

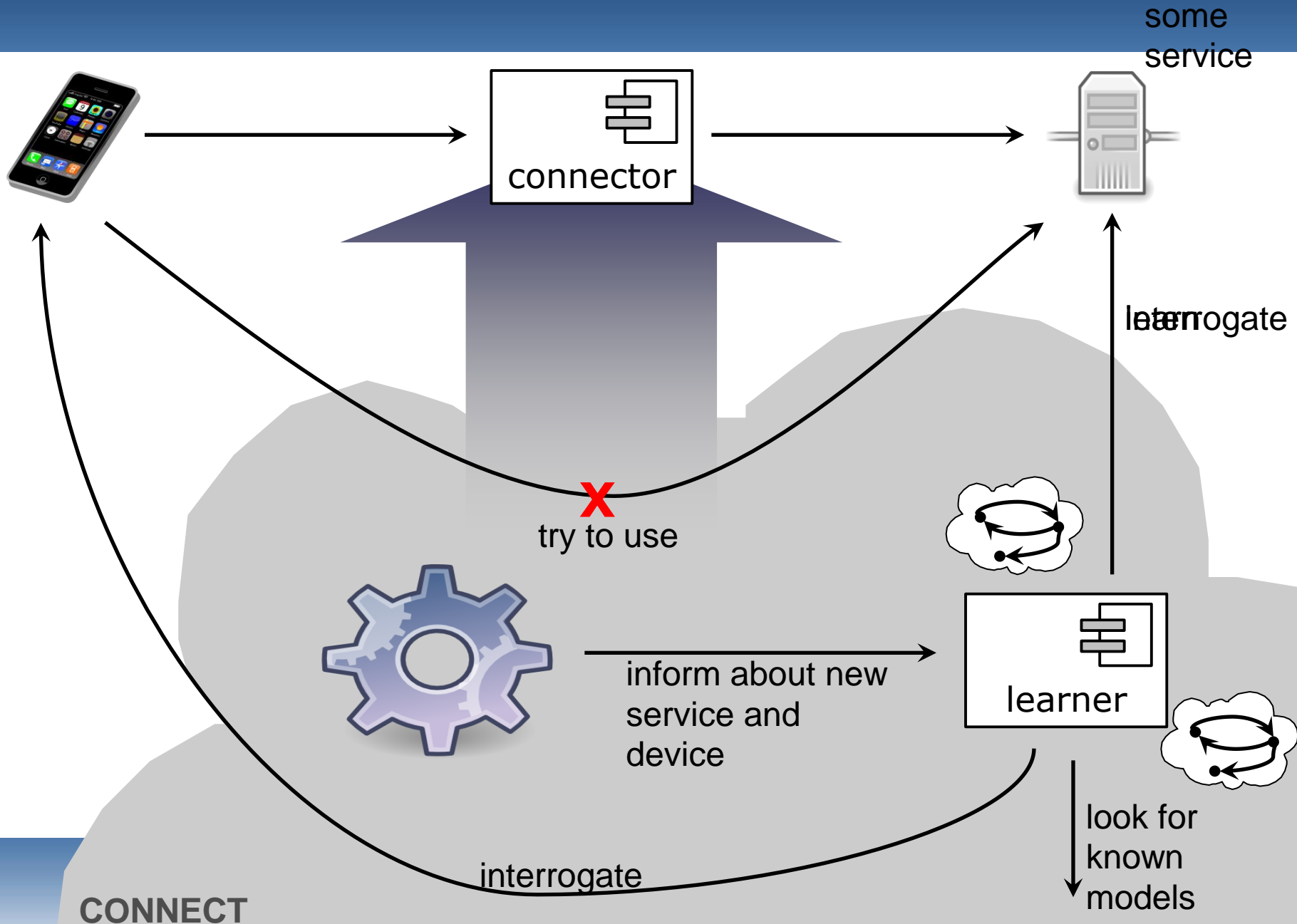
Active Automata Learning: From DFA to Interface Programs and Beyond or

From Languages to Program Executions
or (more technically)

The Power of Counterexample Analysis

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

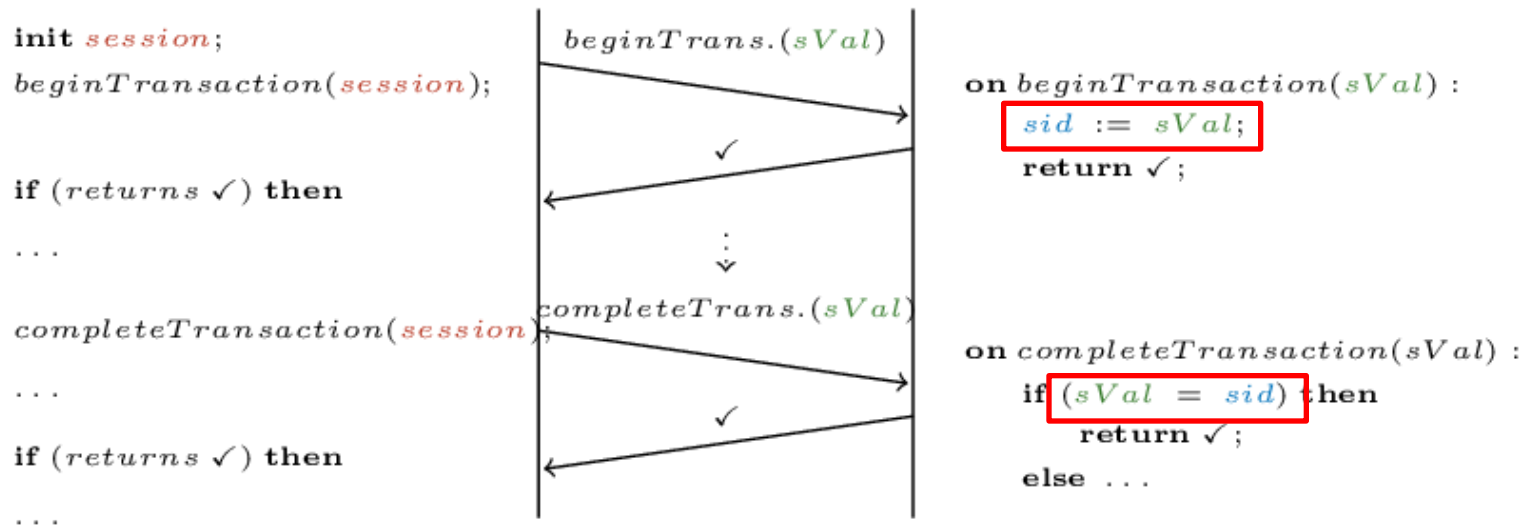
Connect Scenario



Data-Dependent Control

Value-independent Data Dependencies

- automated synthesis of connectors between networked systems
- inference of **executable** “interface programs”



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, ..

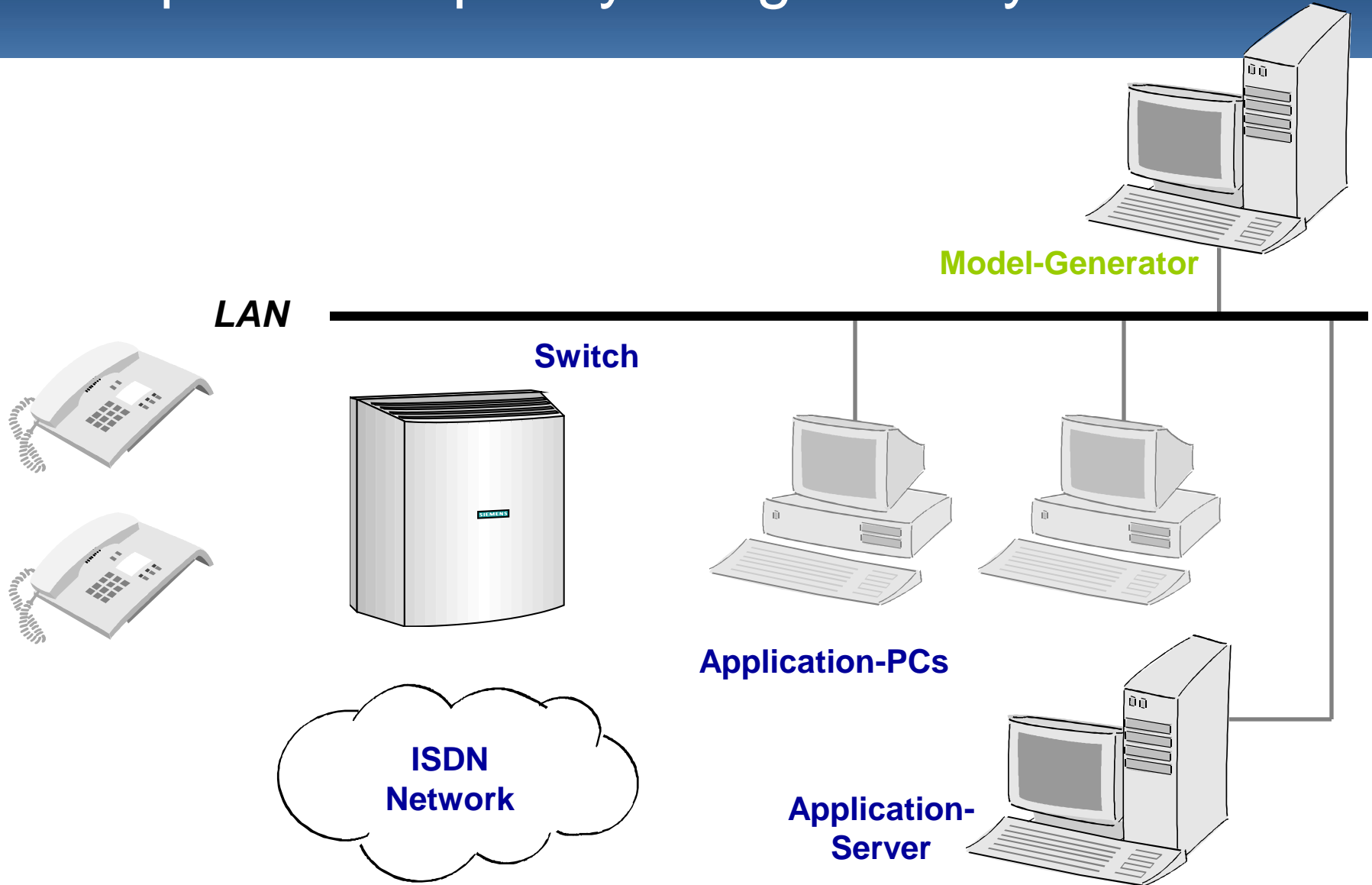
(External) Mapper-Based Data Treatment

Explicit Data Modelling

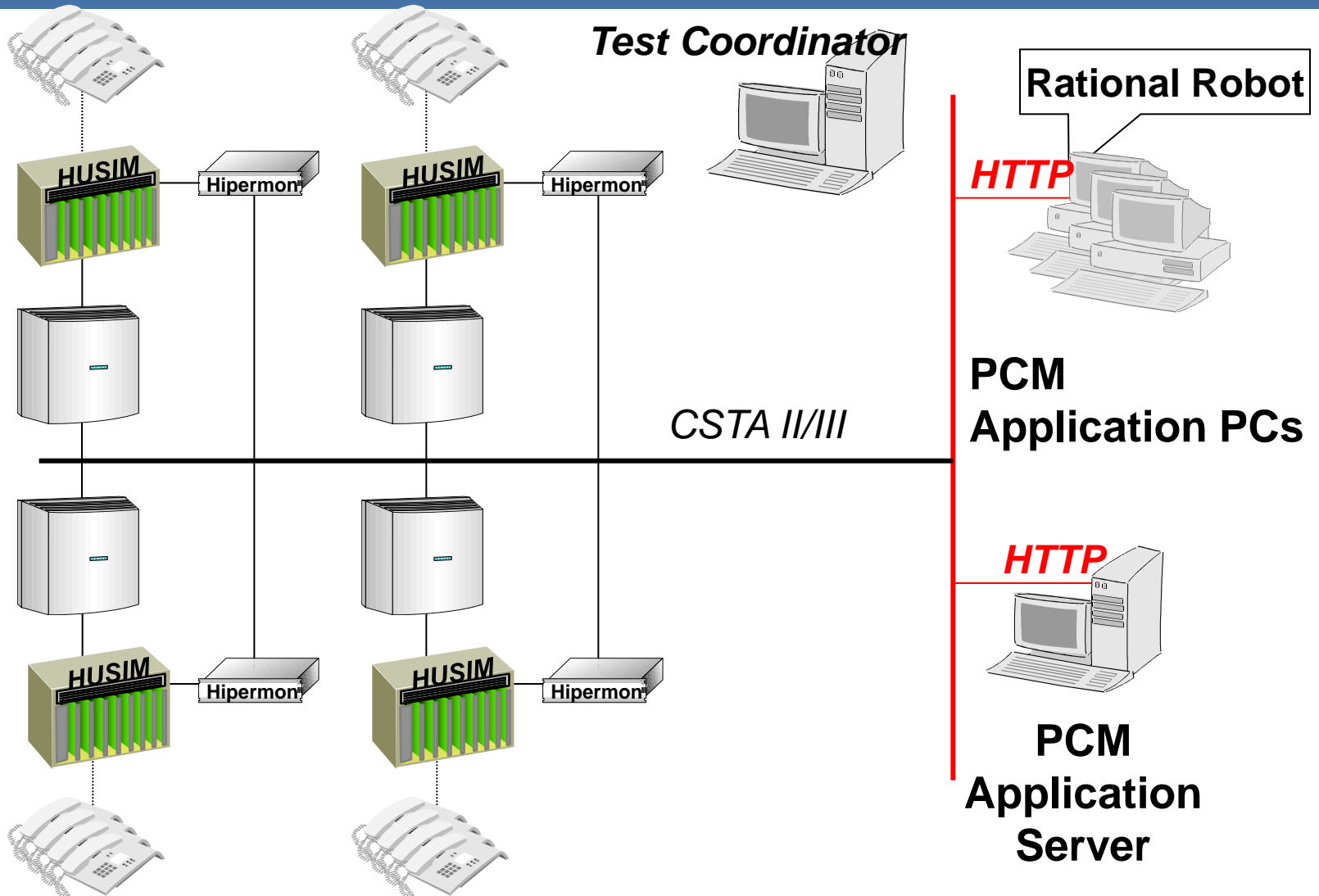
Outline

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

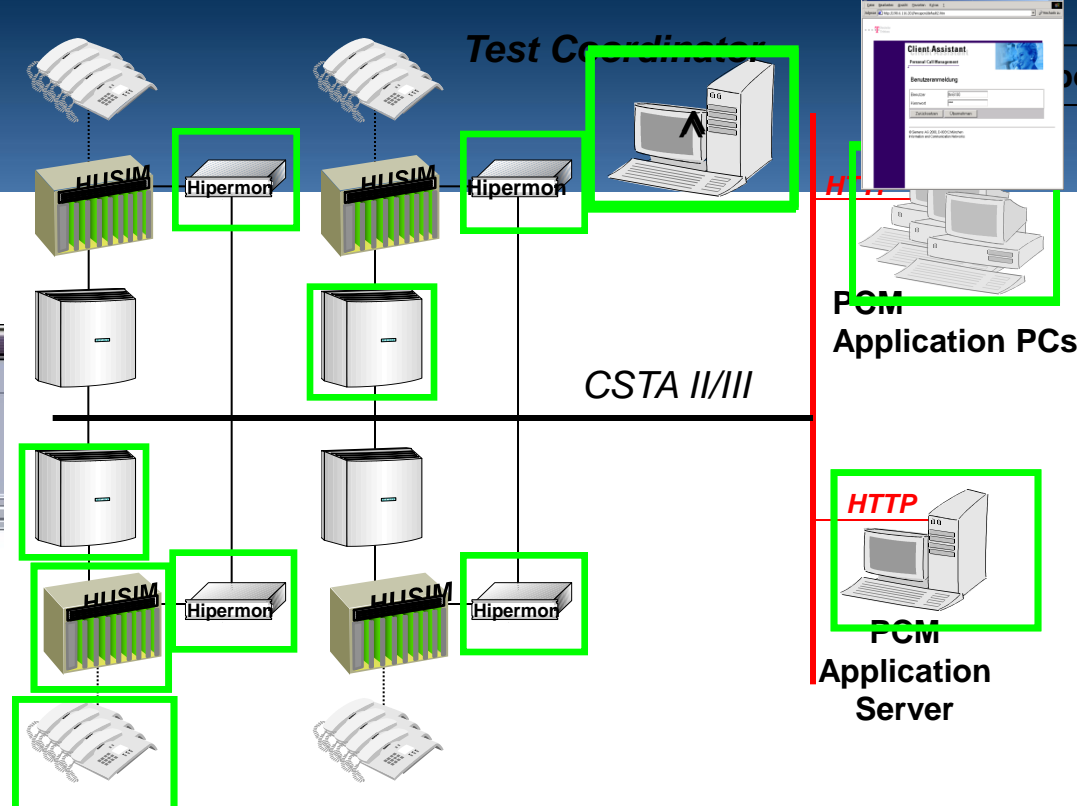
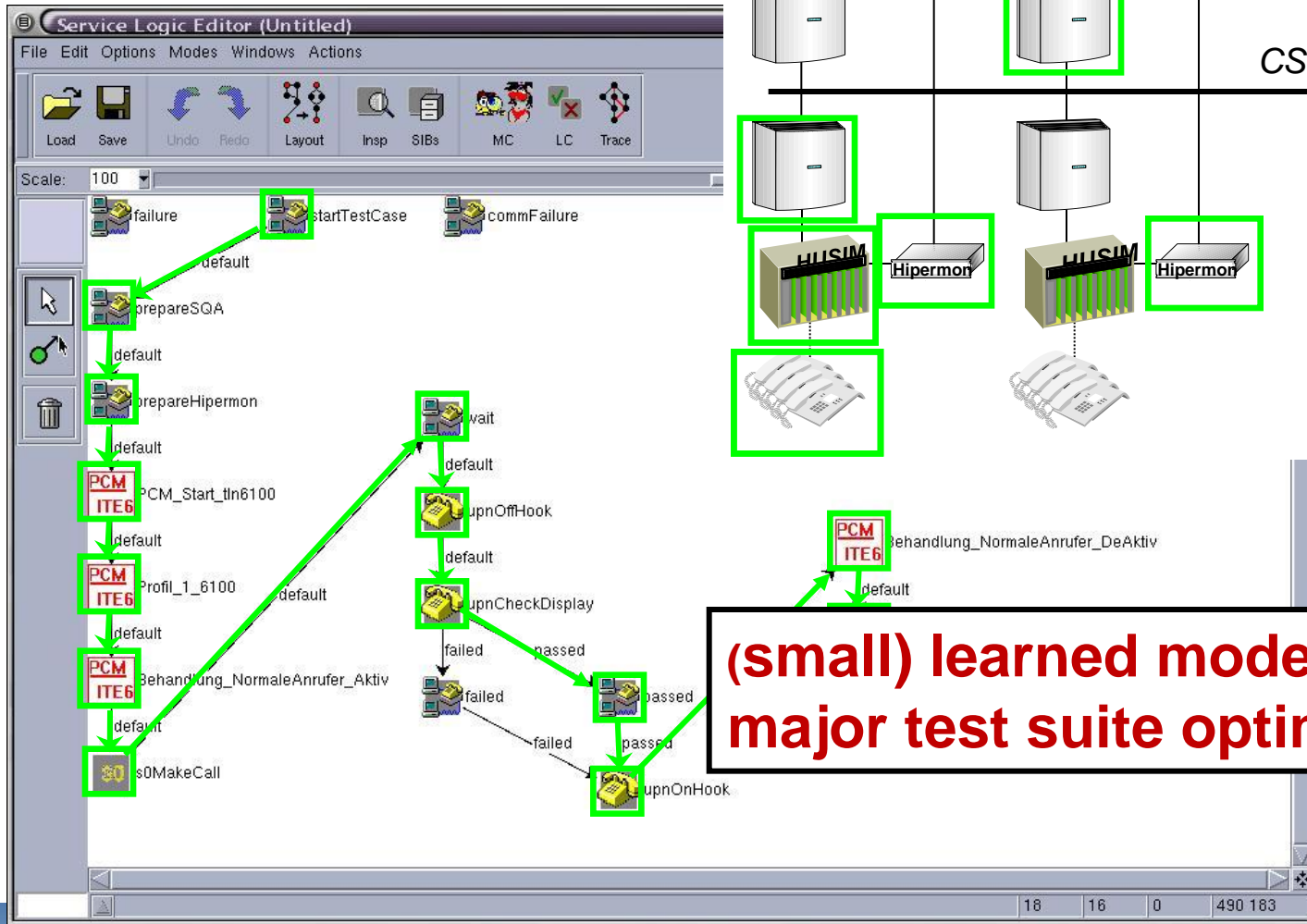
Computer/Telephony Integrated Systems



