Active Automata Learning: From DFA to Interface Programs and Beyond
or
From Languages to Program Executions
or (more technically)
The Power of Counterexample Analysis

Bernhard Steffen, Falk Howar, Malte Isberner
TU Dortmund /CMU
Connect Scenario

CONNECT

- learner
  - look for known models

- connector
  - inform about new service and device
  - try to use
  - interrogate

some service

try to use

interrogate

CONNECT
Data-Dependent Control

Value-independent Data Dependencies

- automated synthesis of connectors between networked systems
- inference of executable "interface programs"

```plaintext
init session;
beginTransaction(session);

if (returns ✓) then
...

completeTransaction(session);
...

beginTrans.(sVal)

on beginTransaction(sVal): 
  sid := sVal;
  return ✓;

on completeTransaction(sVal):
  if (sVal = sid) then
    return ✓;
  else ...
How to Extend w. Data?

Data is crucial for modeling
- Interface specifications
  - relate data in input to data in subsequent output
- Communication protocols
  - sequence numbers, identifiers, ..

(External) Mapper-Based Data Treatment

Explicit Data Modelling
Outline

- Background
- Manual Treatment of Data
- Automated Alphabet Abstraction Refinement
- Modelling Data Explicitly
- Conclusions
The Concrete Scenario

Test Coordinator

Rational Robot

HTTP

PCM
Application PCs

CSTA II/III

PCM
Application Server
Means of Observation

(smallest) learned models imposed major test suite optimizations
Moderated, Regular Extrapolation

- **Extrapolation**
  
  Hypothesis Building beyond known facts

- **Regular**
  
  Extrapolation-Universe: Extended Finite Automata

- **Moderated**
  
  The Extrapolation Process requires targeted interaction

Neither Correct nor Complete!
Models in our Scenario

**Abstract** representation of the protocol-level behaviour.

Abstraction typically concerns replacing "symbolic names" (e.g., "500" for a phone number) and removing "time stamps etc."

```plaintext
{ invokeId = 58391,
  operation-value = 21 (cSTAEventReport),
  {eventSpecificInfo. ... .hookswitch
   {deviceId.dialingNumber = "500"
    hookswitchOnHook= TRUE,
    ...
    timestamp = "20001010095551"
  } }}

