Data-Driven Program Analysis

Hakjoo Oh
Programming Research Laboratory
Korea University

(co-work with Kihong Heo, Kwonsoo Chae, Hongseok Yang, Kwangkeun Yi)

Nov. 9, 2016 @POSTECH
Research Areas

- **Program Analysis** derives specifications from code
- **Program Synthesis** derives code from specifications

```c
int f(int n) {
    int i = 0;
    int r = 1;
    while (i < n) {
        r = r * i;
        i = i + 1;
    }
    return r;
}
```

- $f(1) = 1$
- $f(2) = 2$
- $f(3) = 6$
- $...$
- $f(n) = n!$
Program Analysis

• Predict program behavior automatically
  • static or dynamic: before execution at compile-time / at runtime
  • automatic: sw is analyzed by sw (“program analyzers”)

• Applications
  • bug-finding: e.g., find runtime failures of programs
  • security: e.g., is this app malicious or benign?
  • verification: e.g., does the program meet its specification?
  • compiler optimization: e.g., automatic parallelization
Program Synthesis

• Generate program code from specifications automatically

  • **specification**: logics, examples, implementation, etc
  • **automatic**: sw is generated by sw (“program synthesizers”)

• Applications

  • **programming assistance**: e.g., complete tricky parts of programs
  • **end-user programming**: e.g., automate repetitive tasks
  • **algorithm discovery**: find a new solution for a problem
  • **program optimization**: find a more efficient implementation
  • **automatic patch generation**: automatically fix software bugs
Static Program Analysis

- Program states
- Error states
Static Program Analysis

sound

error states

program states
Static Program Analysis

sound

program states

error states

vs.

unsound

program states

error states
Static Program Analysis

imprecise

program states

error states

false alarms
Static Program Analysis

imprecise vs. precise

program states vs. error states

false alarms

program states vs. error states
Static Program Analysis
Challenge in Static Analysis

scalability

precision
Challenge in Static Analysis

Scalability vs. Precision

Key: "Selectivity"
Flow-Sensitivity

\[ x = y = 0; z = 1 \]
\[ x = z \]
\[ z = z + 1 \]
\[ y = x \]
\[ \text{assert}(y > 0) \]

<table>
<thead>
<tr>
<th></th>
<th>[0,0]</th>
<th>[0,0]</th>
<th>[1,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

precise but costly

<table>
<thead>
<tr>
<th></th>
<th>[1,1]</th>
<th>[0,0]</th>
<th>[1,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>[1,1]</th>
<th>[0,0]</th>
<th>[2,2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>[1,1]</th>
<th>[1,1]</th>
<th>[2,2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Flow-Insensitivity

\[ x = y = 0 \; ; \; z = 1 \]

\[ x = z \]

\[ z = z + 1 \]

\[ y = x \]

\[ \text{assert}(y > 0) \]

cheap but imprecise

<table>
<thead>
<tr>
<th></th>
<th>( x )</th>
<th>( y )</th>
<th>( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>([0, +\infty])</td>
<td>([0, +\infty])</td>
<td>([1, +\infty])</td>
</tr>
</tbody>
</table>
Selective Flow-Sensitivity

\[ x = y = 0; z = 1 \]
\[ x = Z \]
\[ z = z + 1 \]
\[ y = x \]
\[ \text{assert}(y > 0) \]

**FS : \{x, y\}**

| \(x\) | \([0,0]\) |
| \(y\) | \([0,0]\) |
| \(x\) | \([1,+\infty]\) |
| \(y\) | \([0,0]\) |
| \(x\) | \([1,+\infty]\) |
| \(y\) | \([0,0]\) |
| \(x\) | \([1,+\infty]\) |
| \(y\) | \([1,+\infty]\) |

**FI : \{z\}**

| \(z\) | \([1,+\infty]\) |
Selective Flow-Sensitivity

\[ x = y = 0; z = 1 \]

\[ x = z \]

\[ z = z + 1 \]

\[ y = x \]

assert(\(y > 0\))

**FS : \{y,z\}**

<table>
<thead>
<tr>
<th>y</th>
<th>[0,0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td>[1,1]</td>
</tr>
</tbody>
</table>

\[ x = [0, +\infty] \]

