Combining Model-Based Testing and Machine Learning

Roland GROZ
Université Grenoble Alpes (COMUE),
*LIG (Laboratoire d’Informatique de Grenoble)*
France

TAROT Summer School 2016
In a nutshell:

- **Model-based testing**…
- when writing a model is *not an option*!

Testing a system is somehow LEARNING the behaviour of a system

*Problem*: test orderly to learn correct & “complete” behaviour
Outline

- Motivation: why learning?
- ML & Soft. Engineering
- Seminal algorithm: L* (Angluin 87)
- Enhancements for various issues
  - Counter-example processing
  - Tree-based (quotient algo)
  - No Reset
  - Integration
  - EFSM
- Related work
Soft. Engineering trends

- **MDE & MBT**
  - Growing trend in some industries (e.g. embedded)
  - Derive design, code and tests (MBT)
  - Models = 1st class citizens

- **Non formal (e.g. Agile)**
  - Dominant & growing trend
  - Absence of (formal) models
  - Or pb maintaining spec <-> model
  - Often Test Driven Dvt (TDD)
The chore of writing test cases can be GREATLY relieved if:
- Formal specs and models are AVAILABLE
- Test cases can be AUTOMATICALLY generated from models (MBT)

Hum! Matched by the chore of:
- Writing MODELS
- Maintaining them
Component Based Software Engineering

- Requirement Analysis
- High Level System Design
- Components selection & Integration

- Rapid Development
- Reuse Components
- Reduce cost
- Flexibility
- Ease of integration

Integrated System
Typical Issues in System Dvt

Understanding a System of Black Box/3rd party Components is a challenge

How do I perform system behavioural analysis?
How do I identify integration problems?

MODELS could help!

But what if NO model?
MDE & MBT in the reverse

- **MDE assumption**
  - Start from model, formal spec
  - Models = 1st class citizens 😊

- **Test Driven Development (XP, Agile…)**
  - Tests are spec: 1st class citizens
  - Formal models ? No way ! 😞 No time…

- **Proposed approach**
  - Derive models from tests, & combine with MBT
    - = LEARN models from tests
  - **CHALLENGE:** Reconcile Test-Driven (or code-driven) dvt with Models
Partial, incremental and approximate models
Main Technical Goals

- **Reverse Engineering**
  - Understanding the behaviours of the black box components
    - by deriving the *formal models* of the components/system
    - Can also serve documentation purposes (tests for doc)

- **System Validation**
  - Being able to derive new systematic tests
  - Analyzing the system for anomalies
    - by model checking (wrt properties)
    - by developing a *framework for integration testing* of the system of black box components
Objections

- Model is derived from bugged components
  - Derived tests will consider bug=feature

- Incremental: stopping criterion?

Answers

- Unit vs system
  - Combining model-checking & learning
  - Integration testing will reveal errors

- Tunable approximated model of system

- Key notion: counterexamples
Outline

- Motivation: why learning?
- ML & Soft. Engineering
- Seminal algorithm: L* (Angluin 87)
- Enhancements for various issues
  - Counter-example processing
  - Tree-based (quotient algo)
  - No Reset
  - Integration
  - EFSM
- Related work
Various types of Machine Learning

- Artificial Intelligence (& datamining)
  - Ability to infer rules, recognize patterns
  - Learning from samples
  - E.g. neural networks

- Two major techniques (among others)
  - Statistical inference from collection of data -> e.g. Weka tool in (data) testing

- Grammatical inference of language from theoretical computer science
Pioneering inference in SoftEng

- [Peled 1999] Black Box Checking
  - Using L* + Vasilievski’s W-method for Model Checking BB components

- [Steffen, Hagerer 2002] Model generation by Regular extrapolation
  - Applied to testing of telecom switch

- Picked up from 2003 by Dortmund, NASA, Uppsala, Grenoble, Nijmegen, KTH…
Learning languages from samples

"Learning from given positive/negative samples"

• Finding a minimum DFA (Deterministic Finite Automaton) is NP-HARD
  – Complexity of automaton identification from given data. [E. Gold 78]
• Even a DFA with no. of states polynomially larger than the no. of states of the minimum is NP-Complete
  – The minimum consistent DFA problem cannot be approximated within any polynomial. [Pitt & Warmuth 93]
• Probably Approximately Correct (PAC)
  – A theory of the learnable. [L.G. Valiant 84]
Active learning (Query learning)