{obsEvent
  deviceId = A1
  switchOnHook, ...
}]
```
Models comprise state changes as well as UPN- and CSTA-Observations.

Sys_Info

\[
\{ \text{deviceId = A1, hookswitchOnHook, ... } \}
\]

device A1
display(line 1, ...)
LEDs: (1,on) (2,off)
...
...

Sketch of the Model Structure
Active Automata Learning

OT

Distinguishing Futures

Lower Hypothesis Automaton

Unknown System

Reaching Words

Closeness & Consistency Validation

Transitions
### Membership Queries

**Abstract States**

<table>
<thead>
<tr>
<th>OT</th>
<th>ε</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
</tr>
</tbody>
</table>

**Transition Relation**

- Not closed!

**Unknown System**
Closure & Consistency

<table>
<thead>
<tr>
<th>OT</th>
<th>ε</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>ba</td>
<td>0</td>
</tr>
<tr>
<td>bb</td>
<td>0</td>
</tr>
</tbody>
</table>
Equivalence Queries

<table>
<thead>
<tr>
<th>OT</th>
<th>( \varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon )</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>ab</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>ba</td>
<td>0</td>
</tr>
<tr>
<td>bb</td>
<td>0</td>
</tr>
</tbody>
</table>

Unknown System

Counterexample: \( ab \in L \)
## Counter Example-Based Extension

### Unknown System

<table>
<thead>
<tr>
<th>OT</th>
<th>ε</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>ab</td>
<td>1</td>
</tr>
<tr>
<td>ba</td>
<td>0</td>
</tr>
<tr>
<td>bb</td>
<td>0</td>
</tr>
<tr>
<td>aa</td>
<td>0</td>
</tr>
<tr>
<td>aba</td>
<td>0</td>
</tr>
<tr>
<td>abb</td>
<td>1</td>
</tr>
</tbody>
</table>

**Counterexample:** $ab \in L$
### Closure & Consistency

#### Unknown System

**Not consistent:**

row (ε) = row (a), but row (εa) ≠ row (aa)

**New Column:** $a$
### Unknown System

#### Next Iteration

<table>
<thead>
<tr>
<th>OT</th>
<th>( \epsilon )</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \epsilon )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ab</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ba</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bb</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>aa</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>aba</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>abb</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Finished!
Active automata learning: $L^*$

$\Sigma = \{a, b\}$

$aba \in L$?

no

MQ-Oracle

no, $bb \in L$!

EQ-Oracle
Summary of L* algorithm

L* infers Finite State Machine from queries:

1. Pose membership queries until “saturation”
2. Construct Hypothesis from obtained information
3. Pose equivalence query
4. if no look at counterexample and goto 1
5. else return Hypothesis end

- Has been used to learn large automata (≥20k states)
- Adapted for Mealy Machines [Niese et al. 2003]
- and for Interface Automata [Aarts et al. 2010]
- Efficient Tool: LearnLib [TUDortmund]
### Analysis of Counterexamples I

#### Rivest Shapire

A table showing all prefixes of a counterexample, with one essential suffix highlighted.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>bb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>bb</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>bbb</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>aa</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ab</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ba</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

...
Analysis of Counterexamples I

Hyp:

Target:

Hyp:
- Binary search on counterexample
- Requires $O(\log_2(m))$ membership queries
Analysis of Counterexamples I

- Binary search on counterexample
- Requires $O(\log_2(m))$ membership queries
Analysis of Counterexamples I

- Binary search on counterexample
- Requires $O(\log_2(m))$ membership queries
Analysis of Counterexamples I

- Binary search on counterexample
- Requires $O(\log_2(m))$ membership queries
Analysis of Counterexamples I

BOTH SEQUENCES ARE PREFIXES IN TABLE!!!

- Binary search on counterexample
- Requires $O(\log_2(m))$ membership queries
Rivest and Shapire: Analyze counterexample separately (not in the table)

Only add one 'essential' suffix (i.e., witness), as column label to the table

Consequence:
- Guaranteed Consistency!
- Improved worst case complexity

BUT: Hypothesis Automata are no longer guaranteed to be minimal! (cf. Pnueli / Mahler's criticism)
Analysis of Counterexamples I

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>bb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

One essential suffix

Essential suffix

Word from S

Word from SA

as(ε, 0, 0)

as(c₁, c₂, c₃, c₄)

as(c₁, c₂, c₃)

...
Efficient equivalence test heuristics

- **Two paradigmatic changes:**
  1. Subsequent hypothesis models are **evolving**
  2. Equivalence query = **searching** counterexamples
Non-uniform observation tables

• Novel data structure/algorithm, inspired by the idea of ‘observation packs’

• Allows different strategies for handling counterexamples

• Non-uniform configuration => minimal number of tests during learning

Important for Zulu, even more important for CONNECT
### Detailed results by example

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>New Membership</th>
<th>Queries</th>
<th>Rounds</th>
<th>States</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Close</td>
<td>Analyze</td>
<td>Search</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E.H. Blocking</td>
<td>6,744</td>
<td>358</td>
<td>999</td>
<td>259</td>
<td>352</td>
</tr>
<tr>
<td>E.H. Weighted</td>
<td>6,717</td>
<td>340</td>
<td>1,025</td>
<td>262</td>
<td>351</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Falk Howar, TUDortmund
2. Borja Balle, UPC
3. Sarah Eisenstat, MIT

- MQs / EQ: 1-3 (uniform), ca. 3.9 (non-uniform), ca. 4.36 (random)
- MQs / state: ca. 25 (uniform), ca. 19 (non-uniform)
- Random walks: higher cost for analyzing counterexamples
Outline

- Background
- Manual Treatment of Data
- Automated Alphabet Abstraction Refinement
- Modelling Data Explicitly
- Conclusions
Simple Stack