The Concrete Scenario



Means of Observation



(small) 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

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

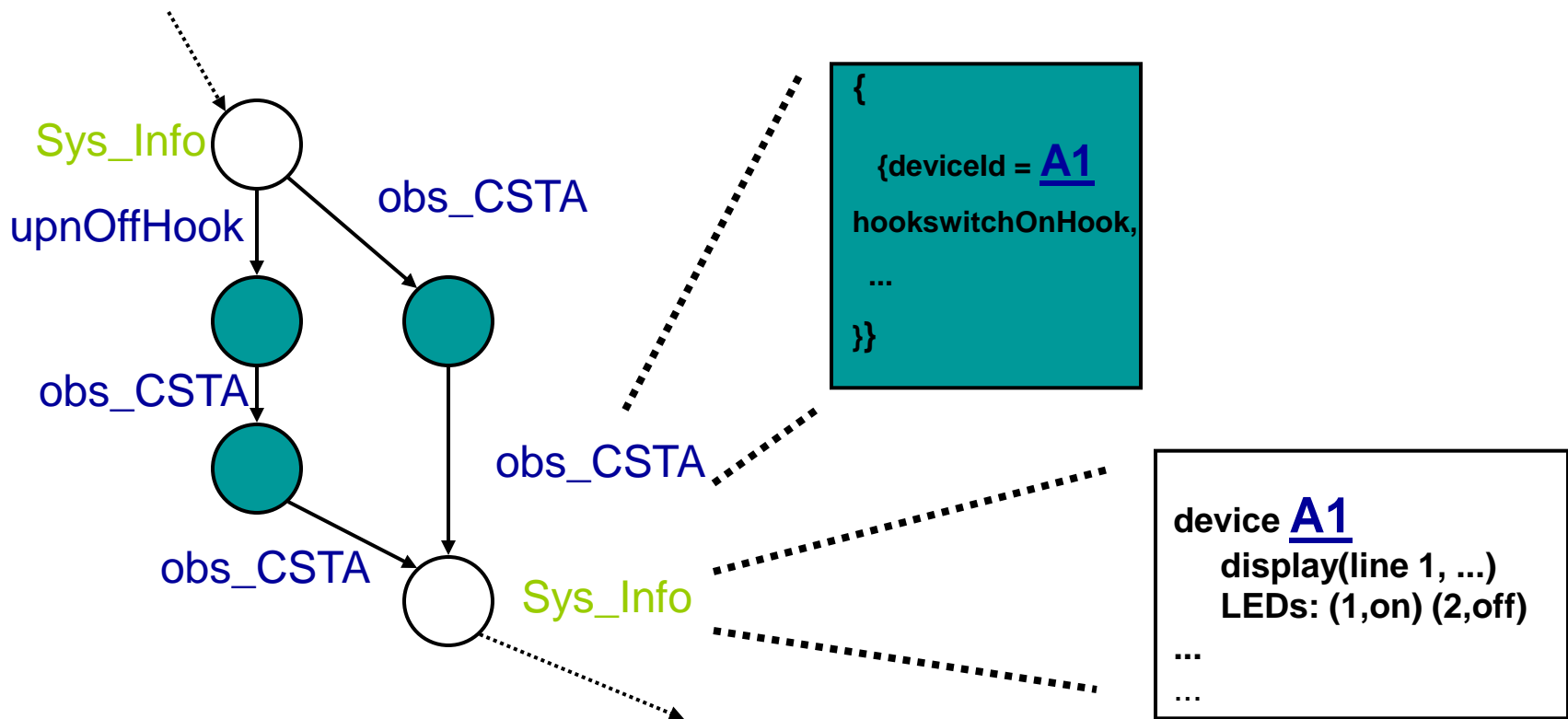
symbolic names
no time stamps etc.



```
{obsEvent  
  deviceId = A1  
  switchOnHook,  
  ...  
}
```

Sketch of the Model Structure

Models comprise state changes as well as UPN- and CSTA-Observations.



Active Automata Learning

Distinguishing Futures

OT
Reaching
Words

Lower Hypothesis
Automaton

Unknown System

Transitions

Closeness & Consistency
Validation

Membership Queries

OT	ε
ε	1
a	1
b	0

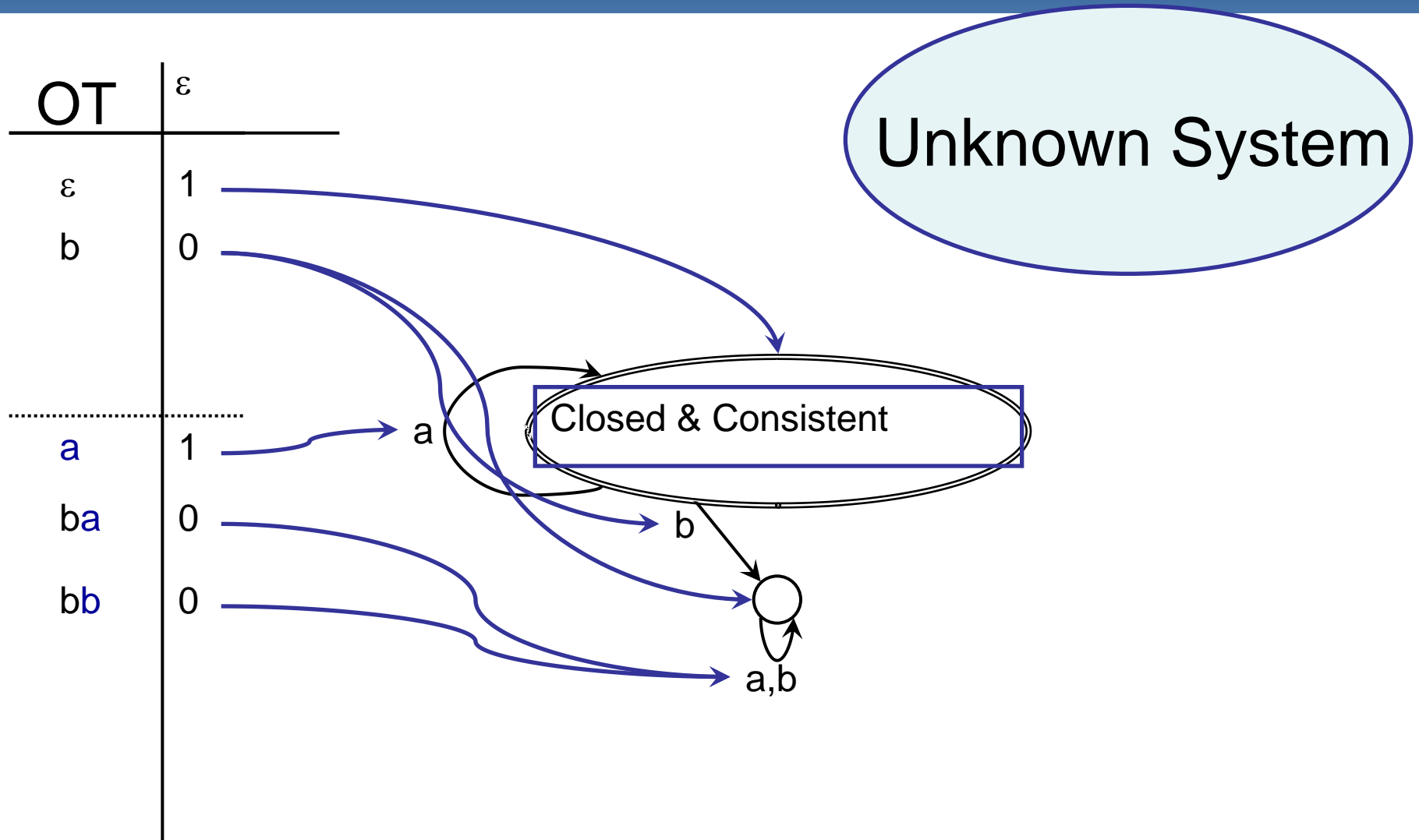
Abstract States

Unknown System

Not closed!

Transition Relation

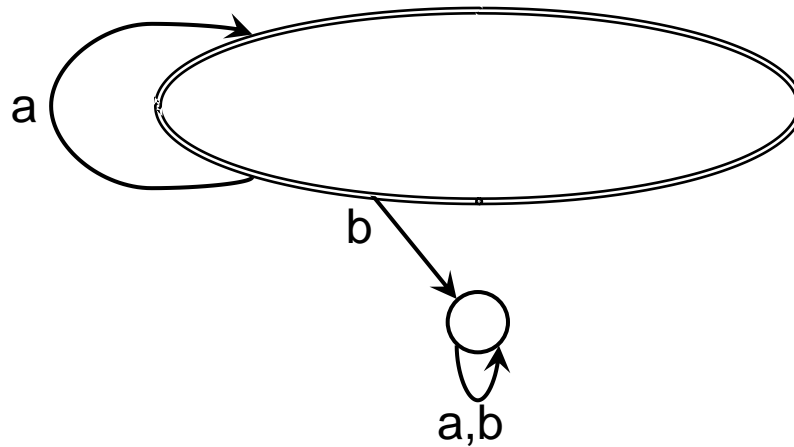
Closure & Consistency



Equivalence Queries

OT	ε
ε	1
b	0
a	1
ab	1
<hr/>	
a	1
ba	0
bb	0

Unknown System

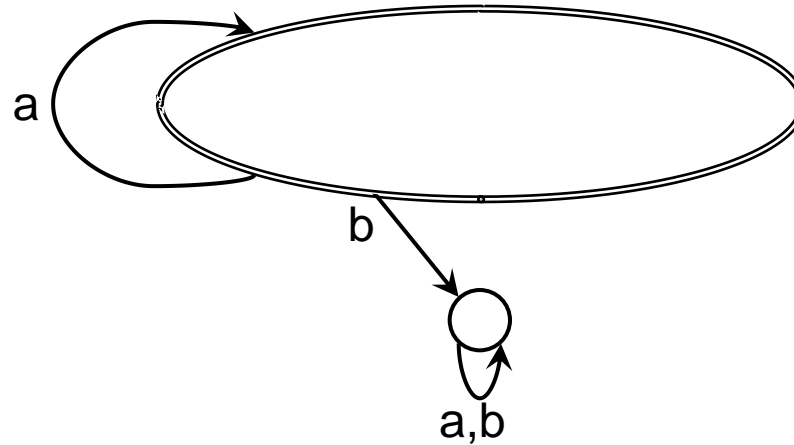


Counterexample: $ab \in L$



Counter Example-Based Extension

Unknown System



Counterexample: $ab \in L$

OT	ε
ε	1
b	0
a	1
ab	1
<hr/>	
ba	0
bb	0
aa	0
aba	0
abb	1

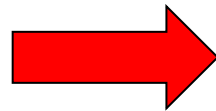
Closure & Consistency

OT	ε
ε	1
b	0
a	1
ab	1
<hr/>	
ba	0
bb	0
aa	0
aba	0
abb	1

Unknown System

Not consistent:

row (ε) = row (a), but row (εa) \neq row (aa)

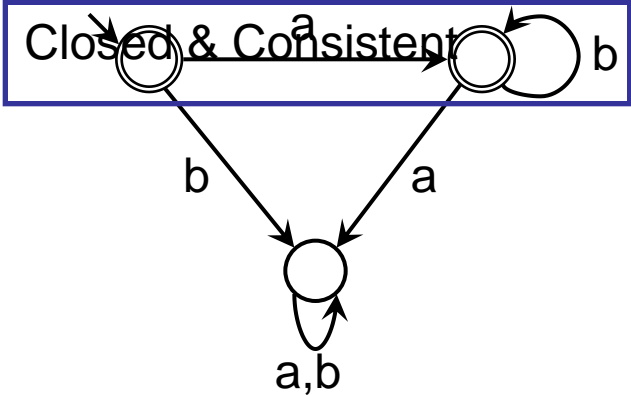


New Column: **a**

Next Iteration

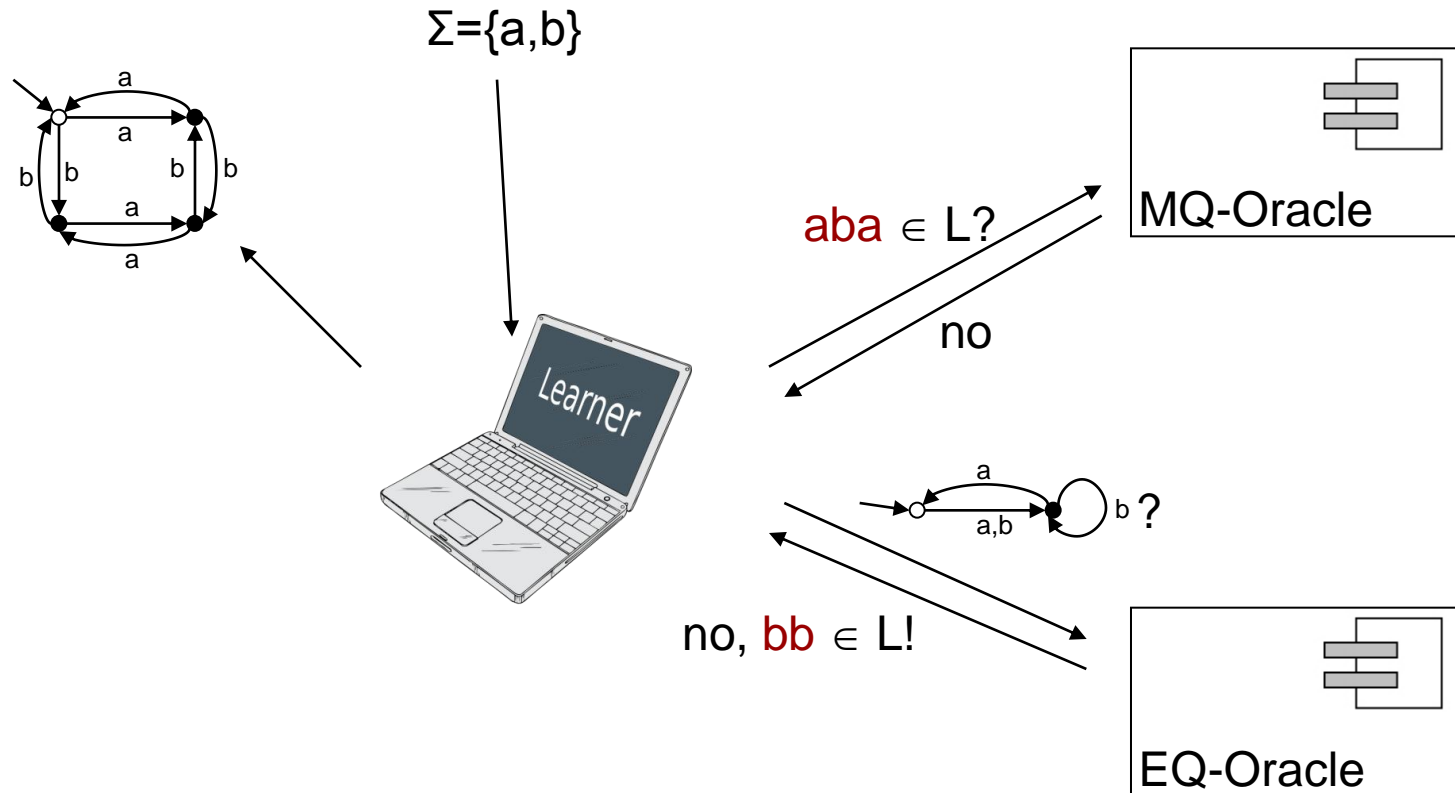
OT	ε	a
ε	1	1
b	0	0
a	1	0
ab	1	0
<hr/>		
ba	0	0
bb	0	0
aa	0	0
aba	0	0
abb	1	0

Unknown System



Finished!

