Learning of Automata Models Extended with Data

Bengt Jonsson

Uppsala University

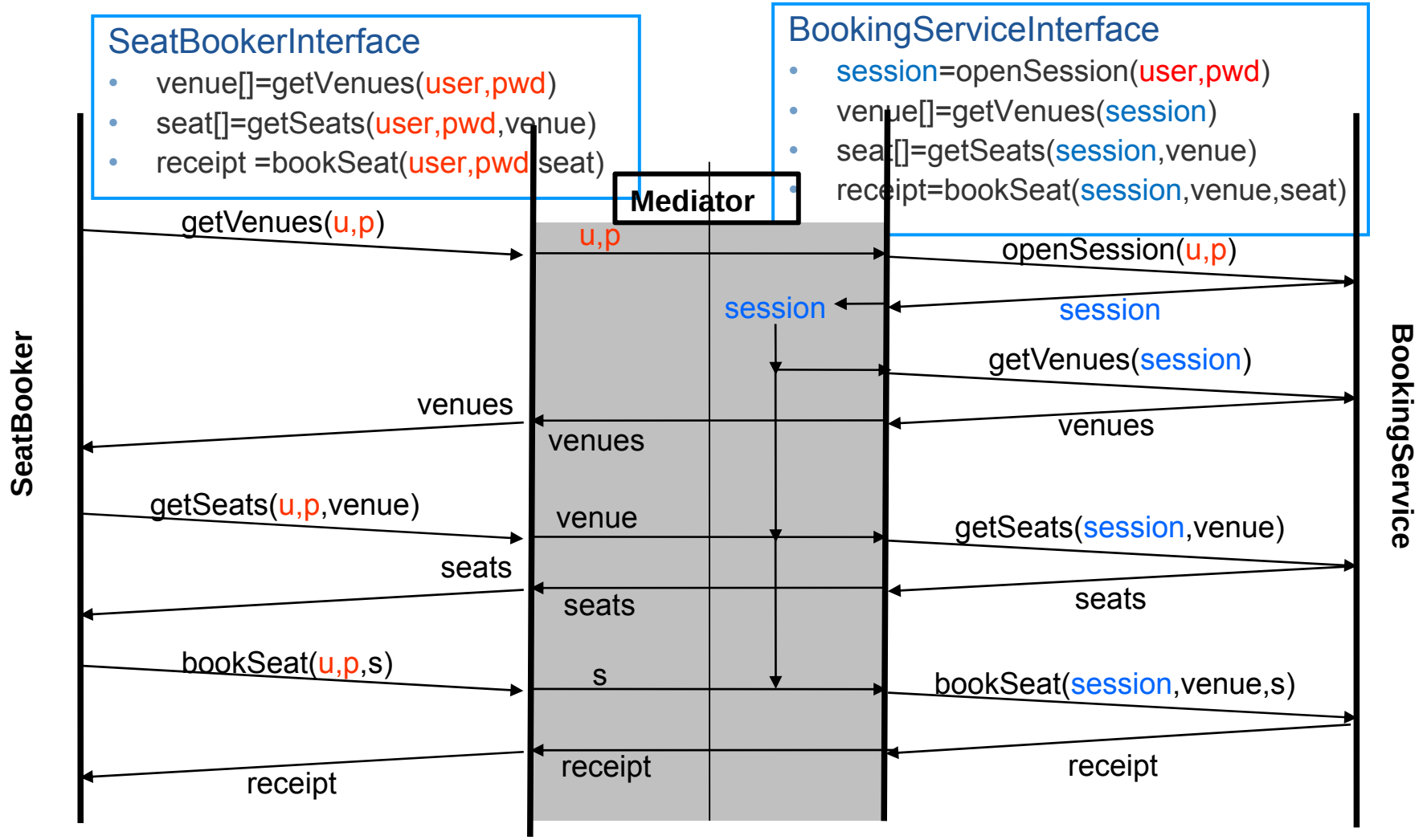
Acknowledgments

Fides Aarts
Therese Bohlin
Olga Grinchtein
Falk Howar
Martin Leucker
Maik Mertens
Harald Raffelt
Bernhard Steffen
Johan Uijen
Frits Vaandrager

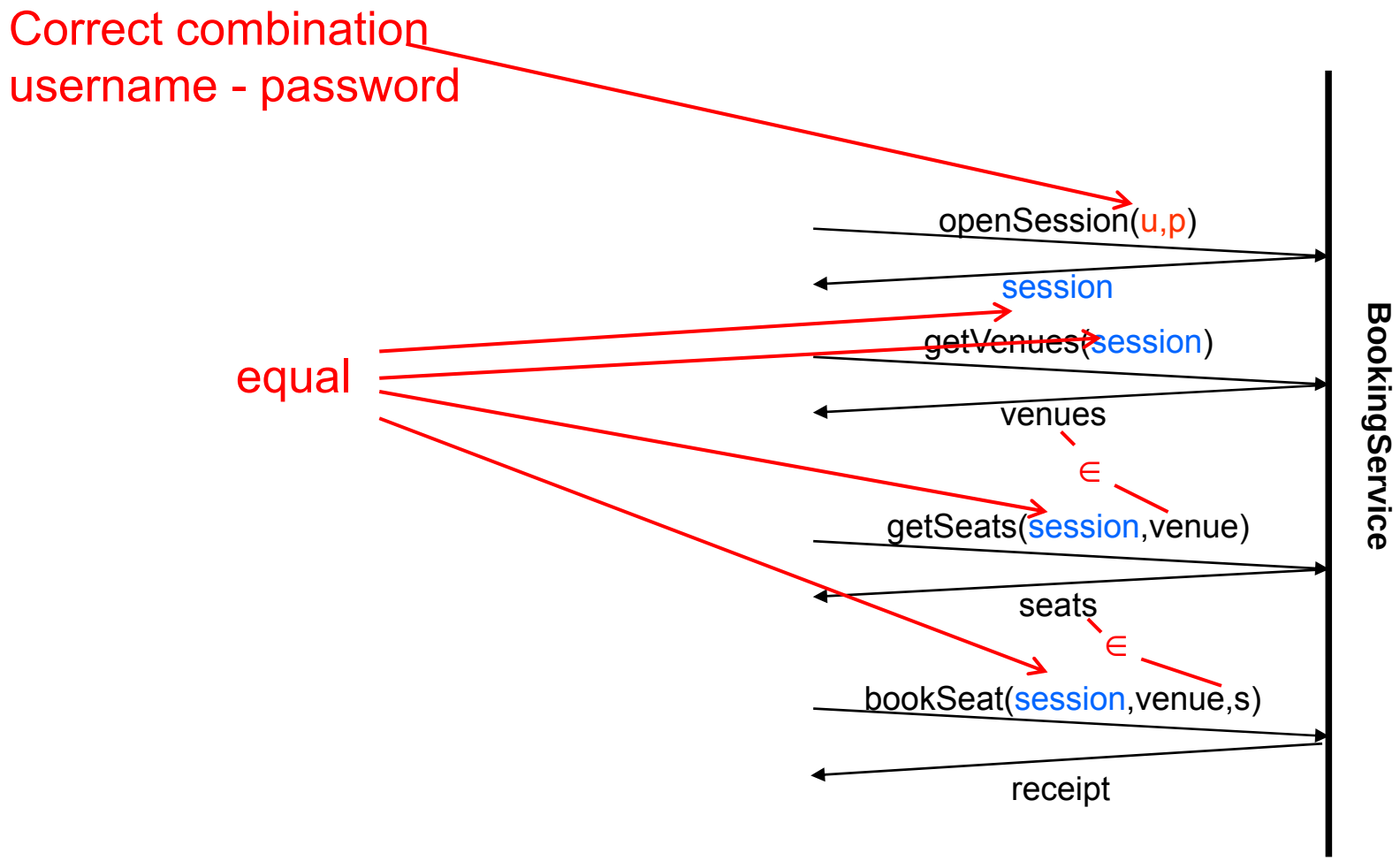
Outline

- Motivation
- Formalisms for Automata with Data
- Abstraction
- Learning Setup
- Some Completeness Result
- Abstraction Refinement
- Applications and Evaluation
- Conclusion and Future Work

Motivating Use Case



Data Relationships

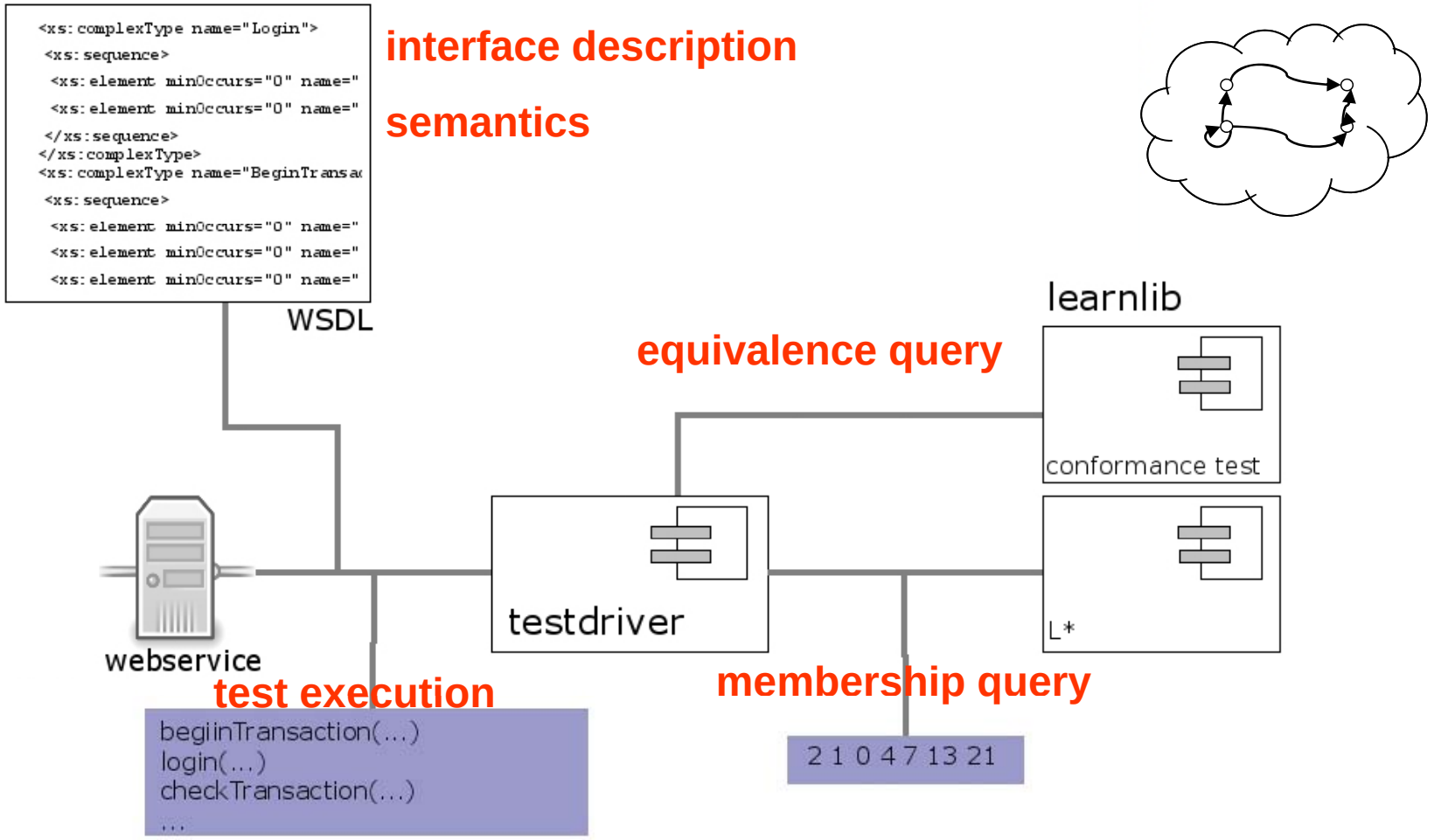


Motivation: More examples

Interface Specifications

- Container classes
  - must keep track of identities of data
  - relate data in input to data in subsequent output
- Communication protocols
  - SIP, TCP, …
  - sequence numbers, identifiers, ..

