Learning Abstractions for Model Checking

Anubhav Gupta
Cadence Berkeley Labs

Overview

- Abstraction for Model Checking $\equiv$ Inductive Learning
  - Learning and Abstraction-Refinement
  - Learning Abstractions without Refinement
Outline

- Machine Learning
- Abstraction
- Learning and Abstraction-Refinement
- Learning Abstractions without Refinement

Machine Learning

- Process that causes a system to improve its performance at a particular task with experience [Mitchell]
Inductive Learning

\[ S \]
\[ \langle x_1, c(x_1) \rangle; \]
\[ \langle x_2, c(x_2) \rangle; \]
\[ \vdots \]
\[ \langle x_k, c(x_k) \rangle \]

\[ f : X \rightarrow C \]

\[ f \in F \]

Classifier

\[ f(x) \]

Generalize

Predict

Inductive Learning: Generalizing from Samples

Inductive Bias

- Generalization requires bias towards certain target functions
  - Completely Unbiased Learner: Learning boolean functions by memorization
- Inductive bias captures the domain-specific assumptions that help in classifying unseen instances
- Two forms on inductive biases:
  - Restriction Bias: Set of candidate functions is restricted
  - Preference Bias: Certain functions preferred over others
Generating Samples

- **Random Sampling:** Training set provided to learner
- **Queries:** Learner asks teacher specific questions about the target function to generate samples
  - **Membership queries**
    - Input: Object
    - Output: Classification
  - **Equivalence queries**
    - Input: Target function
    - Output: Done or Misclassified object with classification

Outline

- **Machine Learning**
- **Abstraction**
- **Learning and Abstraction-Refinement**
- **Learning Abstractions without Refinement**
Model Checking

Model Checking for Safety Properties

State-Explosion Problem

Too many states to handle
Abstraction

Abstraction Function $h : S \rightarrow \hat{S}$

Preserves all the behaviors of the concrete model
Abstraction

- **Preservation Theorem**: If property holds on abstract model then property holds on concrete model

- **Abstraction For Model Checking**: Find a small abstract model on which the property holds
Abstraction Functions

- Candidate abstraction functions are implicitly defined by the technique used for constructing abstract models.
- Two popular techniques:
  - Predicate Abstraction
  - Localization Abstraction

Localization Abstraction

- Partition state variables into visible ($V$) and invisible ($I$) variables:
  - Intuitively, visible variables are the important variables.
- Abstract model consists of only the visible variables.
- Abstraction function maps a concrete state to its projection onto the visible variables.
Abstraction Functions for Localization

\[ V = \{ x_1, x_2, x_3, x_4, x_5, x_6, x_7 \} \]
\[ \mathcal{V} = \{ x_1, x_2, x_4, x_6 \} \]
\[ I = \{ x_3, x_5, x_7 \} \]

Concrete states having the same value for visible variables are mapped to same abstract state.

Localization Abstraction for Circuits

\[ V = \{ x_1, x_2, x_3, x_4, x_5, x_6, x_7 \} \]
\[ \mathcal{V} = \{ x_1, x_2, x_4, x_6 \} \]
\[ I = \{ x_3, x_5, x_7 \} \]

Hence the name localization.
**Abstraction \equiv Inductive Learning**

\[ h : S \rightarrow \hat{S} \]

\[ s \in S \rightarrow h(x) \]

\[ h \in H \]

Goal of abstraction is to learn an abstraction function that classifies the concrete states into abstract states while preserving the property

**Inductive Bias of Abstraction**

- **Restriction Bias**
  - Number of possible abstraction functions is huge
    - Circuit with \( n \) boolean variables
    - Number of ways to partition \( 2^n \) states into disjoint subsets
    - Bell Number
      \[ B_{2^n} \gg 2^{2^n} \]
  - Number of candidate functions is usually much smaller
    - Localization Abstraction: \( 2^n \) abstraction functions
  - Captures domain knowledge: Property is localizable

- **Preference Bias**
  - Smaller abstract models are better
Samples

- What are the samples?
- How are the samples generated?
- How is the abstraction function computed from the samples?