- Active Learning
  - "Learning from Queries": inference algorithm can query an oracle of the language

- Angluin's Algorithm $L^*$ [Angluin 87]
  - Reference algorithm
  - Two types of queries: membership, equivalence
  - Learns Deterministic Finite Automaton (DFA) in polynomial time

- Applied in formal Software Engineering
  - Black Box Checking [Peled 99]
  - Learning and Testing Telecom Systems [Steffen 02-03]
  - Protocol Testing [Shu & Lee 08]
  - ...
Outline

- Motivation: why learning?
- ML & Soft. Engineering
- Seminal algorithm: L* (Angluin 87)
- Enhancements for various issues
  - Counter-example processing
  - Tree-based (quotient algo)
  - No Reset
  - Integration
  - EFSM
- Related work
Concept of the Regular Inference
(Angluin's Algorithm $L^*$)

**Assumptions:**
- The input alphabet $\Sigma$ is known
- Machine can be reset

**Complexity:** $O(|\Sigma| m n^2)$
- $|\Sigma|$ : the size of the input alphabet
- $n$ : the number of states in the actual machine
- $m$ : the length of the longest counterexample

**The Algorithm $L^*$**

- **Input Alphabet** $\Sigma$
- **membership queries from $\Sigma^*$**
- **Black Box Machine**
- **accept / reject**
- **equivalence queries (DFA conjecture)**
- **"yes" or counterexample**
- **Final Minimum DFA Conjecture**
- **Oracle**
Our Context of Inference (testing s/w)

- Components having I/O behaviors
- I/O are structurally complex (parameters)
- Formidable size of input sets

Enhanced State Machine Models
- Mealy Machines
- Parameterized Machines
- More adequate for complex systems
- DFAs may result in transition blow up

Test Strategies and heuristics
Learned Models can be used to generate tests to find discrepancies
Preliminaries

- **Mealy Machine**: \( M = (Q, I, O, \delta, \lambda, q_0) \)
  - \( Q \) : set of states
  - \( I \) : set of input symbols
  - \( O \) : set of output symbols
  - \( \delta \) : transition function
  - \( \lambda \) : output function
  - \( q_0 \) : initial state

- Input Enabled
  - \( \text{dom}(\delta) = \text{dom}(\lambda) = Q \times I \)

Running example

- \( Q = \{q_0, q_1, q_2, q_3\} \)
- \( I = \{a, b\} \)
- \( O = \{x, y\} \)
Mealy Machine Inference Algorithm

The Algorithm $L_M^*$

**Assumptions:**
- The input set $I$ is known
- Machine can be reset
- For each input, the corresponding output is observable
Basic principles of $L_M^*$ algorithm

<table>
<thead>
<tr>
<th>S (span seq for)</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>a</td>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>S . I lookahead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tail state id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>aa</td>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>ab</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

**Observation Table**

Conjecture:
minimal FSM consistent with observations

* $\varepsilon$ is an empty string
Mealy Machine Inference Algorithm $L_M^*$ (1/6)

Initialization

$S_M = \varepsilon$

$E_M = I$

• $\varepsilon$ is an empty string

Output Queries:

$s \cdot e, s \in (S_M \cup S_M \cdot I), e \in E_M$

• $= a / x$
Mealy Machine Inference Algorithm $L_M^*$ (2/6)

Concept: Closed

Concepts:
- Closed: All the rows in $S_M \cdot I$ must be equivalent to the rows in $S_M$
  - Same behaviour = known state
- Consistency

$\epsilon$ is an empty string
Mealy Machine Inference Algorithm $L_M^*$ (3/6)

Making Conjecture

$I = \{a, b\}$

<table>
<thead>
<tr>
<th>$S_M \cdot I$</th>
<th>$E_M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$x$</td>
</tr>
<tr>
<td>$a$</td>
<td>$x$</td>
</tr>
<tr>
<td>$b$</td>
<td>$x$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$x$</td>
</tr>
</tbody>
</table>