Practical Learning Scenario



Finite-State Mealy Machines

Finite State Machines w. input & output

Σ_I input symbols

Σ_O output symbols

Q states

q_0 initial state

δ: $Q \times \Sigma_I \rightarrow Q$ transition function

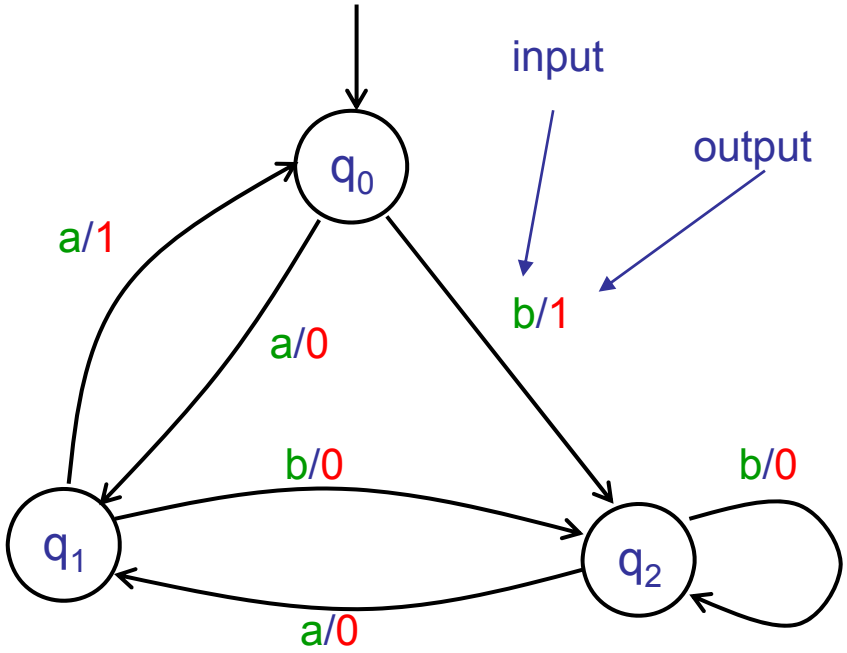
λ: $Q \times \Sigma_I \rightarrow \Sigma_O$ output function

Notation: $q \xrightarrow{a / b} q'$

- Often used for protocol modeling

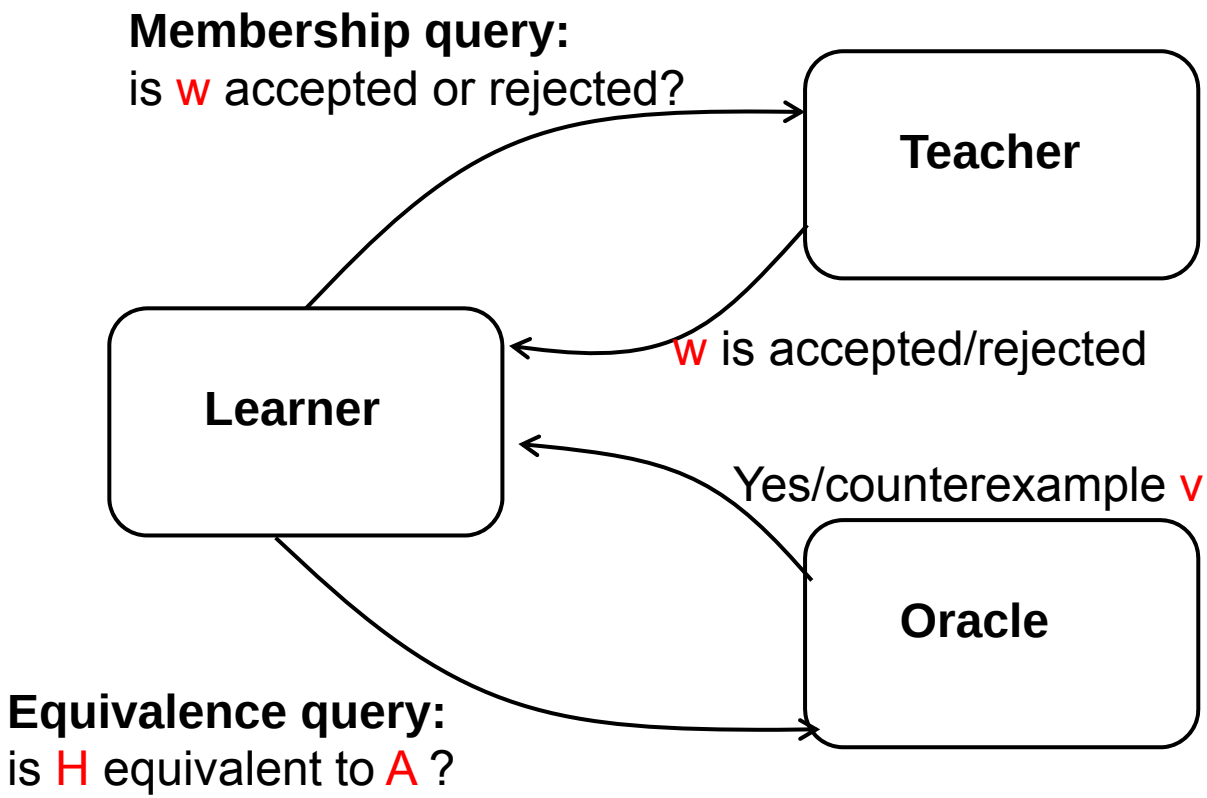
Assumptions:

- Deterministic
- Completely specified



Basic Learning Setup

Same as in L*



Baseline: Automata Learning

L* infers Finite State Machine from membership queries:

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

- Needs $O(n^3)$ queries to form Hypothesis of size n
  - In practice, often $O(n^2 \log n)$ queries
  - Domain-specific optimizations can help a lot
- Has been used to learn large automata (≥20 kstates)
- Adapted for Mealy Machines (by Niese et al. 2003)

How to Extend w. Data?

Extend Mealy Machine Model

- Input and output symbols parameterized by data values.
- State variables remember parameters in received input
- Types of parameters could be, .e.,g
  - Identifiers of connections, sessions, users
  - Sequence numbers
  - Time values

Extend Learning Techniques

- Several conceivable approaches
- We will attempt to reuse L* approach
  - Augment by Abstraction Techniques

Input and Output Symbols

Assume

- Domains, e.g.,
  - STRING e.g., ‘Mary’, ‘174’, …
  - SESSION e.g., 0,1,2,3, …
  - SEAT e.g., 1,2,3, …., 167
- (Input and Output) Actions: with arities, e.g.,

openSession	STRING x STRING x SESSION
getSeat	SESSION x SEAT

- Symbols

openSession(‘Mary’, ’188H#4’, 42)

action parameters

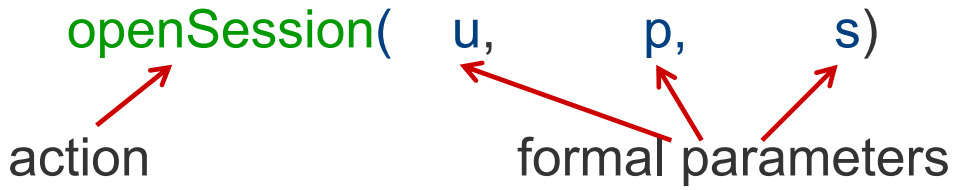
Input and Output Symbols

Assume

- Domains, e.g.,
  - STRING e.g., ‘Mary’, ‘174’, …
  - SESSION e.g., 0,1,2,3, …
  - SEAT e.g., 1,2,3, …., 167
- (Input and Output) Actions: with arities, e.g.,

openSession	STRING x STRING x SESSION
getSeat	SESSION x SEAT

- Parameterized Symbols



Guards and Expressions

Assume

- Domains, e.g.,
  - STRING e.g., ‘Mary’, ‘174’, …
  - SESSION e.g., 0,1,2,3, …
  - SEAT e.g., 1,2,3, …., 167
- Relations on Data, e.g.,

=	
∈	SEAT x SEATS
has_passwd	STRING x STRING

Symbolic Mealy Machine

A Symbolic Mealy Machine consists of

- I Input Actions
- O Output Actions
- L Locations
- l_0 Initial location
- X State variables (typed)
- → Symbolic Transitions

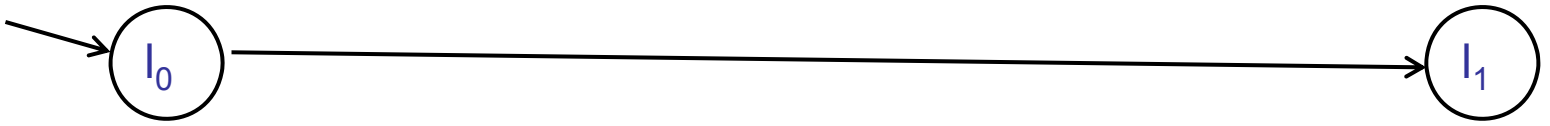
State Variables
cur_session : SESSION
cur_seats : SEATS
booked : SEATS

Input Action

Formal parameters

getSeat(s,seat)
[s = cur_session ∧ seat ∈ cur_seats]/
booked := booked ∪ seat ;
bookedSeat(seat)

Parameterized input symbol
guard
assignment
output expression



Example

State Variables
cur_session : SESSION
cur_seats : SEATS
booked : SEATS

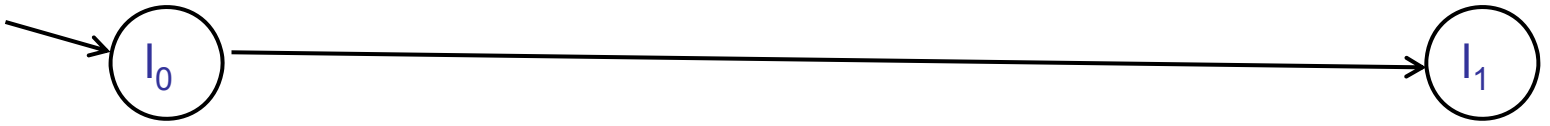
(Maybe complete the Example Here *)*

Input Action

Formal parameters

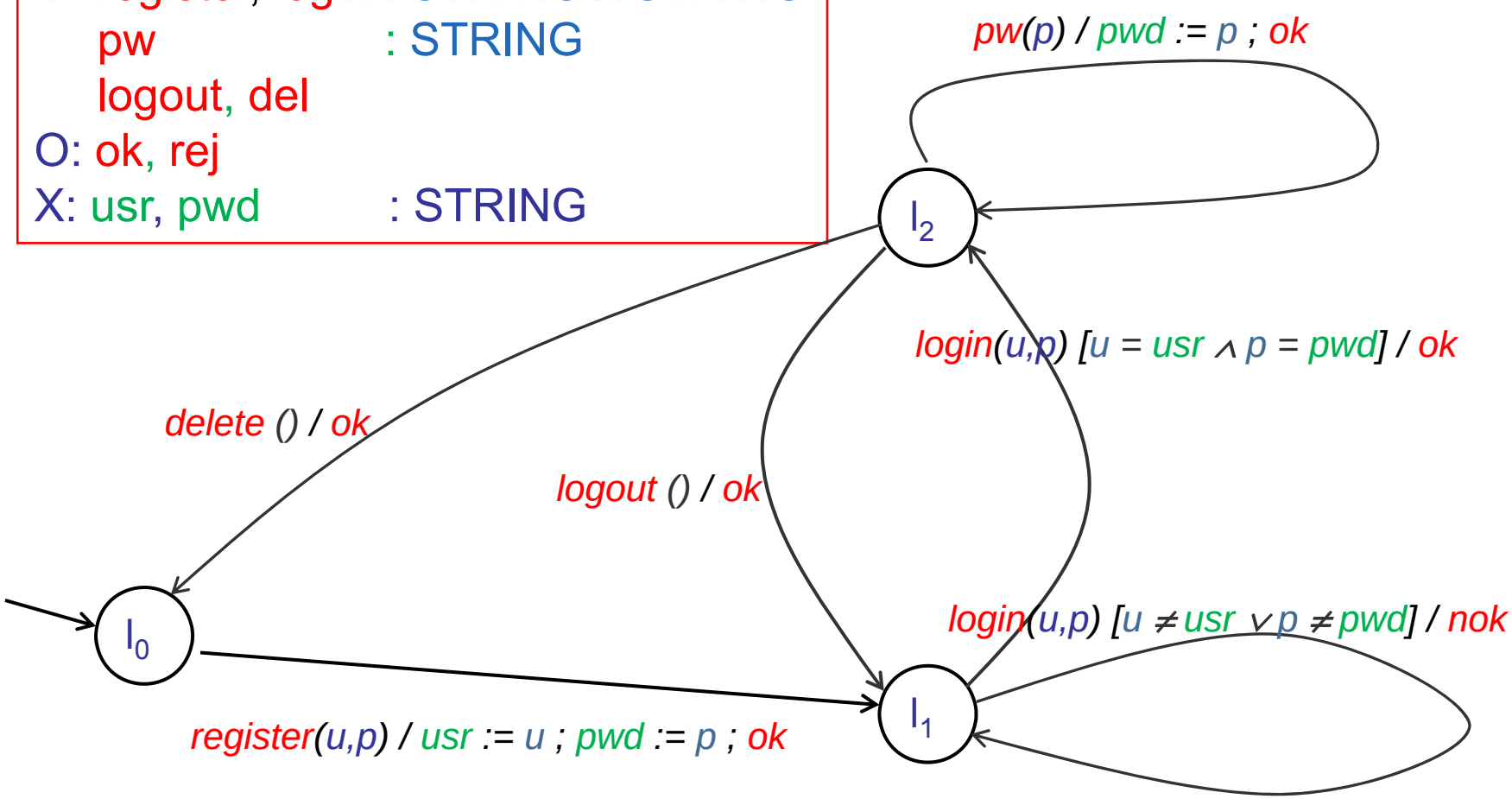
getSeat(s,seat)
[s = cur_session ∧ seat ∈ cur_seats]/
booked := booked ∪ seat ;
bookedSeat(seat)

