y = 0, x = 1\}\) 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 \(F(i-1,l_0) \subseteq true\) for every frame \(F(i-1,l_0)\), we get

\[gen(i,1)(\{y = 0, x = 1\}, e_{l_0\to 1}) = \{y = 0\}\]
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.

$$\forall \hat{g} \subset g. \hat{g} \text{ is not relative inductive with respect to } F$$
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.
  \[ \forall \hat{g} \subset g. \hat{g} \text{ is not relative inductive with respect to } F \]
- if we encounter cube $c$ at frame $F(i-1,l)$ and edge $e$ again, then
  \[ F \sqsubseteq F(i-1,l) \implies g \subseteq \text{gen}(i,l')(c, e) \subseteq c \]
Example

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

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

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

\[
\{y = 0\} \subseteq \text{gen}_{(i,1)}(\{y = 0, x = 1\}, e_{l_0 \rightarrow 1}) \subseteq \{y = 0, x = 1\}
\]

i.e. \( \text{gen}_{(i,1)}(\{y = 0, x = 1\}, e_{l_0 \rightarrow 1}) = \emptyset \equiv \text{true} \) is not possible
Multiple Predecessors I

Example

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

\[
\begin{align*}
&l_0 \quad y > 0 \\
&1 \quad x = 1; x := x - 1 \\
&2 \quad y = 0; x = 0 \\
&l_E \quad y := y + 1; y > -3
\end{align*}
\]
Multiple Predecessors I

Example

- generalise cube \( c = \{y=0, x=0\} \) at location 2
- compute edge-based generalisations with respect to \( e_1 \rightarrow 2 \) and \( e_2 \rightarrow 2 \)
Multiple Predecessors I

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

Observation
It holds that
\[ \text{gen}_{(i,2)}(c) = \bigcup_{e \in G} \text{gen}_{(i,2)}(c, e) \]
such that
\[ \forall e \in G. \quad \text{gen}_{(i,2)}(c, e) \subseteq \text{gen}_{(i,2)}(c) \subseteq c \]
Multiple Predecessors II

First Edge

- assume that $\text{gen}_{(i,2)}(\{y=0, x=0\}, e_{1\rightarrow 2}) = \{y=0\}$
Multiple Predecessors II

First Edge
- assume that \( \text{gen}_{(i,2)}(\{y=0, x=0\}, e_{1\rightarrow 2}) = \{y=0\} \)

Second Edge
- in consequence, we get
  \[ \{y=0\} \subseteq \text{gen}_{(i,2)}(\{y=0, x=0\}, e_{2\rightarrow 2}) \subseteq \{y=0, x=0\} \]
Observation

- given three cubes to be blocked, e.g.
  \[ c_1 = \{ p_1, p_2 \}, \quad c_2 = \{ p_1, p_3, p_4 \}, \quad c_3 = \{ p_1, p_3, p_5 \} \]

- 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

\[ l_0 \xrightarrow{y > 0} 1 \xrightarrow{x = 0} 2 \xrightarrow{y = 0; x = 0} l_E \]

y := y + 1; y > −3

Observation

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

\[
\begin{align*}
\text{l}_0 \quad & y > 0 & \quad \rightarrow & \quad 1 \\
& & \quad \rightarrow & \quad \text{l}_E \\
& x = 0 & \quad \rightarrow & \quad 2 \\
& y = 0; x = 0 & \quad \rightarrow & \quad \text{l}_E \\
\end{align*}
\]

\[
y := y + 1; y > -3
\]

Observation

- so far, drop literals **syntactically** guaranteed by transition \( T_e \)
- e.g. \( \text{gen}_{(i,2)}(\{y = 0, x = 0\}, e_{1 \rightarrow 2}) = \{y = 0\} \)

\[
\begin{align*}
\text{l}_0 \quad & y > 0 & \quad \rightarrow & \quad 1 \\
& & \quad \rightarrow & \quad 2 \\
& x = 1; x := x - 1 & \quad \rightarrow & \quad \text{l}_E \\
& y = 0; x = 0 & \quad \rightarrow & \quad \text{l}_E \\
\end{align*}
\]

\[
y := y + 1; y > -3
\]

