Learning of Automata Models
Extended with Data

Bengt Jonsson
Uppsala University
Acknowledgments

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

SeatBookerInterface
- venue[] = getVenues(user,pwd)
- seat[] = getSeats(user,pwd,venue)
- receipt = bookSeat(user,pwd,seat)

BookingServiceInterface
- session = openSession(user,pwd)
- venue[] = getVenues(session)
- seat[] = getSeats(session,venue)
- receipt = bookSeat(session,venue,seat)

Mediator
getVenues(session)
openSession(u,p)

getSeats(session,venue)
getSeats(u,p,venue)

bookSeat(session,venue,s)
bookSeat(u,p,s)

receipt
receipt

http://connect-forever.eu/
Correct combination
username - password
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

interface description
semantics

equivalence query

test execution

membership query

beginTransaction(...)
login(...)
checkTransaction(...)

210471321
Finite-State Mealy Machines

Finite State Machines w. input & output

- $\Sigma_I$: input symbols
- $\Sigma_O$: output symbols
- $Q$: states
- $q_0$: initial state
- $\delta: Q \times \Sigma_I \rightarrow Q$: transition function
- $\lambda: Q \times \Sigma_I \rightarrow \Sigma_O$: output function

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

- Often used for protocol modeling

Assumptions:
- Deterministic
- Completely specified
Basic Learning Setup

Same as in L*

### Membership query:
is w accepted or rejected?

#### Learner

#### Teacher

w is accepted/rejected

#### Oracle

Yes/counterexample v

### Equivalence query:
is H equivalent to A?
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 ($\geq 20$ 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)`
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**
  - openSession( u, p, s )
    - action
    - formal parameters
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.,

\[
= \\
\in \quad \text{SEAT} \times \text{SEATS} \\
\text{has\_passwd} \quad \text{STRING} \times \text{STRING}
\]
A Symbolic Mealy Machine consists of
- **I** Input Actions
- **O** Output Actions
- **L** Locations
- **l** Initial location
- **X** State variables (typed)
- **→** Symbolic Transitions

State Variables
- `cur_session` : SESSION
- `cur_seats` : SEATS
- `booked` : SEATS

Parameterized input symbol
- `getSeat(s, seat)`
  - `[s = cur_session ∧ seat ∈ cur_seats]`
  - guard
  - assignment
  - output expression

Parameterized input symbol
- `booked := booked ∪ seat`
- `bookedSeat(seat)`
- `l_0 → l_1`
State Variables

\( \text{cur\_session} : \text{SESSION} \)
\( \text{cur\_seats} : \text{SEATS} \)
\( \text{booked} : \text{SEATS} \)

(* Maybe complete the Example Here *)

Input Action

\( \text{getSeat}(s, \text{seat}) \)
\( [s = \text{cur\_session} \land \text{seat} \in \text{cur\_seats}] / \)
\( \text{booked} := \text{booked} \cup \text{seat} ; \)
\( \text{bookedSeat}(\text{seat}) \)

Formal parameters

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

\[\text{login}(u, p) \quad [u = \text{usr} \land p = \text{pwd}] / \text{ok}\]

\[\text{login}(u, p) \quad [u \neq \text{usr} \lor p \neq \text{pwd}] / \text{nok}\]

\[\text{register}(u, p) / \text{usr} := u ; \text{pwd} := p ; \text{ok}\]

\[\text{pw}(p) / \text{pwd} := p ; \text{ok}\]

\[\text{logout}() / \text{ok}\]

\[\text{delete}() / \text{ok}\]
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

\[ M \prec M^A\]
Abstraction in Verification

Problem:

\[ M \text{ satisfies } \varphi ? \]

Transformed into:

\[ M^A \text{ satisfies } \varphi^A ? \]
Define an abstraction \( \alpha \)
- \( \alpha \) 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 \( \alpha \) on \( M^A \)
- i.e., \( M = \alpha^{-1}(M^A) \)

If \( M^A \) is not adequate, refine \( \alpha \)
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.