Parameterized input symbol
guard
assignment
output expression



Example: XMPP protocol

I: register, login : STRING x STRING
pw : STRING
logout, del
O: ok, rej
X: usr, pwd : STRING



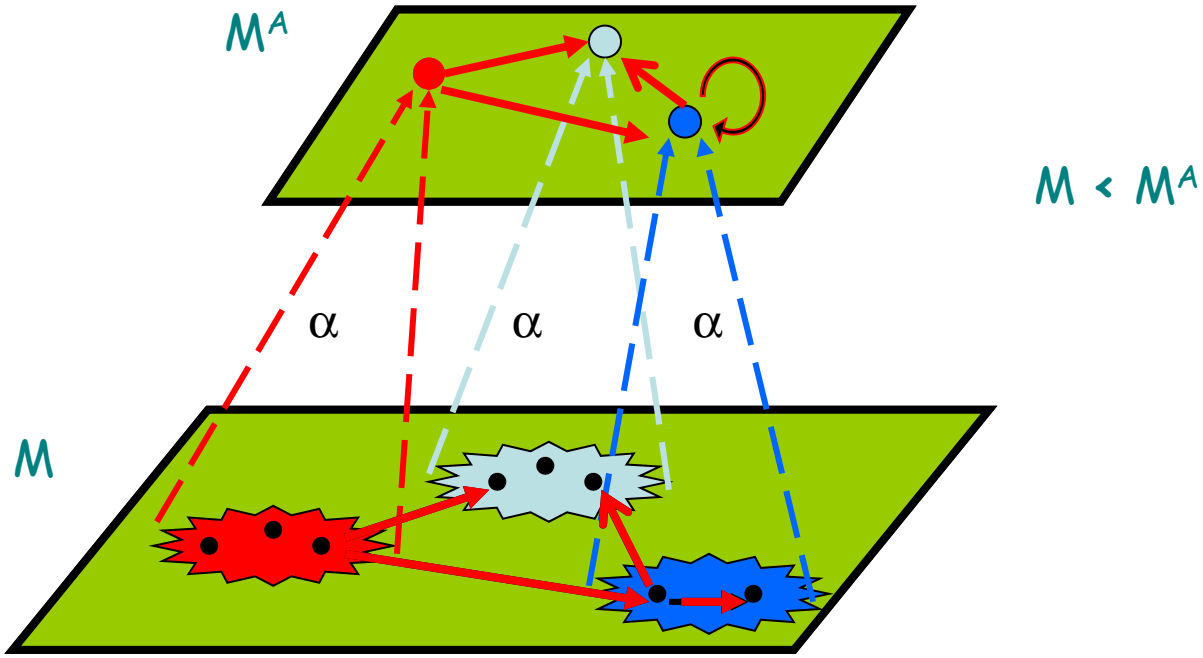
How to Adapt Learning?

- How to use L* to infer Symbolic Mealy Machines?
- L* works on finite-state Mealy machines
- SMMs are infinite state, with infinite alphabets.

IDEA: Use abstraction (from Verification/Model Checking)

- Fides Aarts, Bengt Jonsson, and Johan Uijen: *Generating Models of Infinite-State Communication Protocols using Regular Inference with Abstraction.* ICTSS 2010
- Falk Howar, Maik Merten, Bernhard Steffen *Automata Learning with Automated Alphabet Abstraction Refinement,* VMCAI 2011

Abstraction: the General Idea



Abstraction in Verification

Problem:

$$M \quad \text{satisfies} \quad \varphi \quad ?$$

Transformed into:

$$M^A \quad \text{satisfies} \quad \varphi^A \quad ?$$

Adaptation in Learning

Define an abstraction α

- α transforms the Model M into M^A

Use L* to infer M^A

- works if M^A is deterministic and finite-state

Reverse effect of α on M^A

- i.e., $M = \alpha^{-1}(M^A)$

If M^A is not adequate, refine α

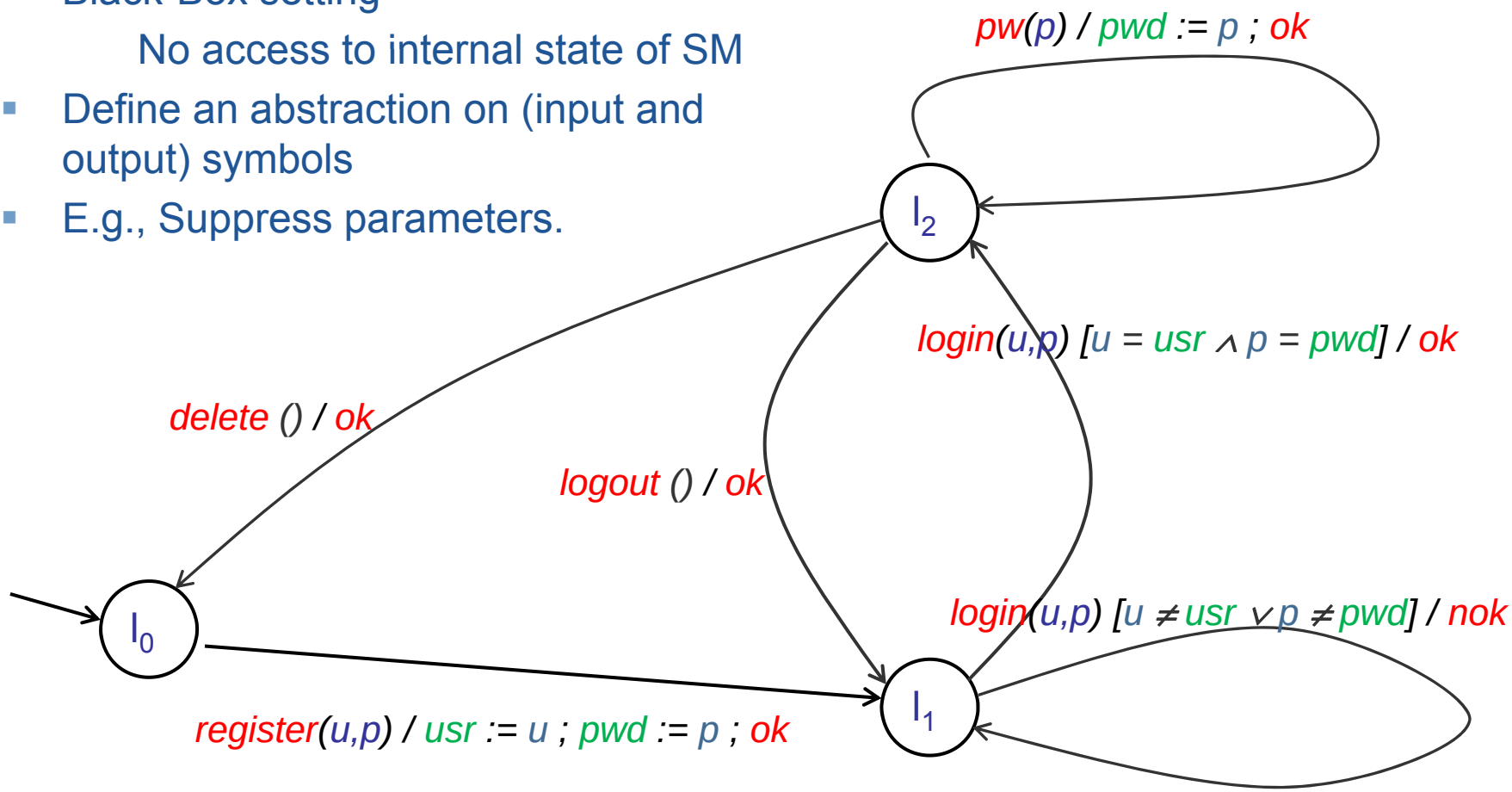
Abstraction in Learning?

- Black-Box setting -> We do not have access to internal state of SM
- Define an abstraction on (input and output) symbols

- E.g., Suppress parameters.

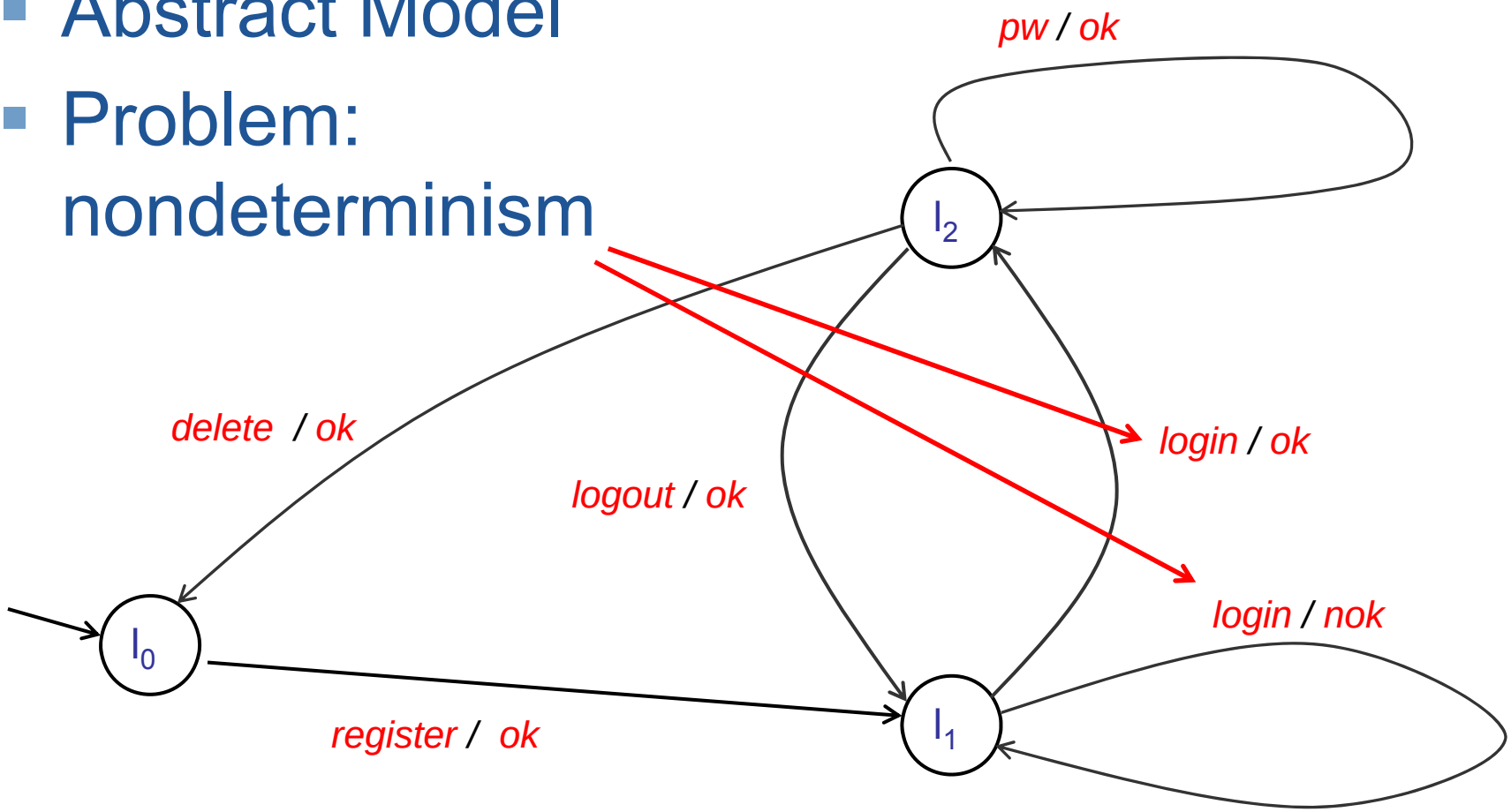
Application to Example

- Black-Box setting ->
  No access to internal state of SM
- Define an abstraction on (input and output) symbols
- E.g., Suppress parameters.