Idea

- drop literals **semantically** guaranteed by transition \( T_e \)
- e.g. \( \text{gen}_{(i,2)}(\{y = 0, x = 0\}, e_{1 \rightarrow 2}) = \{y = 0\} \)
Predecessor Computation I

\[ y > 0 \]

\[ x = 1; (x := x - 1 \lor x := x + 1) \]

\[ y := y + 1; y > -3 \]

Observation

- consider cube \( c = \{ y = 0, x = 0 \} \) at location 2
  and block predecessor cubes with respect to edge \( e_1 \rightarrow 2 \)
**Predecessor Computation I**

\[ x = 1; (x := x - 1 \square x := x + 1) \]
\[ y := y + 1; y > -3 \]

**Observation**
- Consider cube \( c = \{y = 0, x = 0\} \) at location 2 and block predecessor cubes with respect to edge \( e_1 \rightarrow 2 \).
- \( wep(e_1 \rightarrow 2, c) = (x = 1 \land ((y = 0 \land x - 1 = 0) \lor (y = 0 \land x + 1 = 0))) \)
Observation

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

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)
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
  
  \[
  \text{wep}(e_{1 \rightarrow 2}', c) = \{ x=1, y=0, x-1=0 \} \\
  \text{wep}(e_{1 \rightarrow 2}'', c) = \{ x=1, y=0, x+1=0 \}
  \]
**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
  
  \[
  \text{wep}(e'_1 \rightarrow 2, c) = \{x=1, y=0, x-1=0\} \\
  \text{wep}(e''_1 \rightarrow 2, c) = \{x=1, y=0, x+1=0\}
  \]

- 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 $\text{gen}_{(i,l')}(e, c)$ from predecessor cubes
Predecessor Cubes I

Idea

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

- generalise cube \( c \) with respect to edge \( e \), derive generalisation \( \text{gen}_{(i, l')}(e, c) \) from predecessor cubes
- compute predecessor \( \text{wep}(e, c) \) of cube \( c \)
- find superset \( \hat{c} \) of \( \text{wep}(e, c) \) in predecessor frame \( F_{(i-1, l)} \), i.e. \( \hat{c} \supseteq \text{wep}(e, c) \)
Predecessor Cubes I

Idea

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

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

Example

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

Example

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

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

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

$$\text{gen}_{(i,2)}(\{y=0, x=0\}, e_{2 \rightarrow 2}) = \{y=0\}$$
so far, check \( SAT(F_{i-1,l} \land T_e \land c') \)
Alternative Relative Inductiveness

Relative Inductiveness

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

\[
F_{i-1,l} \xrightarrow{T_e} c' \equiv F_{i-1,l} \land \text{wep}(e, c)
\]

Alternative Relative Inductiveness

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

Relative Inductiveness
- so far, check $\text{SAT}(F_{(i-1,l)} \land T_e \land c')$

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

Experimental Results
- alternative relative inductiveness check performs significantly better
- $\text{wep}(e, c)$ 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 $(A \land B)$ 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 $(A \land B)$ be unsatisfiable. An interpolant $X$ for $A$ and $B$ satisfies

- $A \Rightarrow X$,
- $X \Rightarrow \neg B,$

$A = c'$

$(B = F_{(i-1,l)} \land T_e)$

Diagram:

- $F_{(i-1,l)}$
- $c'$
- $X$
- $T_e$
Interpolation I

Idea

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

Interpolant

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

- $A \Rightarrow X$,
- $X \Rightarrow \neg B$,
- $\text{Var}(X) \subseteq \text{Var}(A) \cap \text{Var}(B)$

\[
(A = c') \\
(B = F_{(i-1,l)} \land T_e)
\]
Observation

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

- consider location 2 with self loop $e_{2\rightarrow 2}$
- IC3CFA derives proof obligations for $(y =  \hat{z} \land x = 0)$, 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

\[ \begin{align*}
\text{location 2 only reachable with at least 2 steps} \\
\text{false is safe overapproximation of reachable states } R_{(1,2)}
\end{align*} \]
Improved Initialisation

\[ y > 0 \]
\[ x = 1; x := x - 1 \]
\[ y = 0; x = 0 \]
\[ y := y + 1; y > -3 \]

<table>
<thead>
<tr>
<th>i:</th>
<th>l₀</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
<td>...</td>
<td>false</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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
try to generalise cube $c$

$F_{(i-1,l)}$ $\rightarrow_{Te}^{}$ $c'$

CTGs

Idea

try to generalise cube $c$
try to generalise cube $c$

possible generalisation $g \subseteq c$ is not relative inductive
CTGs

**Idea**

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

![Diagram](image)
Outline

1 Preliminaries

2 IC3CFA

3 Experimental Results

4 Conclusion
Evaluation I

<table>
<thead>
<tr>
<th>Algorithm</th>
<th># solved</th>
<th>t solved</th>
<th>score</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC3CFA</td>
<td>117 / 150</td>
<td>10,700 s</td>
<td>194</td>
<td>14,230 MB</td>
</tr>
<tr>
<td>IC3CFAOld</td>
<td>101 / 150</td>
<td>29,200 s</td>
<td>165</td>
<td>30,820 MB</td>
</tr>
</tbody>
</table>
Reduction of SMT calls

- best case: 895,207 → 2,924 (factor 306)
- overall: 24,700k → 249k (average factor 67)
<table>
<thead>
<tr>
<th>Algorithm</th>
<th># solved</th>
<th>t solved</th>
<th>score</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC3CFAOld</td>
<td>101 / 150</td>
<td>29,200 s</td>
<td>165</td>
<td>30,820 MB</td>
</tr>
<tr>
<td>CPAchecker</td>
<td>120 / 150</td>
<td>12,680 s</td>
<td>193</td>
<td>44,000 MB</td>
</tr>
</tbody>
</table>

### Graph

- X-axis: IC3CFAOld [s]
- Y-axis: CPAchecker [s]

- Logarithmic scale for both axes.
- Data points plotted for each algorithm, showing the relationship between execution times.
Evaluation III

<table>
<thead>
<tr>
<th>Algorithm</th>
<th># solved</th>
<th>t solved</th>
<th>score</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC3CFA</td>
<td>117 / 150</td>
<td>10,700 s</td>
<td>194</td>
<td>14,230 MB</td>
</tr>
<tr>
<td>CPAchecker</td>
<td>120 / 150</td>
<td>12,680 s</td>
<td>193</td>
<td>44,000 MB</td>
</tr>
</tbody>
</table>
Evaluation III

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

**Observation**

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

1 Preliminaries

2 IC3CFA

3 Experimental Results

4 Conclusion
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
Contributions

- ...

Implementation

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

- frame $F_i$ overapproximates $i$-step reachable program states $R_i$
Frames

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

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

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

- 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

$F(0,l_0) = \text{true}$, $\forall l \neq l_0$, $F(0,l) = \text{false}$ (initial location $l_0$)

$\forall l \in L, 0 \leq i < k$. $F(i,l) \Rightarrow F(i+1,l)$ (CFA locations $L$)

$\forall l' \in L \setminus \{l_E\}$, $e_l \rightarrow l' \in G$, $0 \leq i < k$. $F(i,l) \land T e_l \rightarrow l' \Rightarrow F'(i+1,l')$ (CFA edges $G$)

$\forall 0 \leq i \leq k$. $\neg \exists F(i,l_E)$. (error location $l_E$)
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

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