Counterexample:

```
 a b a b b a a
```

Component's response: $x\ x\ x\ x\ x\ b$

Conjecture's response: $x\ x\ x\ x\ x\ y$

Black Box Mealy Machine Component
Mealy Machine Inference Algorithm $L_M^*$ (4/6)

Processing Counterexamples

**Observation Table** $(S_M, E_M, T_M)$

**Counterexample**: $a \ b \ a \ b \ b \ a \ a$

**Method**:

Add all the prefixes of the counterexample to $S_M$
Mealy Machine Inference Algorithm $L_M^*$ (5/6)

Concept: Consistency

Concepts:

- **Closed**

- Consistency: All the successor rows of the equivalent rows must also be equivalent

- First inconsistency
  - $\varepsilon$ and $ab$ look similar… but not $\varepsilon.a$ and $ab.a$

- Later inconsistency:
  - $ab$ and $aba$, but not $aba$ and $abaa$

- ...
**Mealy Machine Inference Algorithm** $L_M^*$ (6/6)

Termination: Conjecture = Black Box

---

**Final Observation Table** $(S,M,E,T)$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>aa</th>
<th>aaa</th>
<th>baa</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>x</td>
<td>x</td>
<td>xy</td>
<td>xxx</td>
<td>xxxy</td>
</tr>
<tr>
<td>a</td>
<td>y</td>
<td>x</td>
<td>yy</td>
<td>xxx</td>
<td>xxxx</td>
</tr>
<tr>
<td>ab</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xyy</td>
<td>xxxx</td>
</tr>
<tr>
<td>aba</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xxx</td>
<td>xxxx</td>
</tr>
<tr>
<td>abab</td>
<td>y</td>
<td>x</td>
<td>yy</td>
<td>xxx</td>
<td>xxxx</td>
</tr>
<tr>
<td>ababb</td>
<td>x</td>
<td>x</td>
<td>xy</td>
<td>xxx</td>
<td>xxxx</td>
</tr>
<tr>
<td>ababba</td>
<td>x</td>
<td>x</td>
<td>xy</td>
<td>xxx</td>
<td>xxxx</td>
</tr>
<tr>
<td>ababaaa</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xyy</td>
<td>xxxx</td>
</tr>
<tr>
<td>b</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xyy</td>
<td>xxxx</td>
</tr>
<tr>
<td>aa</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xyy</td>
<td>xxxx</td>
</tr>
<tr>
<td>abb</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xxx</td>
<td>xxxx</td>
</tr>
<tr>
<td>abaa</td>
<td>y</td>
<td>x</td>
<td>yy</td>
<td>xxx</td>
<td>xxxx</td>
</tr>
<tr>
<td>ababa</td>
<td>x</td>
<td>x</td>
<td>xy</td>
<td>xxx</td>
<td>xxxx</td>
</tr>
<tr>
<td>ababbaa</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xyy</td>
<td>xxxx</td>
</tr>
<tr>
<td>ababbaab</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xxx</td>
<td>xxxx</td>
</tr>
</tbody>
</table>

**Complexity:** $O( |\Sigma| \cdot m \cdot n^2 )$

- $|\Sigma|$: the size of the input alphabet
- $n$: the number of states in the actual machine
- $m$: the length of the longest counterexample
Outline

- Motivation: why learning?
- ML & Soft. Engineering
- Seminal algorithm: L* (Angluin 87)
- Enhancements for various issues
  - Counter-example processing
  - Tree-based (quotient algo)
  - No Reset
  - Integration
  - EFSM
- Related work
Other algorithms derived from $L^*$

- Counter-example processing
  - Rivest & Schapire (1993)
    - Do not add prefixes (avoid compatibility check)
    - Dichotomic search for discriminating suffix
      - Complexity falls to $O(|\Sigma|n^2 + n \log m)$
      - But flawed (Balcazar 97)
    - Corrected by Shahbaz, Irfan and Groz (2009):
      - Suffix1by1
  - Only membership queries
    - Howar: Zulu competition at ICGI 2010
Processing Counterexamples avoiding consistency checks

Observation Table \((S_M, E_M, T_M)\) before processing counterexample

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\varepsilon)</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>a</td>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>b</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>aa</td>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>ab</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Counterexample