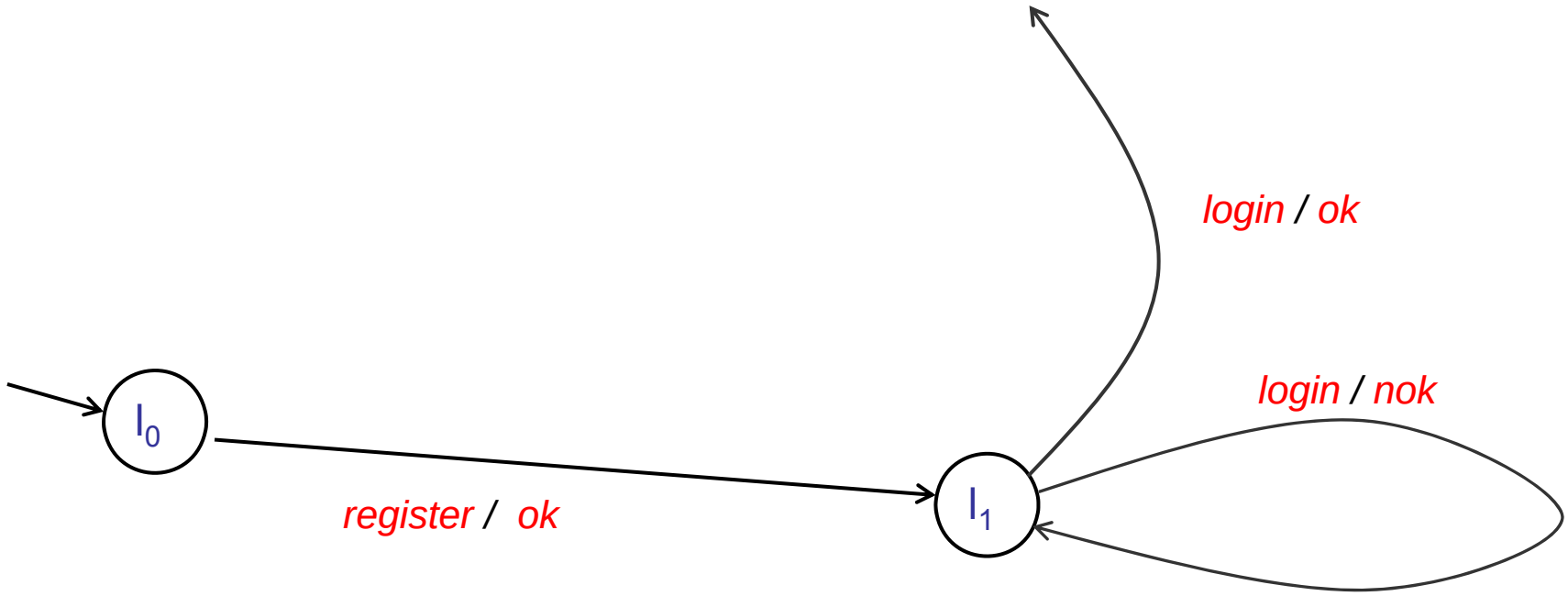


Inadequate Model

- Abstract Model
- Problem: nondeterminism

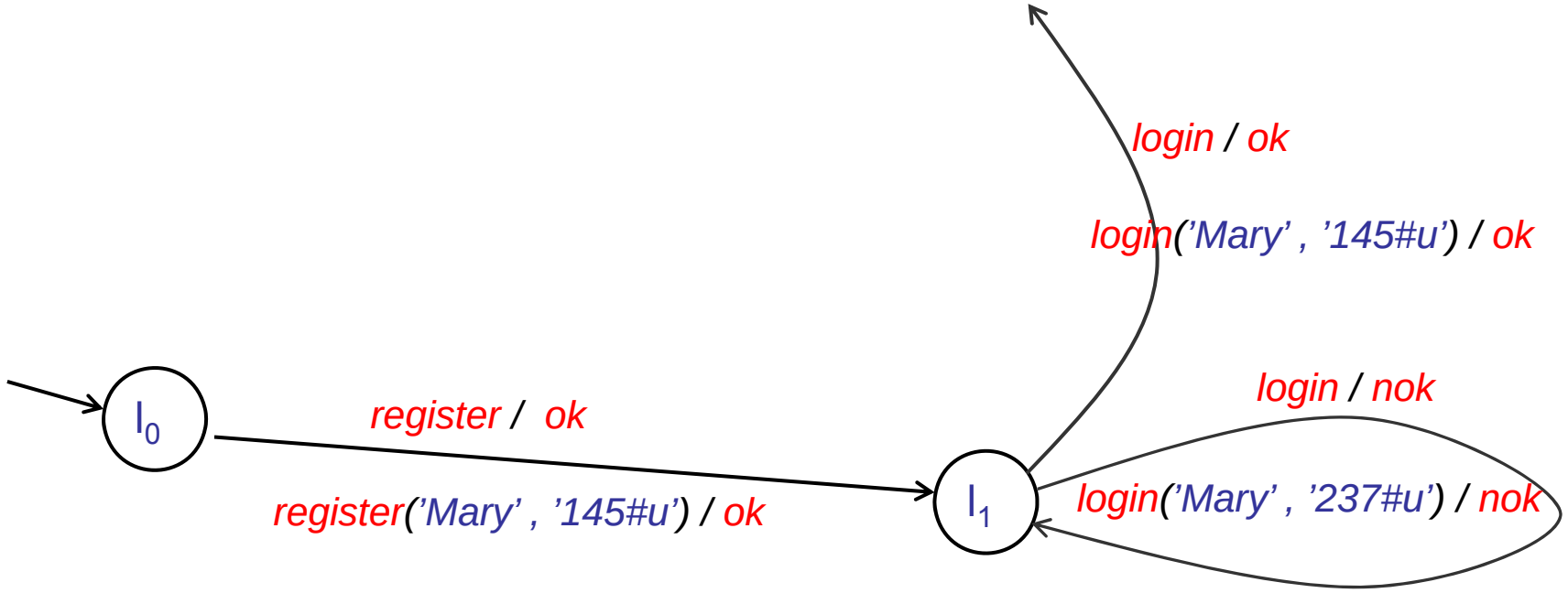


Fixing Nondeterminism-Problem



Fixing Nondeterminism-Problem

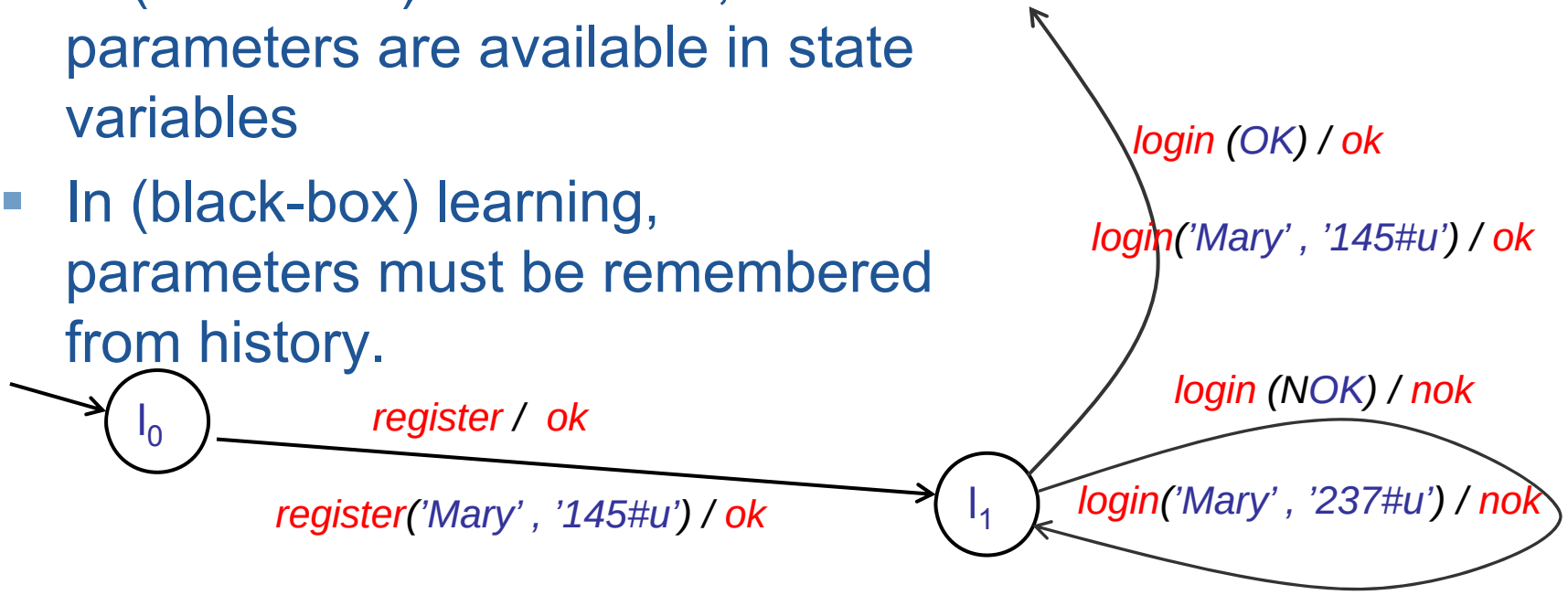
Abstraction depends on
parameters **and**
previous history



