Learning Abstractions for Model Checking

Anubhav Gupta
Cadence Berkeley Labs
Overview

- Abstraction for Model Checking ≡ 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

Generalize

Predict

\[ x \]

\[ f(x) \]

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
Model Checking

Model Checking for Safety Properties
State-Explosion Problem

Too many states to handle
Abstraction

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

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

Abstract model may exhibit spurious behavior. Try another abstraction function.
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 (\(\mathcal{V}\)) and invisible (\(\mathcal{I}\)) 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

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

\[ \mathcal{V} = \{x_1, x_2, x_4, x_6\} \]

\[ \mathcal{I} = \{x_3, x_5, x_7\} \]

Concrete states having the same value for visible variables are mapped to same abstract state
Localization Abstraction for Circuits

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

Hence the name localization
Abstraction $\equiv$ Inductive Learning

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
      \[ B_{2n} \gg 2^{2^n} \]
  - 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
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.

$h'$ is a refinement of $h$. 
Abstraction-Refinement Loop

- **Abstract**
  - $M, h$

- **Model Check**
  - $\hat{M}$
  - Fail
  - **Refine**
    - $h'$
    - Spurious
    - Check
    - Counterexample
    - Real
    - Bug

- **Check**
  - Pass
  - No 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.]
Put $\mathcal{B}(s_f) = \{ R(s_f, s_{f+1}) \land C_f(s_f) \land C_{f+1}(s_{f+1}) \}_{i=1}^{f-1} \land \{ C_f(s_f) \land C_{f+1}(s_{f+1}) \}_{i=1}^{f}$.
Refinement

Put deadend and bad states in separate abstract states
State-Separation Problem

Refinement: Find subset $u$ of $I$ that separates all pairs of deadend and bad states and make the visible states separate the set.
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

\[ S_D \cup S_B \]

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

\[ h : S \rightarrow S' \]

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

Classifier

\[ \forall d \in D. b \in B. \quad h(d) \neq h(b) \]
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

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

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

\(v_i = 1\) means that \(v_i\) 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
$\{v_1, v_2, v_4\}$
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

\[ Sep = \{\} \]

1. Run SAT-solver on \( \Phi(Sep) \)
   - Unsatisfiable: STOP
   - Satisfiable: Add to sample set

2. 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: Ip.solve
- 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>IU30</td>
<td>7.3</td>
<td>8.0</td>
<td>3</td>
<td>20</td>
<td>7.5</td>
<td>3</td>
</tr>
<tr>
<td>IU35</td>
<td>19.1</td>
<td>11.8</td>
<td>4</td>
<td>21</td>
<td>12.7</td>
<td>4</td>
</tr>
<tr>
<td>IU40</td>
<td>53.6</td>
<td>25.9</td>
<td>6</td>
<td>23</td>
<td>19.0</td>
<td>5</td>
</tr>
<tr>
<td>IU45</td>
<td>226.1</td>
<td>28.3</td>
<td>5</td>
<td>22</td>
<td>25.3</td>
<td>5</td>
</tr>
<tr>
<td>IU50</td>
<td>1754</td>
<td>160.4</td>
<td>13</td>
<td>32</td>
<td>85.1</td>
<td>10</td>
</tr>
<tr>
<td>IU55</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IU60</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IU65</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IU70</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IU75</td>
<td>-</td>
<td>1080</td>
<td>21</td>
<td>38</td>
<td>586.7</td>
<td>16</td>
</tr>
<tr>
<td>IU80</td>
<td>-</td>
<td>1136</td>
<td>21</td>
<td>38</td>
<td>552.5</td>
<td>16</td>
</tr>
<tr>
<td>IU85</td>
<td>-</td>
<td>1162</td>
<td>21</td>
<td>38</td>
<td>581.2</td>
<td>16</td>
</tr>
<tr>
<td>IU90</td>
<td>-</td>
<td>965</td>
<td>20</td>
<td>37</td>
<td>583.3</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Design</th>
<th>Line</th>
<th>Chauhan</th>
<th>EqvQ, Inp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>S</td>
<td>L</td>
</tr>
<tr>
<td>M9</td>
<td>TRUE</td>
<td>10.2</td>
<td>2</td>
</tr>
<tr>
<td>M6</td>
<td>TRUE</td>
<td>44.3</td>
<td>4</td>
</tr>
<tr>
<td>M16</td>
<td>TRUE</td>
<td>1162</td>
<td>61</td>
</tr>
<tr>
<td>M17</td>
<td>TRUE</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>D6</td>
<td>20</td>
<td>917</td>
<td>46</td>
</tr>
<tr>
<td>IUc1</td>
<td>TRUE</td>
<td>3530</td>
<td>13</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.]
- All of these are Heuristics!
- Identify common variable assignments across 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! Identifying common variable assignments across multiple counterexamples [Glusman et. al.]
- SAT-Proof based refinement [Chauhan et. al.]
- Refinement by failure-state analysis [Yuan Lu et. al.]
Drawbacks of Failure-State Splitting