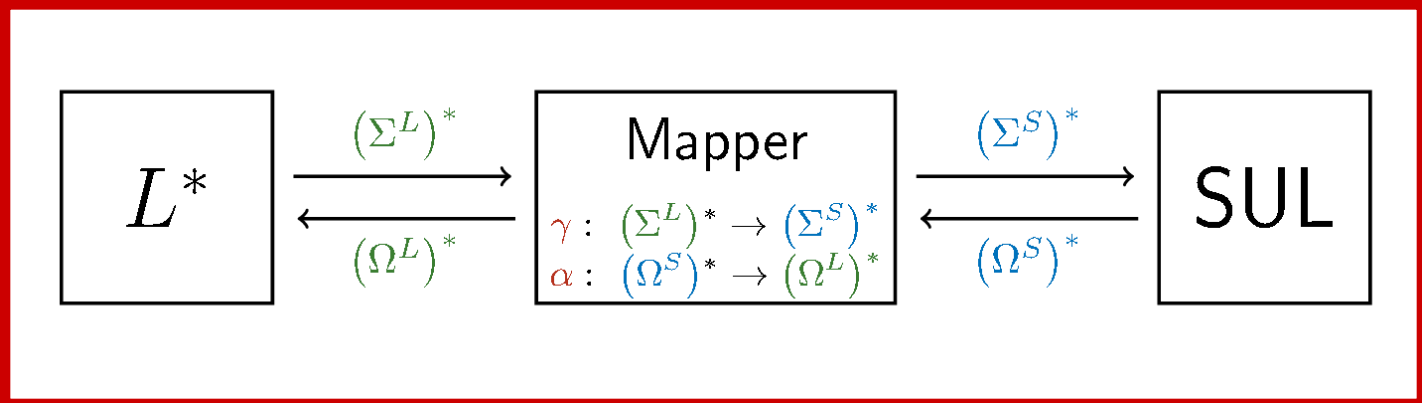
Active automata learning: L^*



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.



- H
- A
- and for Interface Automata [Aarts et al. 2010]
- Efficient Tool: LearnLib [TUDortmund]

Analysis of Counterexamples I

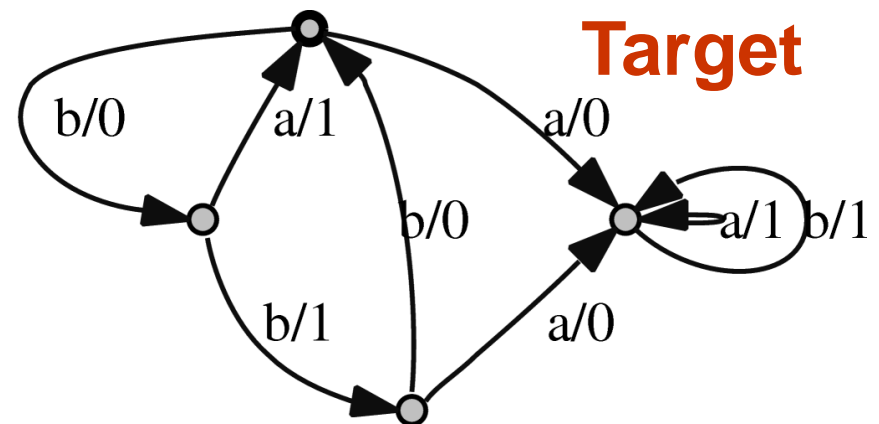
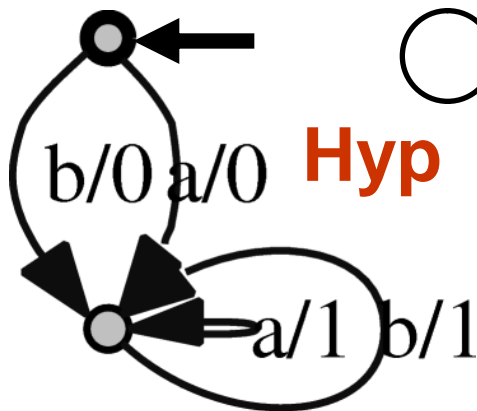
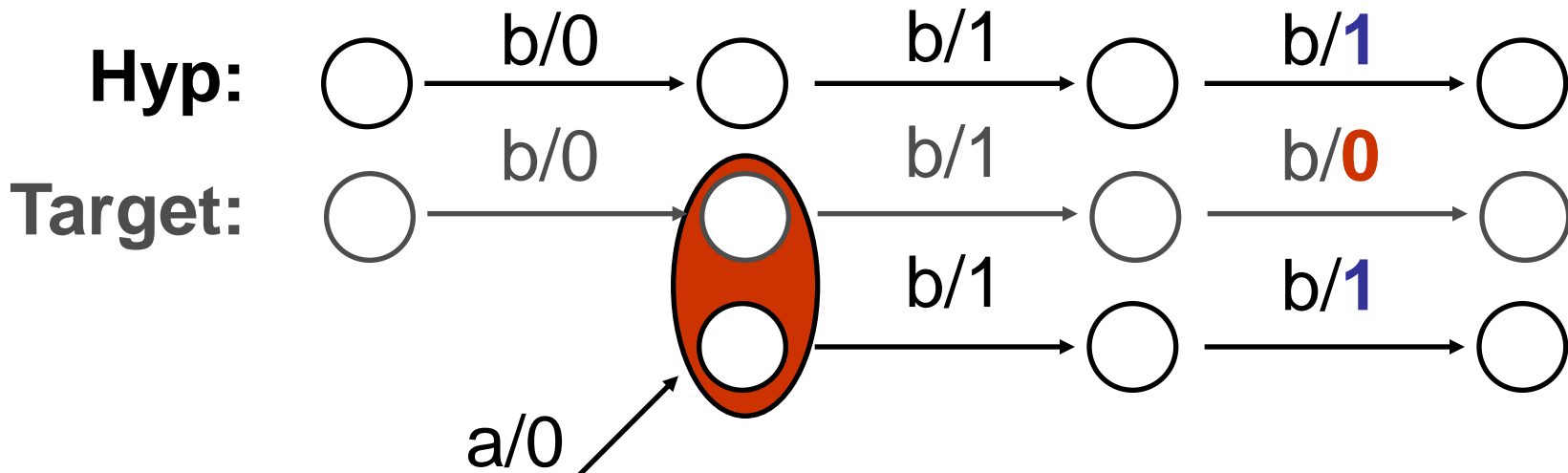
Rivest Shapire

one essential suffix

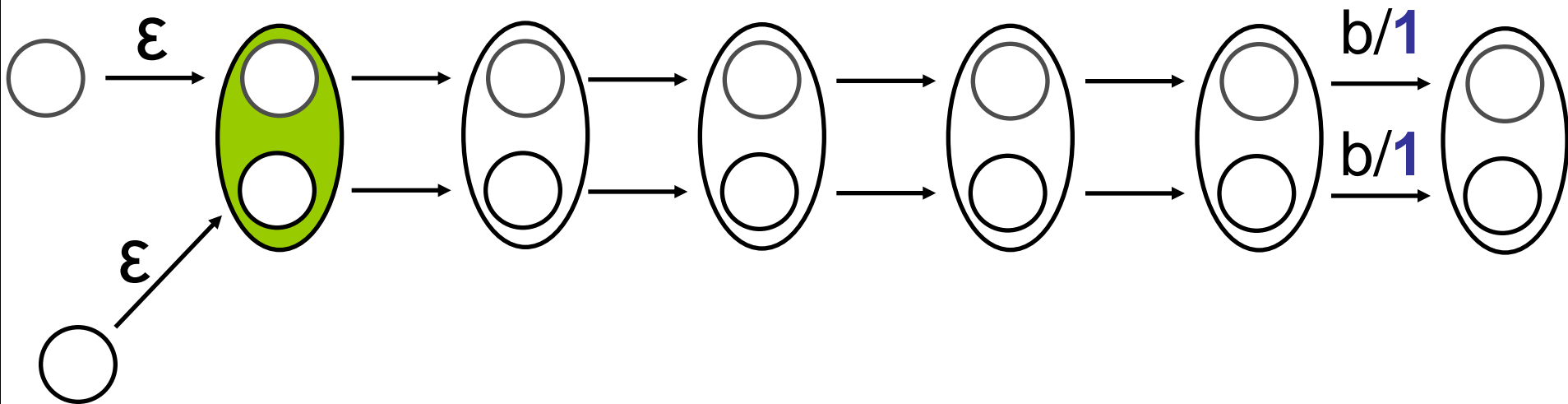
All prefixes of
counterexample
...

	a	b	bb
ϵ	0	0	
a	1	1	
b	1	1	
bb	0	0	
bbb	0	0	
aa	1	1	
ab	1	1	
ba	0	0	
...	

Analysis of Counterexamples I

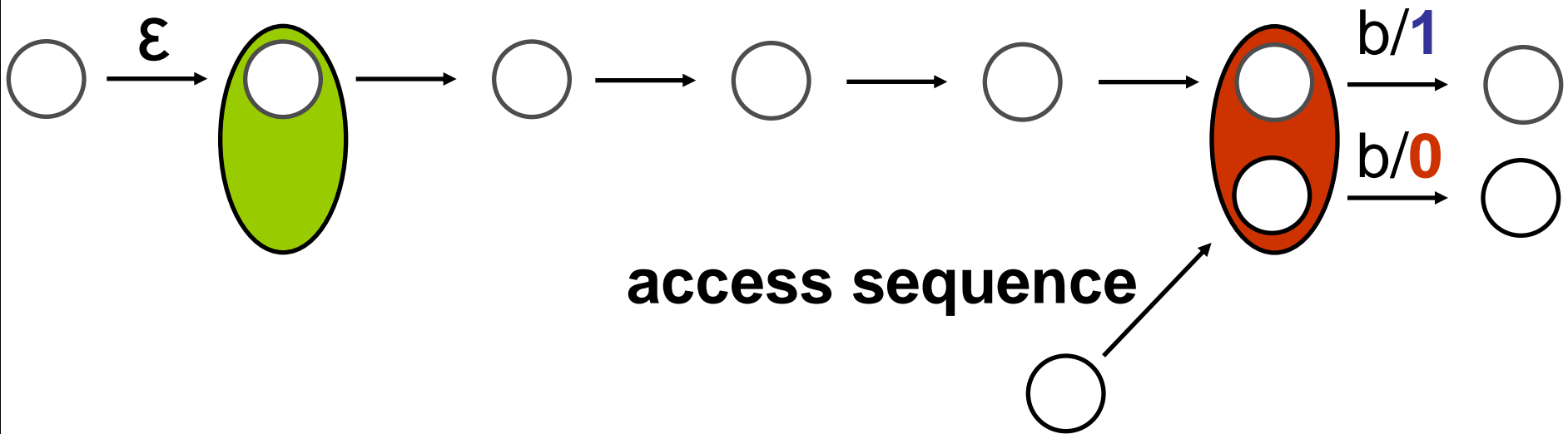


Analysis of Counterexamples I



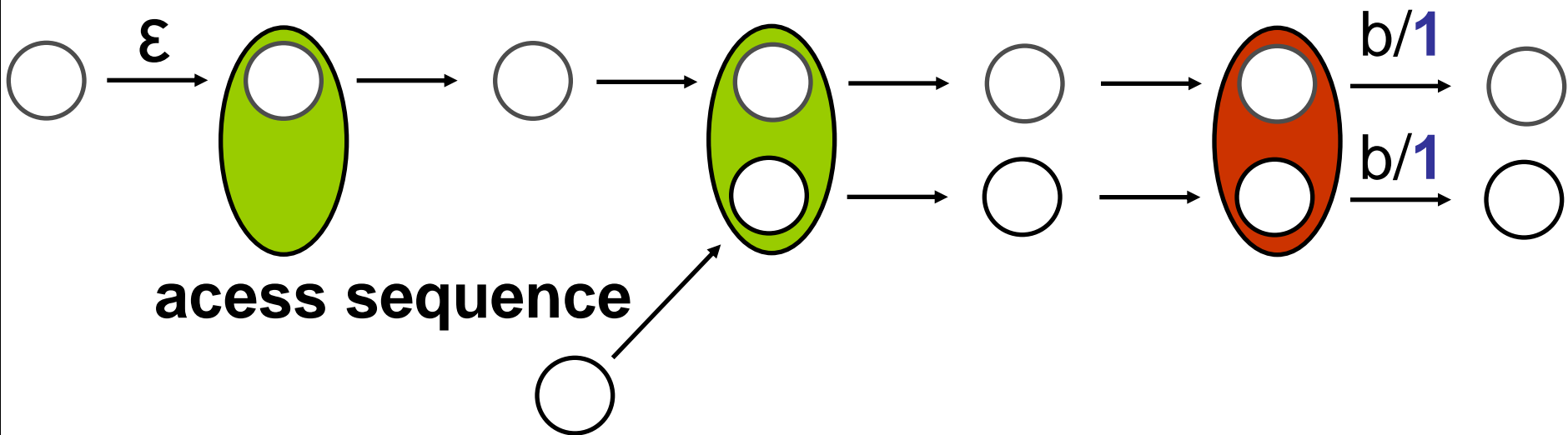
- Binary search on counterexample
- Requires $O(\log_2(m))$ membership queries

Analysis of Counterexamples I



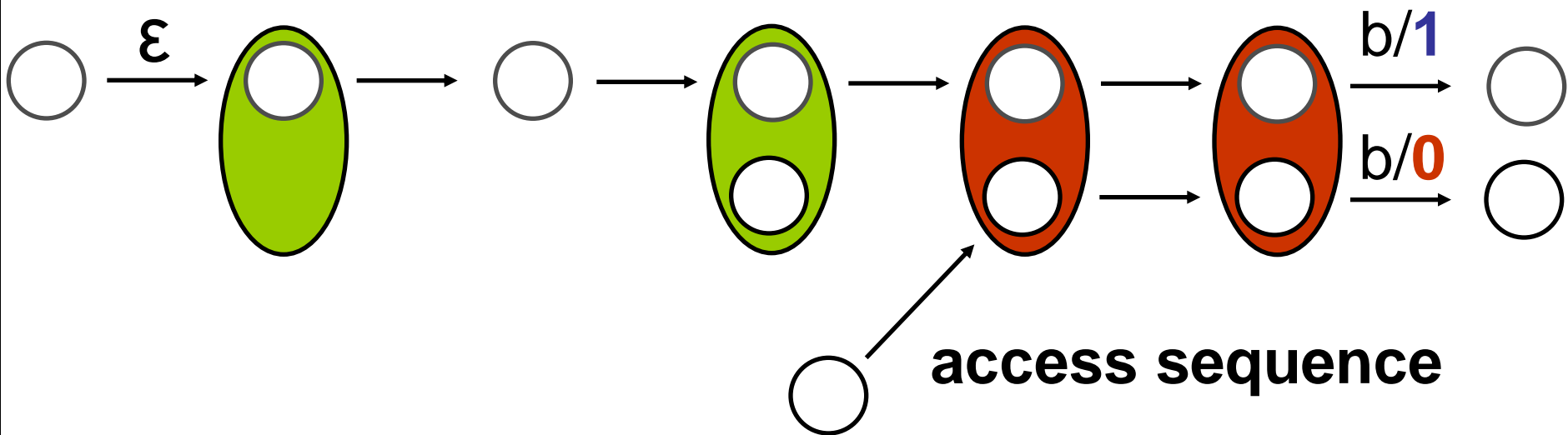
- Binary search on counterexample
- Requires $O(\log_2(m))$ membership queries