Add all the suffixes to \(E_M\)

Observation Table \((S_M, E_M, T_M)\) after processing counterexample

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>aa</th>
<th>baa</th>
<th>bbbaa</th>
<th>abbbaa</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\varepsilon)</td>
<td>x</td>
<td>x</td>
<td>xy</td>
<td>xxx</td>
<td>xxxy</td>
<td>xxxxx</td>
</tr>
<tr>
<td>a</td>
<td>y</td>
<td>x</td>
<td>yy</td>
<td>xxx</td>
<td>xxxx</td>
<td>yxxxx</td>
</tr>
<tr>
<td>b</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xxy</td>
<td>xxxx</td>
<td>xxxxy</td>
</tr>
<tr>
<td>aa</td>
<td>y</td>
<td>x</td>
<td>xx</td>
<td>xxx</td>
<td>xxxxy</td>
<td>xxxxx</td>
</tr>
<tr>
<td>ab</td>
<td>x</td>
<td>x</td>
<td>yy</td>
<td>xxx</td>
<td>xxxx</td>
<td>yxxxx</td>
</tr>
</tbody>
</table>
Comparison of the two Methods

Total Output Queries in $L_{M^+}$: 64

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>aa</th>
<th>baa</th>
<th>bbbaa</th>
<th>abbbba</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>x</td>
<td>x</td>
<td>xy</td>
<td>xxx</td>
<td>xxxy</td>
</tr>
<tr>
<td>a</td>
<td>y</td>
<td>x</td>
<td>yy</td>
<td>xxx</td>
<td>xxxx</td>
</tr>
<tr>
<td>b</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xxy</td>
<td>xxxxy</td>
</tr>
<tr>
<td>ab</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xxx</td>
<td>xxyx</td>
</tr>
<tr>
<td>aa</td>
<td>y</td>
<td>x</td>
<td>yy</td>
<td>xxx</td>
<td>xxyx</td>
</tr>
<tr>
<td>ba</td>
<td>x</td>
<td>x</td>
<td>xy</td>
<td>xxx</td>
<td>xxyx</td>
</tr>
<tr>
<td>bb</td>
<td>x</td>
<td>x</td>
<td>xy</td>
<td>xxx</td>
<td>xxyx</td>
</tr>
<tr>
<td>aba</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xxy</td>
<td>xxyx</td>
</tr>
<tr>
<td>abb</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xxy</td>
<td>xxyx</td>
</tr>
</tbody>
</table>

Final Observation Table ($S_{M^+}E_{M^+}T_{M^+}$) after processing counterexample according to $L_{M^+}$

Total Output Queries in $L_{M^*}$: 86

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>x</th>
<th>yy</th>
<th>xxx</th>
<th>xxx</th>
</tr>
</thead>
<tbody>
<tr>
<td>aba</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xxy</td>
<td>xxx</td>
</tr>
<tr>
<td>ababba</td>
<td>x</td>
<td>x</td>
<td>xy</td>
<td>xxx</td>
<td>xxx</td>
</tr>
<tr>
<td>ababbaa</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xxy</td>
<td>xxx</td>
</tr>
<tr>
<td>ababbaaab</td>
<td>x</td>
<td>x</td>
<td>xx</td>
<td>xxy</td>
<td>xxx</td>
</tr>
</tbody>
</table>

Final Observation Table ($S_{M^*}E_{M^*}T_{M^*}$) after processing counterexample according to $L_{M^*}$
Comparison of the two Methods

Total Output Queries in $L_M^*$: 86
Total Output Queries in $L_M^+$: 64

Observation Table ($S_M, E_M, T_M$) after processing counterexample according to $L_M^*$

Observation Table ($S_M, E_M, T_M$) after processing counterexample according to $L_M^+$

Complexity of $L_M^*$:
$O(|I|^2 n m + |I|^2 m n^2)$

Complexity of $L_M^+$:
$O(|I|^2 n + |I|^2 m n^2)$

- $I$: the size of the input set
- $n$: the number of states in the actual machine
- $m$: the length of the longest counterexample
Outline