Outline

- Machine Learning
- Abstraction
- Learning and Abstraction-Refinement
- Learning Abstractions without Refinement
Refinement

For localization, refinement corresponds to making more variables visible.

Abstraction-Refinement Loop

- Abstract
- Model Check
- Refine
- Check Counterexample

- Pass: No Bug
- Fail: Spurious, Real Bug

- SAT-Based Concretization
Many Refinement Heuristics

- Identifying conflicting latches with 3-valued simulation of counterexample [Wang et. al.]
- Identify common variable assignments across multiple counterexamples [Glusman et. al.]
- SAT-Proof based refinement [Chauhan et. al.]
- Refinement by failure-state splitting [Yuan Lu et. al.]

Refinement

Put $d\delta(s_f) = \{ R(s_f, s_{f+1}) \land C_f(s_f) \land C_{f+1}(s_{f+1}) \}$
**Refinement**

Put deadend and bad states in separate abstract states

**State-Separation Problem**

Refinement: Find subset $U$ of $I$ that separates all pairs of dead and bad states, while making the visible state separable.
A Simple Approach

- Generate all the deadend and bad states
  - Explicitly
  - Symbolically
- Compute the separating set from these
- Previous work [Yuan Lu et. al.] generated BDDs for deadend and bad states

- Infeasible for large systems

Sampling

- Learn the separating set from samples of deadend and bad states
- Use SAT-solvers to generate multiple samples efficiently
Learning and Abstraction-Refinement

\[ S_D \cup S_B \]

\[ \begin{array}{cc}
d_1 & b_1 \\
d_2 & b_2 \\
\vdots & \vdots \\
d_p & b_q \\
\end{array} \]

\[ h : S \rightarrow \tilde{S} \]

\[ s \in S \]

\[ h(s) \]

**Computing the Separating Set**

- **Integer Linear Programming (ILP)**
  - Smallest separating set
  - Computationally expensive

- **Decision Tree Learning**
  - Computationally efficient
  - Non-optimal
Computing Separating Set using ILP

\[ \text{Min } \sum_{i=1}^{\left| \mathcal{I} \right|} v_i \]

subject to: \((\forall d \in S_D) \ (\forall b \in S_B) \ \sum_{1 \leq i \leq |\mathcal{I}|, \text{ } d,b \text{ differ at } v_i} v_i \geq 1 \]

\[ v_i = 1 \text{ means that } v_i \text{ is in the separating set} \]

Computing Separating Set using Decision Tree Learning

- Decision Tree Learning constructs a decision tree that classifies a set of samples using a set of attributes
- Samples: \( S_D \cup S_B \)
- Attributes: \( \mathcal{I} \)
- ID3 algorithm
  - Construct small tree
- Separating set consists of variables on the nodes of the decision tree

Separating Set \( \{x_1, x_2, x_4\} \)
Generating Samples

- **Random Sampling**
  - Generate multiple satisfying assignments using SAT-solver on $\mathcal{D}$ and $\mathcal{B}$

- **Equivalence Queries**
  - Query the teacher for samples that are not separated by the current separating set
  - Teacher:
    \[
    \Phi(Sep) \equiv \mathcal{D}(v_i) \land \mathcal{B}(v'_i) \land \bigwedge_{v_i \in Sep} v_i = v'_i
    \]

---

Sampling with Equivalence Queries

- $Sep = \{\}$
  - Run SAT-solver on $\Phi(Sep)$
    - unsatisfiable: STOP
    - satisfiable: add to sample set
    - Compute new $Sep$

Generating Good Samples

- Deadend and bad state pairs that differ in small number of variables are good
  - Eliminate a larger portion of the search space
  - Faster convergence to the separating set
- Can be formulated as optimization problem with Pseudo-Boolean Constraints
  - Solved with Pseudo-Boolean Solver (PBS)

Metrics for Quality of Abstract Models

- Number of State Variables
- Number of Gates
- Number of Inputs
Experimental Evaluation

- **ABSREF Tool**
  - Implemented inside NuSMV
  - SAT-solver: zChaff
  - ILP-solver: lpsolve
  - Model Checker: Cadence SMV

- Compared with
  - BDD-based Model Checking (Cadence SMV)
  - SAT-Proof based refinement [Chauhan et. al.]

