Learning Linear Invariants using Decision Trees

Siddharth Krishna
New York University

Introduction

- **Inferring invariants** for loops is a fundamental problem in program verification.
- Existing approaches (abstract interpretation, predicate abstraction, etc) are limited or incur a high complexity when it comes to inferring invariants in the form of arbitrary boolean combinations of linear inequalities.
- An invariant separates reachable states from states that lead to an error. Thus, is nothing but a **binary classifier** [Sharma et al. CAV'12].
- Thus, we can use Machine Learning.
- Our contribution: A fast, simple, and elegant learning algorithm based on Decision Trees that successfully learns invariants in the form of arbitrary boolean combinations of linear inequalities.

Preliminaries

A **Program**:

$\quad$ x $\leftarrow$ P ; $\qquad$ `/* precondition */`
$\quad$ **while** $x \in E$ **do** x $\leftarrow$ F (x);
$\quad$ `assert` (x $\in$ Q) ; $\qquad$ `/* postcondition */`

Example:

$\quad$ x $\leftarrow$ 9, y $\leftarrow$ 0 ;
$\quad$ **while** $y < 9$ **do** x $\leftarrow$ x -1, y $\leftarrow$ y $+1$;
$\quad$ `assert` (x == 0)

States: values of variables at loop head.
Good states G are all states reachable from precondition P.

$$G = \{(9,0), (8,1), (7,2), \dots\}$$

Bad states B are all states that can reach error state $\neg Q$.

$$B = \{(1,9), (0,8), (-1,9), \dots\}$$

An **invariant** is I s.t.:

- Holds at loop entry: $P \subseteq I$
- Maintained by loop: $F(I) \subseteq I$
- Implies postcondition: $I \cap \neg E \subseteq Q$

Thus, $G \subseteq I$ and $I \cap B = \emptyset$.
Our example has the invariant (Figure 1):

$$x + y = 9 \wedge x \geq 0$$

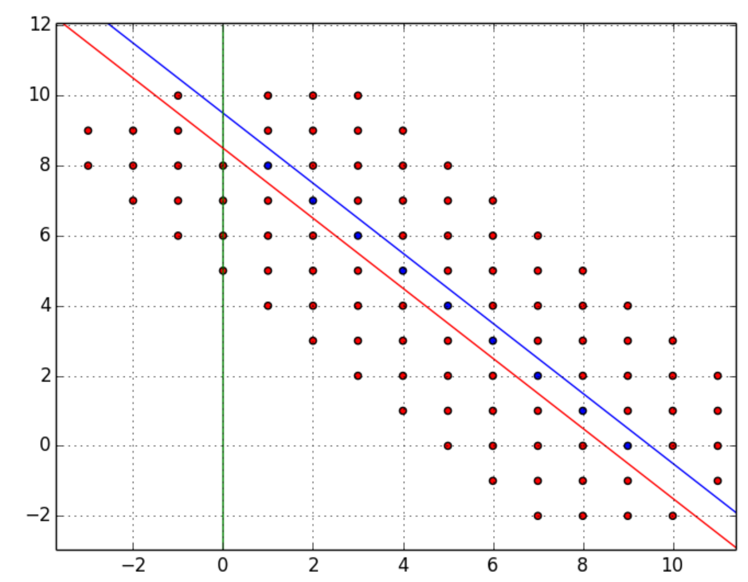

Figure 1: Good and bad states

Problem

Restrict programs to use *linear operations*. Invariant must be a *boolean combination of linear inequalities*.
Problem: Given good and bad states as sets of points in $\mathbb{Z}^d$, find a boolean combination of linear inequalities I that separate them.

Algorithm

- Choose a set of candidate hyperplane slopes $H = \{\vec{w}_1, \vec{w}_2, \dots\}$.
- Process data: new features are: $z_i = \vec{x} \cdot \vec{w}_i$.
- Run a **Decision Tree** on processed data.
- Splitting z_i at t corresponds to the linear inequality: $\vec{x} \cdot \vec{w}_i \leq t$.
- A DT is thus a boolean combination of such linear inequalities.

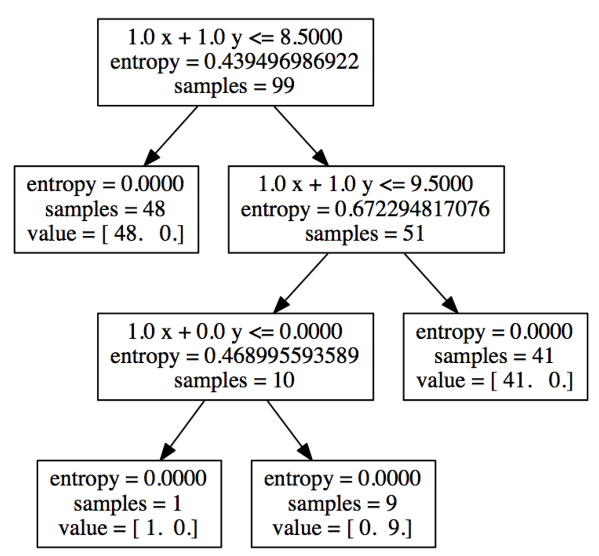

Figure 2: Decision Tree output for example program data from Figure 1. Converting this tree to a formula yields: $x+y > 8.5 \wedge x+y \leq 9.5 \wedge x >= 0$

Evaluation

- We chose benchmarks that were reported to be challenging to other tools such as ICE, MCMC, CPAchecker, InvGen, and HOLA (Table 1).
- Note: some benchmarks require disjunctions of conjunctions of inequalities, something that other invariant generation tools find hard.
- **Sampling:**
  - Good states: run program on different inputs satisfying precondition.
  - Bad states: for all points around good states, check if loop exits and assert fails.
- **Candidate hyperplanes:** we used the commonly used abstract domain of octagons : hyperplanes of the form $\pm x_i \pm x_j \geq c$.
- Correctness of invariant verified by theorem prover (we used Boogie & Z3).