- Motivation: why learning?
- ML & Soft. Engineering
- Seminal algorithm: L* (Angluin 87)
- Enhancements for various issues
  - Counter-example processing
  - Tree-based (quotient algo)
  - No Reset
  - Integration
  - EFSM
- Related work
Other active learning algorithms

- Other data structures: trees vs tables
  - Kearns & Vazirani (1994): binary tree
    - $O(|\Sigma|n^3 + nm)$
  - Z-quotient: tree & quotient automata
    - Petrenko, Li, Groz (HASE 2014)
  - TTT
    - Isberner, Howar, Steffen (RV 2014)
Mealy Machine Quotients

- Let $\Phi$ be a set of strings from $I$ then
  - the states $s_1$ and $s_2$ are $\Phi$-equivalent if they produce same outputs for all the strings in $\Phi$
  - A quotient based upon $\Phi$-equivalence is called $\Phi$-quotient

\[ \Phi = \{a, b, ab, ba, bb, bba\} \]

$q_0$ and $q_2$ are $\Phi$-Equivalent
$q_1$ and $q_3$ are $\Phi$-Equivalent
Relation between the Conjecture and the Black Box Machine

Closed (and Consistent) Observation Table \((S_M, E_M, T_M)\)

\[
\begin{array}{ccc}
S_M & a & b \\
\varepsilon & x & x \\
a & y & x \\
b & x & x \\
aa & y & x \\
ab & x & x \\
\end{array}
\]

Conjecture from the Observation Table \((S_M, E_M, T_M)\)

Black Box Mealy Machine

\(E_M\)-Quotient
Initial k-Quotient

Machine $M$

$q_0$ and $q_2$ are 1-Equivalent: $a/0, b/0$
$q_1$ and $q_3$ are 1-Equivalent: $a/1, b/1$

$q_0$ and $q_2$ are still 2-Equivalent
$q_1$ and $q_3$ are 2-Distinct. $a/1, a/\_?

3-Quotient($M$) $\equiv$ $M$

2-Quotient of $M$

1-Quotient of $M$
Inferring a k-quotient
(example with k=1)

- BFS exploration of traces of increasing length
- Pruning under node k-equiv to another one
- Final step: merging node when trace included, and redirecting transitions

Groz, Li, Petrenko, Shahbaz TestCom 2008

Extended to arbitrary $\Sigma$-quotients $\Sigma \subseteq \mathcal{I}^*$
Outline

- Motivation: why learning?
- ML & Soft. Engineering
- Seminal algorithm: L* (Angluin 87)
- Enhancements for various issues
  - Counter-example processing
  - Tree-based (quotient algo)
  - No Reset
  - Integration
  - EFSM
- Related work
Motivational example

- Reverse-engineer models of Web applications to detect security vulnerabilities
- E-Health app provided by Siemens as a Virtual Machine
  
  ![VirtualBox Diagram](Image)
  
  Learner
  
  - single I/O RTT over LAN: < 1 ms
  - reset=reboot VM: ~1 minute

- Timewise: reset is $O(10^5)$ RTT in example
- Many systems CANNOT be reset AT ALL.
Key difficulties when no reset

- How can we know in which state seq is applied?
- No backtrack possible to check other sequence
- Losing track: we no longer know from where we apply an input
  - localizer procedure

*Can we infer a Black-Box machine without reset?*
Problem, assumptions, result

Groz, Simao et al 2015

- Known bound $N$ on nb of states: $n \leq N$

- Known $W$-set for BB
  - $\text{Card}(W) = p$
  
  \textbf{Algo: polynomial in } N
  
  $\ll O(f N^{p+2})$ bound
  
  \textit{but mean } $O(f N^{1.9})$ for $p=2$

\textbf{Stronger assumptions}

Rivest & Schapire 1993

- Oracle knows BB, can answer yes or no

- Oracle can provide CE
  - $|\text{Largest CE}| = m$

- Known Homing Sequence for BB

  \textbf{Algo: polynomial in } n
  
  $\sim O(f m n^3)$

\textbf{Lower practical complexity for } $p \leq 2$
Example: \( W = \{a, b\}, \ N=3 \)

Localizer seq. \( L = a^5b \)

\[
\begin{array}{c}
1 \\
2 \\
3 \\
\end{array}
\]

\[
\begin{array}{c}
a/1 \\
a/0 \\
b/0 \\
\end{array}
\]

\[
\begin{array}{c}
a/0 \\
a/0 \\
b/1 \\
\end{array}
\]

\[
\begin{array}{c}
a/0 \\
b/0 \\
\end{array}
\]

\[
\begin{array}{c}
a/1 \\
b/0 \\
\end{array}
\]

\[
\begin{array}{c}
a/0 \\
a/0 \\
b/1 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
a/0 \\
b/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]

\[
\begin{array}{c}
b/1 \\
a/0 \\
\end{array}
\]
Example (end)
It pays off to learn without reset!