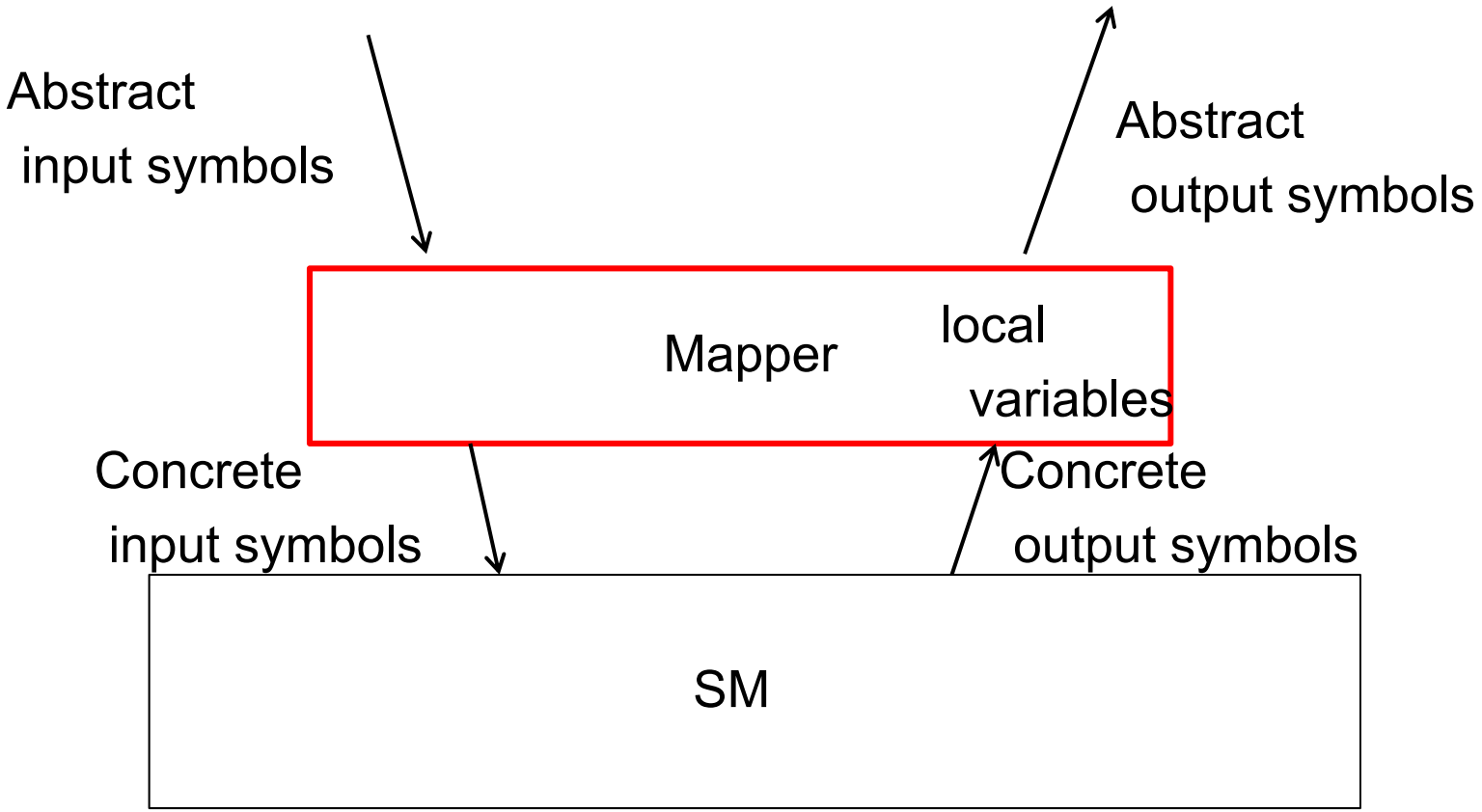
Fixing Nondeterminism-Problem

Abstraction depends on
parameters **and**
previous history

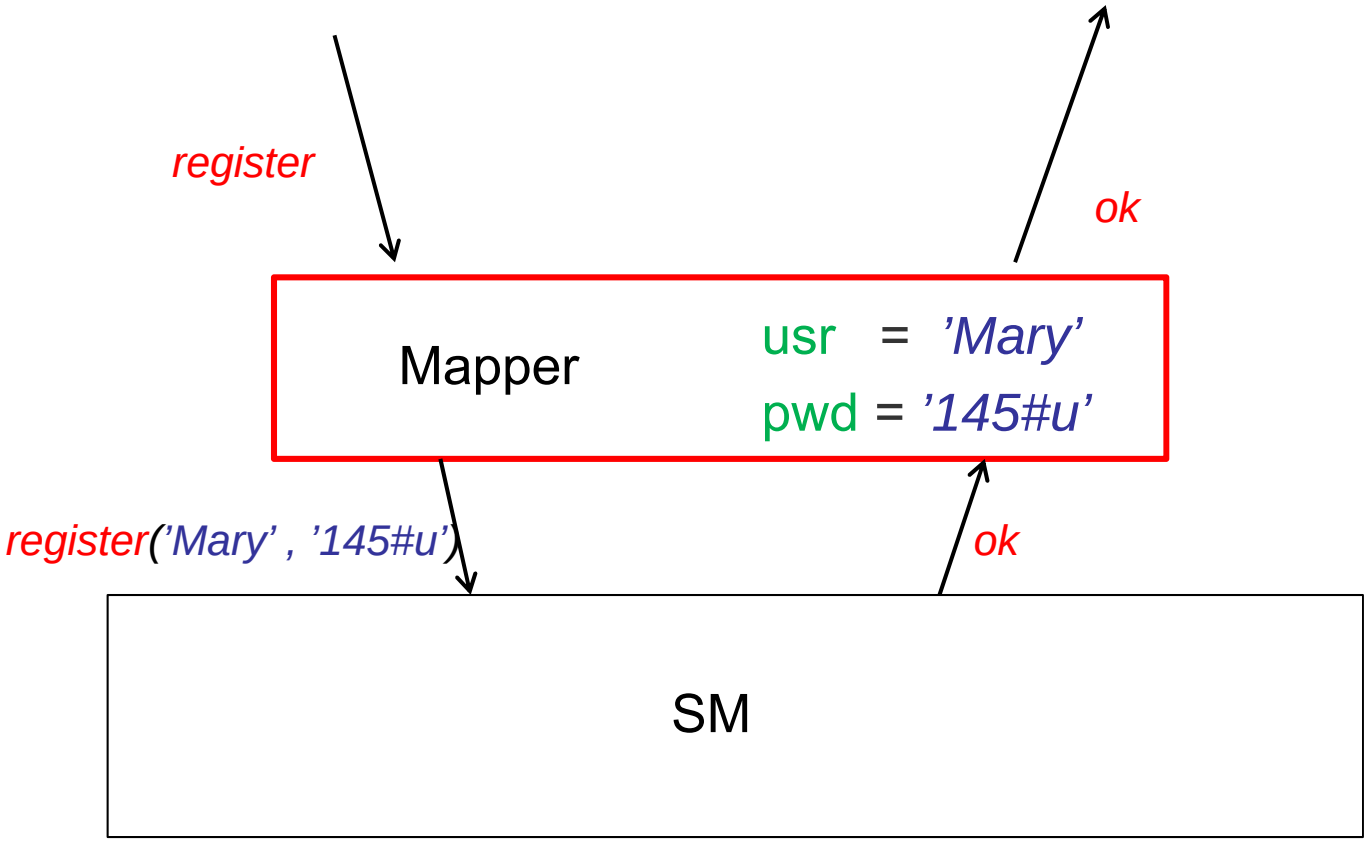
- In (white-box) verification, parameters are available in state variables
- In (black-box) learning, parameters must be remembered from history.



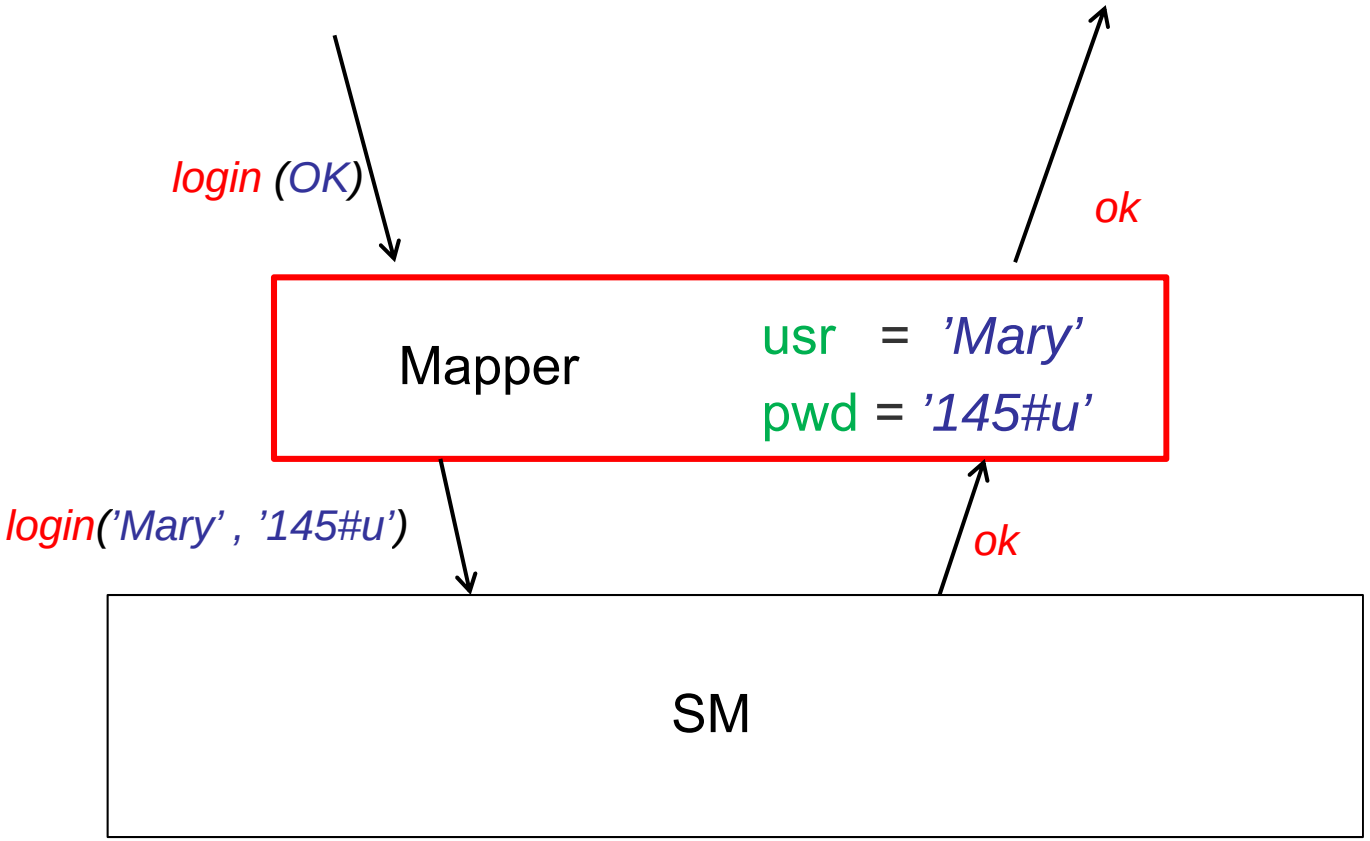
Organization of Abstraction



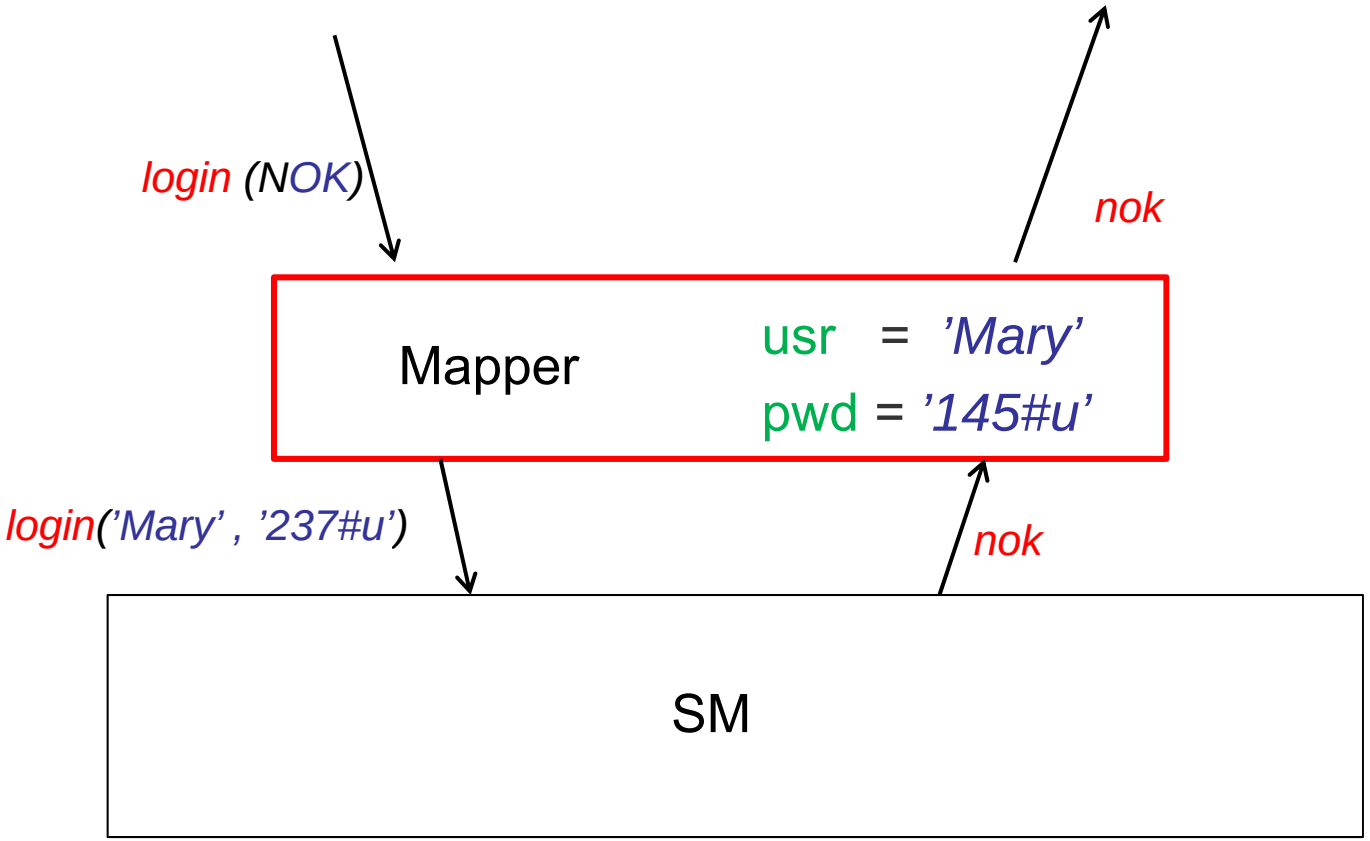
Organization of Abstraction



Organization of Abstraction



Organization of Abstraction



Abstraction: Formal definition

M

Σ_I , Σ_O symbols

Q , q_0 states , initial state

δ: $Q \times \Sigma_I \rightarrow Q$ transition function

λ: $Q \times \Sigma_I \rightarrow \Sigma_O$ output function

Mapper

Σ_I^A , Σ_O^A abstract symbols

R , r_0 states , initial state

δ^R: $R \times (\Sigma_I \cup \Sigma_O) \rightarrow R$ update

α_I: $R \times \Sigma_I \rightarrow \Sigma_I^A$ input abstraction

α_O: $R \times \Sigma_O \rightarrow \Sigma_O^A$ output abstraction

Combined Mealy Machine

Σ_I^A , Σ_O^A abstract symbols

$Q \times R$, $\langle q_0, r_0 \rangle$ states , initial state

In general Nondeterministic

Whenever

$$q \xrightarrow{a \,/\, b} q'$$

we have

$$\langle q , r \rangle \xrightarrow{\alpha_I (r , a) \,/\, \alpha_O (\delta^R(r , a) , b)} \langle q' , \delta^R(\delta^R(r , a) , b) \rangle$$

Application to XMPP

XMPP:

$\langle l_0$, usr=⊥ , pwd=⊥$\rangle$ —*register('Mary' , '145#u') / ok*→ $\langle l_0$, usr = *'Mary'* , pwd = *'145#u'* $\rangle$

Mapper:

Maps *register('Mary' , '145#u')* *to* *register*

Assigns usr := *'Mary'* ; pwd:= *'145#u'*

Combination:

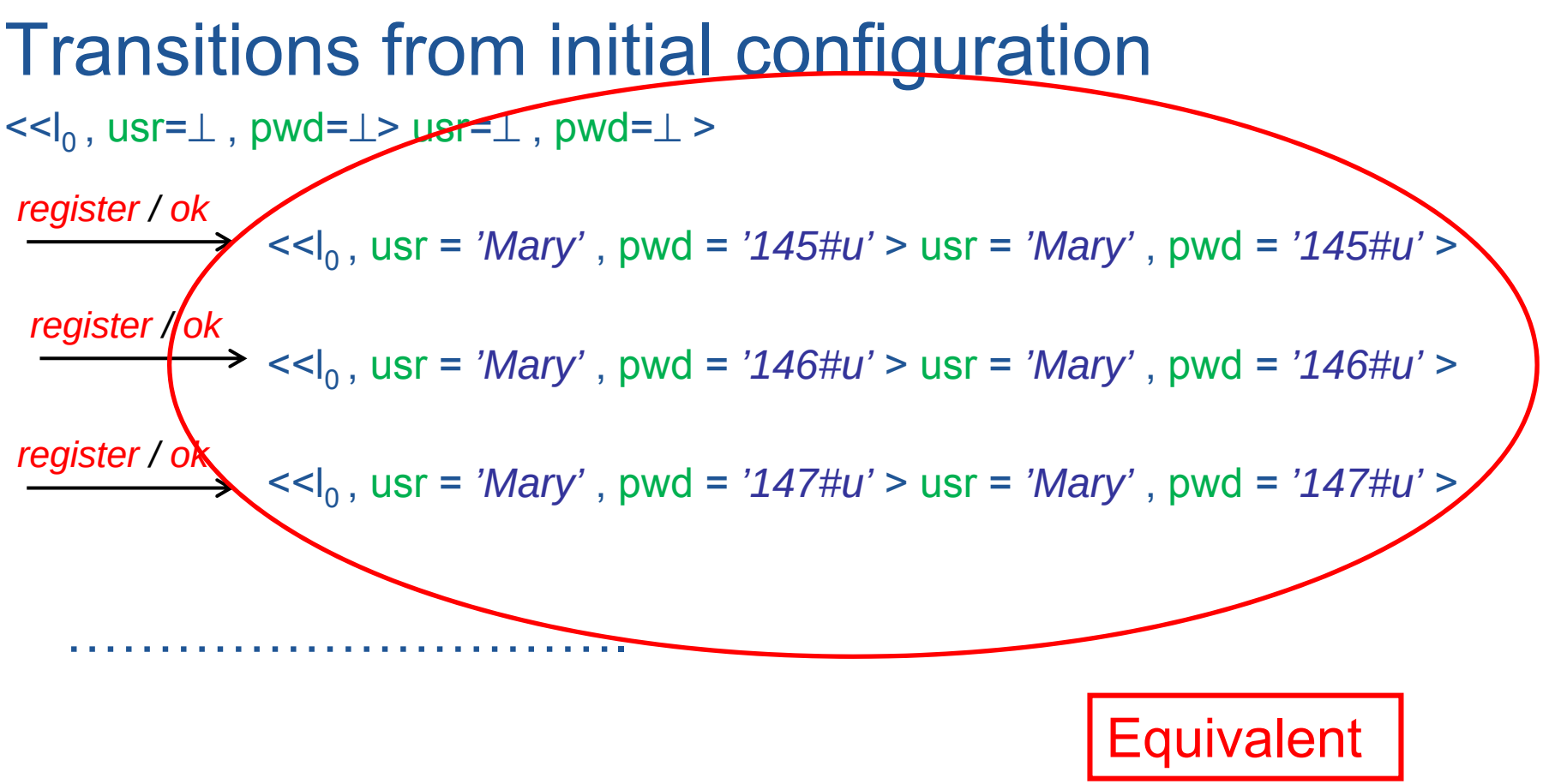
$\langle\langle l_0$, usr=⊥ , pwd=⊥$\rangle$ usr=⊥ , pwd=⊥ $\rangle$

—*register / ok*→

$\langle\langle l_0$, usr = *'Mary'* , pwd = *'145#u'* $\rangle$ usr = *'Mary'* , pwd = *'145#u'* $\rangle$

Potential Nondeterminism

Transitions from initial configuration



Equivalent

Result of Good Abstraction

- Combined Model is equivalent to a finite-state Mealy Machine M^A
- If so, we can obtain M by reversing effect of introducing Mapper

Combined Mealy Machine

Whenever

$$q \xrightarrow{a/\,b} q'$$

we have

$$\langle q , r\rangle \xrightarrow{\alpha_I(r,a)\,/\,\alpha_O(\delta^R(r,a),\,b)} \langle q' , \delta^R(\delta^R(r,a),b)\rangle$$

Result of Good Abstraction

Combined Mealy Machine

Whenever

$$q \xrightarrow{a \,/\, b} q'$$

we have

$$\langle q , r \rangle \xrightarrow{\alpha_I (r , a) \,/\, \alpha_O (\delta^R(r , a) , b)} \langle q' , \delta^R(\delta^R(r , a) , b) \rangle$$

Removing Effect of Mapper

Whenever

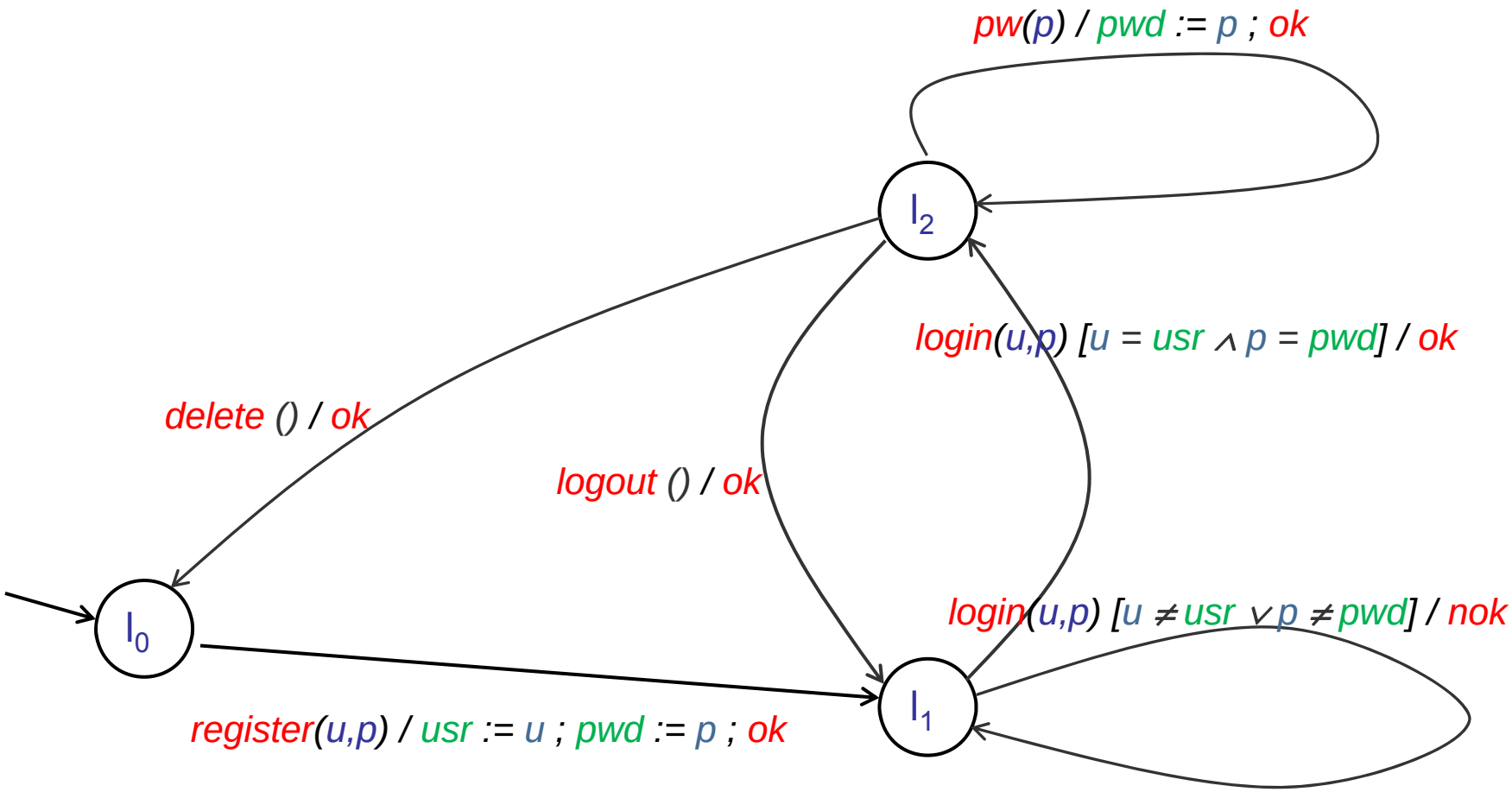
$$q^A \xrightarrow{\alpha_I (r , a) \,/\, \alpha_O (\delta^R(r , a) , b)} q^{A'}$$

Can be Nondeterministic

we have

$$\langle q^A, r \rangle \xrightarrow{a \,/\, b} \langle q^{A'}, \delta^R(\delta^R(r , a) , b) \rangle$$

Application to XMPP Example



Adaptation of Learning

Definition of Mapper

State Variables:

- usr , pwd
- Updated after *register*(*u*,*p*), *pw*(*p*)

Abstractions of symbols:

- *login*(*u*,*p*)

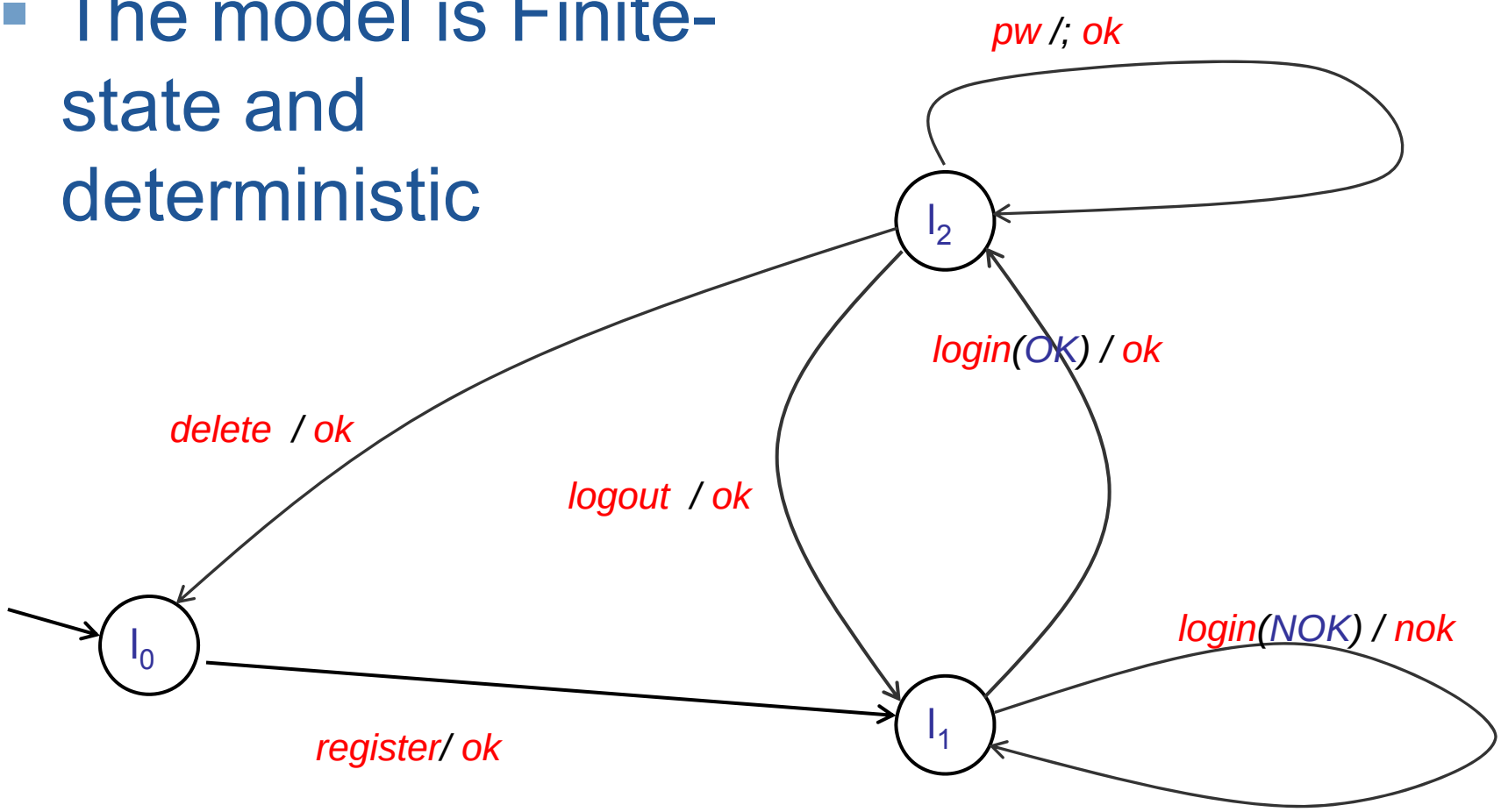
  mapped to *login*(*OK*) *or* *login*(*NOK*)