```java
public class Stack {
    private int capacity = 3;
    private int size = 0;
    private Object elements[] = new Object[capacity];

    public boolean push(Object o) {
        if (size == capacity) return false;
        elements[size++] = o; return true;
    }

    public Object pop() {
        if (size == 0) return null;
        return elements[--size];
    }
}
```
Mappers

\[ L^* \quad \xrightarrow{(\Sigma^L)^*} \quad \text{Mapper} \quad \xrightarrow{(\Sigma^S)^*} \quad \text{SUL} \]

\[ (\Omega^L)^* \]

\[ \gamma : (\Sigma^L)^* \rightarrow (\Sigma^S)^* \]

\[ \alpha : (\Omega^S)^* \rightarrow (\Omega^L)^* \]
Learning the stack as a language

- `stack.push(1)`
- `stack.pop()`
- `true`, `false`, `null`, `1`

\[ L^* \quad \text{push, pop} \quad \in L, \notin L \]

Mapper

SUL
Introducing outputs: Mealy machines

\[
\begin{align*}
&l_0 \\ \downarrow \quad \text{push} / \text{OK} & l_1 & l_2 & l_3 \\
&\quad \text{pop} / 1 & \text{push} / \text{OK} & \text{push} / \text{OK} & \text{push} / \text{NOK} \\
&\quad \text{pop} / \text{null} & \text{pop} / 1 & \text{pop} / 1 & \\
&l_0 \\ \downarrow \quad \text{push} / \text{OK} & l_1 & l_2 & l_3 \\
&\quad \text{pop} / 1 & \text{push} / \text{OK} & \text{push} / \text{O}
\end{align*}
\]
Introducing outputs: Mealy machines

**L***

- push1, push2, pop
- OK, NOK, null, 1, 2

**Mapper**

- stack.push(1)
- Stack.push(2)
- stack.pop()
- true, false, null, 1, 2

**SUL**
Outline

- Background
- Manual Treatment of Data
- Automated Alphabet Abstraction Refinement
- Modelling Data Explicitly
- Conclusions
Automated Alphabet Abstraction Refinement

<presence type=… />
<iq type="result" />

Test-driver

Available
OK

LearnLib

Static alphabet abstraction
Automated Alphabet Abstraction Refinement

Learning relative to a given representation system

Static alphabet abstraction

<presence type=… />
<iq type=“result“ />
The Mod-k Stack

finite set of outputs, e.g.: odd / even

- push
- push' push push pop / odd push push' pop / even

\[ L^* \]

push, push', pop

OK, NOK, null, odd, even

Mapper

stack.push(51);
stack.push(2012);
stack.pop()

SUL

true, false, null, 51, 2012
Analysis of Counterexamples II
Counter Examples and Witnesses

Separating pattern

- $\gamma(\alpha(c_1))$
- $\gamma(\alpha(c_2))$
- $\gamma(\alpha(c_3))$
- $\gamma(\alpha(c_4))$
- $d$
- $c_4$
- $p$
- $c_5$
- $c_6$

$p$ state
$c_4$ representation
$d$ future
\[ \Sigma_C \setminus \alpha_{\text{old}}(c) \]

\[ \gamma(\alpha(p)) \times d \in F \iff \gamma(\alpha(p)) \cdot c \cdot d \in F \]

\[ \gamma_{\text{old}}(\alpha_{\text{old}}(c)) \]

\[ \Sigma_C \]

\[ \text{push} \]

\[ \text{push}' \]
Case Study

- Biometric Passport
- [Aarts et. al, 2010]

262 Concrete symbols, 256 x readFile(i).

- 1 initial abstract symbols
- 8 alphabet refinements, to split readFile
- 9 final abstract symbols

‘read file(i)‘ aggregated according to the required authentication
Outline

- Background
- Manual Treatment of Data
- Automated Alphabet Abstraction Refinement
- Modelling Data Explicitly
- Conclusions
How to Extend with Data?

Data is crucial for modeling

- Interface specifications
  - relate data in input to data in subsequent output
- Communication protocols
  - sequence numbers, identifiers, ..

Extend automaton model

- Data parameters in actions
- State variables to remember parameter values

How to extend the learning techniques?
Register Automata

- locations
- registers (e.g., $id$, $pw$)
- transitions with:
  - actions with formal parameters
  - guards
  - assignments to registers

Diagram:

- States: $l_0$, $l_1$, $l_2$
- Edges:
  - $l_0$ to $l_0$: $\text{delete}()$ | $true$
  - $l_0$ to $l_1$: $\text{register}(p_1, p_2)$ | $true$
    - $id := p_1, pw := p_2$
