Mining Anomaly Detectors

Paolo Tonella

Software Engineering Research Unit
Fondazione Bruno Kessler
Trento, Italy
http://se.fbk.eu/tonella
Outline

• Role and classification of (mined) oracles
• Oracle mining techniques
• Empirical validation of mined oracles
• Future research directions
Role of oracles

For a given program $P$, what combination of tests $T$ and oracle $O$ achieves the highest fault revealing level?

Mutation testing & testability

Mutation adequacy (revised for any arbitrary $o$):

$$\text{Mut}_M(p \times s \times TS \times o) \Rightarrow \forall m \in M, \exists t \in TS: \neg o(t, m)$$

Effectiveness of mutation testing depends on the power of $o$.

**Testability** of program location $loc$ is defined as the probability that the system fails if location $loc$ is faulty.

Propagation probability (revised): probability that a perturbed value of $a$ at location $loc$ affects a variable used by oracle $o$.

Testability of a program depends also on the oracle. Low testability locations can be made more testable by using a more powerful oracle.
Oracle comparison

Oracle power \((o_1 \geq_{TS} o_2)\): \(\forall t \in TS, o_1(t, p) \Rightarrow o_2(t, p)\)

Oracle power is a partial order relation (not all pairs of oracles satisfy the oracle power relation in either direction), hence there are un-comparable oracles according to power.

Probabilistic better \((o_1 PB_{TS} o_2)\):
For a randomly selected \(t \in TS\): \(P[o_1(t, p) = F] \geq P[o_2(t, p) = F]\)

Probabilistic better is a total order relation. Probabilistic better is weaker than (subsumed by) the oracle power relation.
Classes of oracles

Complete oracle: \( corr(t, p, s) \Rightarrow o(t, p) \)

- Faults revealed by \( o \) are real faults; pass runs may miss a fault.

Sound oracle: \( o(t, p) \Rightarrow corr(t, p, s) \)

- Oracle proves correctness; no fault is missed.

Perfect oracle: \( o(t, p) \iff corr(t, p, s) \)

corr\((t, p, s)\): spec \( s \) holds for \( p \) when \( t \) is run.

1. Unsound/complete [FN ≥ 0; FP = 0]
   - Pre/post-conditions; invariants; assertions

2. Unsound/incomplete [FN ≥ 0; FP ≥ 0]
   - Anomaly detectors (oracle/spec mining/learning)
Mining oracles

1. Mining finite state machines
2. Mining temporal properties / association rules
3. Mining data invariants

Common assumption [well-enough debugged program]: during mining (training) only or mostly correct program behaviors are observed.

**INPUT**: static traces (paths) or dynamic traces (logs).
**OUTPUT**: oracles/specifications, that can be checked dynamically or statically (e.g., through model checking).
Mining finite state machines

Dynamic traces (execution logs)

FSM inference
State abstraction

Execution logs

ADABU [Dallmeier et al.; WODA 2006]

```
[in=In@6f3321a3,out=Out@5d0385c1] println
[in=In@6f3321a3,out=Out@5d0385c1] Formatter
[in=In@6f3321a3,out=Out@5d0385c1] close
[in=null,out=Out@5d0385c1] println

[in=In@4a3922f3,out=Out@5f0476d2] println
[in=In@4a3922f3,out=Out@5f0476d2] Formatter
[in=In@4a3922f3,out=Out@5f0476d2] format
[in=In@4a3922f3,out=Out@5f0476d2] close
[in=null,out=Out@5f0476d2] println

[in=In@1b25672c,out=Out@34ab4411] println
[in=In@1b25672c,out=Out@34ab4411] Formatter
[in=In@1b25672c,out=Out@34ab4411] format
[in=In@1b25672c,out=Out@34ab4411] format
[in=In@1b25672c,out=Out@34ab4411] close
[in=null,out=Out@34ab4411] println
```
**Event sequence abstraction**

<table>
<thead>
<tr>
<th>Execution logs</th>
<th>kTail [Biermann &amp; Feldman; Trans Comp 1972]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>KLFA [Mariani &amp; Pastore; ISSRE 2008]</td>
</tr>
<tr>
<td></td>
<td>Synoptic [Beschastnikh et al; FSE 2011]</td>
</tr>
<tr>
<td>[Ammons et al.; POPL 2002]</td>
<td></td>
</tr>
<tr>
<td>[Whaley et al.; ISSTA 2002]</td>
<td></td>
</tr>
</tbody>
</table>

Based on **grammar inference**, usually under the constraint that: no negative example is available.
Based on a sample of strings that belong to a language $L$, we want to build a regular grammar whose accepted language is as close as possible to $L$.

