A Quick Introduction to Invariants

Definition

A logical assertion at a location over the program variables that remain same whenever the location is reached

What are they used for?

Synthesizing an invariant is a key concept in formal verification to ensure correctness of programs.

e.g. the program variables stay within some bounds

How to find Invariant?

By inferring a stronger form of the invariant i.e. inductive invariant
A Concoction of Zonotope Abstraction and Constraint Programming for finding an Invariant

Problem Statement

Related Work

Zonotopes

Future Scope

References

Example

Octagon example [Miné et al. (2015)]

\[
x := \text{input } [-1, 1] \\
y := \text{input } [-1, 1] \\
\text{while true do} \\
x' := \frac{\sqrt{2}}{2} \times (x - y) \\
y' := \frac{\sqrt{2}}{2} \times (x + y) \\
x := x' \quad y := y' \\
\text{done}
\]

\[
E := [-1, 1] \times [-1, 1] \\
\text{(entry states)}
\]

\[
F(Y) := \{(\frac{\sqrt{2}}{2} (x - y), \frac{\sqrt{2}}{2} (x + y)) | (x, y) \in Y\} \\
\text{(loop body)}
\]
Invariant versus inductive invariant

- Let us take a box $I := [-2, 2] \times [-2, 2]$
- All program states i.e. $(x, y)$ lie inside $I$ every time the execution reaches the loop head $\therefore I$ is invariant.
- $I$ is not an inductive invariant because $F(I) \not\subseteq I$ (also in next slide)
- Thus, invariants are not always inductive invariants
Method to Infer Inductive Invariant

Abstract Interpretation

- \( \text{lfp } F = \inf \{ F(G) \subseteq G \} \) [Tarski (1955)] (smallest inductive invariant)
- The well-known and dominant approach for finding inductive invariant for program verification is *abstract interpretation* [Cousot et al. (1977)]
- A classical way to find \( \text{lfp } F \) is to use Kleene iterations (\( F \) is continuous and semantic domain is complete partial order)
  \( X^0 = E, X^1 = F(X^0), \ldots, X^{k+1} = X^k \cup F(X^k) \)
- Kleene iterations on the example do converge but to a useless one \( [-\infty, \infty] \times [-\infty, \infty] \) i.e. \( (\top) \)
Inferences Drawn

- $I := [-2, 2] \times [-2, 2]$ is not an inductive invariant \( \therefore F(I) \not\subset I \)
- In fact, no box is inductive
- If we have a union of boxes $G$ with the spiky edges removed then the set is inductive (shown in figure [Miné COVERIF meeting])

- We can apply constraint solving with splitting
- Also, abstract domain to be chosen is very crucial for $F$
Algorithm: in Brief

- Replacing each operation in the program with associated operation of the chosen abstract domain
- Forming set of constraints and solving them using constraint solver based on propagation and splitting
- The algorithm starts with a single box $S$ (here $S = [-2, 2] \times [-2, 2]$) and iteratively split [Pelleau et al. (2013)] or discard until the inductive invariant properties are satisfied or may stop if the size of the boxes is small enough

Properties to infer inductive invariant

- $E \subseteq \bigcup_i S_i$ (set of boxes contains the entry)
- $S_i \subseteq I$ (set of boxes entails the invariant)
  - this particular property makes possible for the constraint solving to be applied
- $F^\#(S_i) \subseteq \bigcup_i S_i$ (set of boxes is inductive)
Based on the properties discussed earlier, boxes are classified into various categories.

**Box classes**

- **doomed**, if $F^\#(S_k) \cap (\bigcup_i S_i) = \emptyset$

  Even splitting cannot help it; are only discarded.

- **benign**, if $F^\#(S_k) \subseteq \bigcup_i S_i$

  Measure of Benign: $\text{coverage}(S_k) := \frac{\sum_i \text{vol}(F^\#(S_k) \cap S_i)}{\text{vol}(F^\#(S_k))}$ (The ultimate aim is to have $\forall k : \text{coverage}(S_k) = 1$ which implies the algorithm returns when the box has a coverage of 1)
necessary, if $S_k \cap E \neq \emptyset$

- if *necessary*, we split
- else check for usefulness (next slide) or size or coverage and decide to discard or split until $\epsilon_s$ (here it is $0.01 \times size([-2, 2] \times [-2, 2])$
Box Classification