Analysis of Counterexamples I



- Binary search on counterexample
- Requires $O(\log_2(m))$ membership queries

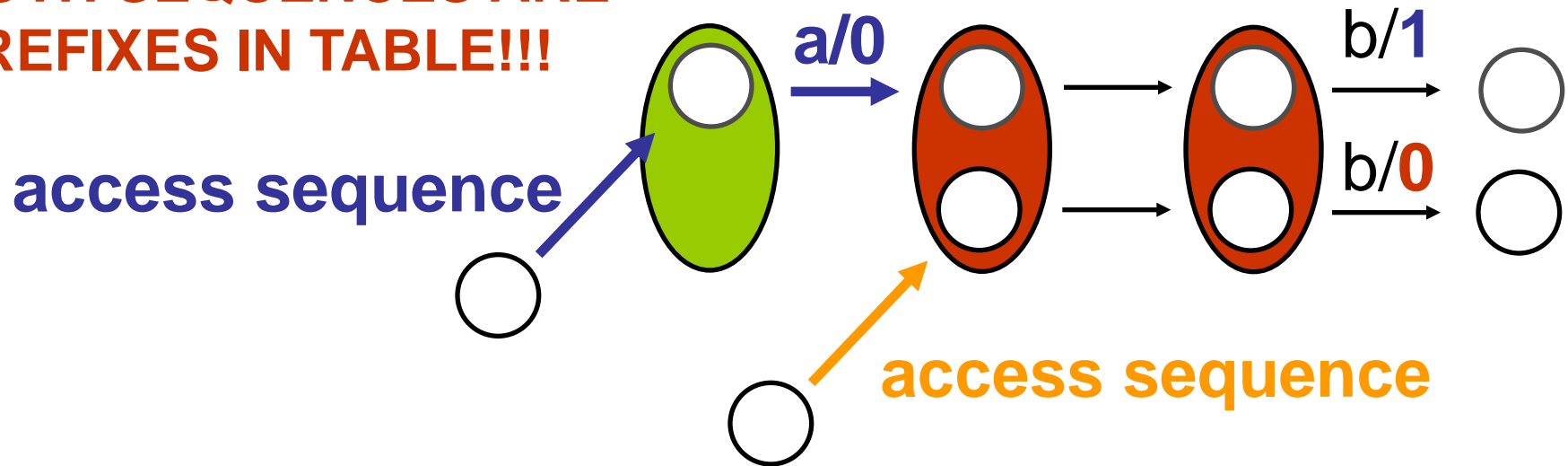
Analysis of Counterexamples I



- 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

Effect: Reduced Observation Table

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:

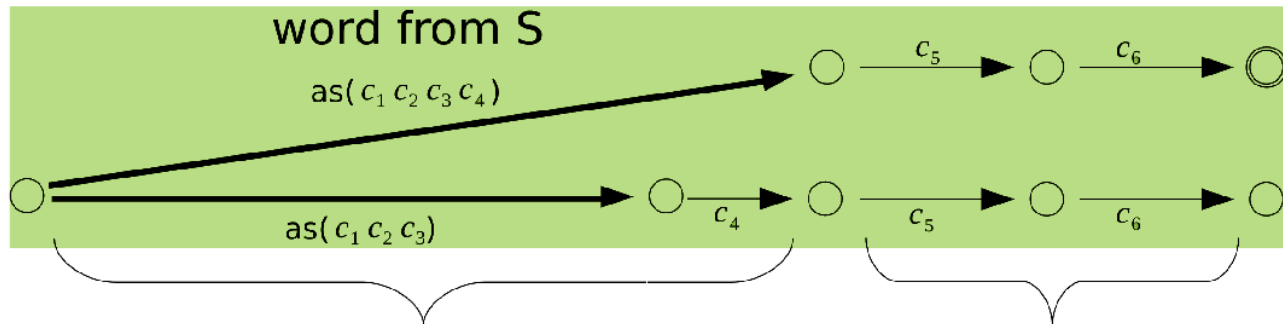
- Guaranteed Consistency!
- Improved worst case complexity

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

Analysis of Counterexamples I

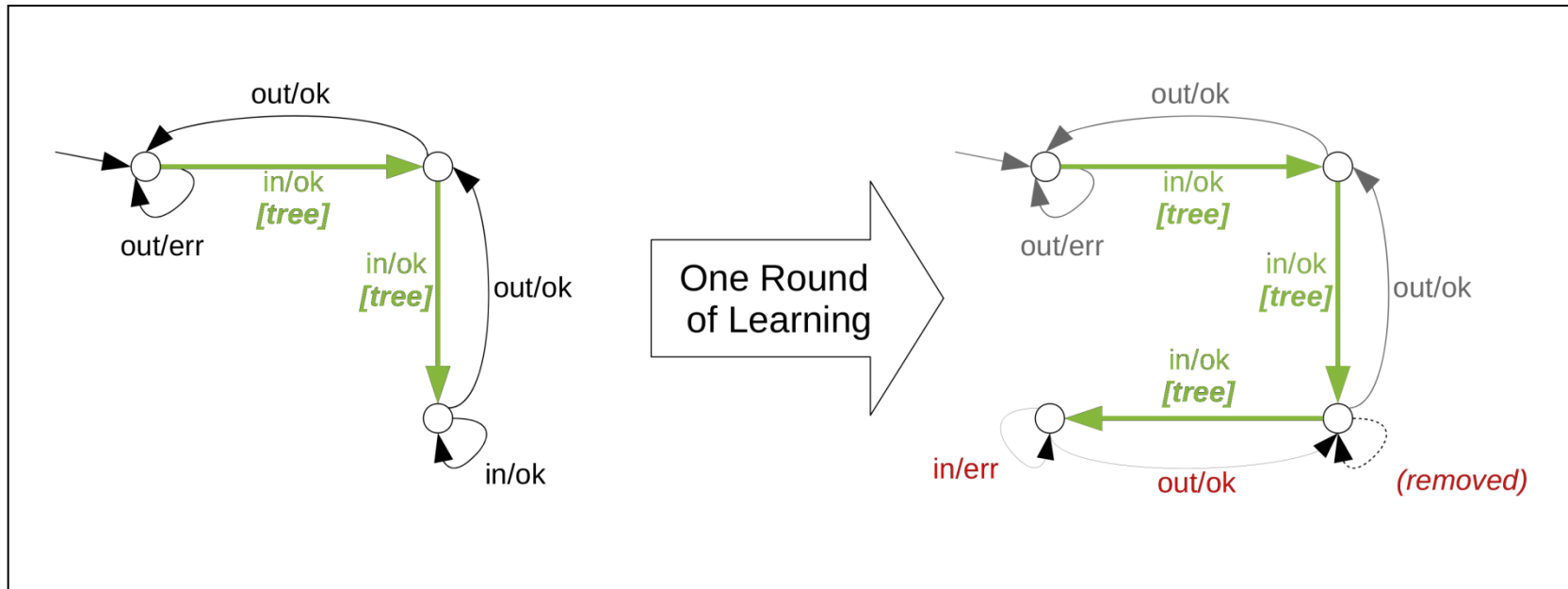
one essential suffix

	a	b	bb
ϵ	0	0	



...	
-----	-----	-----	--

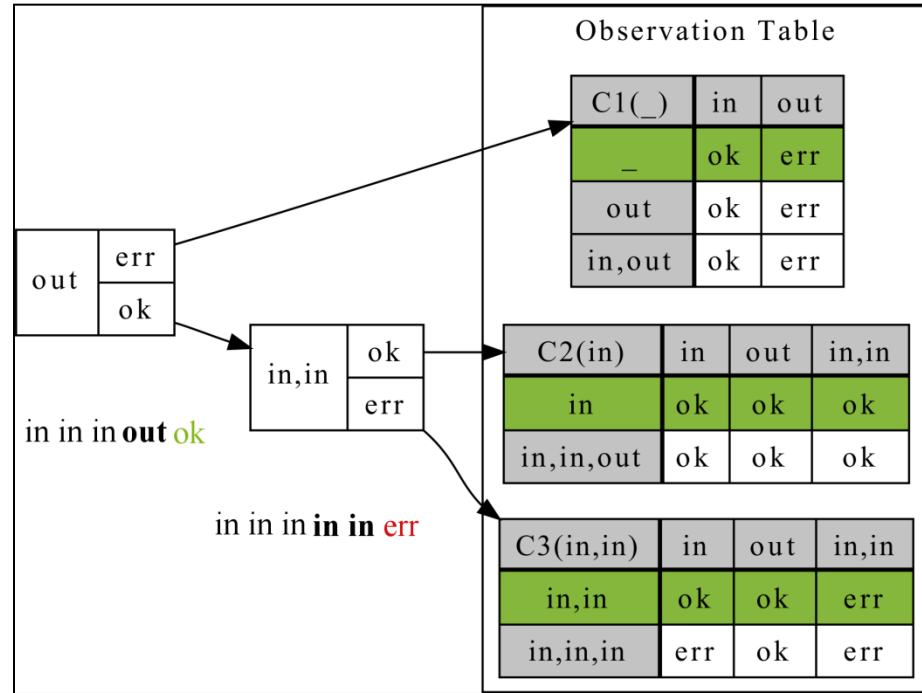
Efficient equivalence test heuristics



- 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



Important for Zulu,
even more important for CONNECT

Detailed results by example

Algorithm	New Membership Queries			Rounds	States	Score
	Close	Analyze	Search			
E.H.Blocking	6,744	358	999	259	352	94.11
F.H.Weighted	6,717	340	1,035	262	351	94.61
F				228	332	93.28
r				5	312	74.89
r				6	319	73.06
r				6	319	74.39

ZULU Problem 49763507

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

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

finite capacity



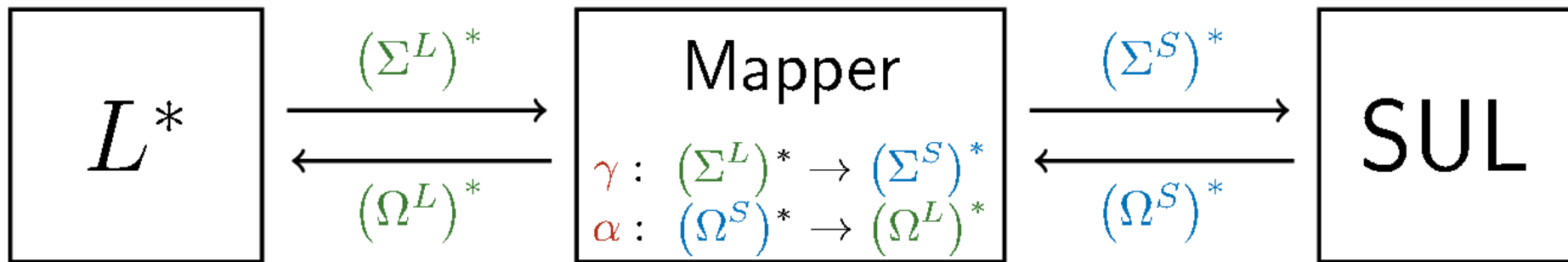
```
public boolean push(Object o) {  
    if (size == capacity) return false;  
    elements[size++] = o; return true;  
}
```

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

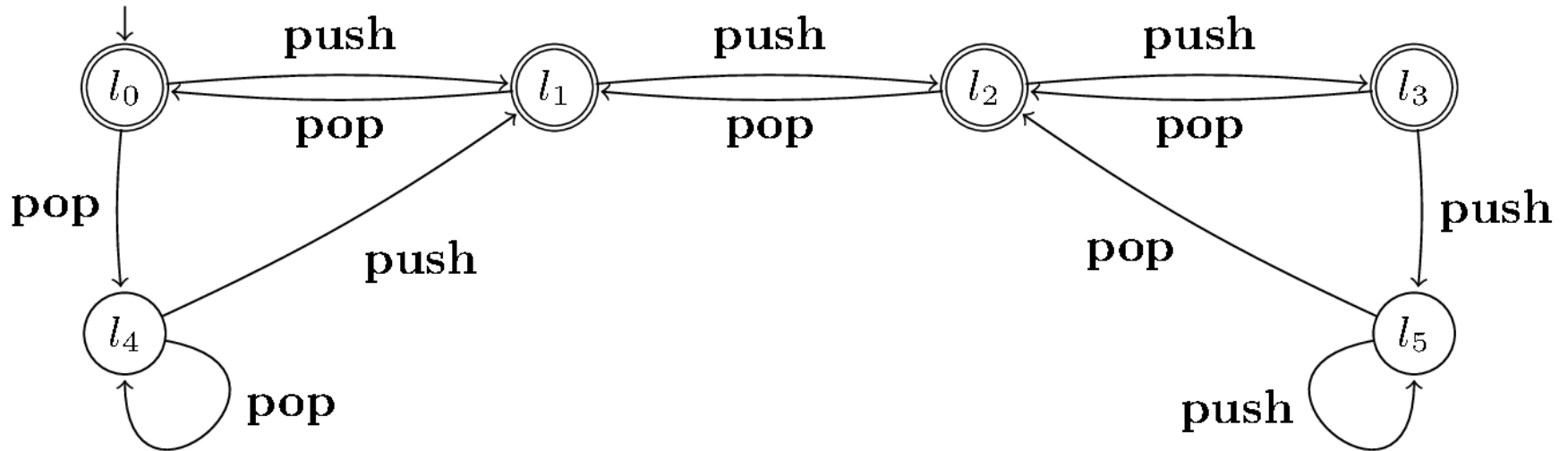
```
}
```



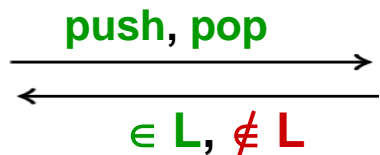
Mappers



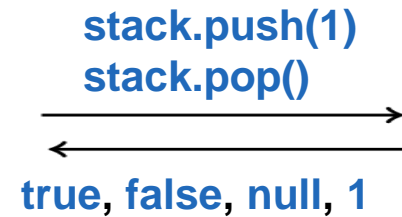
Learning the stack as a language



L^*

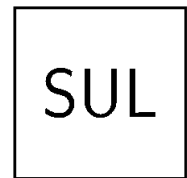
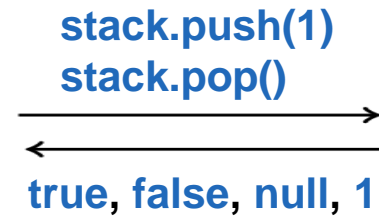
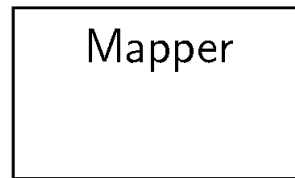
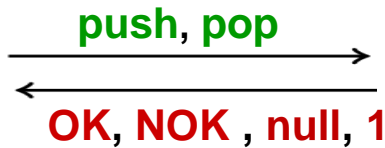
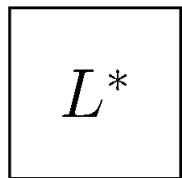
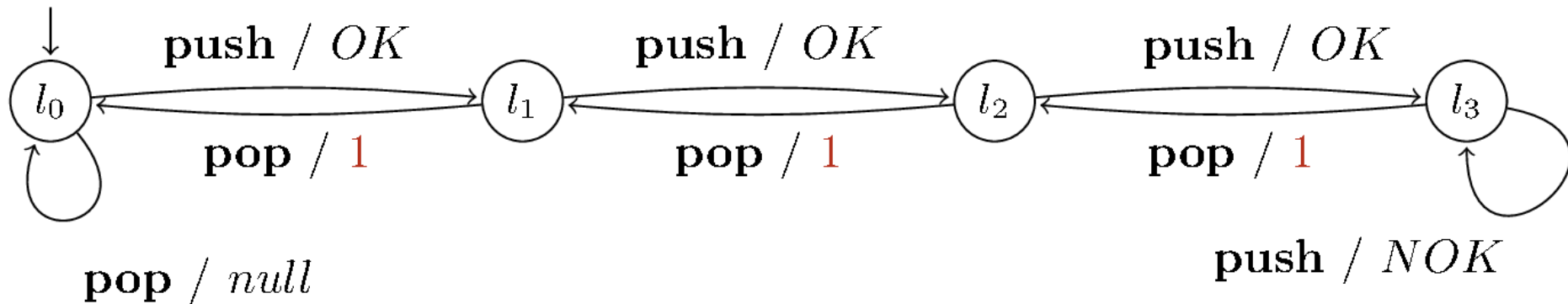


Mapper

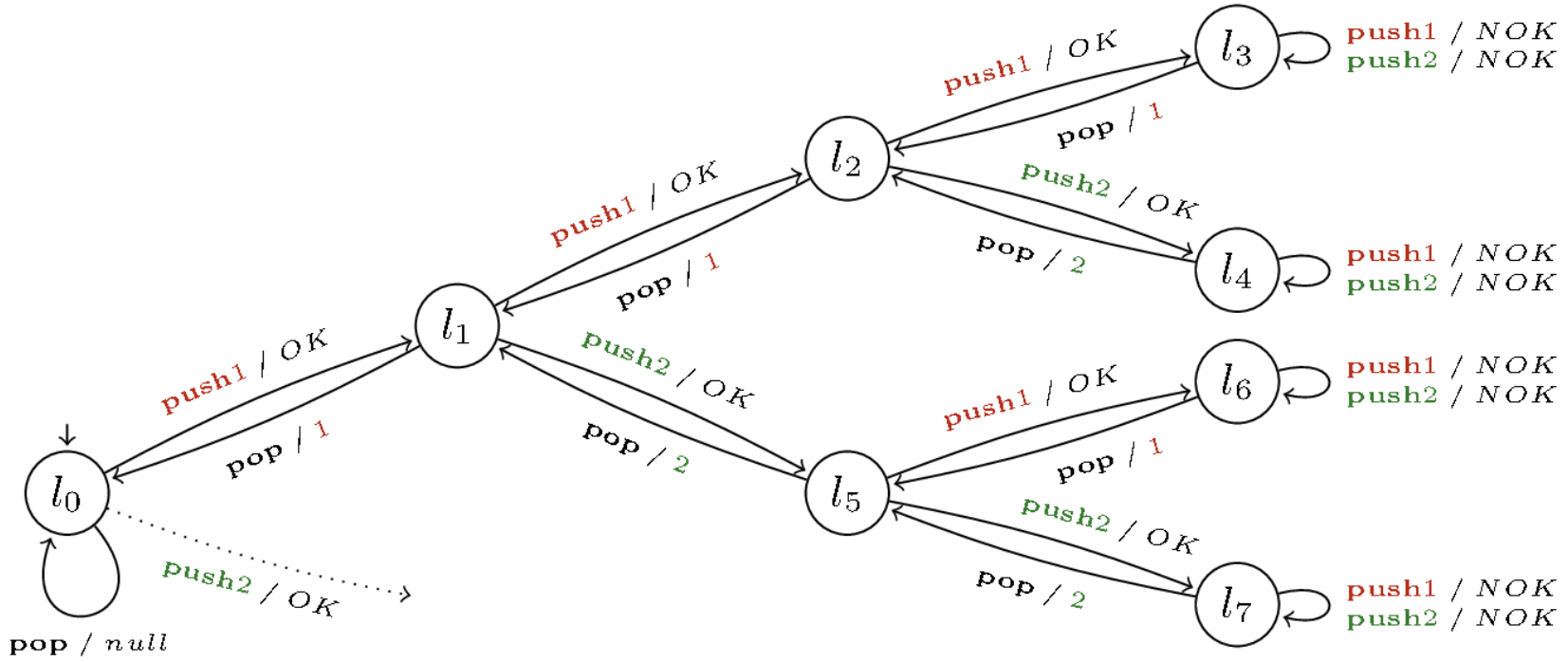


SUL

Introducing outputs: Mealy machines



Introducing outputs: Mealy machines



L^*

$\xrightarrow{\text{push1, push2, pop}}$
 $\xleftarrow{\text{OK, NOK, null, 1, 2}}$