Results

- Our algorithm was able to successfully find invariants for all the programs that we considered. It also was faster on most benchmarks.
- We can handle larger candidate sets H and sample sets as compared to similar ML based techniques, due to small learning complexity.

Table 1: Comparison of running times in seconds.

Name	ICE	MCMC0	MCMC1	CPA	InvGen	DT
cegar2	4.86	17.30	30.66	1.97	X	0.01
ex23	X	0.01	0.02	19.77	0.02	0.12
fig1	0.38	5.13	13.19	1.75	X	14.95
fig6	0.30	0.00	0.01	1.68	0.01	0.01
fig9	0.33	0.00	0.00	1.73	0.01	0.01
gopan	X	TO	TO	63.85	X	0.03
hola10	49.21	TO	TO	2.03	X	0.04
hola15	X	0.04	TO	X	0.02	0.52
hola18	TO	68.78	21.93	TO	X	1.63
hola19	X	TO	TO	X	X	0.19
nested2	62.02	0.09	0.15	1.86	0.03	0.04
nested5	60.95	31.28	63.68	2.08	0.03	2.47
popl07	X	TO	TO	110.81	X	0.04
prog2	0.34	0.00	0.02	4.39	0.01	0.02
prog4	X	0.13	0.58	X	X	2.34
sum1	1.32	39.81	29.04	X	X	0.02
test1	0.39	TO	TO	1.71	0.04	0.92

Examples

- Disjunction of conjunctions:

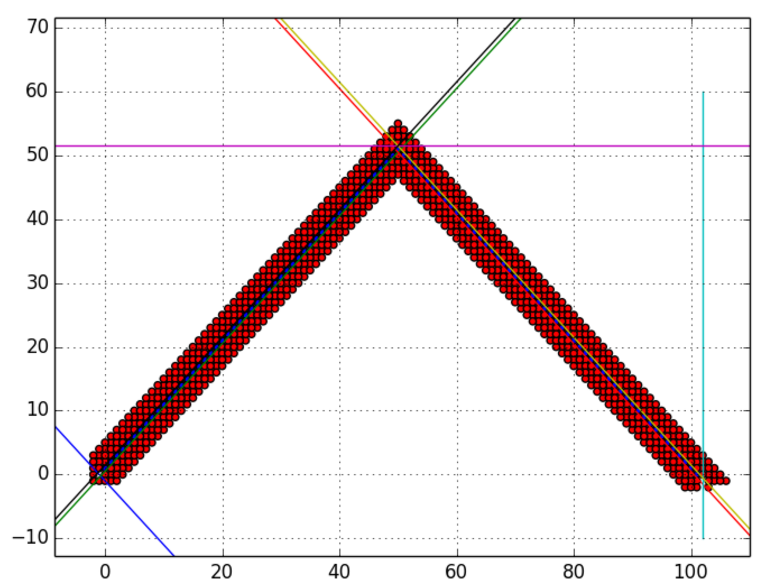

Figure 3: gopan

- Infinite reachable sets:

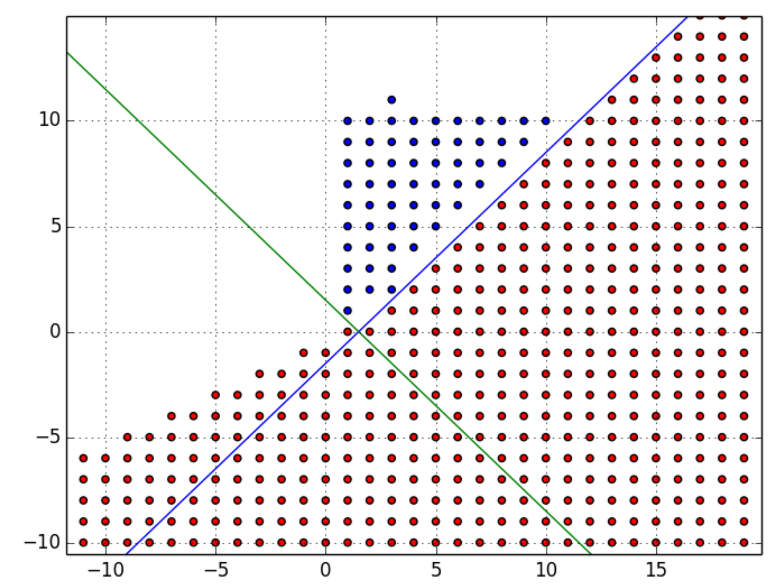

Figure 4: simple2

Ongoing & Future Work

- We have already extended our algorithm to handle non-linear functions such as mod and quadratic functions.
- Future: theoretical guarantees on convergence, make use of implication counter-examples.
- We take a long time on some benchmarks mainly due to our naive sampling.
- Future: combine with static analysis techniques to make more robust.

Joint work with:
Christian Puhrsch and Thomas Wies.