  - $l_2$ to $l_1$: $\text{login}(p_1, p_2)$ | $id = p_1 \land pw = p_2$
  - $l_2$ to $l_2$: $\text{change\_pw}(p_1)$ | $true$
    - $pw := p_1$
  - $l_1$ to $l_1$: $\text{logout}()$ | $true$
  - $l_1$ to $l_1$: $\text{login}(p_1, p_2)$ | $id \neq p_1 \lor pw \neq p_2$
The Impact of Register Automata

Query: push(p₀)/OK push(p₁)/OK pop()/p₂

L* push(p)/OK, pop()/o(p), ...

Mapper

stack.push(51);
stack.push(2012);
stack.pop()

e L, f L

ture, false, null, 51, 2012

SUL
Principles of L* learning

L* exploits Nerode congruence:

- Membership queries formed as compositions of
  - finite set of prefixes
    - represent states of automaton
    - extended until all states have been covered
  - finite set of suffixes
    - allow to approximate **Nerode congruence** on prefixes (states)
    - extended until any inequivalent states separated by suffixes

- How to extend to languages with data?
  - Define Nerode congruence and adapt the above.
  - But: this has been very difficult for many automata models with data (e.g., timed automata).
A Data-Aware Nerode-Relation

**Problem:** classical Nerode-relation has infinite index.

\[ \text{register}(\text{Bob}, 123) \text{ login}(\text{Bob}, 123) \]
\[ \text{register}(\text{Alice}, 456) \text{ login}(\text{Alice}, 456) \]

Let \( \mathcal{W}_D \) be the set of all data words.

**Definition (Equivalence wrt. \( \mathcal{L}_D \))**

Two words \( u, u' \in \mathcal{W}_D \) are equivalent wrt. \( \equiv \) iff there exists a permutation \( \pi \) on \( D \) s.t. for all \( v \in \mathcal{W}_D \)

\[
u v \in \mathcal{L}_D \iff v \pi(v) \in \mathcal{L}_D
\]

For \( \pi(\text{Bob}) = \text{Alice}, \pi(123) = 456, \) and \( \pi = \text{id} \) otherwise

\[ \text{register}(\text{Bob}, 123) \equiv L \text{ register}(\text{Alice}, 456) \]
### Reusing structure of $L^*$

$L$-essential prefixes *(just enough equal data values)*

#### Abstract suffixes

#### Red prefixes

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon$</th>
<th>$\text{login}(z_1, z_2)$</th>
<th>$\text{logout()}\text{login}(z_1, z_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>($l_0$)</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>register(1, 2)</td>
<td>($l_1$)</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>register(1, 2)$\text{login}(1, 2)$</td>
<td>($l_2$)</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
</tbody>
</table>

#### Blue prefixes

<table>
<thead>
<tr>
<th></th>
<th>$\text{login}(1, 2)$</th>
<th>$\text{logout()}\text{login}(1, 2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>register(1, 2)$\text{login}(3, 4)$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>register(1, 2)$\text{login}(1, 2)\text{ch_pw}(3)$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>register(1, 2)$\text{login}(1, 2)\text{logout()}$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
</tbody>
</table>

**id**

(1,1) (3,2)

---

[Visit Connect-Forever.eu](http://connect-forever.eu)
Counterexample Analysis for inferring

- New locations
- New registers
- New transitions
CE: New location
CE: New location
CE: New location
CE: New location

Diagram showing a network flow with nodes and labels such as "u", "v", and "X(v)", with transitions labeled "a" and "ua". The diagram includes nodes labeled "l0", "l1", and "l2" with transitions labeled "register(p1, p2)" and "login(q1, q2)".
CE: New location
CE: New location
CE: New register

\[ l_0 \xrightarrow{u} l_1 \xrightarrow{a} l_2 \]

SUL

\[ l_0 \xrightarrow{\text{register}(p_1, p_2)} \]
\[ l_0 \xleftarrow{\text{keep } p_1 \text{ and } p_2} \]
\[ l_1 \xleftarrow{\text{login}(q_1, q_2)} \]
\[ l_2 \xrightarrow{\text{if } q_1 = p_1 \text{ and } q_2 = p_2} \]

Hyp.

\[ l_0 \xrightarrow{\text{register}(p_1, p_2)} \]
\[ l_1 \xleftarrow{\text{Hyp.}} \]
CE: New register
