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 reachabled 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

`x ← P; /* precondition */
while x ∈ E do x ← F (x);
assert (x ∈ Q); /* postcondition */`

Example:

`x ← 9, y ← 0;
while y < 9 do x ← x - 1, y ← y + 1;
assert (x == 0)`

Good states I are all states reachable from precondition P:

`G = {(9, 0), (8, 1), (7, 2), . . . }`

Bad states B are all states that can reach error state ¬Q:

`B = {(1, 9), (0, 8), (−1, 9), . . . }`

An invariant is I s.t.

- Holds at loop entry: P ⊆ I
- Maintained by loop: F(I) ⊆ I
- Implies postcondition: I ∩ ¬E ⊆ Q

Thus, G ⊆ I and I ∩ B = ∅.

Our example has the invariant (Figure 1):

\[ x + y = 9 \land x \geq 0 \]

Problem

Restrict programs to use linear operations. Invariant must be a boolean combination of linear inequalities.

Algorithm

Choose a set of candidate hyperplane slopes:

\[ H = \{w_1, w_2, \ldots \} \]

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.

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 \( \pm x_i\pm y \geq c \).
- Correctness of invariant verified by theorem prover (we used BooQi & Z3).

Examples

- Disjunction of conjunctions:

```
+-----------------------+
<table>
<thead>
<tr>
<th>Name</th>
<th>ICE</th>
<th>MCMC0</th>
<th>MCMC1</th>
<th>CPA</th>
<th>InvGen</th>
<th>DT</th>
</tr>
</thead>
<tbody>
<tr>
<td>cegar2</td>
<td>4.86</td>
<td>17.30</td>
<td>30.66</td>
<td>1.99</td>
<td>X</td>
<td>0.01</td>
</tr>
<tr>
<td>ex23</td>
<td>0.01</td>
<td>0.02</td>
<td>19.77</td>
<td>0.02</td>
<td>0.12</td>
<td></td>
</tr>
<tr>
<td>fig1</td>
<td>0.38</td>
<td>5.31</td>
<td>17.19</td>
<td>1.75</td>
<td>X</td>
<td>14.95</td>
</tr>
<tr>
<td>fig6</td>
<td>0.30</td>
<td>0.00</td>
<td>0.01</td>
<td>1.68</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>fig9</td>
<td>0.33</td>
<td>0.00</td>
<td>0.00</td>
<td>1.73</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>gopan</td>
<td>X</td>
<td>TO</td>
<td>63.85</td>
<td>X</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>hola10</td>
<td>49.21TO</td>
<td>2.01</td>
<td>X</td>
<td>0.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hola15</td>
<td>X</td>
<td>0.04</td>
<td>TO</td>
<td>X</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>hola18</td>
<td>TO</td>
<td>68.78</td>
<td>TO</td>
<td>1.63</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hola19</td>
<td>X</td>
<td>TO</td>
<td>X</td>
<td>X</td>
<td>0.19</td>
<td></td>
</tr>
<tr>
<td>nestle2</td>
<td>62.02</td>
<td>0.09</td>
<td>0.15</td>
<td>1.86</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>nestle5</td>
<td>60.95</td>
<td>31.28</td>
<td>63.68</td>
<td>2.08</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>pop87</td>
<td>X</td>
<td>TO</td>
<td>110.81</td>
<td>X</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>prop2</td>
<td>0.34</td>
<td>0.00</td>
<td>4.59</td>
<td>0.01</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>prop4</td>
<td>X</td>
<td>0.13</td>
<td>0.58</td>
<td>X</td>
<td>2.34</td>
<td></td>
</tr>
<tr>
<td>sum1</td>
<td>1.32</td>
<td>39.81</td>
<td>29.04</td>
<td>X</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>test1</td>
<td>0.39</td>
<td>TO</td>
<td>1.71</td>
<td>0.04</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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

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.