Mining Anomaly Detectors

Paolo Tonella

Software Engineering Research Unit

Fondazione Bruno Kessler

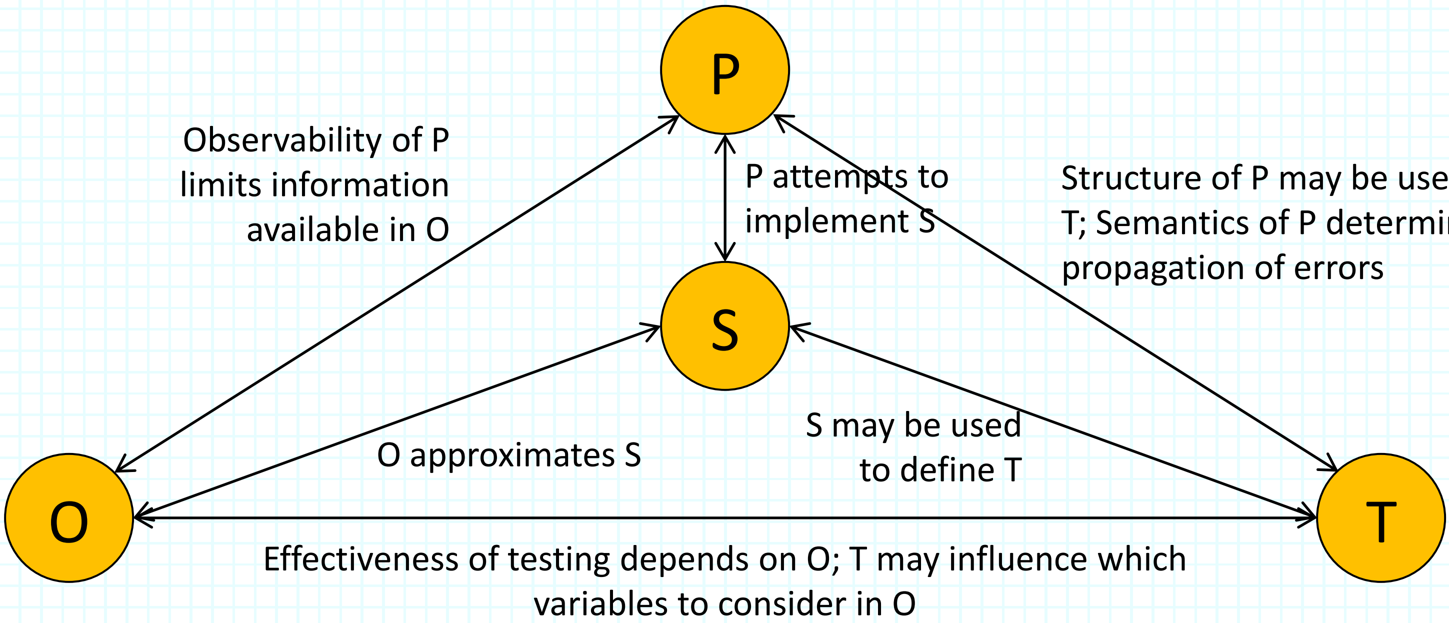
Trento, Italy

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 <u>and oracle O</u> achieves the highest fault revealing level?

M. Staats, M. W. Whalen and M. P. E. Heimdahl, *Programs, Tests, and Oracles: The Foundations of Testing Revisited*. ICSE 2011.

