Fast Numerical Program Analysis with Reinforcement Learning

Gagandeep Singh  Markus Püschel  Martin Vechev

Department of Computer Science
ETH Zürich
ML-Based Solvers

Many solvers and analyzers are based on heuristics.

Trade-off precision vs. scalability.

Key idea: apply machine learning to learn optimal strategy.

This work: ML for numerical static analysis.
Reinforcement learning for analysis

Linux device driver
> 2,300 program points

ELINA
state-of-the-art Polyhedra analysis
fixed point
analysis time: 877 sec

= 

Poly-RL
RL-based analysis
fixed point
analysis time: 6.6 sec
Static analysis: trade-offs

$T_{fast}$  

$T_{precise}$  

Precise fixpoint  

Imprecise fixpoint
Static analysis: trade-offs

- $T_{fast}$
- $T_{precise}$

Precise fixpoint
Imprecise fixpoint

Slow, but precise
Static analysis: trade-offs

Fast, but imprecise

\( T_{fast} \)

\( S_0 \)

\( T_{precise} \)

Precise fixpoint

Imprecise fixpoint

Slow, but precise
Static analysis: trade-offs

Goal: find such a sequence of transformers
Reinforcement learning

Learn a policy that for any state selects the action maximizing long term rewards
Reinforcement learning for analysis

Learn a policy that for any state selects the action maximizing long term rewards

Learn a policy that for any abstract state selects the transformer maximizing speed and precision of analysis
Agents maintains a function $Q : S \times \mathcal{A} \rightarrow \mathbb{R}$

\[ Q(s, a) = \sum_{j=1}^{l} \theta_j \phi_j(s, a) \]

$\phi_j$ are features on (state, action) pairs
$\theta_j$ are parameters to be learned via Q-learning,
learning uses reward function $r(s_t, a_t, s_{t+1})$

\[ a_t = \text{argmax}_{a \in \mathcal{A}} Q(s_t, a) \]
Step 1: Define space of transformers $\mathcal{A}$

Input state $s$

$\begin{align*}
& x_1 - x_2 + x_3 \leq 0, \\
& x_2 + x_3 + x_4 \leq 0, \\
& x_2 - x_3 \leq 0, \\
& x_3 + x_4 \leq 0, \\
& x_4 - x_5 \leq 0, \\
& x_4 - x_6 \leq 0
\end{align*}$

Approximate Transformer I

Remove constraints

$\begin{align*}
& x_2 + x_3 + x_4 \leq 0, \\
& x_3 + x_4 \leq 0
\end{align*}$

$\begin{align*}
& x_1 - x_2 + x_3 \leq 0, \\
& x_2 - x_3 \leq 0
\end{align*}$

$x_5 := x_4 - x_6$

$x_5 := x_4 - x_6$

$\begin{align*}
& x_4 - x_5 \leq 0, \\
& x_4 - x_6 \leq 0
\end{align*}$

$x_5 := x_4 - x_6$

$\begin{align*}
& x_4 - x_5 - x_6 = 0, \\
& x_4 - x_6 \leq 0
\end{align*}$

$x_5 := x_4 - x_6$

Optimal Transformer

$x_5 := x_4 - x_6$

$\begin{align*}
& x_1 - x_2 + x_3 \leq 0, \\
& x_2 + x_3 + x_4 \leq 0, \\
& x_2 - x_3 \leq 0, \\
& x_3 + x_4 \leq 0, \\
& x_4 - x_5 - x_6 = 0, \\
& x_4 - x_6 \leq 0
\end{align*}$

Approximate Transformer II

Remove constraints

$\begin{align*}
& x_4 - x_5 \leq 0, \\
& x_4 - x_6 \leq 0
\end{align*}$

$x_5 := x_4 - x_6$

$\begin{align*}
& x_1 - x_2 + x_3 \leq 0, \\
& x_2 + x_3 + x_4 \leq 0, \\
& x_2 - x_3 \leq 0, \\
& x_3 + x_4 \leq 0
\end{align*}$

$x_5 := x_4 - x_6$

$\begin{align*}
& x_4 - x_5 - x_6 = 0, \\
& x_4 - x_6 \leq 0
\end{align*}$
Step 2: Define features $\phi_j(s, a)$

Feature are proxy for **precision** of input $s$ and **performance** of transformer $a$

### State $s$

- $\{x_1 - x_2 + x_3 \leq 0, x_2 - x_3 \leq 0\}$
- $\{x_4 - x_5 \leq 0, x_4 - x_6 \leq 0\}$
- $\{x_7 = 0, x_8 + x_7 \leq 0\}$

<table>
<thead>
<tr>
<th>Precision features</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># of variables with finite upper and lower bound</td>
<td>$1 {x_7}$</td>
</tr>
<tr>
<td># of variables with either finite upper or lower bound</td>
<td>$2 {x_1, x_8}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance features</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># of blocks</td>
<td>$3$</td>
</tr>
<tr>
<td>Maximum # of variables in a block</td>
<td>$3$</td>
</tr>
<tr>
<td>Minimum # of variables in a block</td>
<td>$2$</td>
</tr>
<tr>
<td>Average # of variables in a block</td>
<td>$8/3$</td>
</tr>
</tbody>
</table>
Step 3: Define reward function $r(s_t, a_t, s_{t+1})$