**FI : \{x\}**

<table>
<thead>
<tr>
<th>y</th>
<th>[0,0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td>[1,1]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>y</th>
<th>[0, +\infty]</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td>[2,2]</td>
</tr>
</tbody>
</table>

fail to prove
Hard Search Problem

- Intractably large space, if not infinite
- \(2^{\text{Var}}\) different abstractions for FS
- Most of them are too imprecise or costly

\[ P(\{x,y,z\}) = \{\emptyset, \{x\}, \{y\}, \{z\}, \{x,y\}, \{y,z\}, \{x,z\}, \{x,y,z\}\} \]
Our Research

• How to automatically find a good abstraction?
  • pre-analysis approach [PLDI’14, TOPLAS’16]
  • data-driven approaches [OOPSLA’15, SAS’16, APLAS’16]

pre-analysis \{x,y,z,…\} main analysis

learn a good strategy from data via machine learning techniques
Our Learning Approaches

• Learning via black-box optimization [OOPSLA’15]
• Learning via white-box optimization [APLAS’16]
• Learning from automatically labelled data [SAS’16]
• Learning with automatically generated features (in progress)
• …
Static Analyzer

\[ F(p, a) \Rightarrow n \]

- **abstraction** (e.g., a set of variables)
- **number of proved assertions**
Overall Approach
Overall Approach

• Parameterized adaptation strategy

\[ S_w : \text{pgm} \rightarrow 2^{\text{Var}} \]
Overall Approach

- Parameterized adaptation strategy

\[ S_w : \text{pgm} \rightarrow 2^{\text{Var}} \]

- Learn a good parameter $W$ from existing codebase
Overall Approach

• Parameterized adaptation strategy

\[ S_w : \text{pgm} \rightarrow 2^{\text{Var}} \]

• Learn a good parameter \( W \) from existing codebase

\[
\begin{array}{c}
P_1, P_2, \ldots, P_m \\
\Rightarrow \\
\text{Codebase} \\
\Rightarrow \\
W
\end{array}
\]

• For new program \( P \), run static analysis with \( S_w(P) \)
I. Parameterized Strategy

\[ S_w : \text{pgm} \rightarrow 2^{\text{Var}} \]

(1) Represent program variables as feature vectors.

(2) Compute the score of each variable.

(3) Choose the top-k variables based on the score.
(1) Features

• Predicates over variables:

\[ f = \{ f_1, f_2, \ldots, f_5 \} \quad (f_i : \text{Var} \rightarrow \{0,1\}) \]

• 45 simple syntactic features for variables: e.g,
  
  • local / global variable, passed to / returned from malloc, incremented by constants, etc
(1) Features

- Represent each variable as a feature vector:

\[ f(x) = \langle f_1(x), f_2(x), f_3(x), f_4(x), f_5(x) \rangle \]

\[ f(x) = \langle 1,0,1,0,0 \rangle \]
\[ f(y) = \langle 1,0,1,0,1 \rangle \]
\[ f(z) = \langle 0,0,1,1,0 \rangle \]
(2) Scoring

• The parameter $w$ is a real-valued vector: e.g.,

$$w = \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle$$

• Compute scores of variables:

$$score(x) = \langle 1,0,1,0,0 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.3$$
$$score(y) = \langle 1,0,1,0,1 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.6$$
$$score(z) = \langle 0,0,1,1,0 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.1$$
(3) Choose Top-k Variables

• Choose the top-k variables based on their scores: e.g., when $k=2$,

\[
\begin{align*}
\text{score}(x) &= 0.3 \\
\text{score}(y) &= 0.6 \\
\text{score}(z) &= 0.1
\end{align*}
\]

\{x, y\}

• In experiments, we chosen 10% of variables with highest scores.
2. Learn a Good Parameter

\[
P_1, P_2, \ldots, P_m \quad \Rightarrow \quad W
\]

- Solve the optimization problem:

Find \( w \) that maximizes

\[
\sum_{P_i} F(P_i, S_w(P_i))
\]
Learning via Random Sampling

repeat $N$ times

pick $w \in \mathbb{R}^n$ randomly

evaluate $\sum_{P_i} F(P_i, S_w(P_i))$

return best $w$ found
Learning via Random Sampling

![Histogram Graph]

### Table 4.

<table>
<thead>
<tr>
<th>Trial</th>
<th>Quality (%)</th>
<th>Cost Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80.3%</td>
<td>118.2 x</td>
</tr>
<tr>
<td>2</td>
<td>72.4%</td>
<td>79.4 x</td>
</tr>
<tr>
<td>3</td>
<td>76.3%</td>
<td>108.8 x</td>
</tr>
<tr>
<td>4</td>
<td>73.7%</td>
<td>42.0 x</td>
</tr>
<tr>
<td>5</td>
<td>98.5%</td>
<td>258.3 x</td>
</tr>
</tbody>
</table>