### Results

<table>
<thead>
<tr>
<th>Circuit</th>
<th>SMV</th>
<th>Rand, ILP</th>
<th>Rand, DTL</th>
<th>EqvQ, DTL</th>
<th>Chauhan</th>
<th>EqvQ, Inp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>S</td>
<td>L</td>
<td>Time</td>
<td>S</td>
<td>L</td>
</tr>
<tr>
<td>1/140</td>
<td>1.5</td>
<td>84</td>
<td>54</td>
<td>1.7</td>
<td>21</td>
<td>1.9</td>
</tr>
<tr>
<td>1/430</td>
<td>19.1</td>
<td>11.4</td>
<td>17</td>
<td>21</td>
<td>11.9</td>
<td>21</td>
</tr>
<tr>
<td>1/440</td>
<td>23.9</td>
<td>25.9</td>
<td>20</td>
<td>22</td>
<td>10.1</td>
<td>22</td>
</tr>
<tr>
<td>1/445</td>
<td>28.1</td>
<td>28.1</td>
<td>25</td>
<td>22</td>
<td>22.1</td>
<td>22</td>
</tr>
<tr>
<td>1/460</td>
<td>1754</td>
<td>160.4</td>
<td>13</td>
<td>22</td>
<td>55.1</td>
<td>10</td>
</tr>
<tr>
<td>1/465</td>
<td>61.9</td>
<td>4</td>
<td>20</td>
<td>56.34</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>1/470</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1/475</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1/480</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1/485</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1/490</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Design</th>
<th>Length</th>
<th>Time</th>
<th>S</th>
<th>L</th>
<th>Time</th>
<th>S</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>M9</td>
<td>TRUE</td>
<td>10.2</td>
<td>2</td>
<td>36</td>
<td>29</td>
<td>1</td>
<td>38</td>
</tr>
<tr>
<td>M6</td>
<td>TRUE</td>
<td>44.9</td>
<td>4</td>
<td>20</td>
<td>18.4</td>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>M16</td>
<td>TRUE</td>
<td>1162</td>
<td>61</td>
<td>35</td>
<td>41.7</td>
<td>3</td>
<td>34</td>
</tr>
<tr>
<td>M17</td>
<td>TRUE</td>
<td>783</td>
<td>0</td>
<td>39</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D6</td>
<td>20</td>
<td>917</td>
<td>40</td>
<td>89</td>
<td>1773</td>
<td>43</td>
<td>92</td>
</tr>
<tr>
<td>IUD1</td>
<td>TRUE</td>
<td>3390</td>
<td>13</td>
<td>19</td>
<td>-</td>
<td>9</td>
<td>41</td>
</tr>
</tbody>
</table>
Outline

- Machine Learning
- Abstraction
- Learning and Abstraction-Refinement
- Learning Abstractions without Refinement

Many Refinement Heuristics

- Identifying conflicting latches with 3-valued simulation of counterexample [Dong Wang et. al.]
- Idee! All of these are Heuristics! Cross multiple counterexamples [Glusman et. al.]
- SAT-Proof based refinement [Chauhan et. al.]
- Refinement by failure-state analysis [Yuan Lu et. al.]
Many Refinement Heuristics

- Identifying conflicting latches with 3-valued simulation of counterexample [Dong Wang et. al.]
- All of these are Heuristics! [Glusman et. al.]
- SAT-Proof based refinement [Chauhan et. al.]
- Refinement by failure-state analysis [Yuan Lu et. al.]

Drawbacks of Failure-State Splitting

\[ \mathcal{V} = \{x\} \quad \mathcal{I} = \{y, z\} \]

Separating Set: \( \{y, z\} \)
Drawbacks of Failure-State Splitting

$\mathcal{V} = \{x, y, z\}$

Drawbacks of Failure-State Splitting

$\mathcal{V} = \{x, y\}$
Drawbacks of Abstraction-Refinement

- Adds details to abstract model; never removes anything
  - Information added to eliminate counterexample might also eliminate previously seen counterexample
- Does not look at many counterexamples of different lengths simultaneously
  - Abstract model depends on what counterexamples are considered and in what order
- Abstraction-Refinement cannot find the smallest abstract model
- This drawback is present no matter what heuristic is used to compute the refinement

What is needed?