Reward favors high precision of output state $s_{t+1}$ and speed for transformer $a_t$

Output state $s_{t+1}$

\[
\begin{align*}
\{x_1 - x_2 + x_3 &\leq 0, \\
x_2 - x_3 &\leq 0 \} \\
\{x_4 - x_5 - x_6 = 0, \\
x_5 = 0, \\
x_6 \leq 2, -x_6 &\leq 0 \}
\end{align*}
\]

- $n_s$: # of variables with singleton bounds
- $n_b$: # of variables with finite lower and upper bound
- $n_{hb}$: # of variables with only finite lower or upper bound
- $cyc$: # of cycles for computing the transformer

\[
r(s_t, a_t, s_{t+1}) = 3. n_s + 2. n_b + n_{hb} - \log_{10}(cyc)
\]

Feature | Value on $s_{t+1}$
---|---
$n_s$ | 1 ($x_5$)
$n_b$ | 2 ($x_4, x_6$)
$n_{hb}$ | 1 ($x_1$)
cyc | 10
$r(s_t, a_t, s_{t+1})$ | 7
Putting it all together

Finally, using the features $\phi_j$, the reward function $r(s_t, a_t, s_{t+1})$, and transformers $\mathcal{A}$, we can apply Q-learning and learn $\theta_j$

Then, we can perform **RL-based analysis**

$$Q(s, a) = \sum_{j=1}^{l} \theta_j \phi_j(s, a)$$

$$a_t = \arg\max_{a \in \mathcal{A}} Q(s_t, a)$$
Experimental setup

Dataset from SVCOMP
70 benchmarks for training, 30 benchmarks for testing

Poly-RL vs
• ELINA: state-of-the-art Polyhedra library [ground truth]
• Poly-Fixed: fixed strategy
• Poly-Init: use precise transformer 50% of the time

Precision = % of program points with same invariants as ELINA
## Experimental results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Number of program points</th>
<th>ELINA</th>
<th>Poly-RL</th>
<th>Poly-Fixed</th>
<th>Poly-Init</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>time (s)</td>
<td>time(s)</td>
<td>precision</td>
<td>time(s)</td>
</tr>
<tr>
<td>wireless_airo</td>
<td>2372</td>
<td>877</td>
<td>6.6</td>
<td>100</td>
<td>6.7</td>
</tr>
<tr>
<td>net_ppp</td>
<td>689</td>
<td>2220</td>
<td>9.1</td>
<td>87</td>
<td>TO</td>
</tr>
<tr>
<td>mfd_sm501</td>
<td>369</td>
<td>1596</td>
<td>3.1</td>
<td>97</td>
<td>1421</td>
</tr>
<tr>
<td>ideapad_laptop</td>
<td>461</td>
<td>172</td>
<td>2.9</td>
<td>100</td>
<td>157</td>
</tr>
<tr>
<td>pata_legacy</td>
<td>262</td>
<td>41</td>
<td>2.8</td>
<td>41</td>
<td>2.5</td>
</tr>
<tr>
<td>usb_ohci</td>
<td>1520</td>
<td>22</td>
<td>2.9</td>
<td>100</td>
<td>34</td>
</tr>
<tr>
<td>usb_gadget</td>
<td>1843</td>
<td>66</td>
<td>37</td>
<td>60</td>
<td>35</td>
</tr>
<tr>
<td>wireless_b43</td>
<td>3226</td>
<td>19</td>
<td>13</td>
<td>66</td>
<td>TO</td>
</tr>
<tr>
<td>lustre_lllite</td>
<td>211</td>
<td>5.7</td>
<td>4.9</td>
<td>98</td>
<td>5.4</td>
</tr>
<tr>
<td>usb_cx231xx</td>
<td>4752</td>
<td>7.3</td>
<td>3.9</td>
<td>(\approx)100</td>
<td>3.7</td>
</tr>
<tr>
<td>netfilter_ipvs</td>
<td>5238</td>
<td>20</td>
<td>17</td>
<td>(\approx)100</td>
<td>9.8</td>
</tr>
</tbody>
</table>

Poly-RL produces the same invariant at assertion points as ELINA on all benchmarks.

Timeout: 1hr, Memory limit: 100 GB
Summary

Many analyzers/solvers based on heuristics

Use Reinforcement Learning to find heuristics

Instantiate for Polyhedra via Q-learning

Promising results and future work

Step 1: Define space of transformers $\mathcal{A}$
Step 2: Define features $\phi_j(s,a)$
Step 3: Define reward function $r(s_t, a_t, s_{t+1})$

$$Q(s,a) = \sum_{j=1}^i \theta_j \phi_j(s,a) \quad a_t = \text{argmax}_{a \in \mathcal{A}} Q(s_t, a)$$