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

\[ (x) \quad (0) \quad (1) \quad (2) \quad (3) \quad (4) \]

Failure State

Deadend State

Bad States

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

\[ \mathcal{V} = \{x, y, z\} \]
Drawbacks of Failure-State Splitting

\[ \nu = \{x, y\} \]

\[
\begin{array}{cccccc}
(x, y) & (0, 0) & (1, 0) & (2, 0) & (3, 0) & (4, 0) \\
(z) & & & & & \\
(0) & & & & & \\
(1) & & & & & \\
(0) & & & & & \\
(1) & & & & & \\
(x, y) & (0, 1) & (1, 1) & (2, 1) & (3, 1) & (4, 1) \\
\end{array}
\]
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
\[\langle (s_1, t_1), (s_2, t_2), (s_3, t_3), (s_4, t_4), (s_5, t_5) \rangle\]
Broken Traces And Abstract Counterexamples
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

\[ S_T \]

\[ t_1 \]

\[ t_2 \]

\[ \cdot \]

\[ \cdot \]

\[ \cdot \]

\[ t_p \]

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

\[ \forall t \in T. \ h \text{ eliminates } t \]

Classifier

\[ s \in S \Rightarrow h(s) \text{ Predict} \]
Learning Abstractions (LearnAbs)

- Broken Trace
- Samples
- Abstract Model
- Eliminating Abstraction Function
- Property Holds
- Real Bug
- Broken Traces
Computing the Eliminating Model

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

Eliminating Set for the Broken Trace
Computing Eliminating Model

Find subset $\mathcal{V}$ of variables that hits the eliminating set of all broken trace samples.

Find the smallest $\mathcal{V}$. 

$T_1$

$T_2$

$T_3$

$T_4$
Computing Eliminating Model

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

- **Approximate algorithms**
  - Faster but non-optimal
Learning Abstractions (LearnAbs)