Learner

- single I/O RTT over LAN: <1ms
- reset=reboot VM: ~1 minute

Cost of single reset ~sequence of $10^5$ inputs
- If we know $W$ of 2 elements, it is FASTER to learn WITHOUT reset!
- If we know $W$ of 3 elements, it may still pay off depending on number and length of queries
Outline

- Motivation: why learning?
- ML & Soft. Engineering
- Seminal algorithm: L* (Angluin 87)
- Enhancements for various issues
  - Counter-example processing
  - No Reset
  - Integration
  - EFSM
- Related work
Integration testing

- Popular issues
  - Architecture, testability
  - Integration order, stubbing
  - Interoperability testing

- Combining integration with Model learning
  - Unit learning (1st approach)
  - Deriving integration tests from combined learned models
Integration exposes models

Component U: \( I_U = \{X,y\} \)

Component V: \( I_V = \{a,b\} \)

Composed Model: \( X a y b y a y b y a y \ldots \) Livelock!
Analysing the problem

- Artefact?
  - Possibly: models are approximate
- Check sequence on real system
  1. If Livelock confirmed: report error
  2. If Real sequence differ: counter example
Integration provides counter-examples

Component U: \( I_U = \{X, y\} \)
Component V: \( I_V = \{a, b\} \)

Composed Model:
\[
X \rightarrow a \rightarrow y \rightarrow b \rightarrow y \rightarrow a \rightarrow y \rightarrow b \rightarrow y \rightarrow a \rightarrow y \rightarrow b \rightarrow y \rightarrow a \rightarrow y \rightarrow \ldots
\]
Livelock!

Real:
\[
X \rightarrow a \rightarrow y \rightarrow b \rightarrow y \rightarrow a \rightarrow y \rightarrow b \rightarrow y \rightarrow a \rightarrow y \rightarrow b \rightarrow C
\]

\( \rightarrow \) Refine U model with (projected) counter-example
System architecture & assumptions

- System of communicating Mealy Machine Components
- Components are deterministic and input-enabled
- System has External and Internal i/o interfaces
  - External interface is controllable
  - External and Internal interfaces are observable
- Single Message in Transit and Slow Environment
Overview (simplified)

Verification

Inference

Components' Analysis & Testing

**Given:** Modular System

**Goal:** Error detection

Legend:
- Tasks performed by testing real components
- Tasks performed on models

Step 1
- Model Extraction

Step 2
- Reachability Analysis

Step 3
- Test generation
  - Pb on real System?

Step 4
- Refinement
Iterations

U. T.: Unit testing
I. T.: Integration testing
C.E.: Counter-example
Learning & Testing Framework

Step 1: Learn Models

Step 2(a): Compose Models

Step 2(b): Analyze Product

Step 2(c): Confirm Problem on System

Step 3: Refine Models

Step 4: Generate Tests from Product

Step 5: Resolve Discrepancy (exception, crash, out of memory,...?)

No Compositional Problems

[problem as counterexample]

[composition problem]

[problem confirmed]

[error found]

[discrepancy as counterexample]

[no discrepancy]
Outline

- Motivation: why learning?
- ML & Soft. Engineering
- Seminal algorithm: L* (Angluin 87)
- Enhancements for various issues
  - Counter-example processing
  - No Reset
  - Integration
  - EFSM
- Related work
Learning extended FSM

- Dealing with boolean variables

- Parameterized inputs/outputs
  - no var, arbitrary I/O functions: Shahbaz 2007
  - Var. with equality: Berg, Jonsson… 2008