Mapper

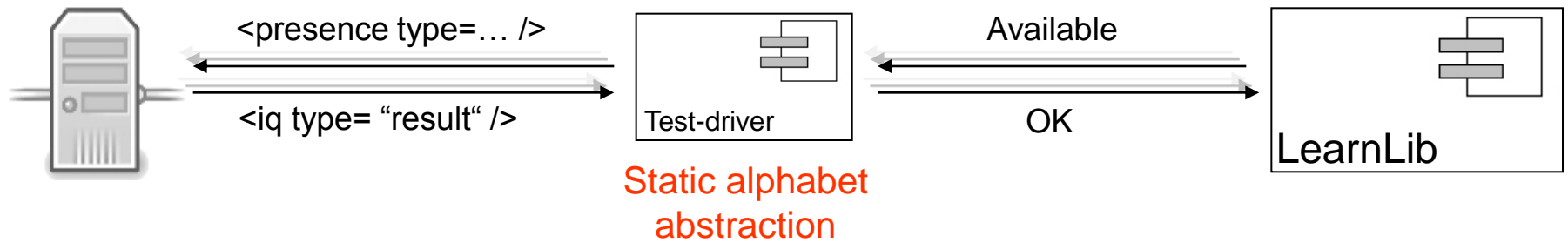
$\xrightarrow{\text{stack.push(1), Stack.push(2), stack.pop()}}$
 $\xleftarrow{\text{true, false, null, 1, 2}}$

SUL

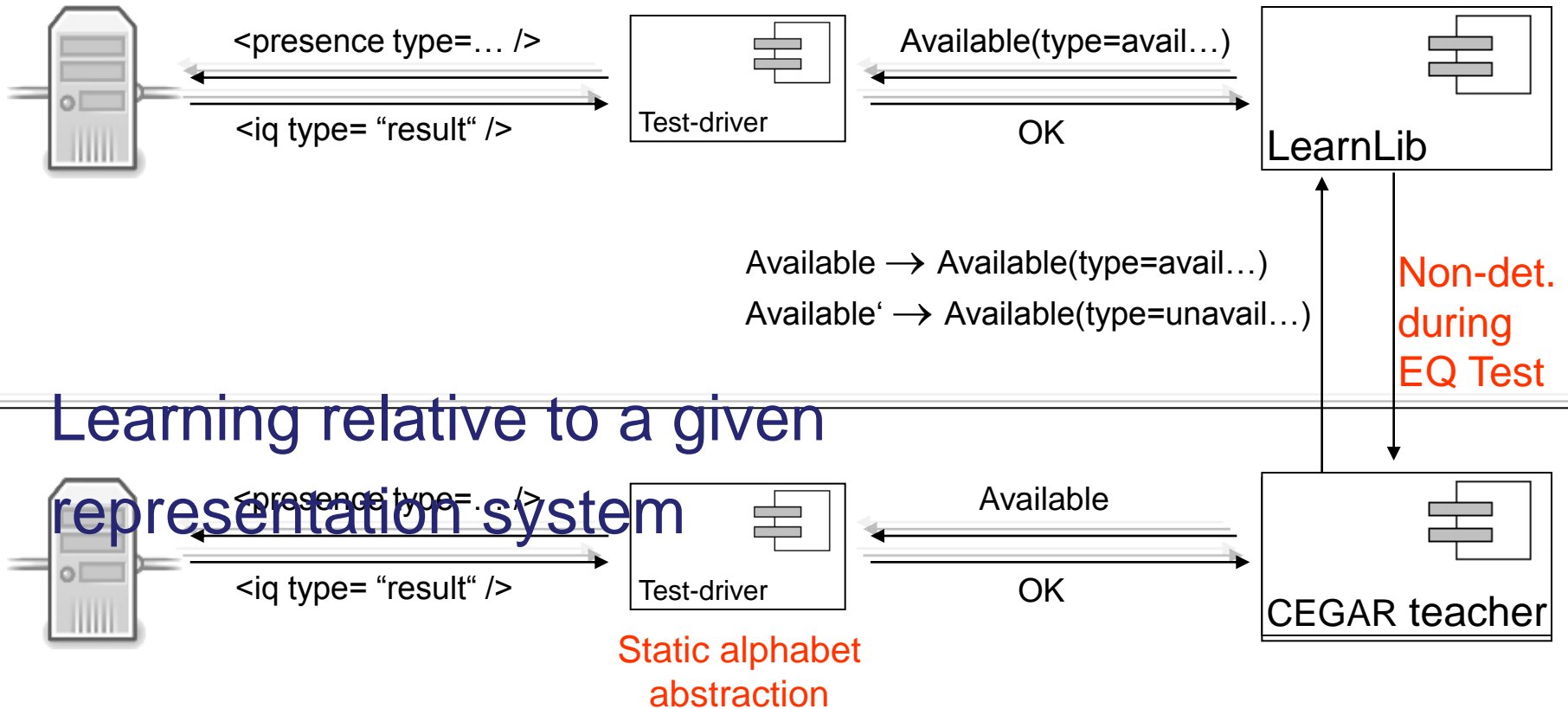
Outline

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

Automated Alphabet Abstraction Refinement

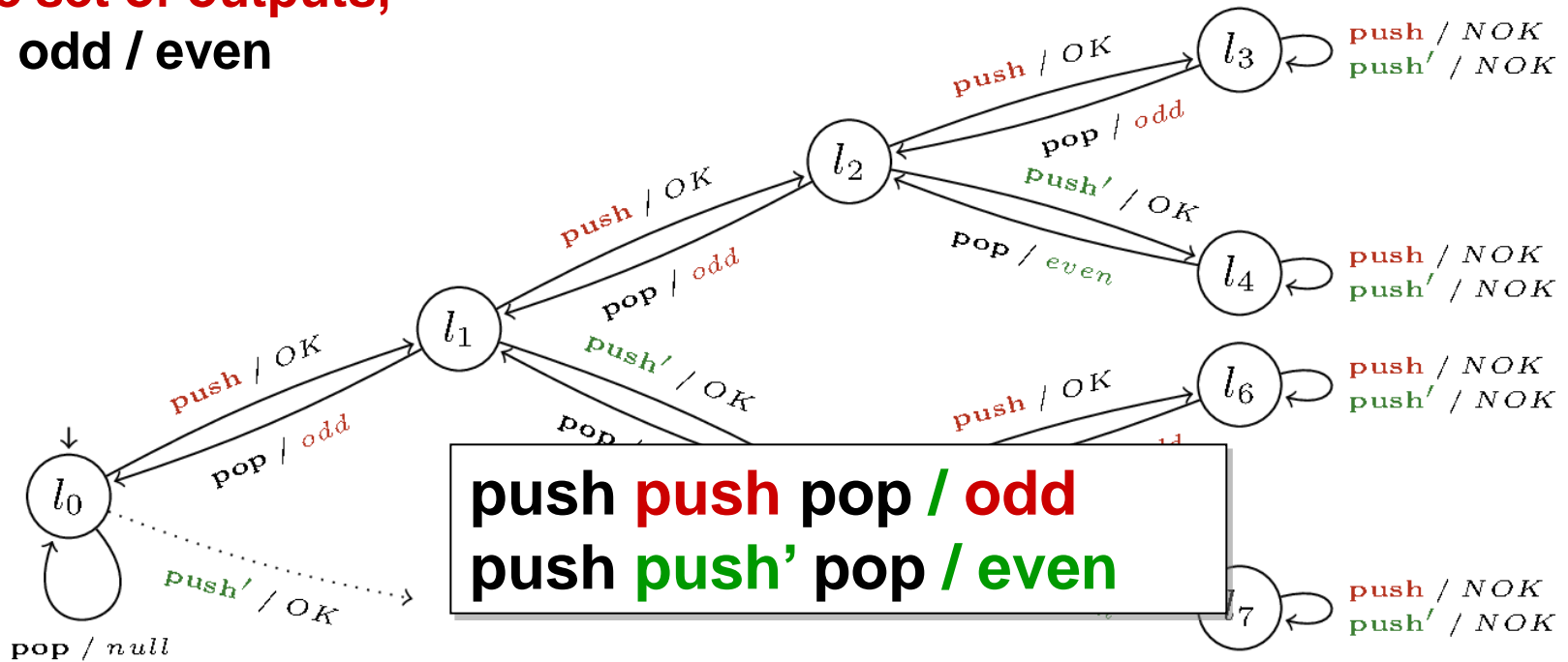


Automated Alphabet Abstraction Refinement



The Mod-k Stack

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



L^*

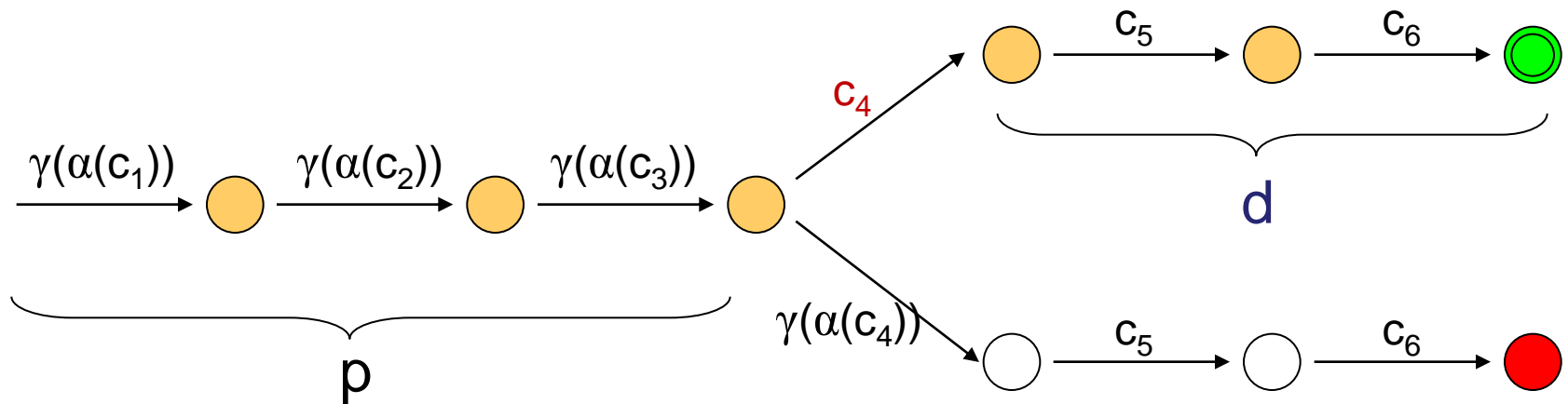
$\xrightarrow{\text{push, push', pop}}$
 $\xleftarrow{\text{OK, NOK, null, odd, even}}$

Mapper

$\xrightarrow{\text{stack.push(51); stack.push(2012); stack.pop()}}$
 $\xleftarrow{\text{true, false, null, 51, 2012}}$

SUL

Counter Examples and Witnesses



Separating pattern

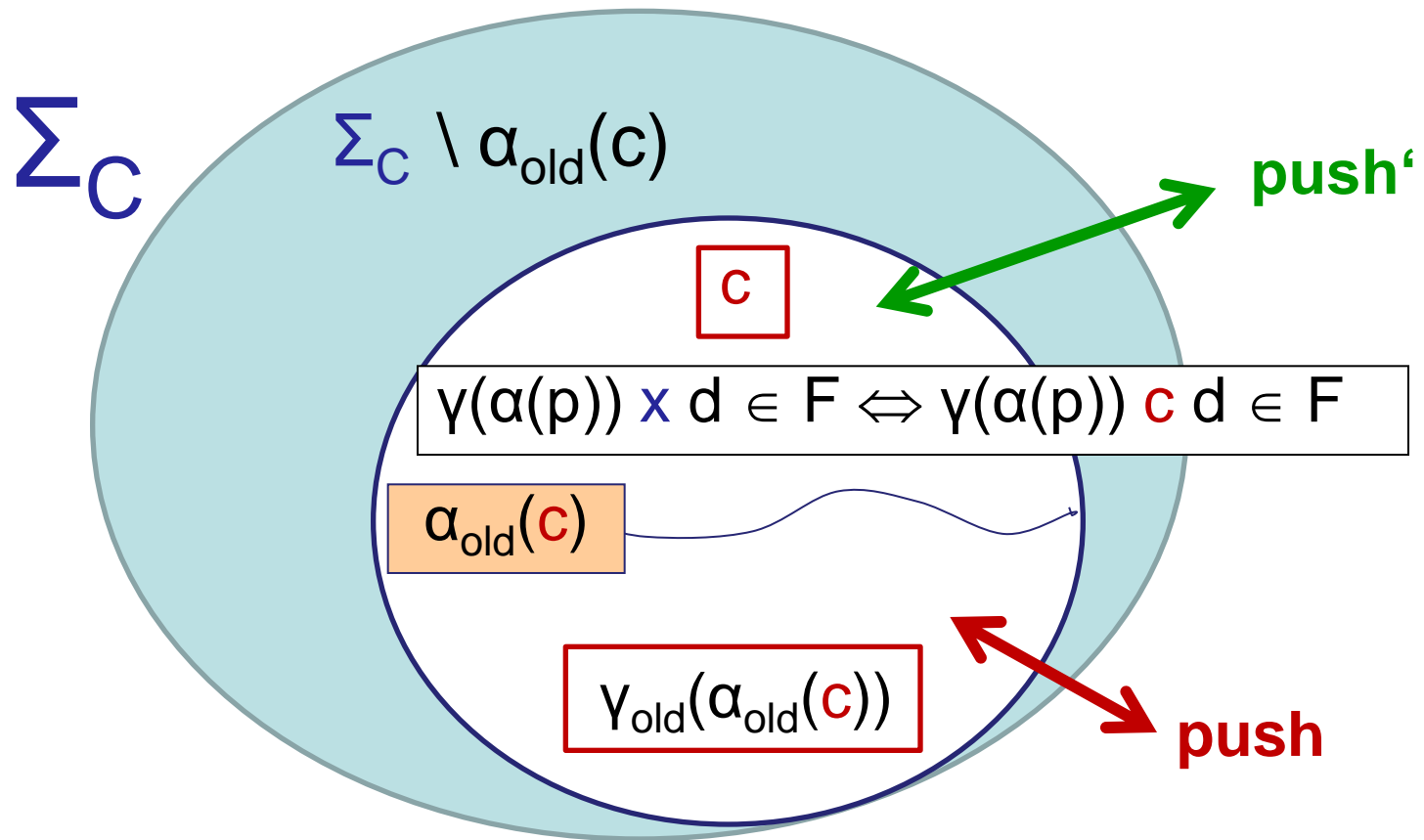
p
state

c_4
representation

d
future



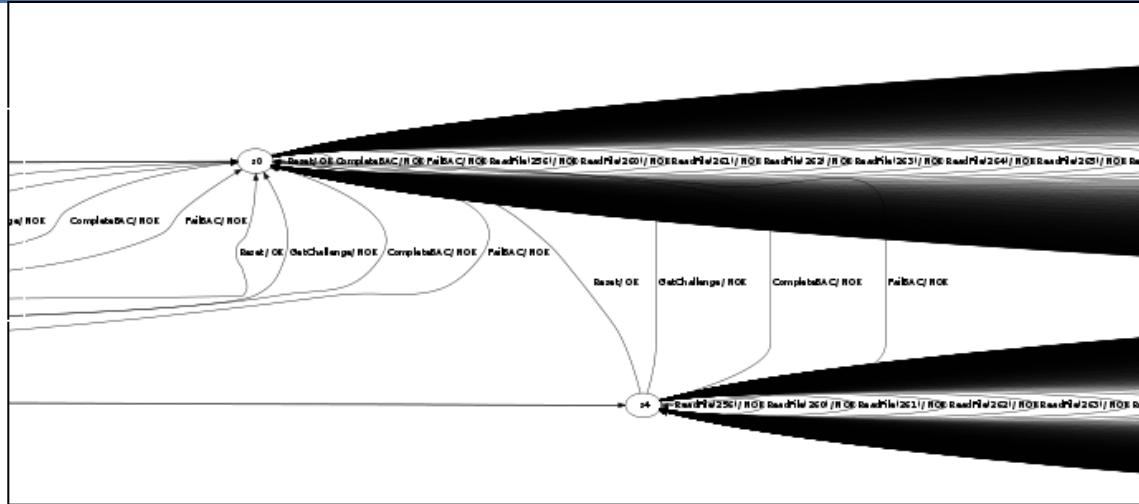
Alphabet Abstraction Refinement



Case Study

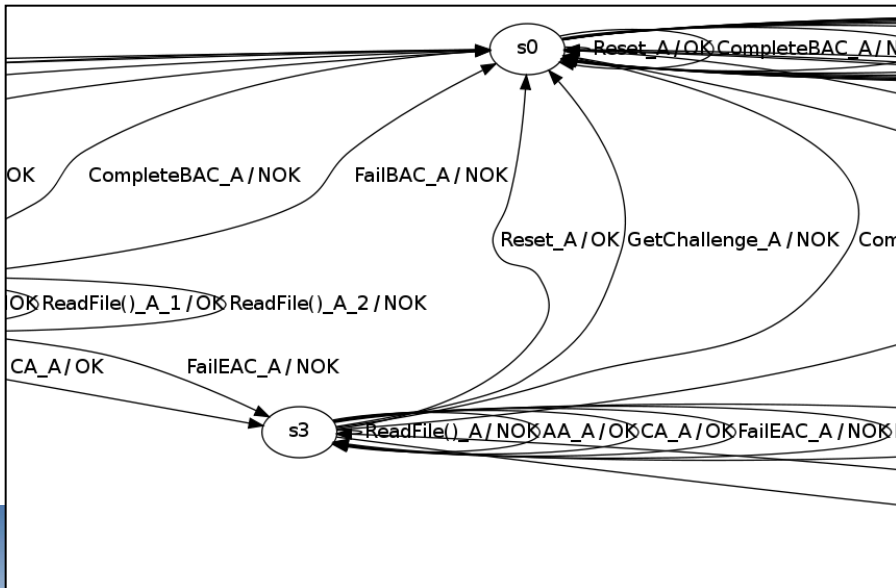
- Biometric Passport
- [Aarts et. al, 2010]

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, ..