### Table 5.

<table>
<thead>
<tr>
<th>Trial</th>
<th>Quality (%)</th>
<th>Cost Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80.0%</td>
<td>112.1 x</td>
</tr>
</tbody>
</table>

### Figures

- Figure 2. Comparison of Bayesian optimisation with random sampling

### Programs

- Baseline analyser: supports full C language and has been developed for seven years.
- Features: numeric and pointer information tracking, interval abstract domain for numeric values, allocation-site-based heap abstraction for pointer values.
- Properties: field-sensitive, flow-sensitive, not context-sensitive.
- Sparse analysis technique applied for scalability.
- Partially flow-sensitive analyser implemented.
- Bayesian optimisation based learning algorithm.

### Experiments

- 30 open source programs from GNU and Linux packages.
- Key questions: learning algorithm performance and Bayesian optimisation benefits.
- Cross validation method used.
- Implementation available at [link].
Bayesian Optimization

• A powerful method for solving difficult black-box optimization problems.

• Especially powerful when the objective function is expensive to evaluate.

• Key idea: use a probabilistic model to reduce the number of objective function evaluations.
Learning via Bayesian Optimization

repeat N times

select a promising w using the model

evaluate \( \sum_{P_i} F(P_i, S_w(P_i)) \)

update the probabilistic model

return best w found

- Probabilistic model: Gaussian processes
- Selection strategy: Expected improvement
Learning via Bayesian Optimization

![Histogram of quality distribution](image_url)
The random sampling method converged to the quality of the best parameter found in 69.6% of the FS-only queries, compared to 70.0% for Bayesian optimization. The difference becomes apparent after the first 30 evaluations. The figure shows the quality of the best parameter found over the number of evaluations for random sampling and Bayesian optimization. Bayesian optimization outperforms random sampling by a significant margin, especially after the first 30 evaluations.
Effectiveness

• Implemented in Sparrow, an interval analyzer for C
• Evaluated on 30 open-source programs
  • 20 for training, 10 for testing
Effectiveness

• Implemented in Sparrow, an interval analyzer for C
• Evaluated on 30 open-source programs
  • 20 for training, 10 for testing

Precision

FI                SFS                FS
0                 70                 100
Effectiveness

- Implemented in Sparrow, an interval analyzer for C
- Evaluated on 30 open-source programs
  - 20 for training, 10 for testing

### Precision

- FI
- SFS: 70
- FS: 100

### Cost

- FI
- SFS: 2x
- FS: 18x
Limitations

• While promising, the method has limitations:
  • black-box optimization is inherently inefficient
  • manual feature engineering is needed
• Follow-up work to overcome the limitations:
  • improving the efficiency [APLAS’16, SAS’16]
  • automating feature engineering [on-going]
Improving Efficiency

- A white-box optimization method [APLAS’16]

\[ \mathcal{O}_P : \mathcal{J}_P \rightarrow \mathbb{R}. \]

Find \( w^* \) that minimizes \( \sum_{j \in \mathcal{J}_P} (score^w_P(j) - \mathcal{O}(j))^2 \)

- A supervised learning method [SAS’16]

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>-a</th>
<th>b</th>
<th>-b</th>
<th>c</th>
<th>-c</th>
<th>i</th>
<th>-i</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
</tr>
<tr>
<td>-a</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
</tr>
<tr>
<td>b</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
</tr>
<tr>
<td>-b</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
</tr>
<tr>
<td>c</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
</tr>
<tr>
<td>-c</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
</tr>
<tr>
<td>i</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
</tr>
<tr>
<td>-i</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
<td>T</td>
<td>✭</td>
</tr>
</tbody>
</table>
Manual Feature Engineering

- The success of ML heavily depends on the “features”
- Feature engineering is nontrivial and time-consuming
- Features do not generalize to other tasks

<table>
<thead>
<tr>
<th>Type</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Manual Feature Engineering</td>
</tr>
</tbody>
</table>

Type A features not because they are important for flow-sensitivity. We included them an argument to a function that does memory allocation. Note that we included these functions having a structure as an argument functions having a pointer argument functions having more than one argument functions having no arguments invoked with an unknown value constant is passed to an argument directly invoked with a constant directly return a reallocated memory indirectly return a constant expression read from a structure field write to a global variable function using a string-related library function function containing a loop function containing realloc

flow-sensitivity
context-sensitivity
widening thresholds
Automatic Feature Generation