- **useful**, if $S_k \cap (\bigcup_i F^\#(S_i)) \neq \emptyset$
- In the figure, the box $S_k$ intersects the image of a box $S_l$ under $F^\#$ which implies that $F^\#(S_l) \subseteq \bigcup_i S_i$ i.e. $S_k$ helps make $S_l$ benign
- if $S_k$ is discarded it leaves $S_l$ non-benign i.e. $F^\#(S_l) \not\subseteq \bigcup_{i \neq k} S_i$ and eventually failure
Motivation for using zonotopic abstraction

- To have a more precise $F^\#$ and also tractable
- To have less splitting
- To sum up, we are trying to see how much refinement is possible with such an abstraction when combined with constraint solving.
A Brief Introduction on Zonotopes

An affine form of a variable $x$

$$\hat{x} := \alpha_0^x + \sum_{i=1}^{n} \alpha_i^x \varepsilon_i$$

Zonotope is the geometric concretization of sets of values taken by the affine form

$$\hat{x} = 20 - 3\varepsilon_1 + 5\varepsilon_2 + 2\varepsilon_3 + 1\varepsilon_4 + 3\varepsilon_5$$

$$\hat{y} = 10 - 4\varepsilon_1 + 2\varepsilon_2 + 1\varepsilon_4 + 5\varepsilon_5$$
A Brief Introduction on Zonotopes

A Concoction of Zonotope Abstraction and Constraint Programming for finding an Invariant

Zonotope is represented by a center \( c \in \mathbb{R}^n \) and generators \( g^{(i)} \in \mathbb{R}^n \)

\[
Z = \{ x \in \mathbb{R}^n | x = c + \sum_{i=1}^{p} \varepsilon_i g^{(i)} \}
\]

\[
\hat{x} = 20 - 3\varepsilon_1 + 5\varepsilon_2 + 2\varepsilon_3 + 1\varepsilon_4 + 3\varepsilon_5
\]

\[
\hat{y} = 10 - 4\varepsilon_1 + 2\varepsilon_2 + 1\varepsilon_4 + 5\varepsilon_5
\]

\[
c = \begin{bmatrix} 20 \\ 10 \end{bmatrix}
\]

and

\[
g^{(i)} = \begin{bmatrix} -3 & 5 & 2 & 1 & 3 \\ 4 & 2 & 0 & 1 & 5 \end{bmatrix}
\]
Challenges while combining zonotope abstraction and constraint solving

Challenges

- Calculating coverage($S_k$) needs volume computation
  - Volume of zonotopes can be computed by the method in [Gover et al. (2010)]
  - However, this is costly
  - We change this into checking coverage = 1 (just an inclusion of zonotopes) and otherwise the coverage is a heuristic, we can think of estimating the size of the intersection by computing the size of the resulting $\varepsilon_i$ in intervals after using the CAV 2010 intersection

- The size computation
  - one way to characterize size would be to compute sum (or weighted sum) of the norm of the columns of generator matrix [Combastel et al. (2015)] and [Le et al. (2013)]
Challenges while combining zonotope abstraction and constraint solving

Splitting with overlap

- A zonotope can be split into two zonotopes with overlap by splitting the $j^{th}$ generator [Althoff et al. (2008)]
- We split the generator with longest length ($\|g^{(i)}\|_1$) to ensure less overlap
A Concoction of Zonotope Abstraction and Constraint Programming for finding an Invariant

Problem Statement
Related Work
Zonotopes
Future Scope
References

Challenges while combining zonotope abstraction and constraint solving

Splitting without overlap

A zonotope can be split into disjoint union of parallelotopes
Challenges while combining zonotope abstraction and constraint solving

Checking for intersection

- Let \( Z_1 = (c_1, < g_1, \cdots, g_k >) \) and \( Z_2 = (c_2, < h_1, \cdots, h_m >) \)
- \( Z_1 \cap Z_2 \neq \emptyset \) if \( c_1 - c_2 \) is entailed in \( (0, < g_1, \cdots, g_k, h_1, \cdots, h_m >) \)
- We use LP solver for checking point inclusion inside zonotope
A Concoction of Zonotope Abstraction and Constraint Programming for finding an Invariant

Bibek Kabi – Eric Goubault – Sylvie Putot

Problem Statement
Related Work
Zonotopes
Future Scope
References

Intersecting case
Non-intersecting case
Future Scope

- Use CORA Toolbox [Althoff (2015)] to develop the complete algorithm
- Using of efficient data structures (hierarchical representation of sets of zonotopes) inorder to make it faster
References


References II