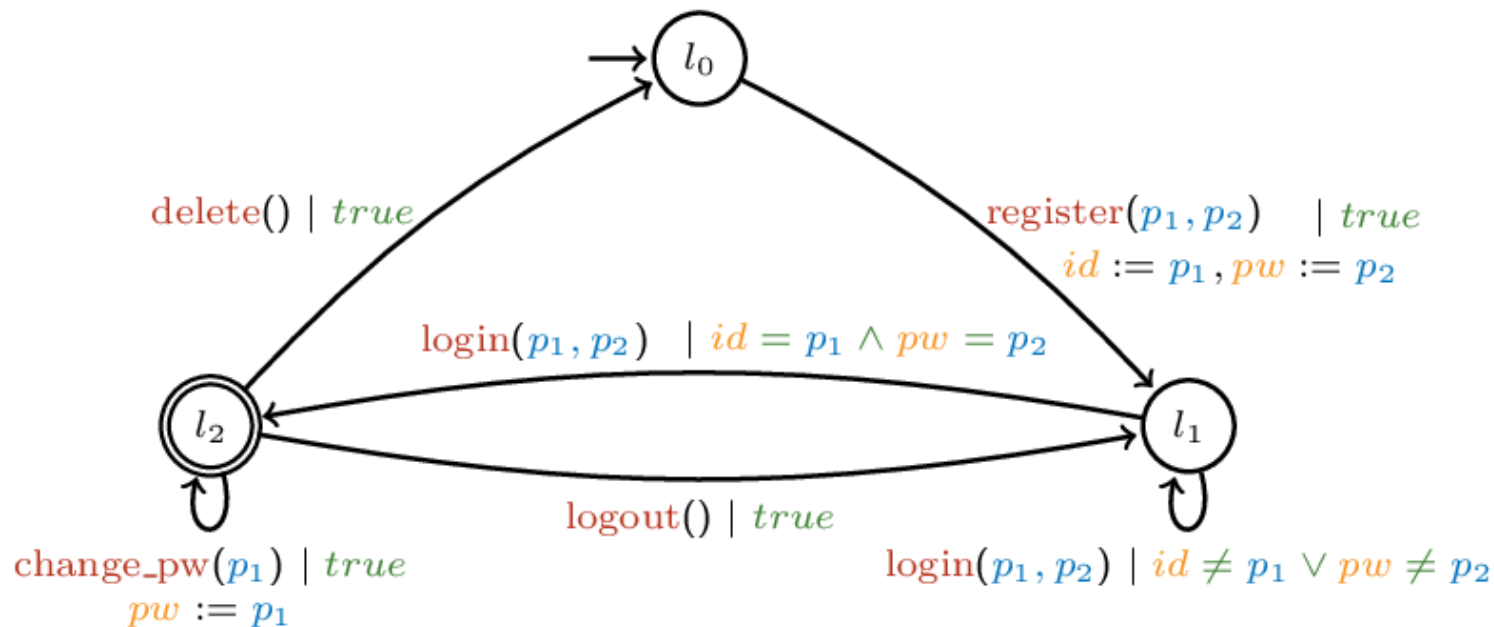
Extend automaton model

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

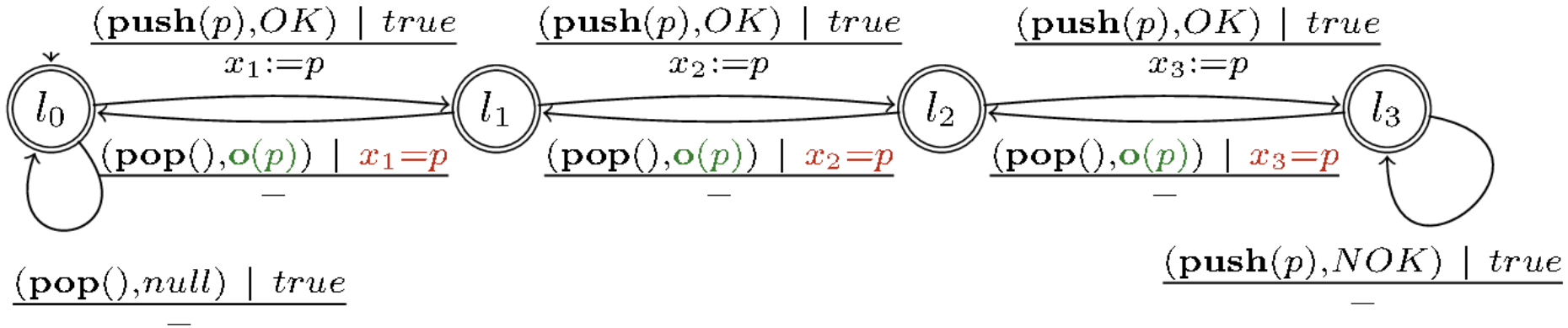
How to extend the learning techniques?

Register Automata

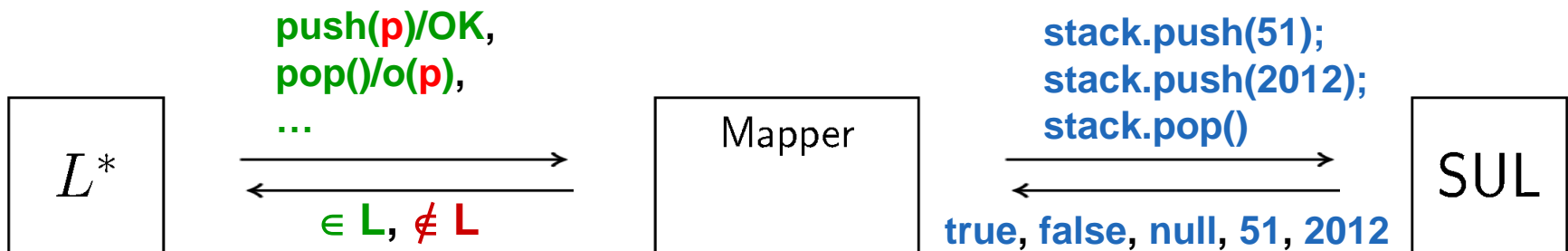
- locations
- registers (e.g., *id*, *pw*)
- transitions with:
 - actions with formal parameters
 - guards
 - assignments to registers



The Impact of Register Automata



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



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.

register(Bob,123) login(Bob,123)
register(Alice,456) login(Alice,456)

Let $\mathcal{W}_{\mathcal{D}}$ be the set of all data words.

Definition (Equivalence wrt. $\mathcal{L}_{\mathcal{D}}$)

Two words $u, u' \in \mathcal{W}_{\mathcal{D}}$ are equivalent wrt. $\equiv_{\mathcal{L}}$ iff there **exists** a permutation π on \mathcal{D} s.t. for all $v \in \mathcal{W}_{\mathcal{D}}$

$$uv \in \mathcal{L}_{\mathcal{D}} \Leftrightarrow u\pi(v) \in \mathcal{L}_{\mathcal{D}}$$

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

register(Bob,123) $\equiv_{\mathcal{L}}$ register(Alice,456)

Reusing structure of L^*

\mathcal{L} -essential prefixes (just enough equal data values)

abstract suffixes

red prefixes

id

(1,1)

(3,2)

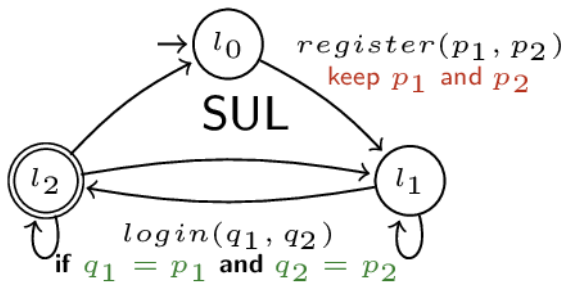
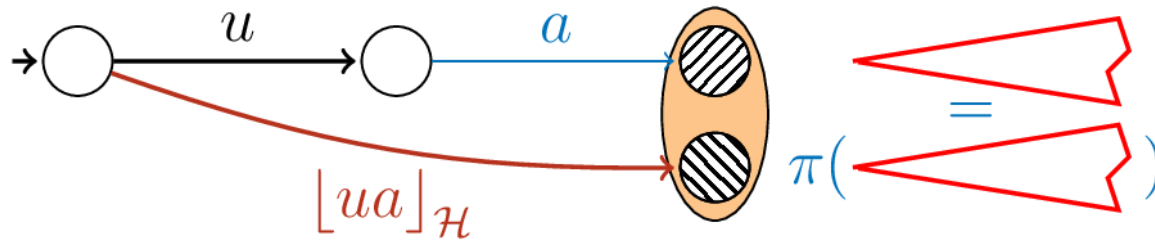
blue prefixes

		ϵ	$\text{login}(z_1, z_2)$	$\text{logout()login}(z_1, z_2)$
ϵ	(l_0)	×	$\text{login}(\bar{1}, \bar{2})$ ×	$\text{logout()login}(\bar{1}, \bar{2})$ ×
register(1, 2)	(l_1)	×	$\text{login}(\bar{1}, \bar{2})$ × $\text{login}(1, 2)$ ✓	$\text{logout()login}(\bar{1}, \bar{2})$ × $\text{logout()login}(1, 2)$ ✓
register(1, 2)login(1, 2)	(l_2)	✓	$\text{login}(\bar{1}, \bar{2})$ ✓	$\text{logout()login}(\bar{1}, \bar{2})$ × $\text{logout()login}(1, 2)$ ✓
register(1, 2)login(3, 4)		×	$\text{login}(\bar{1}, \bar{2})$ × $\text{login}(1, 2)$ ✓	$\text{logout()login}(\bar{1}, \bar{2})$ × $\text{logout()login}(1, 2)$ ✓
register(1, 2)login(1, 2)ch.pw(3)		✓	$\text{login}(\bar{1}, \bar{2})$ ✓	$\text{logout()login}(\bar{1}, \bar{2})$ × $\text{logout()login}(1, 3)$ ✓
register(1, 2)login(1, 2)logout()		×	$\text{login}(\bar{1}, \bar{2})$ × $\text{login}(1, 2)$ ✓	$\text{logout()login}(\bar{1}, \bar{2})$ × $\text{logout()login}(1, 2)$ ✓

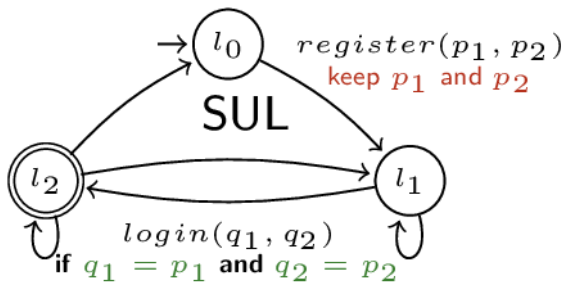
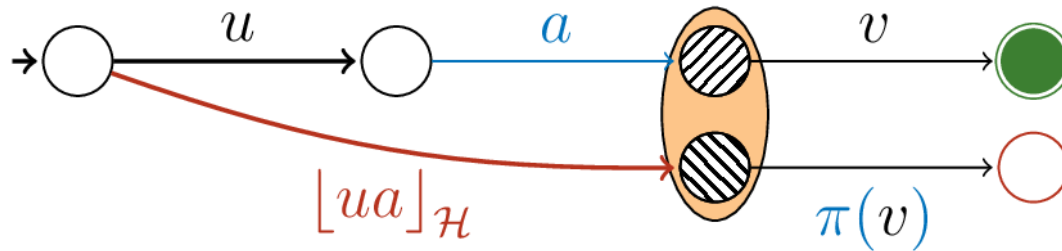
Analysis of Counterexamples III

- Counterexample Analysis for inferring
 - New locations
 - New **registers**
 - New **transitions**

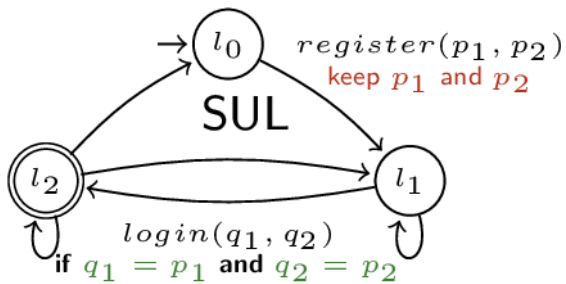
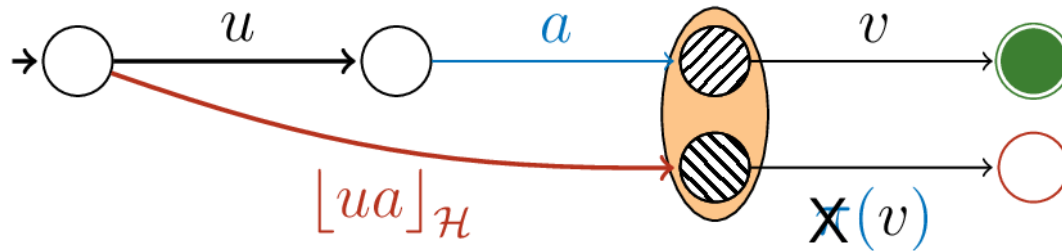
CE: New location



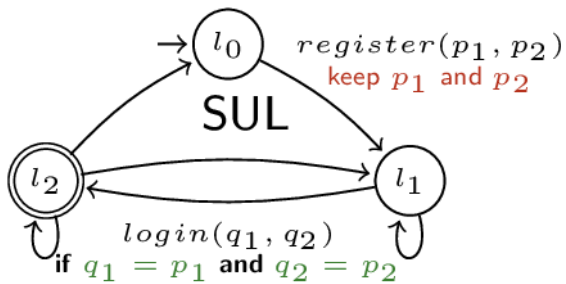
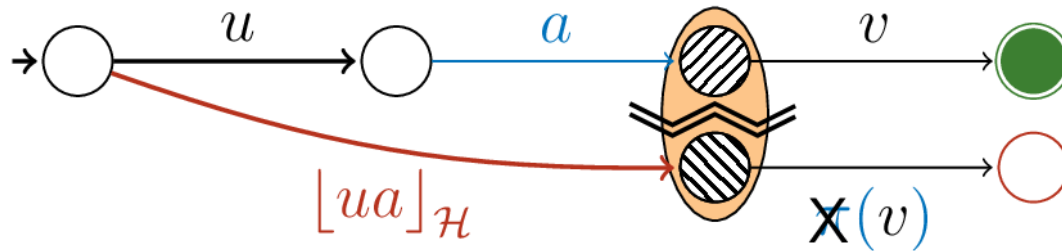
CE: New location



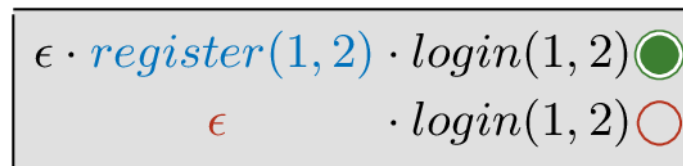
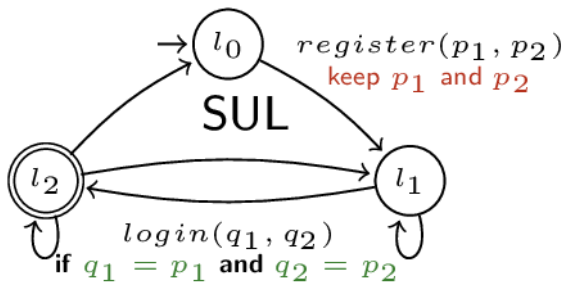
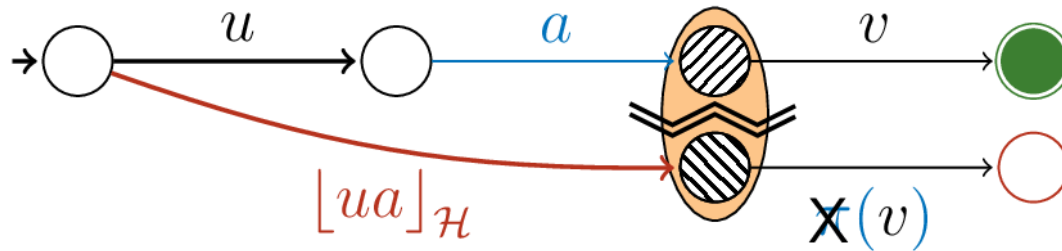
CE: New location