- With variables
  - Register automata: Howar et al VMCAI 2012
  - With Data Mining inference of guards and output functions: Li, Hossen, Groz
Combining state & data inference

- Connecting to Daikon tool, for dynamic invariant detection
  - Shahbaz ISOLA 2007
    
    *Daikon: inductive inference of functions from samples*  
    
    *y=f(x)*  
    
    M. Ernst (U. Washington)

- Weka & FSM inference
  - Dury & Petrenko: security of Web interface
  - Li & Groz: EFSM inference
    
    *Weka: data mining toolset, clustering*  
    
    (U. Waikato)
Inferring for security

- Input parameters critical (e.g. Cross site scripting...)
- Storing past values: cookies, session IDs
- Non-deterministic values: nonces

- Model: Extended FSM with ND values, and storage
Needham Schroeder authentication

Extended FSM model of NSPK Responder

Inferred EFSSM model
State inference // data inference

<table>
<thead>
<tr>
<th></th>
<th>$m_1$</th>
<th>$m_1$</th>
</tr>
</thead>
</table>
| $\varepsilon$ | $(5, m_2)$  
(\text{ndv}_3, m_2)  
(10, $\Omega$)  
(\text{ndv}_3, $\Omega$) |  
| $m_3(10)$ | $(5, m_2)$  
(\text{ndv}_3, m_2)  
(10, KO)  
(\text{ndv}_3, OK) |  
| $m_1(5)$  
$m_1(5)$  
$m_1(5)$  
$m_1(5)m_3(10)$ | $(5, m_2)$  
(\text{ndv}_3, m_2)  
(10, KO)  
(\text{ndv}_3, OK) |  
| $m_3(10)$ | $(5, (0, 0, 0, 10)$  
$\rightarrow (5, 600)),$  
$(0, (0, 0, 0, 10)$  
$\rightarrow (0, 800))$ |  
| $m_1(5)$  
$m_1(5)$  
$m_1(5)m_3(10)$ | $(5, (5, 5, 900,$  
$0) \rightarrow \omega)$,  
$(0, (5, 5, 110,$  
$0) \rightarrow \omega)$ |  
| $m_3(10)$ | $(5, (5, 5, 140,$  
$10) \rightarrow \omega),  
(150, (5, 5, 150,$  
$10) \rightarrow \omega),$ |  
| $\varepsilon$ | $(5, (5, 5, 120,$  
$10) \rightarrow \omega),  
(130, (5, 5,$  
$130, 0) \rightarrow \omega)$ |  
| $m_3(10)$ | $(5, (5, 5, 150,$  
$10) \rightarrow \omega),  
(160, (5, 5,$  
$160, 10) \rightarrow \omega)$ |  
| $m_1(5)m_3(10)$ | $(5, (5, 5, 150,$  
$10) \rightarrow \omega),  
(160, (5, 5,$  
$160, 10) \rightarrow \omega)$ |  

61
Outline

- Motivation: why learning?
- ML & Soft. Engineering
- Seminal algorithm: L* (Angluin 87)
- Enhancements for various issues
  - Counter-example processing
  - No Reset
  - Integration
  - EFSM

- Related work
Related work

- Active learning in Soft. Eng/Testing
  - D. Peled (Bar-Ilan): Black Box Checking (1999)
  - D. Lee & G. Shu (Ohio 2007): Security protocol testing
  - B. Jonsson, T. Berg (Uppsala): Register automata
  - K. Meinke (KTH): Learning Based Testing (& model checking), Congruence on Abstract Data Types
  - F. Vaandrager, S. Verwer (Nijmegen): Smartcard
Related work

- Many other approaches
  - Specification mining, becoming popular
    - May assume code available, often passive
  - Typical papers:
    - Ammons (POPL 2002) coined the word
    - Lorenzoli, Mariani, Pezze (ICSE 2008)
    - Bertolino, Inverardi (FSE 2009)

- Use of (statistical) Machine Learning in testing
  - E.g. for test data classification & partition refinement (Briand 2008)
Reference book on learning automata

• For machine learning in general:
  • Many references,
  • e.g. A. Cornéjuols & L. Miclet

• No book as yet for Software Testing & machine learning
  • Planned April 2017 (Springer): outcome of Dagstuhl seminar 2016