Mutation testing & testability

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

$$Mut_M(p \times s \times TS \times o) \Rightarrow \forall m \in M, \exists t \in TS:\ \neg o(t, m)$$

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: $o(t, p) \Leftrightarrow corr(t, p, s)$

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

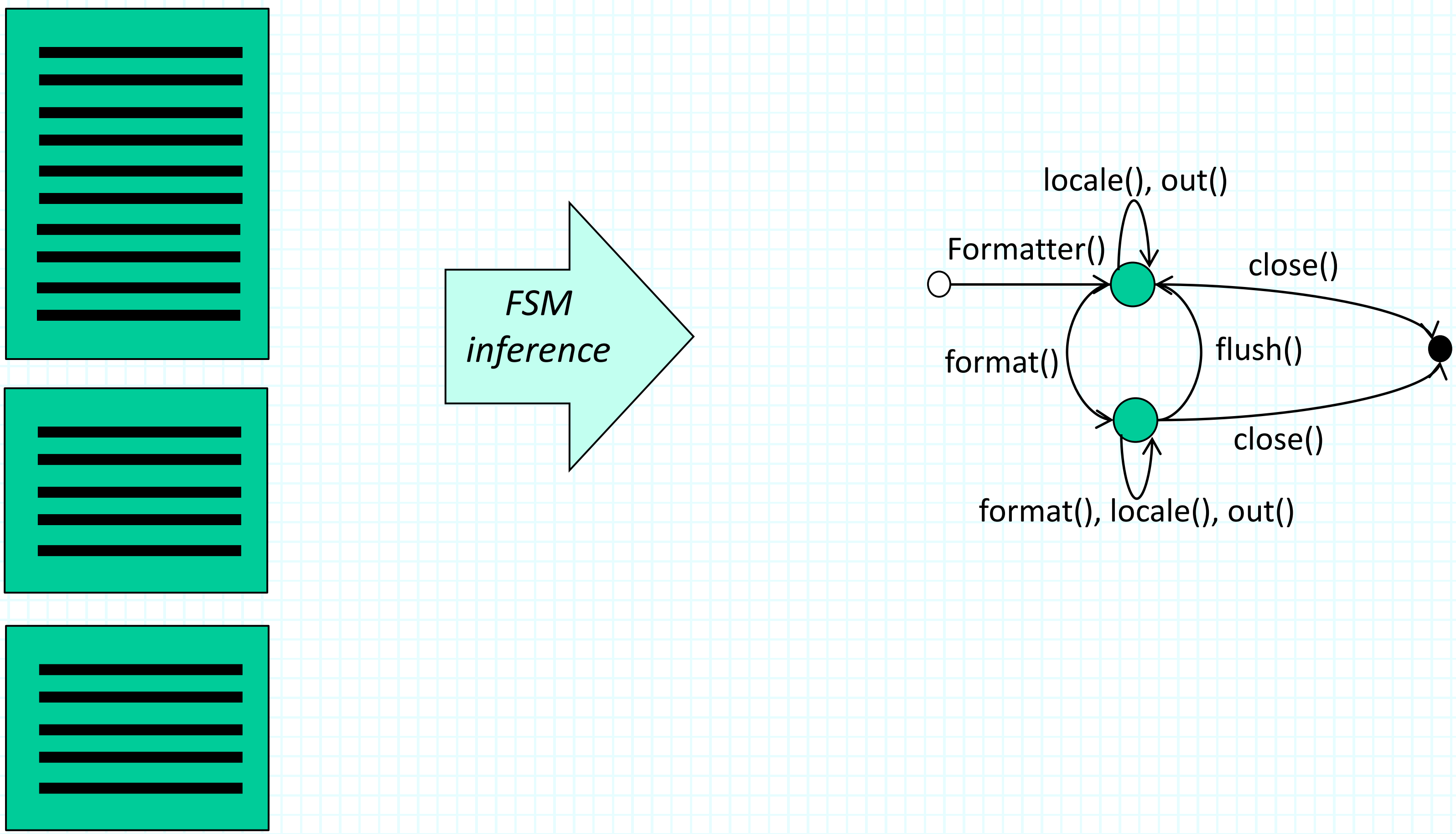
1. Mining finite state machines
2. Mining temporal properties / association rules
3. Mining data invariants

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)