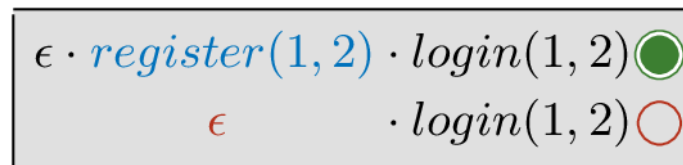
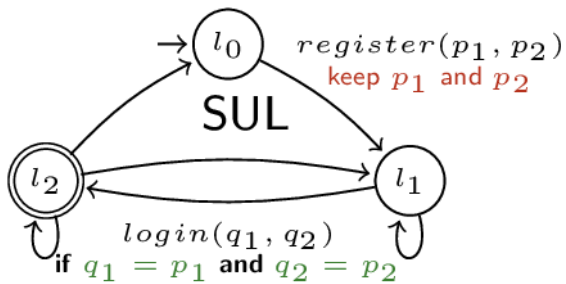
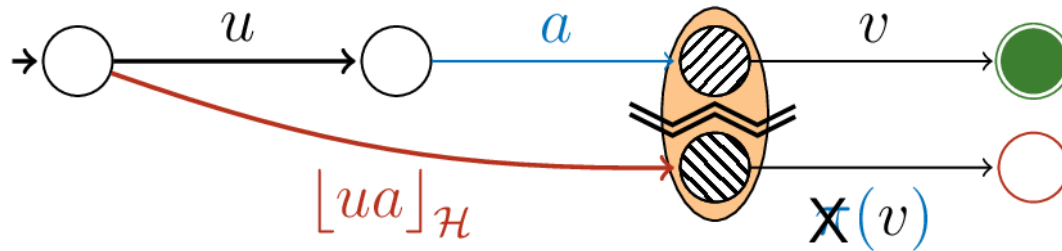
CE: New location



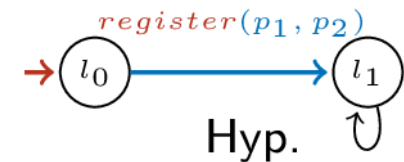
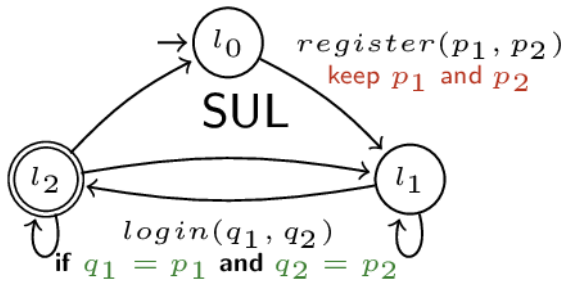
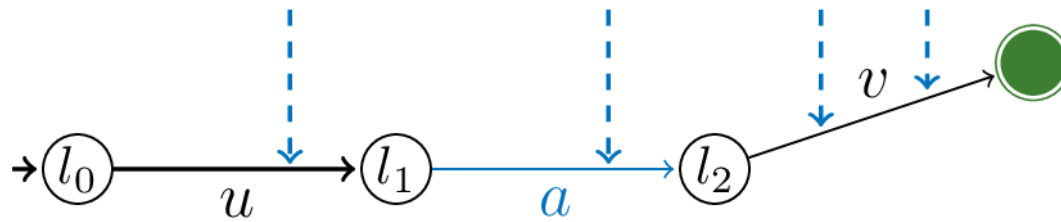
CE: New location



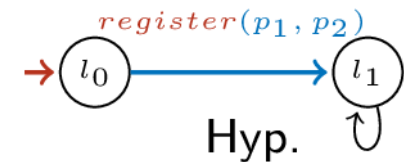
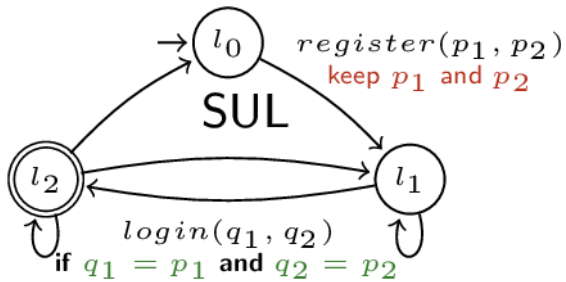
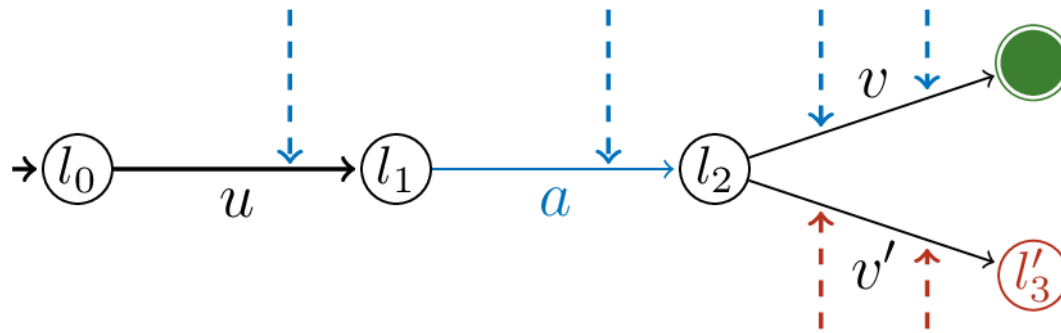
CE: New location



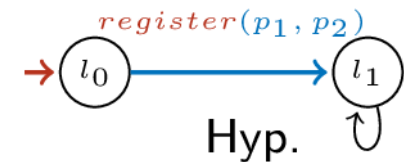
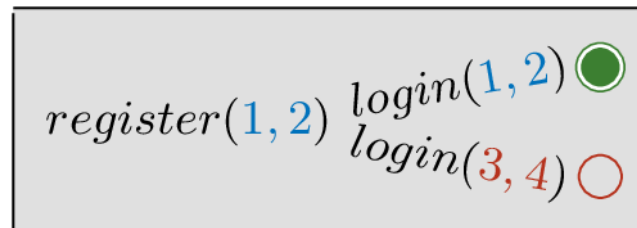
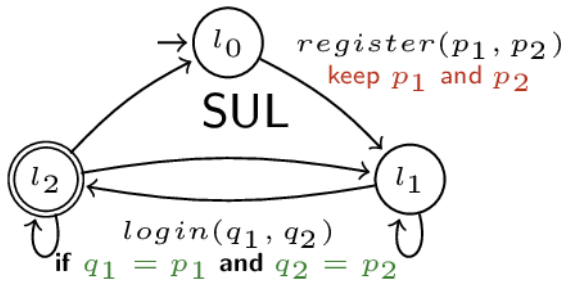
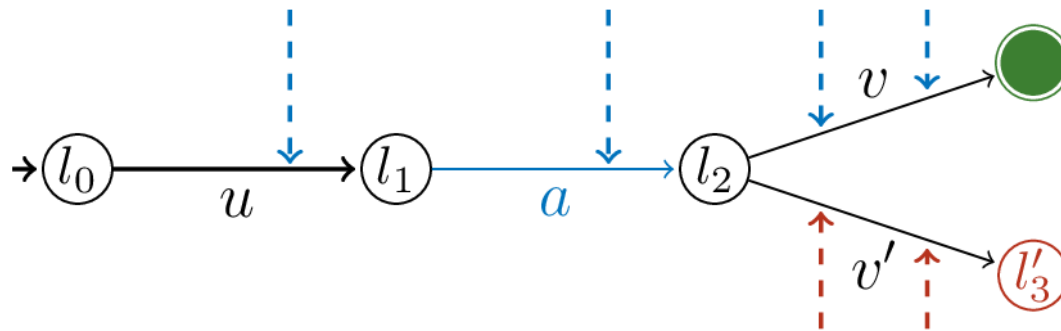
CE: New register



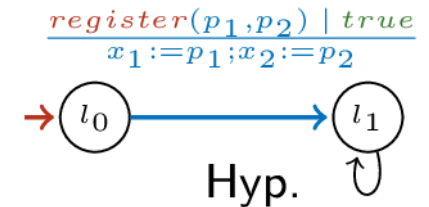
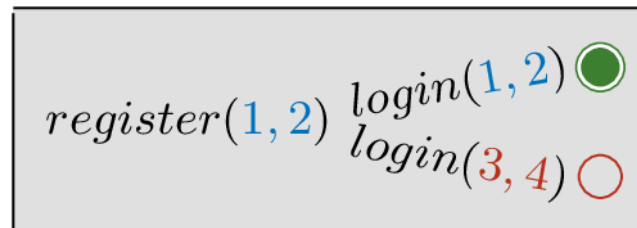
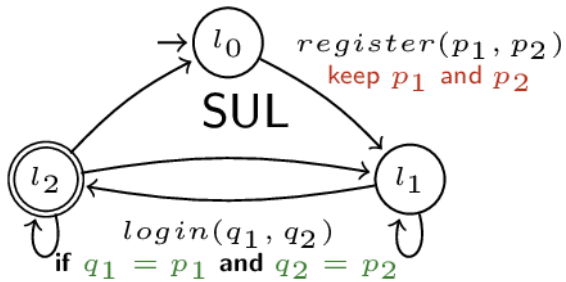
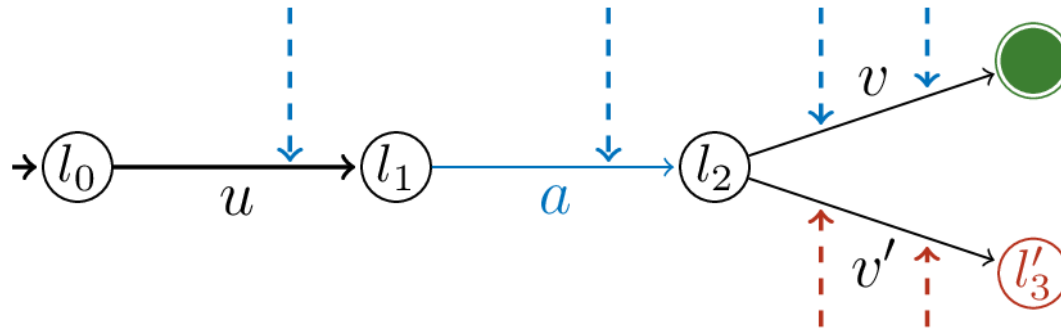
CE: New register



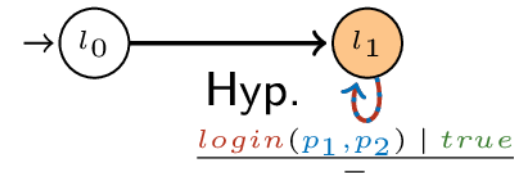
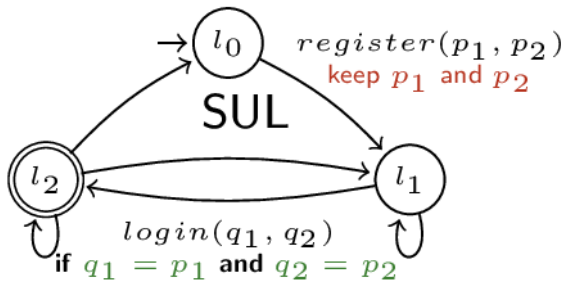
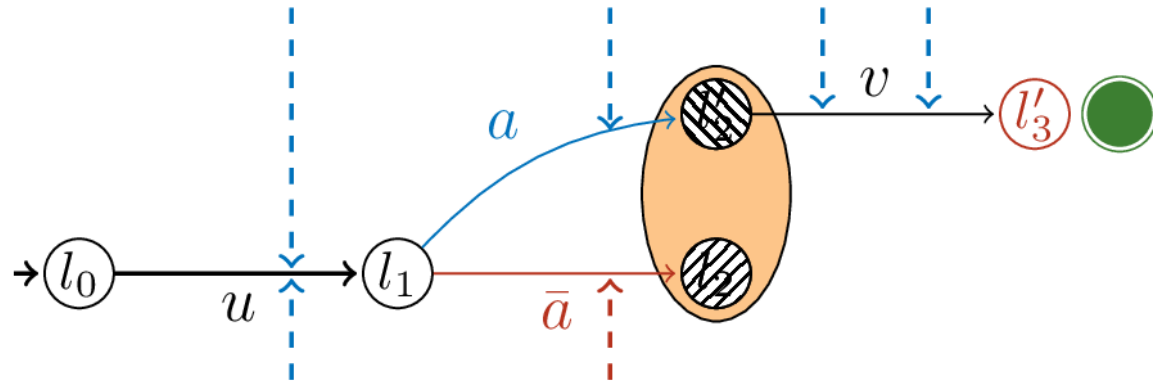
CE: New register



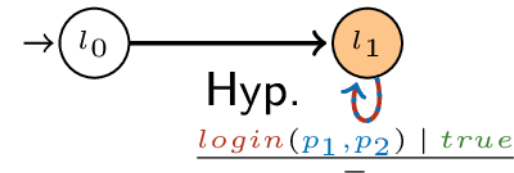
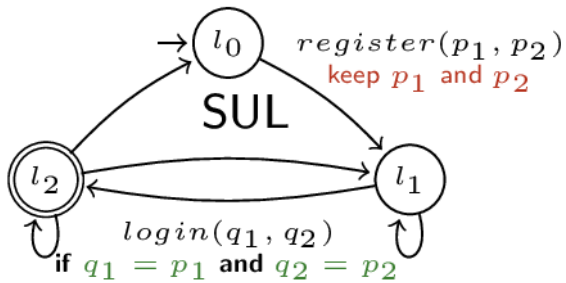
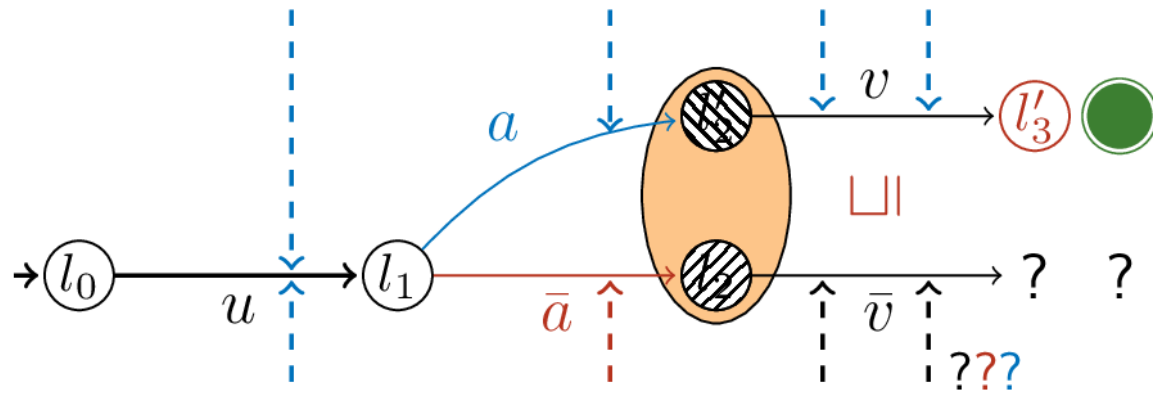
CE: New register



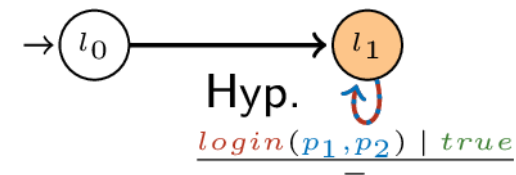
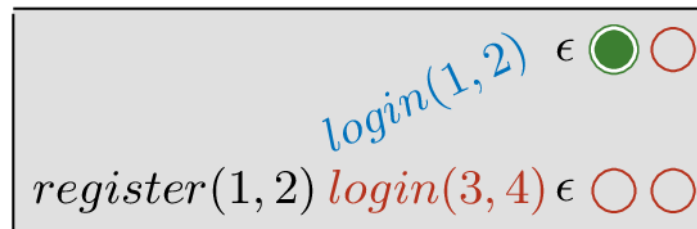
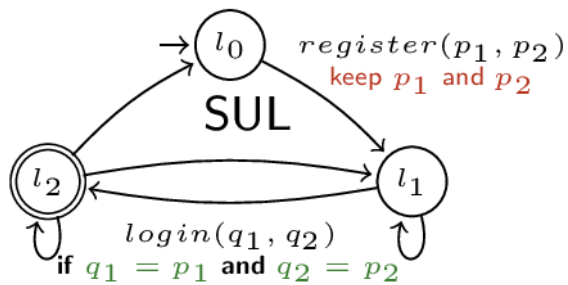
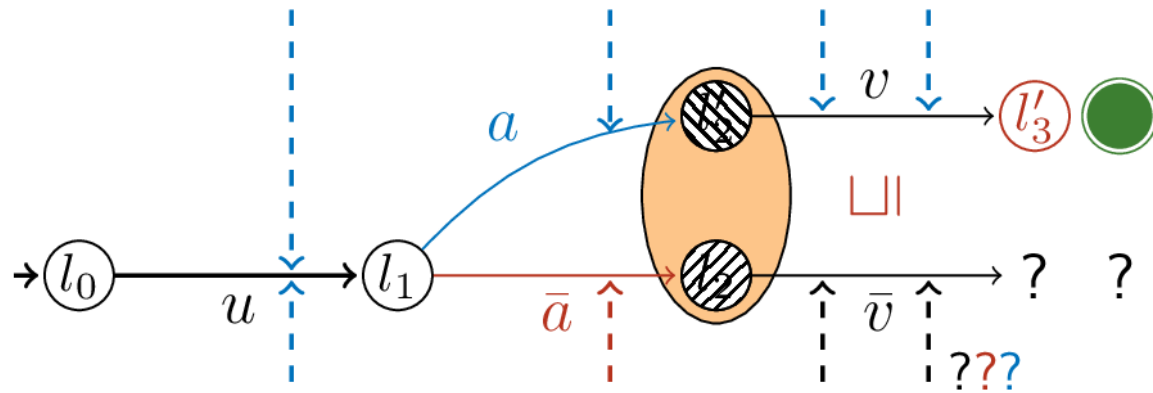
CE: New transition