- We need a strategy of eliminating spurious behavior that is not heuristic
- We need a strategy that is not based on refinement
- We need a strategy that analyzes all the counterexamples simultaneously
Broken Traces

- Broken Traces on concrete model corresponding to an abstraction function
- Sequence of $k$ pairs of concrete states
  $$\langle(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\rangle$$
- Each $s_i$ and $t_i$ map to same abstract state
- $s_1$ is an initial state
- $t_k$ is an error state
- Each $t_i \rightarrow s_{i+1}$ is a concrete transition
- Break at $i$ if $s_i \neq t_i$. No breaks = Real bug
Broken Traces and Abstract Counterexamples

- **Broken Trace Theorem:** There is a counterexample on the abstract model if and only if there is a corresponding broken trace on the concrete model.
Eliminating Broken Traces

- Abstraction function eliminates a broken trace if it maps some \( s_i \) and \( t_i \) into separate abstract states.

Our Abstraction Strategy

- Find an abstraction function that eliminates all broken traces.
- The smallest abstract model that eliminates all broken traces is the smallest abstract model that can prove the property.
Sampling

- Computationally infeasible to generate all broken traces and eliminate them
- Learn the abstraction function from samples of broken traces
- Use abstract counterexamples to guide the search for broken trace samples

Learning Abstractions without Refinement

\[ S_T \]
\[ t_1 \]
\[ t_2 \]
\[ \cdot \]
\[ \cdot \]
\[ t_p \]

\[ h : S \rightarrow \hat{S} \]

Classifier

\[ s \in S \]
\[ h(s) \]

\[ \forall t \in T. \ h \text{ eliminates } t \]
Learning Abstractions (LearnAbs)

![Diagram showing the process of learning abstractions.]

1. Broken Trace
2. Samples
3. Broken Traces
4. Abstract Model
5. Eliminating Function
6. Property Holds
7. Real Bug

Computing the Eliminating Model

\[ ((s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)) \]

**Eliminating Set for the Broken Trace**

\[ s_2 \]

\[ t_2 \]

- 0 1 1 0 1 0 0 1 1 0 1
- 0 1 1 0 0 1 0 1 1 0 0

57
Computing Eliminating Model

Find subset \( V \) of variables that hits the eliminating set of all broken trace samples

- Minimum Hitting Set
  - Can be formulated as an Integer Linear Program
  - Smallest Eliminating Model

- Approximate algorithms
  - Faster but non-optimal
SAT with Hints

- SAT-solver modified to produce a satisfying assignment that is close to a given partial assignment (hint)
  - SAT-solver is forced to first make decisions corresponding to the hint
Generating Broken Traces

- Use SAT with hints
  - Hints from previous state
- Break if necessary
- No expensive BMC unfolding

Experimental Evaluation

- **LEARNABS Tool**
  - Input: Bit-level SMV net-lists
  - SAT-solver: zChaff
  - ILP-solver: CPLEX
  - Model Checker: Cadence SMV

- **Compared with**
  - SAT-Proof based abstraction [Chauhan et. al., McMillan et. al.]
    - Single Counterexample (S) mode: Model Checker called after abstract counterexample is eliminated
    - All Counterexamples (A) mode: Model Checker called after all counterexamples of current length are eliminated
## Results