State abstraction

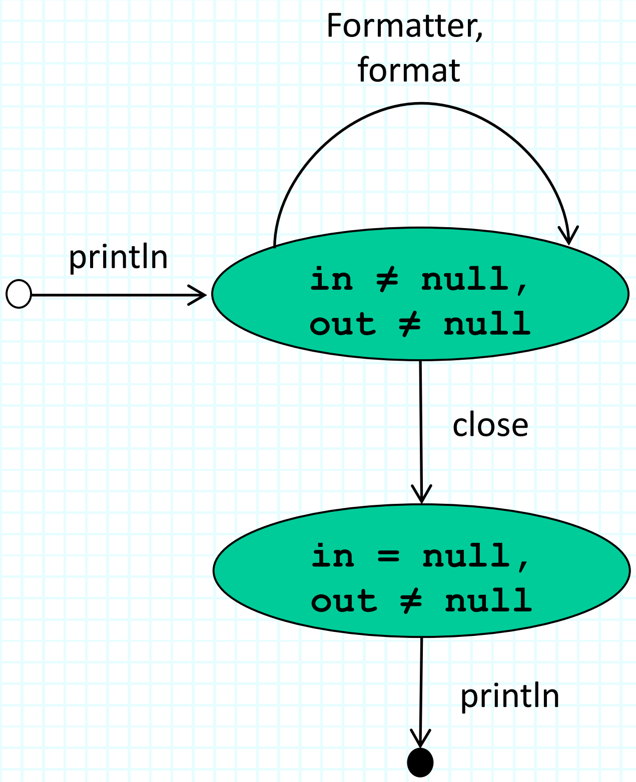
Execution logs

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] format
[in=In@1b25672c,out=Out@34ab4411] close
[in=null,out=Out@34ab4411] println
```



Event sequence abstraction

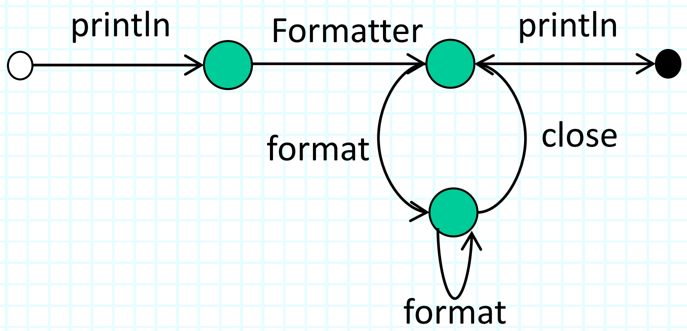
Execution logs