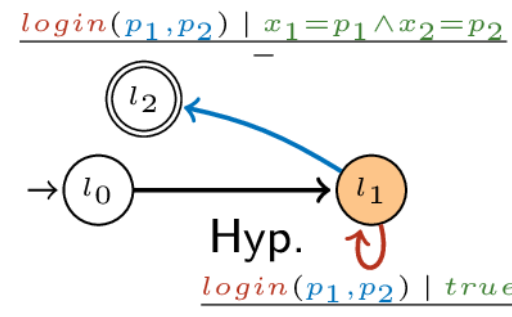
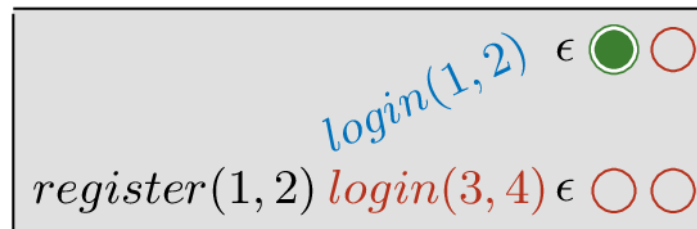
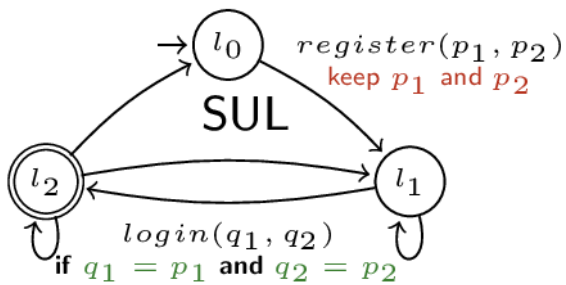
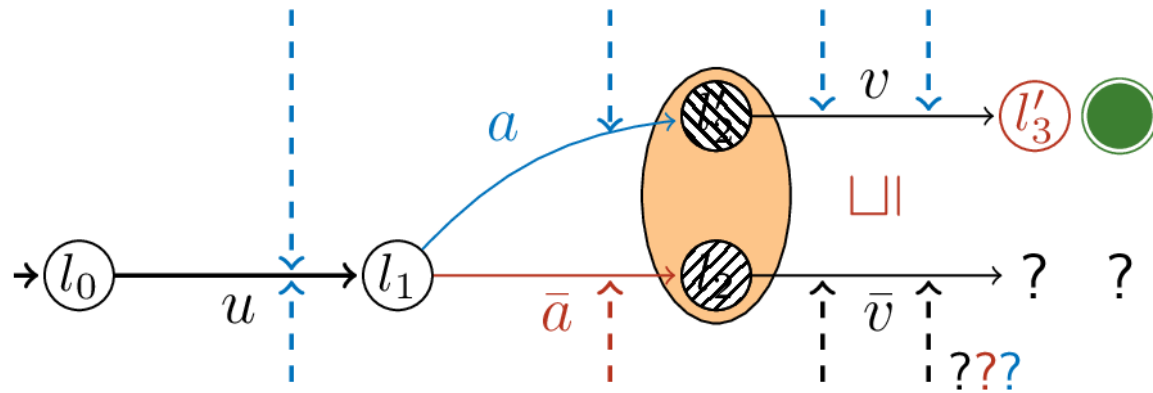
CE: New transition



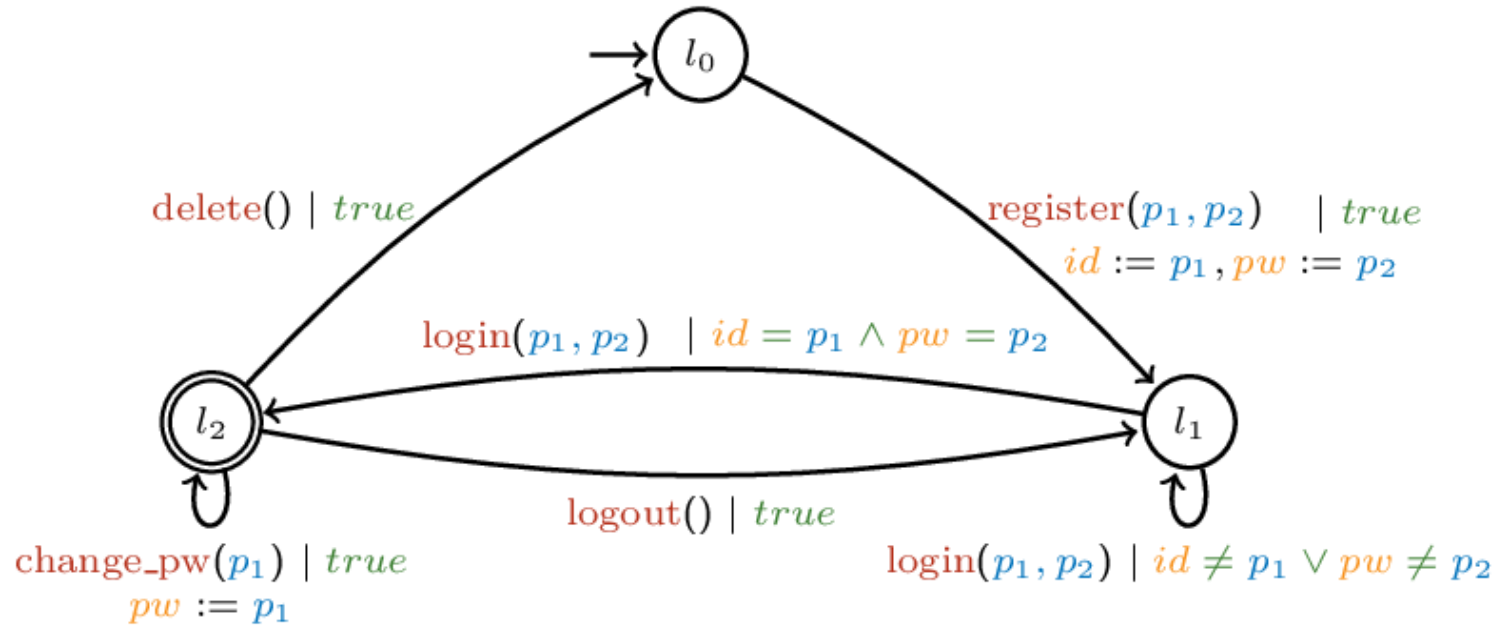
CE: New transition



CE: New transition



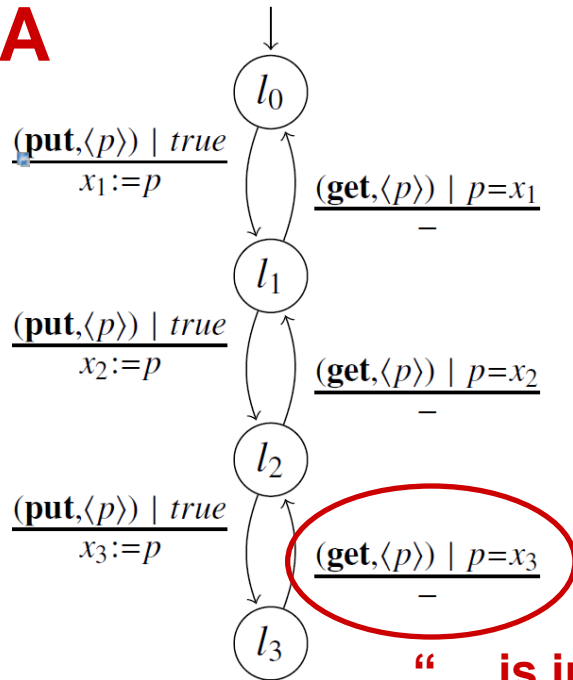
Experimental Evaluation



Setup	# Loc.	# Trans.	MQs	EQs
L^* , no optimization, $ \mathcal{D} = 6$)	73	5,913	415,333	72
L^* , symmetry reduction, $ \mathcal{D} = 6$)	73	5,913	2,776	2
RA learning algorithm	3	16	403	3

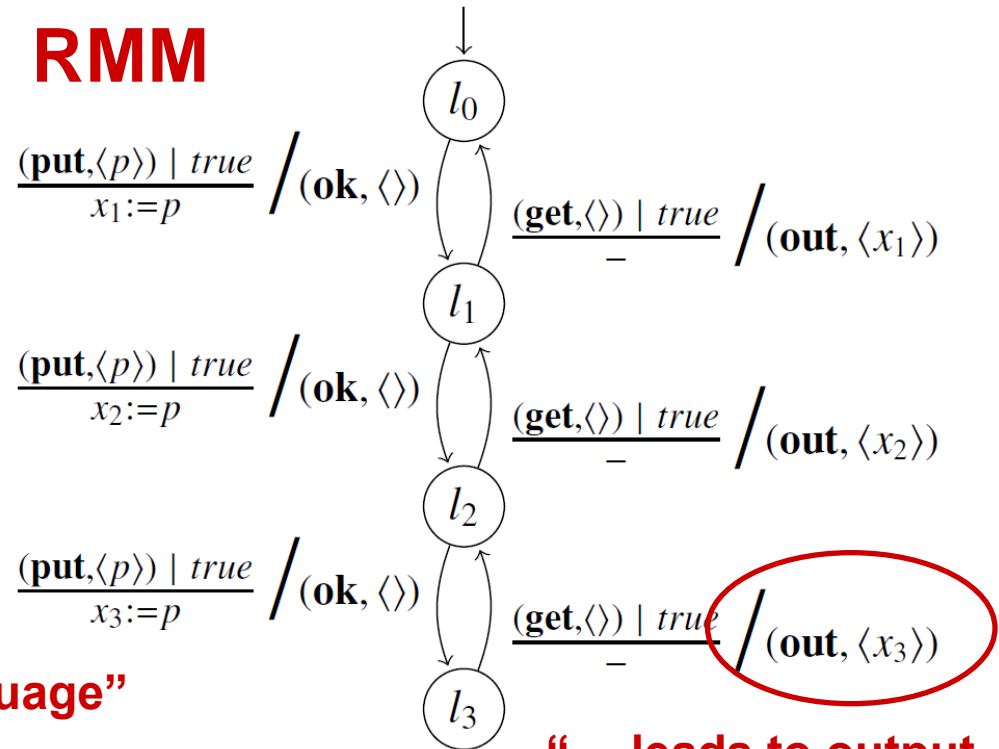
Modeling Output explicitly: RMMs

RA



“... is in language”

RMM

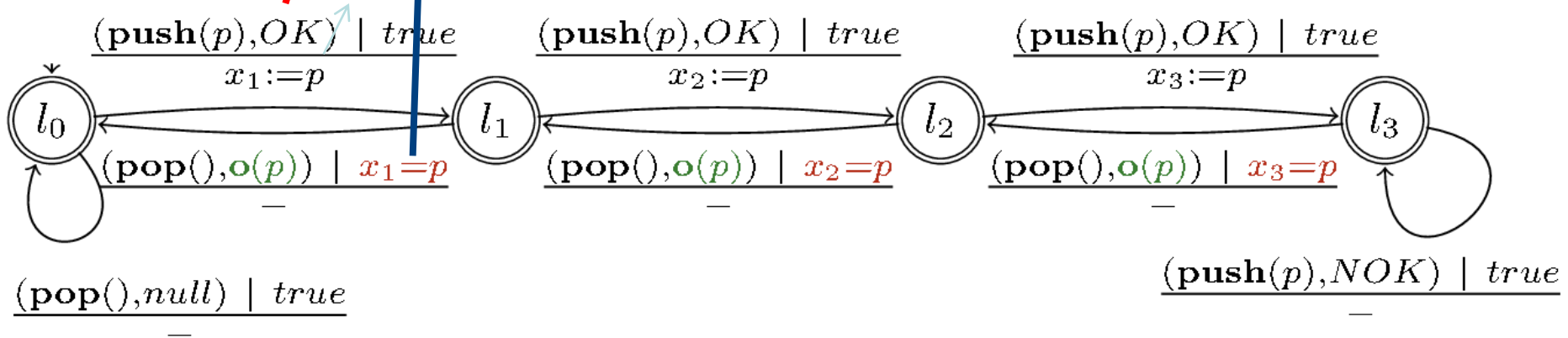
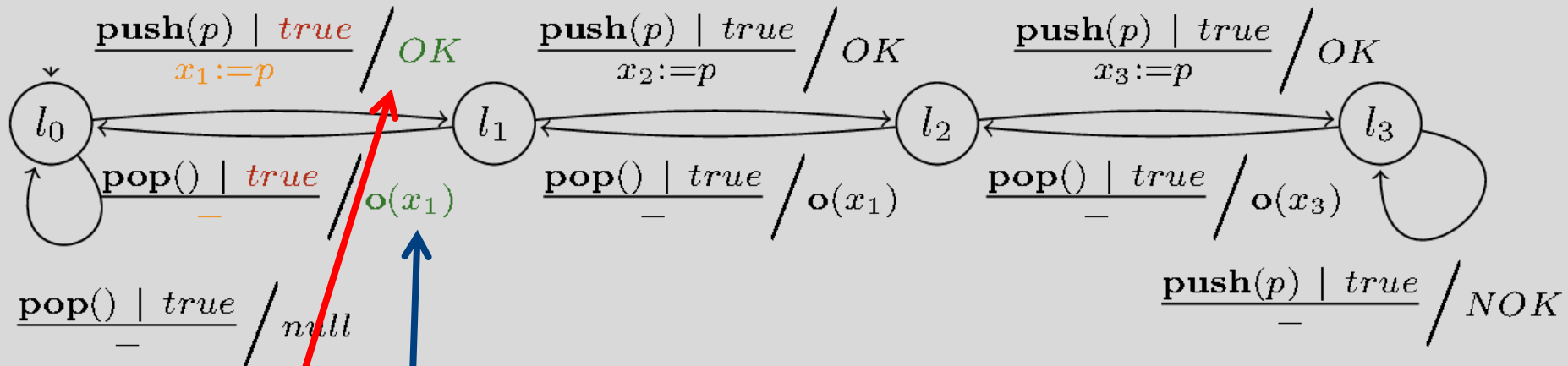


“... leads to output ...”

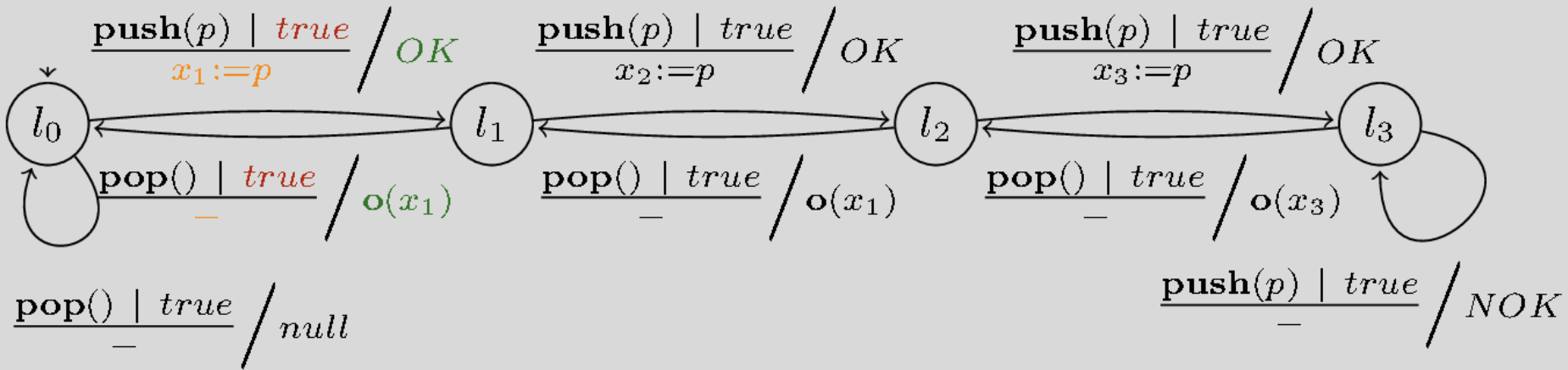
Example: Stack of capacity 3

- RA: output encoded as guarded transition
- RMM: output with data for transitions

RMM: Explicit Output



RMM: Explicit Output



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

L^*

$\xrightarrow{\text{push}(p), \text{pop}()}$
 $\xleftarrow{\text{OK, NOK, null, } p}$

Mapper

$\xrightarrow{\text{stack.push}(51)$
 $\text{stack.push}(2012)$
 $\text{stack.pop}()}$
 $\xleftarrow{\text{true, false, null, } 51, 2012}$

SUL

Inferring RMMs

Name	Mealy ($ \mathcal{D} = 4$)			RA			RMM		
	$ Q $	MQs	EQs	$ L $	MQs	EQs	$ L $	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

- Example: Nested stack of capacity 16
 - RMM: **781** locations, **45k** MQ, **9** EQ, **20 sec.**
 - Mealy, $|\mathcal{D}|=4$: **> 10^9** states

Outline

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

Conclusions and Perspectives

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 or

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