- *All other symbols:*

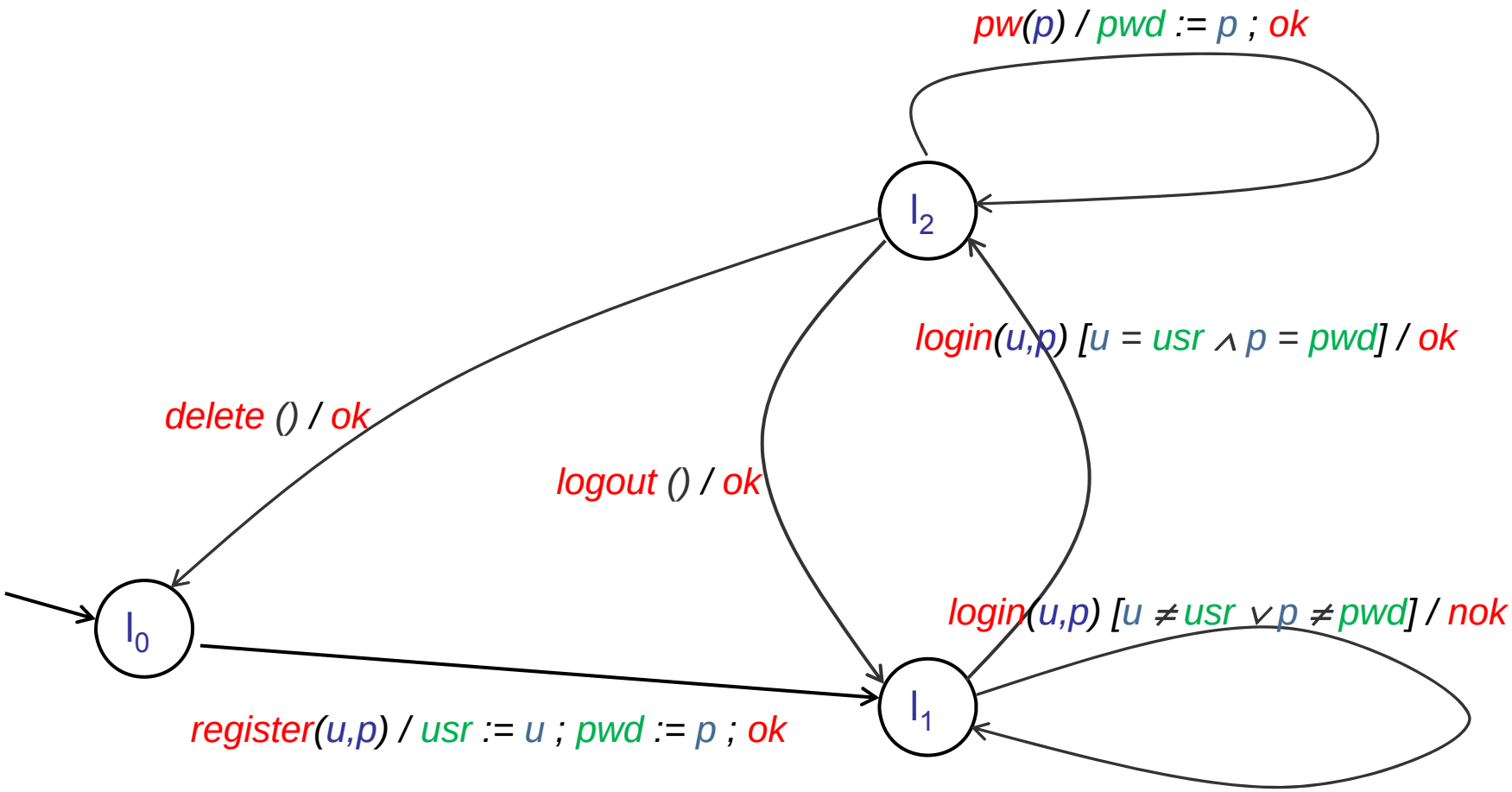
  mapped by suppressing parameters

Abstract Model

- The model is Finite-state and deterministic



Reverse effect of Abstraction



Systematic Construction of Abstractions

Simplifying assumption (for this presentation):

- Outputs do not have parameters

For SMMs with simple operations on data, abstractions can be constructed systematically

- Analogy: “region-graph-like” techniques for model checking infinite-state models
- Assume that we know
  - which parameters M stores from input symbols
  - signature of tests (assume no operations)

Designing a Mapper

We know which parameters M stores

-> define sufficient mapper variables $y_1, \ldots, y_j$

We know signature of tests

-> define *complete guard* as maximal consistent conjunction

-> Mapper maps each input symbol symbol $a\ (d_1, \ldots, d_n)$

to $a\ (p_1, \ldots, p_n)\ [\ g\ ]$

where g is appropriate complete guard over $y_1 \ldots\ y_j\ p_1 \ldots\ p_n$

Designing a Mapper

Assume:

any complete guard over $y_1, \ldots, y_j$ determines

for each input symbol $\alpha\ (p_1, \ldots, p_n)$

the complete guards over $y_1 \ldots y_j\ p_1 \ldots\ p_n$

any complete guard over $y_1, \ldots, y_j\ p_1 \ldots\ p_n$ determines

a unique complete guard over any subset

This assumptions make M^A finite-state and deterministic

Why it works

These assumptions make M^A finite-state and deterministic

because in state of combined model

$$\langle\langle l, x_1=d_1, \ldots x_k = d_k\rangle \;\; , y_1 = d_1', \ldots y_j = d_j'\rangle$$

- control location l
- complete guard g satisfied by $y_1 , \ldots , y_j$
- mapping from $y_1 , \ldots , y_j$ to $x_1 , \ldots , x_k$

uniquely determine future behavior

Why uniquely determined

Namely

in state of combined model

$$\langle\langle l, x_1=d_1, \ldots x_k = d_k\rangle \;, y_1 = d_1', \ldots y_j = d_j'\rangle$$

- An input $a(d_1, \ldots, d_n)$ is mapped to $a(p_1, \ldots, p_n) : g$
- Chosen symbolic transition of M is uniquely determined
- Location, guard and mapping in next state are uniquely determined

Example from XMPP

Abstractions of *pw(d)*

pw(p) [p = usr = pwd]

pw(p) [p = usr ≠ pwd]

pw(p) [p ≠ usr = pwd]

pw(p) [p = pwd ≠ usr]

pw(p) [p ≠ pwd ≠ usr ∧ p ≠ usr]

Inferring Information to Store

- Principle:
  - A parameter is memorable if it influences future behavior
- First case:
  - Parameter appears in output

register('Mary' , '145#u') / ok … askpwd('Mary') / reply('145#u')

- Second case:
  - Parameter influences decision

register('Mary' , '145#u') / ok … login('145#u') / ok

register('Mary' , '145#u') / ok … login('fresh') / nok

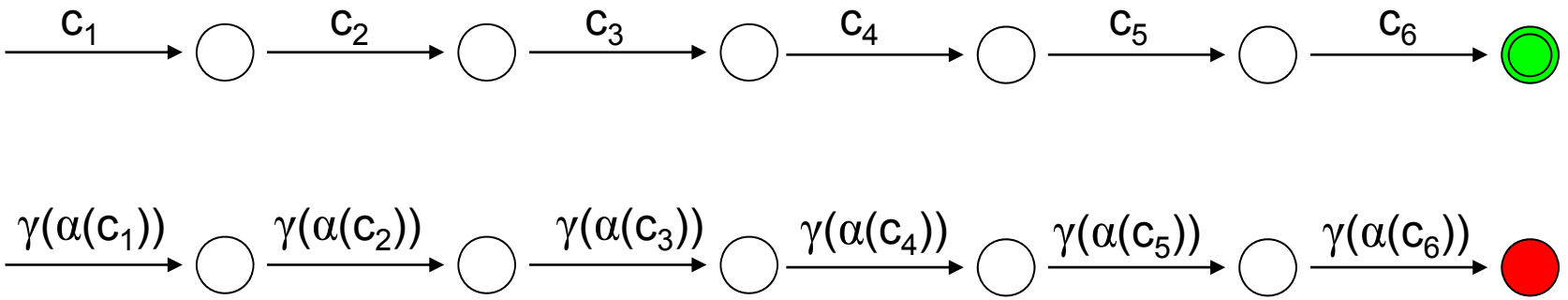
Inferring Guards

Alphabet Abstraction Refinement:

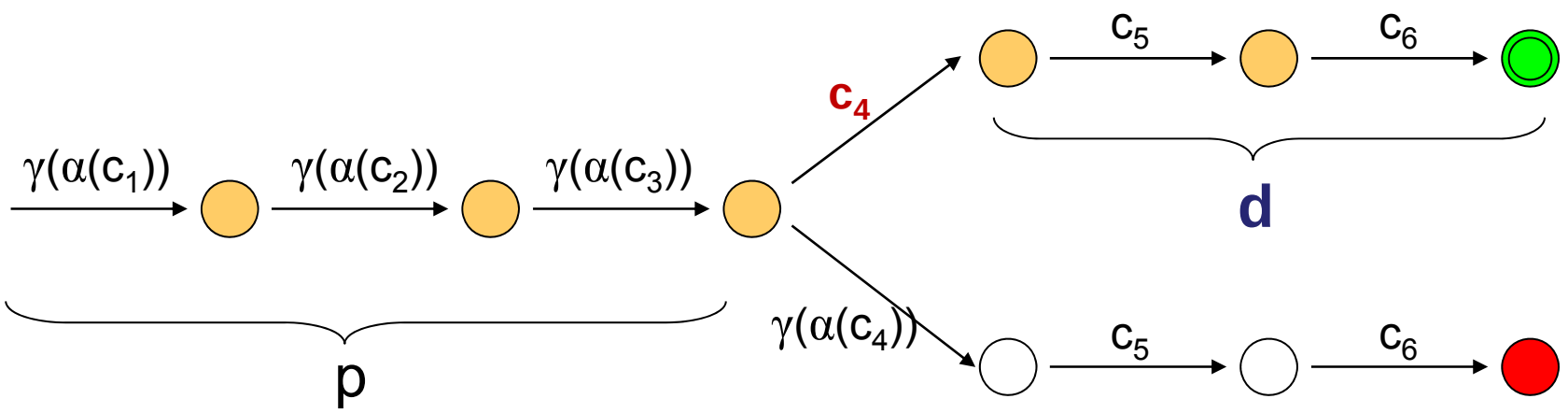
- Start without guards
- Add guards whenever nondeterminism appears.

register/ *ok* … *login*/ *ok* *nok*

Counter Examples and Witnesses



Counter Examples and Witnesses



Separating Pattern

p	c_4	d
state	representation	future

Abstraction Refinement

$$\alpha_{new}(\mathbf{x}) =_{df} \begin{cases} \alpha_{old}(\mathbf{x}) & \textbf{if } \alpha_{old}(\mathbf{x}) \mathbf{<>} \alpha_{old}(c) \\ a_c & \textbf{if } \alpha_{old}(\mathbf{x}) = \alpha_{old}(c) \text{ and} \\ & \quad \gamma(\alpha(p))\ \mathbf{x}\ d \in F \Leftrightarrow \gamma(\alpha(p))\ \mathbf{c}\ d \in F \\ \alpha_{old}(\mathbf{c}) & \textbf{else} \end{cases}$$

where a_c is a new abstract alphabet symbol.

$$\gamma_{new}(a) =_{df} \begin{cases} \gamma_{old}(a) & \textbf{if } a \neq \alpha_{old}(c) \\ \mathbf{c} & \textbf{if } a = a_c \\ \gamma_{old}(a) & \textbf{else} \end{cases}$$

Inferring Guards

Alphabet Abstraction Refinement:

- Start without guards
- Add guards whenever nondeterminism appears.

register/ *ok* … *login*/ *ok* / *nok*

register(*'Mary'* , *'145#u'*) / *ok* … *login*(*'Mary'* , *'145#u'*) / *ok*

register(*'Mary'* , *'145#u'*) / *ok* … *login*(*'Mary'*, *'fresh'*) / *nok*

Inferring Guards

Alphabet Abstraction Refinement:

- Start without guards
- Add guards whenever nondeterminism appears.