- Eliminating Abstraction Function
  - Broken Traces
  - Abstract Model
  - Property Holds
  - Real Bug
  - Broken Trace Samples
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>cex</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>itr</td>
<td>abs</td>
<td>time</td>
<td>itr</td>
<td>abs</td>
</tr>
<tr>
<td>PJ00</td>
<td>348</td>
<td>T</td>
<td>7</td>
<td>6</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>PJ01</td>
<td>321</td>
<td>T</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>PJ02</td>
<td>306</td>
<td>T</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>PJ03</td>
<td>306</td>
<td>T</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>PJ04</td>
<td>305</td>
<td>T</td>
<td>7</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>PJ05</td>
<td>105</td>
<td>T</td>
<td>9</td>
<td>35</td>
<td>8</td>
<td>28</td>
</tr>
<tr>
<td>PJ06</td>
<td>328</td>
<td>T</td>
<td>209</td>
<td>23</td>
<td>7</td>
<td>41</td>
</tr>
<tr>
<td>PJ07</td>
<td>94</td>
<td>T</td>
<td>18</td>
<td>13</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>PJ08</td>
<td>116</td>
<td>T</td>
<td>58</td>
<td>75</td>
<td>7</td>
<td>41</td>
</tr>
<tr>
<td>PJ09</td>
<td>71</td>
<td>T</td>
<td>8</td>
<td>4</td>
<td>6</td>
<td>25</td>
</tr>
<tr>
<td>PJ10</td>
<td>85</td>
<td>T</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>PJ11</td>
<td>294</td>
<td>T</td>
<td>11</td>
<td>5</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>PJ12</td>
<td>312</td>
<td>T</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>PJ13</td>
<td>420</td>
<td>T</td>
<td>10</td>
<td>8</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>PJ14</td>
<td>127</td>
<td>T</td>
<td>25</td>
<td>9</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>PJ15</td>
<td>355</td>
<td>T</td>
<td>184</td>
<td>14</td>
<td>5</td>
<td>23</td>
</tr>
<tr>
<td>PJ16</td>
<td>290</td>
<td>T</td>
<td>248</td>
<td>64</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>PJ17</td>
<td>212</td>
<td>T</td>
<td>2126</td>
<td>1869</td>
<td>20</td>
<td>29</td>
</tr>
<tr>
<td>PJ18</td>
<td>145</td>
<td>T</td>
<td>993</td>
<td>390</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>PJ19</td>
<td>52</td>
<td>T</td>
<td>&gt;2hr</td>
<td>18</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>circuit</td>
<td>reg</td>
<td>ccx</td>
<td>SATProof (S)</td>
<td>LearnAbs(S)</td>
<td>SATProof (A)</td>
<td>LearnAbs(A)</td>
</tr>
<tr>
<td>---------</td>
<td>-----</td>
<td>-----</td>
<td>--------------</td>
<td>-------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>time</td>
<td>itr</td>
<td>abs</td>
<td>time</td>
</tr>
<tr>
<td><strong>RB05 1</strong></td>
<td>313</td>
<td>31</td>
<td>11</td>
<td>10</td>
<td>24</td>
<td>141</td>
</tr>
<tr>
<td><strong>RB09 1</strong></td>
<td>168</td>
<td>T</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td><strong>RB10.1</strong></td>
<td>236</td>
<td>T</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td><strong>RB10.2</strong></td>
<td>236</td>
<td>T</td>
<td>3</td>
<td>6</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td><strong>RB10.3</strong></td>
<td>236</td>
<td>T</td>
<td>5</td>
<td>7</td>
<td>34</td>
<td>2</td>
</tr>
<tr>
<td><strong>RB10.4</strong></td>
<td>236</td>
<td>T</td>
<td>8</td>
<td>10</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td><strong>RB10.5</strong></td>
<td>236</td>
<td>T</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td><strong>RB10.6</strong></td>
<td>236</td>
<td>T</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td><strong>RB11.2</strong></td>
<td>242</td>
<td>T</td>
<td>&gt;1hr</td>
<td>-</td>
<td>-</td>
<td>128</td>
</tr>
<tr>
<td><strong>RB14.1</strong></td>
<td>180</td>
<td>T</td>
<td>37</td>
<td>7</td>
<td>47</td>
<td>3</td>
</tr>
<tr>
<td><strong>RB14.2</strong></td>
<td>180</td>
<td>T</td>
<td>&gt;1hr</td>
<td>-</td>
<td>-</td>
<td>1258</td>
</tr>
<tr>
<td><strong>RB15.1</strong></td>
<td>270</td>
<td>9</td>
<td>2</td>
<td>7</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td><strong>RB16.1 1</strong></td>
<td>1117</td>
<td>8</td>
<td>7</td>
<td>92</td>
<td>324</td>
<td>8</td>
</tr>
<tr>
<td><strong>RB16.2 4</strong></td>
<td>1113</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>40</td>
<td>61</td>
</tr>
<tr>
<td><strong>RB26 1</strong></td>
<td>608</td>
<td>T</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>RB31 2 1</strong></td>
<td>111</td>
<td>T</td>
<td>&gt;1hr</td>
<td>-</td>
<td>-</td>
<td>38</td>
</tr>
<tr>
<td><strong>IUPI</strong></td>
<td>4494</td>
<td>T</td>
<td>mem</td>
<td>1295</td>
<td>18</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