Before [OOPSLA’15, SAS’16, APLAS’16]

New method
Partial Flow-Sensitive Analysis

- A query-based, partially flow-sensitive interval analysis
- The analysis uses a query-classifier $C : \text{Query} \rightarrow \{1,0\}$

```plaintext
1  x = 0; y = 0; z = input(); w = 0;
2  y = x; y++;
3  assert (y > 0);  // Query 1
4  assert (z > 0);  // Query 2
5  assert (w == 0); // Query 3
```
Partial Flow-Sensitive Analysis

- A query-based, partially flow-sensitive interval analysis
- The analysis uses a query-classifier $C: \text{Query} \rightarrow \{1,0\}$

```plaintext
1  x = 0; y = 0; z = input(); w = 0;
2  y = x; y++;
3  assert (y > 0);  // Query 1 provable
4  assert (z > 0);  // Query 2 unprovable
5  assert (w == 0); // Query 3 unprovable
```
Partial Flow-Sensitive Analysis

- A query-based, partially flow-sensitive interval analysis
- The analysis uses a query-classifier $C : \text{Query} \rightarrow \{1, 0\}$

```plaintext
x = 0; y = 0; z = input(); w = 0;
y = x; y++;
assert (y > 0); // Query 1 provable
assert (z > 0); // Query 2 unprovable
assert (w == 0); // Query 3 unprovable
```

<table>
<thead>
<tr>
<th>line</th>
<th>flow-sensitive result</th>
<th>flow-insensitive result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>abstract state</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>${x \mapsto [0,0], y \mapsto [0,0]}$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>${x \mapsto [0,0], y \mapsto [1,1]}$</td>
<td>${z \mapsto [0,0], w \mapsto [0,0]}$</td>
</tr>
<tr>
<td>3</td>
<td>${x \mapsto [0,0], y \mapsto [1,1]}$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>${x \mapsto [0,0], y \mapsto [1,1]}$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>${x \mapsto [0,0], y \mapsto [1,1]}$</td>
<td></td>
</tr>
</tbody>
</table>
Learning a Query Classifier

Standard binary classification:

\[ \{(q_i, b_i)\}_{i=1}^n \]
Learning a Query Classifier

Standard binary classification:

\[ \{(q_i, b_i)\}_{i=1}^{n} \rightarrow \{(v_i, b_i)\}_{i=1}^{n} \]

\[ (v_i \in \mathbb{B}^k) \]
Learning a Query Classifier

Standard binary classification:

\[ \{(q_i, b_i)\}_{i=1}^{n} \rightarrow \{(v_i, b_i)\}_{i=1}^{n} \rightarrow C : \mathbb{B}^k \rightarrow \mathbb{B} \]

- Transform to feature vectors
- Apply standard learning algorithms
Standard binary classification:

\[
\{(q_i, b_i)\}_{i=1}^n \rightarrow \{(v_i, b_i)\}_{i=1}^n \rightarrow C : \mathbb{B}^k \rightarrow \mathbb{B}
\]

- Success relies on how we convert queries to feature vectors
- This feature engineering has been done manually
Conversion from Queries to Feature Vectors

- A set of feature features $\Pi = \{\pi_1, \ldots, \pi_k\}$
- A feature encodes a property about queries
- A procedure to check whether a query satisfies a feature
  \[\text{match} : \text{Query} \times \text{Feature} \rightarrow \mathbb{B}\]
- The feature vector of a query $q$:
  \[\langle \text{match}(q, \pi_1), \ldots, \text{match}(q, \pi_k) \rangle\]
Automatic Feature Generation

• Generate *feature programs* by running reducer
  • small pieces of code that minimally describe when it is worth increasing the precision

• Represent them by *abstract data-flow graphs*
  • generalized form of feature programs
Generating Feature Programs

1. a = 0; b = 0;
2. while (1) {
   b = unknown();
3. if (a > b) reduce(P, φ) =>
4. if (a < 3) assert (a < 5);
5. assert (a < 5);
6. a++;
7. }

• By running a program reducer: e.g., C-Reduce [PLDI’12]

\[ \text{reduce} : \mathbb{P} \times (\mathbb{P} \rightarrow \mathbb{B}) \rightarrow \mathbb{P} \]

• Feature-preserving condition:

\[ \phi(P) \equiv FI(P) = \text{unproven} \land FS(P) = \text{proven} \]
Generalize to Abstract Data-Flow Graphs