- Split *login* into
  - *login*(*u*,*p*) [*u* = *usr* ∧ *p* = *pwd*]
  - *login*(*u*,*p*) [*u* ≠ *usr* ∨ *p* ≠ *pwd*]

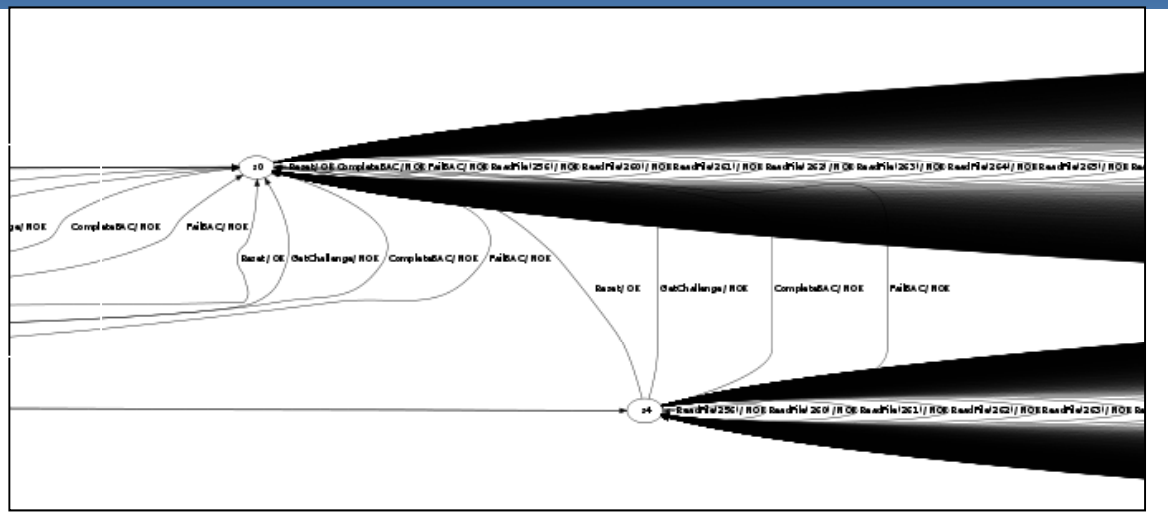
Applications of These Ideas

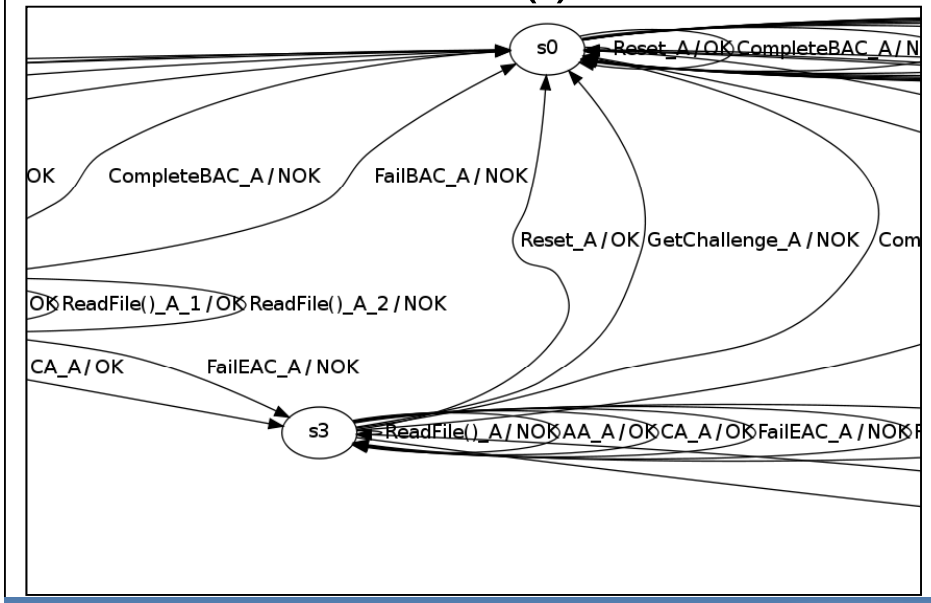
- Feasability studies on fragments of SIP and TCP
  - Implementations from ns-2 [Aarts, Jonsson, Uijen]
- Biometric Passport
  - w. manual abstraction [Aarts, Schmaltz, Vaandrager]
  - w. automated abstraction refinement [Howar, Steffen, Merten]

Passport [Howar et al]

Biometric Passport
[Aarts et. al, 2010]

262 Concrete symbols,
256 x readFile(i).



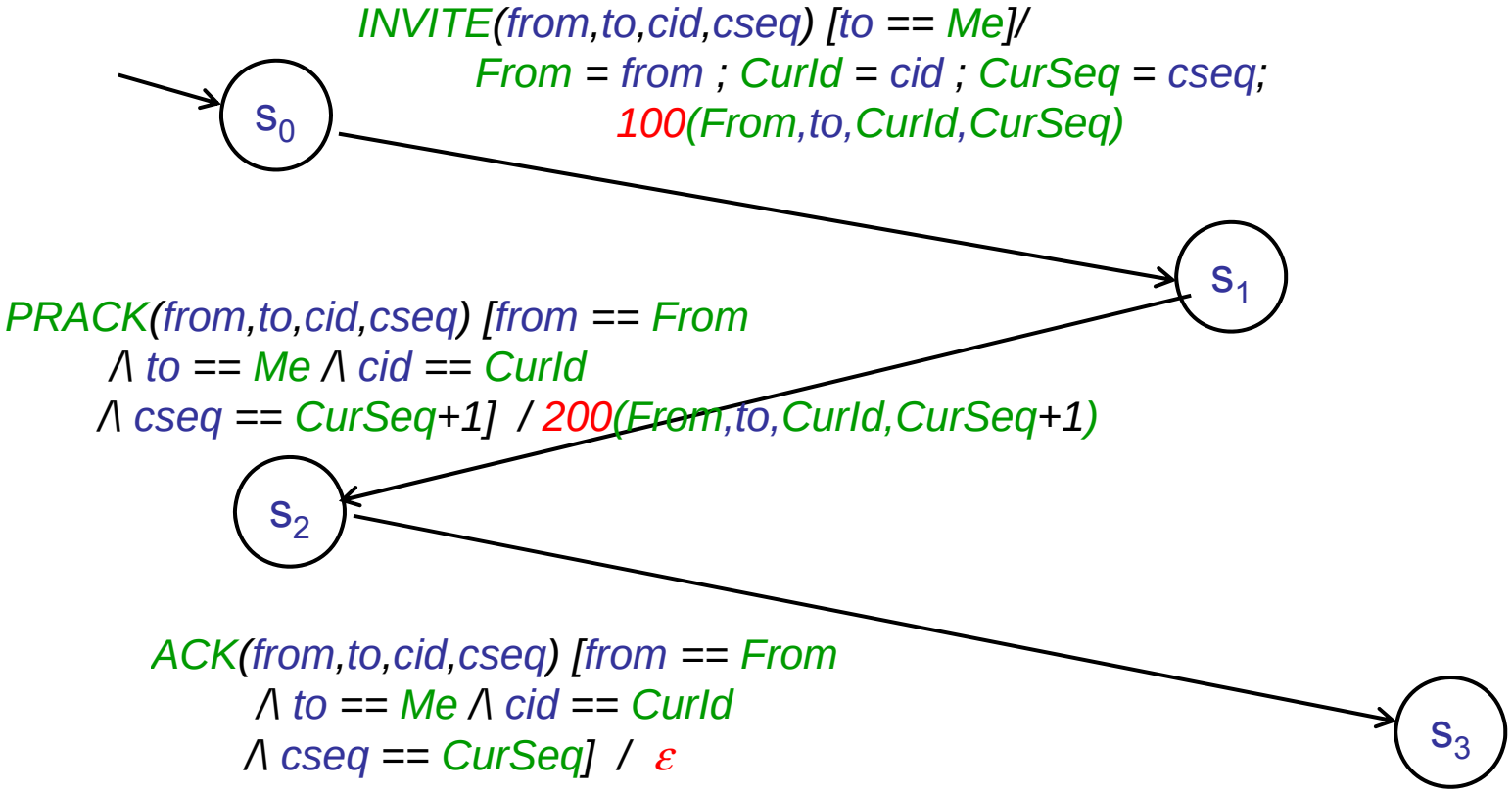


- **1** initial abstract symbols
- **8** alphabet refinements,
  to split readFile
- **9** final abstract symbols

*‚**read file(i)**‘ aggregated according to the required* ***Authentication***

part of SIP Server

Variables: *From, CurId, CurSeq*
Constants: *Me*



Resulting Model

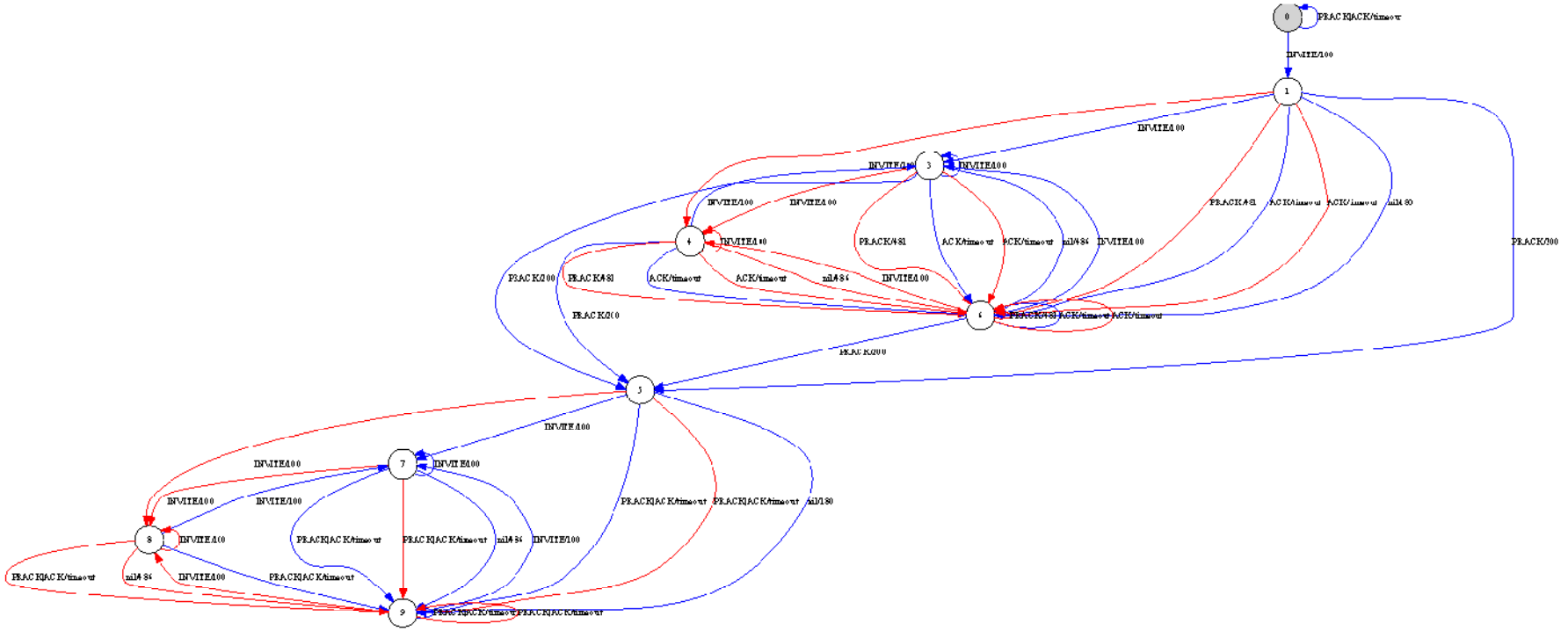
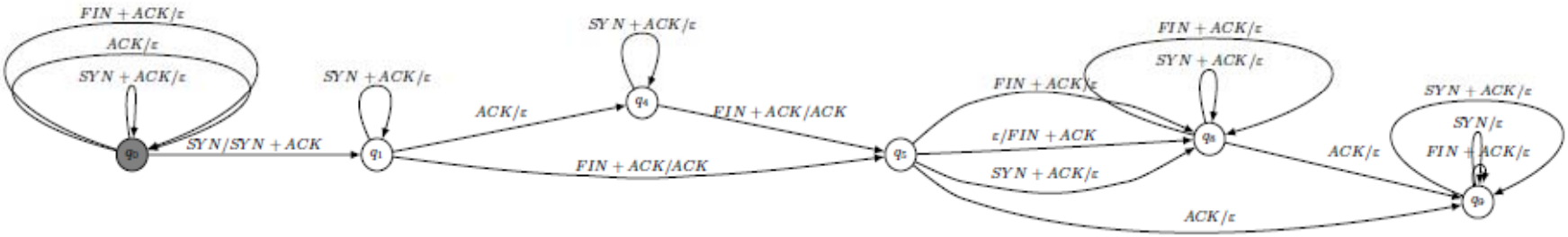


Fig. 3. Full SIP model

TCP

- Model of behavior of TCP in ns-2
- Only transitions with “accepted” values of input parameters are shown.
- Values of parameters not displayed



Conclusions and Future Work

- Data (and data dependencies) Important for Modeling Components and Interfaces
- Abstraction Techniques can be used to make L* Applicable
- In Black-Box Situation, the techniques are less robust
  - Abstraction needs to be carefully designed
- Construction of Abstractions need to combine
  - Storing of “right” information
  - Partitioning of input symbols using guards
- In Progress: Systematic Combination of these for particular signatures, also obtaining canonical models

General Challenges

- Nondeterministic Models/Loose Specifications
- Automated Test-Driver Synthesis