**K-tail principle:**
Two states are merged (matched) if they have the same $k$-tails.
Active learning

LearnLib [Raffelt et al.; STTT 2009]
Mining temporal properties

Micro-pattern templates:
Sequencing: ab
Loop begin: $ab^+$
Loop end: $a^+b$
Pre-condition: $ab$?
Post-condition: $a?b$
Generalized pre-cond: $a^+b^*$
Generalized post-cond: $a^*b^+$
Association rule: $(ab \mid ba)$
General assoc rule: $(a^+b^+ \mid b^+a^+)$

IsEnforcing(sat: int, fail: int) →
{ENFORCE, LEARN, DEAD}

OCD [Gabel & Su; ICSE 2010]

Perracotta [Yang et al.; ICSE 2006]

Alternation rule:
$(a \ b)^*$
E.g.: lock/unlock

Example diagram:

\[ a \quad b \]
Association rule mining

**Itemset** database:
\[ D = \{\{a, b, c, d, e\}, \{a, b, d, e, f\}, \{a, b, d, g\}, \{a, c, h, i\}\} \]

**Support** of itemsets: \( \text{support}(\{a, b, d\}) = 3 \)

**Frequent itemsets** (support > 2):
\[ F = \{\{a\}, \{b\}, \{d\}, \{a, b\}, \{a, d\}, \{b, d\}, \{a, b, d\}\} \]

**Association rules** and **confidence** for frequent itemset \{a, b, d\}:
\[ c(A \Rightarrow B) = P[B \mid A] = \frac{\text{support}(A \cap B)}{\text{support}(A)} \]
\{a\} \Rightarrow \{b, d\} \quad c = \frac{3}{4} = 75\%
\{a, b\} \Rightarrow \{d\} \quad c = 100\%
\{b\} \Rightarrow \{a, d\} \quad c = 100\%

**DynaMine**:
\( a \Rightarrow b \)
Resorts to mining software revisions (co-added method calls) to find rule instances.

**DynaMine** [Livshits & Zimmermann; FSE 2005]
[Thummalapenta & Xie; ICSE 2009]
[Weimer & Necula; TACAS 2005]
Mining data invariants

**Invariant templates:**

- $x == c$
- $a <= x <= b$
- $x = a \cdot y + b \cdot z + c$
- $x = \text{abs}(y)$
- $x = \text{max}(y, z)$
- $x < y$
- $x == y$, $x + y == c$, $x - y == c$
- $\text{sorted}(x[])$
- $\text{subsequence}(x[], y[])$
- $c \text{ in } x[]$, $y \text{ in } x[]$
- $\text{strcmp}(x, y) < 0$

**Daikon** [Ernst et al.; ICSE 1999]

Dynamically discovered invariants are reported if the probability for them to be coincidental is $< \text{confidence threshold (e.g., prob}(N_{\text{occur}}) < 0.01)$.

**Diduce** [Hangal & Lam; ICSE 2002]
Empirical validation

Mined oracles are **unsound** (FN ≥ 0) and **incomplete** (FP ≥ 0). Are they useful in practice?

**Key research questions:**

1. **Missed faults** (FN): how many faults are not exposed by the mined oracle?
2. **False alarms** (FP): how many false alarms are raised by the mined oracle?
3. **Fault characterization** (FC): is there a particular class of faults that is specifically addressed by the mined oracle? How relevant is such fault class?
**Empirical studies**

<table>
<thead>
<tr>
<th>Oracle mining tool</th>
<th>FN</th>
<th>FP</th>
<th>FC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADABU [WODA 2006]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kTail [Trans Comp 1972]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KLFA [ISSRE 2008]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synoptic [FSE 2011]</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>LearnLib [STTT 2009]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCD [ICSE 2010]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perracotta [ICSE 2007]</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>DynaMine [FSE 2005]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Daikon [ICSE 1999]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diduce [ICSE 2002]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Most experimental validations focus on the accuracy of the mined models/specs and conduct in-depth analysis of few sample anomalies, without any attempt of a systematic evaluation.
Future work

Solid, empirical validation of mined oracles:
- Experimental framework
- Benchmark (programs, test cases, traces, faults, ...)
- Key research questions
- Metrics
- Comparative evaluations
- Characterization by fault class

We (probably) do not need more oracle mining techniques; we (definitely) need to better understand and compare the effectiveness of existing techniques.