- The right level of abstraction depends on an analysis
- We choose the best abstraction using a combination of searching and cross-validation

```
1 a = 0;
2 while (1) {
3   if (a < 3)
4     assert (a < 5);
5   a++;
6 }
```
Feature Generation

- Apply the method on codebases:

\[ P_1, P_2, \ldots, P_m \Rightarrow \Pi = \{ \pi_1, \ldots, \pi_k \} \]
Matching Algorithm

\[ \text{match} : \text{Query} \times \text{Feature} \rightarrow \mathbb{B} \]

\[
\begin{align*}
\text{id} & := \top \quad \text{id} > \text{id} \quad \text{id} := \text{id} + c \\
\text{id} & := c \quad \text{id} < c \quad Q(\text{id} < c)
\end{align*}
\]

\[
\begin{align*}
\text{id} & := \text{id} + c \\
\text{id} & := c \quad \text{id} < c \quad Q(\text{id} < c)
\end{align*}
\]

1. \(a = 0; b = 0;\)
2. \(\text{while (1) {}\)}\
3. \(\text{b = unknown();}\)
4. \(\text{if (a > b)}\)
5. \(\quad \text{if (a < 3)}\)
6. \(\quad \text{assert (a < 5);}\)
7. \(\quad a++;\)
8. \}
Matching Algorithm

$$\text{match} : \text{Query} \times \text{Feature} \rightarrow \mathbb{B}$$

1. $$a = 0; b = 0;$$
2. while (1) {
3.   $$b = \text{unknown}();$$
4.   if $$(a > b)$$
5.     if $$(a < 3)$$
6.       assert $$(a < 5);$$
7.     $$a++;$$
8. }

Subgraph inclusion:

$$(N_1, E_1) \subseteq (N_2, E_2) \iff N_1 \subseteq N_2 \land E_1 \subseteq E_2^*$$
Learning a Query Classifier

\[ P_1, P_2, \ldots, P_m \]  

Codebase

\[ \Pi = \{ \pi_1, \ldots, \pi_k \} \]

\[ \{(v_i, b_i)\}_{i=1}^n \]

\[ C : \mathbb{B}^k \rightarrow \mathbb{B} \]
Experiments

Effectiveness of partially flow-sensitive analysis

<table>
<thead>
<tr>
<th>Trial</th>
<th>Query Prediction</th>
<th>Analysis</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
<td>Prove</td>
</tr>
<tr>
<td>1</td>
<td>92.6 %</td>
<td>77.9 %</td>
<td>5,340</td>
</tr>
<tr>
<td>2</td>
<td>78.8 %</td>
<td>73.3 %</td>
<td>2,972</td>
</tr>
<tr>
<td>3</td>
<td>66.7 %</td>
<td>73.3 %</td>
<td>3,984</td>
</tr>
<tr>
<td>4</td>
<td>88.7 %</td>
<td>68.8 %</td>
<td>4,600</td>
</tr>
<tr>
<td>5</td>
<td>89.9 %</td>
<td>79.4 %</td>
<td>2,517</td>
</tr>
<tr>
<td>TOTAL</td>
<td>81.5 %</td>
<td>73.9 %</td>
<td>19,413</td>
</tr>
</tbody>
</table>

Effectiveness of partially relational analysis

<table>
<thead>
<tr>
<th>Trial</th>
<th>Query Prediction</th>
<th>Analysis</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
<td>Prove</td>
</tr>
<tr>
<td>1</td>
<td>74.8 %</td>
<td>81.3 %</td>
<td>3,678</td>
</tr>
<tr>
<td>2</td>
<td>84.1 %</td>
<td>82.6 %</td>
<td>5,845</td>
</tr>
<tr>
<td>3</td>
<td>82.8 %</td>
<td>73.0 %</td>
<td>1,926</td>
</tr>
<tr>
<td>4</td>
<td>77.6 %</td>
<td>85.2 %</td>
<td>2,221</td>
</tr>
<tr>
<td>5</td>
<td>71.6 %</td>
<td>78.4 %</td>
<td>2,886</td>
</tr>
<tr>
<td>TOTAL</td>
<td>79.0 %</td>
<td>79.9 %</td>
<td>16,556</td>
</tr>
</tbody>
</table>
Summary

• Choosing a good abstraction is a key challenge in static program analysis
• New data-driven approach is promising
• Further information:

http://prl.korea.ac.kr
Summary

• Choosing a good abstraction is a key challenge in static program analysis
• New data-driven approach is promising
• Further information:

http://prl.korea.ac.kr

Thank you