CE: New register
CE: New register

\[
\begin{align*}
&l_0 \xrightarrow{u} l_1 \xrightarrow{a} l_2 \\
&l_2 \xrightarrow{v} l_3
\end{align*}
\]

**SUL**

- \(l_0 \xrightarrow{register(p_1, p_2)} l_1\)
- \(l_0 \xrightarrow{login(q_1, q_2)} l_2\)
- \(l_2 \xrightarrow{keep \ p_1 \ and \ p_2} l_1\)
- \(l_2 \xrightarrow{\text{if } q_1 = p_1 \ and \ q_2 = p_2} l_0\)

**Register**

- \(register(1, 2)\)
- \(login(1, 2)\)
- \(login(3, 4)\)

**Hyp.**

\[
\begin{align*}
&x_1 := p_1; x_2 := p_2 \\
&l_0 \xrightarrow{\text{register}(p_1, p_2)} l_1
\end{align*}
\]
CE: New transition

1. From state $l_0$ to $l_1$ with input $u$, transition labeled $\alpha$.
2. From state $l_1$ to $l_3$ with input $v$, transition labeled $\bar{\alpha}$.

**SUL**
- From $l_2$ to $l_0$ with input $\text{register}(p_1, p_2)$, keeping $p_1$ and $p_2$.
- From $l_0$ to $l_2$ with input $\text{login}(q_1, q_2)$ if $q_1 = p_1$ and $q_2 = p_2$.

**Hyp.**
- From $l_0$ to $l_1$ with input $\text{login}(p_1, p_2)$ if true.
CE: New transition

\[ l_0 \xrightarrow{u} l_1 \xrightarrow{a} l_3 \]

- \( l_0 \rightarrow l_1 \) over \( u \)
- \( l_1 \) to \( l_3 \) over \( a \)

\[ \square \]

- \( l_1 \xrightarrow{\bar{a}} ? \)
- \( l_3 \xrightarrow{v} ? \)

\[ \square \]

- \( l_0 \rightarrow l_1 \) over \( \text{SUL} \)
- \( l_0 \rightarrow l_2 \) over \( \text{register}(p_1, p_2) \), keep \( p_1 \) and \( p_2 \)
- \( l_2 \rightarrow l_1 \) over \( \text{login}(q_1, q_2) \), if \( q_1 = p_1 \) and \( q_2 = p_2 \)
- \( l_1 \rightarrow l_0 \) over \( \text{login}(p_1, p_2) \), true

Hyp.
CE: New transition

- Transition from $l_0$ to $l_1$ is marked by $u$.
- Transition from $l_1$ to $l_3$ is marked by $v$.
- Transition from $l_3$ to $\_\_\_\_$ is marked by $??$.

- States $l_0$, $l_1$, and $l_3$ are connected by transition $u$.
- States $l_1$ and $l_3$ are connected by transition $v$.

- Transition $\alpha$ is marked between $l_0$ and $l_1$.
- Transition $\bar{\alpha}$ is marked between $l_1$ and $l_3$.

- The SUL (Specified Use Level) includes states $l_0$, $l_1$, and $l_2$.
- Transitions include:
  - $i_0$ to $l_0$: $register(p_1, p_2)$, keep $p_1$ and $p_2$.
  - $i_1$ to $l_1$: $login(q_1, q_2)$ if $q_1 = p_1$ and $q_2 = p_2$.

- Additional transitions include:
  - $l_1$ to $l_0$: $login(1, 2)$ $\in$ $\emptyset$.
  - $l_1$ to $l_0$: $register(1, 2) login(3, 4) \in \emptyset$.

- Hypothesis: $login(p_1, p_2)$ is true.
CE: New transition

\[ l_0 \xrightarrow{u} l_1 \xrightarrow{\bar{a}} l_3 \]

\[ \text{register}(p_1, p_2) \quad \text{keep } p_1 \text{ and } p_2 \]

\[ \text{login}(q_1, q_2) \quad \text{if } q_1 = p_1 \text{ and } q_2 = p_2 \]

\[ \text{login}(1,2) \quad \text{login}(3,4) \]

\[ \text{Hyp.} \]

\[ \text{login}(p_1, p_2) \quad \text{true} \]
Experimental Evaluation

\[
\begin{array}{ccc}
\text{delete()} & \text{true} & \text{register}(p_1, p_2) \\
& & \text{id} := p_1, pw := p_2 \\
\text{login}(p_1, p_2) & \text{id} = p_1 \land pw = p_2 & \text{logout()} \text{ true} \\
\text{change-pw}(p_1) & \text{true} & \text{login}(p_1, p_2) \text{ id} \neq p_1 \lor pw \neq p_2 \\
& \text{pw} := p_1 & \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Setup</th>
<th># Loc.</th>
<th># Trans.</th>
<th>MQs</th>
<th>EQs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(L^*), no optimization, (</td>
<td>D</td>
<td>= 6)</td>
<td>73</td>
<td>5,913</td>
</tr>
<tr>
<td>(L^*), symmetry reduction, (</td>
<td>D</td>
<td>= 6)</td>
<td>73</td>
<td>5,913</td>
</tr>
<tr>
<td>RA learning algorithm</td>
<td>3</td>
<td>16</td>
<td>403</td>
<td>3</td>
</tr>
</tbody>
</table>

http://connect-forever.eu/
Modeling Output explicitly: RMMs

Example: Stack of capacity 3
- RA: output encoded as guarded transition
- RMM: output with data for transitions