```
println
Formatter
close
println
```

```
println
Formatter
format
close
println
```

```
println
Formatter
format
format
format
close
println
```

kTail [Biermann & Feldman; Trans Comp 1972]
KLFA [Mariani & Pastore; ISSRE 2008]
Synoptic [Beschastnikh et al; FSE 2011]
[Ammons et al.; POPL 2002]
[Whaley et al.; ISSTA 2002]



Based on <u>grammar inference</u>, usually under the constraint that:
no negative example is available.

Grammar inference

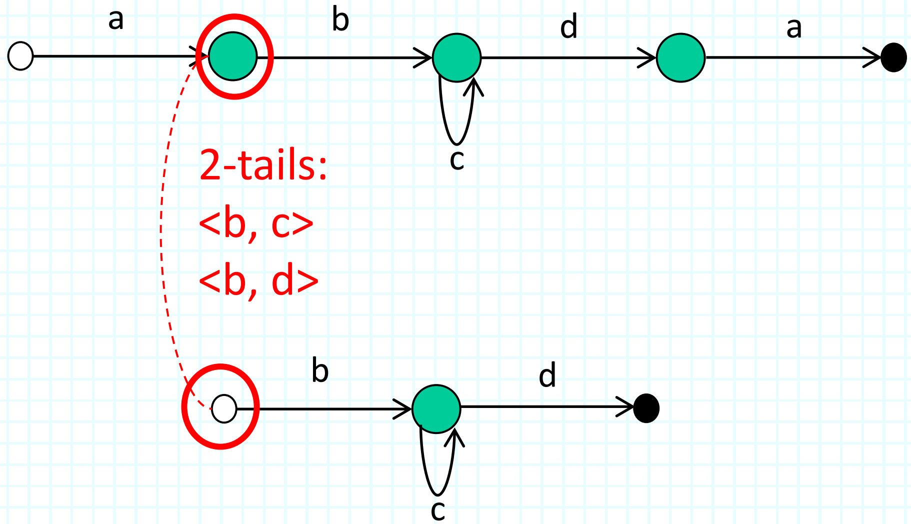
Based on a sample of strings that belong to a language L, we want to build a regular grammar whose accepted language is as close as possible to L.

a b c c c c d a

a b c c d a

b c c c c d

b c c c d

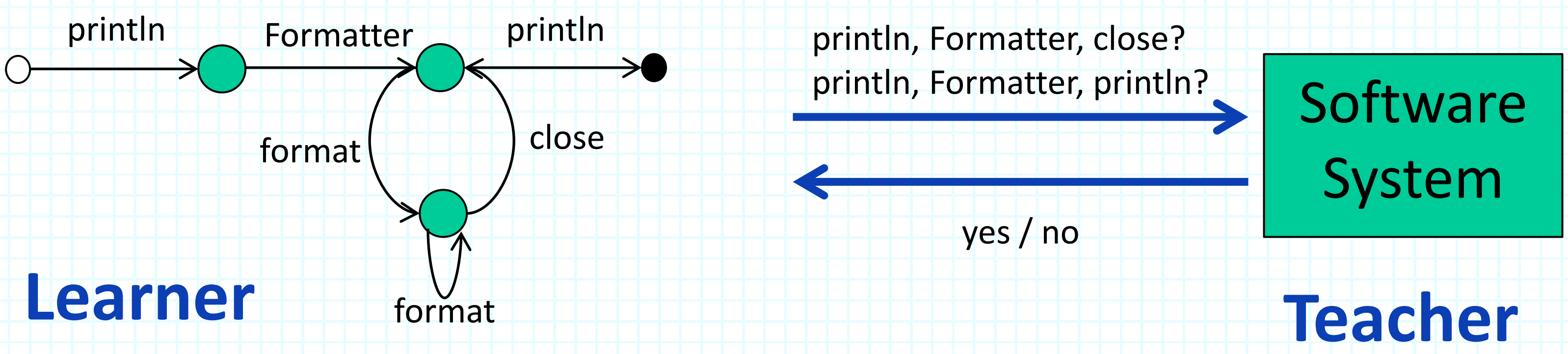


K-tail principle:

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 | ba)

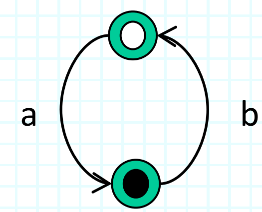
General assoc rule: $(a^+b^+ \mid b^+a^+)$

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



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: 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}:

$c(A \Rightarrow B) = P[B \mid A] = \text{support}(A\ B) / \text{support}(A)$

{a} ⇒ {b, d}   c = ¾ = 75%

{a, b} ⇒ {d}   c = 100%

{b} ⇒ {a, d}   c = 100%

DynaMine: a ⇒ 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 y + b z + c
x = abs(y)
x = max(y, z)
x < y
x == y, x + y == c, x - y == c
sorted(x[])
subsequence(x[], y[])
c in x[], y in x[]
strcmp(x, y) < 0

Daikon [Ernst et al.; ICSE 1999]

Dynamically discovered invariants are reported if the probability for them to be coincidental is < confidence threshold (e.g., prob(N_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

Oracle mining tool	FN	FP	FC
ADABU [WODA 2006]			
kTail [Trans Comp 1972]			
KLFA [ISSRE 2008]			
Synoptic [FSE 2011]		✓	
LearnLib [STTT 2009]			
OCD [ICSE 2010]			
Perracotta [ICSE 2007]		✓	
DynaMine [FSE 2005]			
Daikon [ICSE 1999]			
Diduce [ICSE 2002]			

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.