<table>
<thead>
<tr>
<th>circuit</th>
<th>reg</th>
<th>cox</th>
<th>SATProof (s)</th>
<th>LearnAbs (s)</th>
<th>SATProof (A)</th>
<th>LearnAbs (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>ltr</td>
<td>abs</td>
<td>time</td>
<td>ltr</td>
<td>abs</td>
</tr>
<tr>
<td>P000</td>
<td>348</td>
<td>T</td>
<td>3</td>
<td>9</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>P010</td>
<td>321</td>
<td>T</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>P020</td>
<td>306</td>
<td>T</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>P030</td>
<td>306</td>
<td>T</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>P040</td>
<td>305</td>
<td>T</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>P050</td>
<td>105</td>
<td>T</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>P060</td>
<td>328</td>
<td>T</td>
<td>709</td>
<td>7</td>
<td>56</td>
<td>7</td>
</tr>
<tr>
<td>P070</td>
<td>94</td>
<td>T</td>
<td>18</td>
<td>11</td>
<td>36</td>
<td>13</td>
</tr>
<tr>
<td>P080</td>
<td>116</td>
<td>T</td>
<td>58</td>
<td>6</td>
<td>41</td>
<td>7</td>
</tr>
<tr>
<td>P090</td>
<td>71</td>
<td>T</td>
<td>8</td>
<td>6</td>
<td>31</td>
<td>4</td>
</tr>
<tr>
<td>P100</td>
<td>85</td>
<td>T</td>
<td>8</td>
<td>6</td>
<td>31</td>
<td>4</td>
</tr>
<tr>
<td>P110</td>
<td>704</td>
<td>T</td>
<td>11</td>
<td>5</td>
<td>17</td>
<td>5</td>
</tr>
<tr>
<td>P120</td>
<td>517</td>
<td>T</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P130</td>
<td>420</td>
<td>T</td>
<td>10</td>
<td>5</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>P140</td>
<td>177</td>
<td>T</td>
<td>7</td>
<td>6</td>
<td>97</td>
<td>9</td>
</tr>
<tr>
<td>P150</td>
<td>368</td>
<td>T</td>
<td>184</td>
<td>4</td>
<td>47</td>
<td>14</td>
</tr>
<tr>
<td>P160</td>
<td>290</td>
<td>T</td>
<td>238</td>
<td>6</td>
<td>44</td>
<td>14</td>
</tr>
<tr>
<td>P170</td>
<td>217</td>
<td>T</td>
<td>2126</td>
<td>14</td>
<td>43</td>
<td>138</td>
</tr>
<tr>
<td>P180</td>
<td>145</td>
<td>T</td>
<td>993</td>
<td>22</td>
<td>49</td>
<td>35</td>
</tr>
<tr>
<td>P190</td>
<td>57</td>
<td>T</td>
<td>2</td>
<td>&gt;2hr</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

## Results

<table>
<thead>
<tr>
<th>circuit</th>
<th>reg</th>
<th>cox</th>
<th>SATProof (s)</th>
<th>LearnAbs (s)</th>
<th>SATProof (A)</th>
<th>LearnAbs (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>ltr</td>
<td>abs</td>
<td>time</td>
<td>ltr</td>
<td>abs</td>
</tr>
<tr>
<td>RO05</td>
<td>513</td>
<td>31</td>
<td>11</td>
<td>10</td>
<td>24</td>
<td>141</td>
</tr>
<tr>
<td>RO10</td>
<td>168</td>
<td>T</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>RO15</td>
<td>236</td>
<td>T</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>RO20</td>
<td>236</td>
<td>T</td>
<td>3</td>
<td>6</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>RO25</td>
<td>36</td>
<td>T</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>RO30</td>
<td>262</td>
<td>T</td>
<td>1</td>
<td>&gt;2hr</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RB10</td>
<td>180</td>
<td>T</td>
<td>31</td>
<td>&gt;1hr</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RB15</td>
<td>180</td>
<td>1</td>
<td>&gt;1hr</td>
<td>1258</td>
<td>60</td>
<td>34</td>
</tr>
<tr>
<td>RO50</td>
<td>270</td>
<td>5</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>RO60</td>
<td>1117</td>
<td>T</td>
<td>8</td>
<td>7</td>
<td>99</td>
<td>394</td>
</tr>
<tr>
<td>RO80</td>
<td>1113</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>40</td>
<td>61</td>
</tr>
<tr>
<td>RO100</td>
<td>408</td>
<td>T</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>RO120</td>
<td>141</td>
<td>1</td>
<td>&gt;1hr</td>
<td>128</td>
<td>26</td>
<td>25</td>
</tr>
<tr>
<td>RP10</td>
<td>4404</td>
<td>1</td>
<td>mem</td>
<td>1285</td>
<td>10</td>
<td>8</td>
</tr>
</tbody>
</table>
Conclusion

- This work has shown the viability of using machine learning techniques to improve abstraction-based model checking
  - Machine learning techniques help the model checker to efficiently identify the relevant information in the model