```
RA

l0
(put,(p)) | true
x1:=p

l1
(get,(p)) | p=x1

l2
(put,(p)) | true
x2:=p

l3
(get,(p)) | p=x2

```

```
RMM

l0
(put,(p)) | true
x1:=p

l1
(get,()) | true
x1:=p

l2
(put,(p)) | true
x2:=p

l3
(get,()) | true
x2:=p

```

“… is in language”

“… leads to output …”
RMM: Explicit Output

Query: $\text{push}(p_1) \text{push}(p_2) \text{pop()} / p_2$

$L^*$  \[\text{push}(p), \text{pop()} \Rightarrow \text{OK, NOK, null, p}\]

Mapper  \[\text{stack.push}(51)\]

SUL  \[\text{stack.push}(2012)\]

\[\text{stack.pop()}\]

true, false, null, 51, 2012
Inferring RMMs

- **Example:** Nested stack of capacity 16
  - RMM: 781 locations, 45k MQ, 9 EQ, 20 sec.
  - Mealy, \(|D|=4\): > 10⁹ states

| Name     | Mealy (\(|D|=4\)) | RA | RMM |
|----------|--------------------|----|-----|
|          | |MQs| EQs| |MQs| EQs| |MQs| EQs|
| Stack (1)| 5  | 17 | 0  | 3  | 35 | 2  | 2  | 10 | 0  |
| Stack (2)| 21 | 53 | 1  | 4  | 135| 4  | 3  | 18 | 0  |
| Stack (3)| 85 | 232| 3  | 5  | 554| 6  | 4  | 38 | 1  |
| Stack (4)| 341| 854| 4  | 6  | 2998| 8 | 5  | 53 | 2  |
Outline

- Background
- Manual Treatment of Data
- Automated Alphabet Abstraction Refinement
- Modelling Data Explicitly
- Conclusions
Main Practical Challenges are

- Search for Counterexamples
- **Counterexample Analysis**

**Question**: How much can counter examples tell about a system?

We have seen scenarios for (beside the classical locations),

- Optimal Alphabet Abstraction
- Optimal Register Allocation
- Optimal Transition Functions

We have seen how to get

- From DFA to Interface Programs
- From Languages to Program Executions
Conclusions and Perspectives

Beyond: Investigation of language extensions
- Extended Guards
- Actions with Effect
- Procedural Structure?

Hybrid Approaches and Case Studies

Experimental Evaluation and Performance Analysis

The RERS Greybox Challenge 2012