\[
\text{login}(u, p) [u = \text{usr} \land p = \text{pwd}] / \text{ok}
\]

\[
\text{login}(u, p) [u \neq \text{usr} \lor p \neq \text{pwd}] / \text{nok}
\]

\[
\text{register}(u, p) / \text{usr} := u ; \text{pwd} := p ; \text{ok}
\]

\[
\text{logout()} / \text{ok}
\]

\[
\text{delete()} / \text{ok}
\]

\[
pw(p) / \text{pwd} := p ; \text{ok}
\]
Inadequate Model

- Abstract Model
- Problem: nondeterminism
Fixing Nondeterminism-Problem

Diagram:
- Initial state $I_0$
- Transition: $I_0 \xrightarrow{\text{register / ok}} I_1$
- Transition: $I_1 \xrightarrow{\text{login / ok}}$
- Transition: $I_1 \xrightarrow{\text{login / nok}}$
Fixing Nondeterminism-Problem

Abstraction depends on parameters and previous history
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

Abstract
input symbols

Concrete
input symbols

Mapper

Abstract
output symbols

Concrete
output symbols

local
variables

SM
register('Mary', '145#u') ok

SM

Mapper

usr = 'Mary'
pwd = '145#u'

register

ok
Organization of Abstraction

**Mapper**
- **usr** = 'Mary'
- **pwd** = '145#u'

**SM**

- `login('Mary', '145#u')`
- `login (OK)`
- `ok`
- `ok`
Organization of Abstraction

Mapper

usr = 'Mary'
pwd = '145#u'

SM

\text{login} (NOK)
\text{nok}

\text{login}('Mary', '237#u')
\text{nok}
Abstraction: Formal definition

**M**
- $\Sigma_i$, $\Sigma_o$ symbols
- $Q$, $q_0$ states, initial state
- $\delta: Q \times \Sigma_i \rightarrow Q$ transition function
- $\lambda: Q \times \Sigma_i \rightarrow \Sigma_o$ output function

**Mapper**
- $\Sigma_i^A$, $\Sigma_o^A$ abstract symbols
- $R$, $r_0$ states, initial state
- $\delta^R: R \times (\Sigma_i \cup \Sigma_o) \rightarrow R$ update
- $\alpha_i: R \times \Sigma_i \rightarrow \Sigma_i^A$ input abstraction
- $\alpha_o: R \times \Sigma_o \rightarrow \Sigma_o^A$ output abstraction

**Combined Mealy Machine**
- $\Sigma_i^A$, $\Sigma_o^A$ abstract symbols
- $Q \times R$, $<q_0, r_0>$ states, initial state

Whenever
- $q \xrightarrow{a/b} q'$
- $\alpha_i(r, a) / \alpha_o(\delta^R(r, a), b)$
- $<q, r> \xrightarrow{a/b} <q', \delta^R(\delta^R(r, a), b)>$

**In general Nondeterministic**
Application to XMPP

XMPP:

\[ \text{register('Mary', '145#u') / ok} \]

\[ \langle l_0, \text{usr}=\perp, \text{pwd}=\perp \rangle \rightarrow \langle l_0, \text{usr} = 'Mary', \text{pwd} = '145#u' \rangle \]

Mapper:

Maps \[ \text{register('Mary', '145#u')} \] to \[ \text{register} \]
Assigns \[ \text{usr} := 'Mary'; \text{pwd} := '145#u' \]

Combination:

\[ \langle \langle l_0, \text{usr}=\perp, \text{pwd}=\perp \rangle \text{usr}=\perp, \text{pwd}=\perp \rangle \rightarrow \langle \langle l_0, \text{usr} = 'Mary', \text{pwd} = '145#u' \rangle \text{usr} = 'Mary', \text{pwd} = '145#u' \rangle \]
Potential Nondeterminism

Transitions from initial configuration

$$\langle l_0, \text{usr}=\bot, \text{pwd}=\bot \rangle \rightarrow \langle \text{usr}=\bot, \text{pwd}=\bot \rangle$$

register / ok

$$\langle l_0, \text{usr} = 'Mary', \text{pwd} = '145#u' \rangle \rightarrow \langle \text{usr} = 'Mary', \text{pwd} = '145#u' \rangle$$

register / ok

$$\langle l_0, \text{usr} = 'Mary', \text{pwd} = '146#u' \rangle \rightarrow \langle \text{usr} = 'Mary', \text{pwd} = '146#u' \rangle$$

register / ok

$$\langle l_0, \text{usr} = 'Mary', \text{pwd} = '147#u' \rangle \rightarrow \langle \text{usr} = 'Mary', \text{pwd} = '147#u' \rangle$$

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

$$\alpha_I (r, a) / \alpha_O (\delta^R(r, a), b)$$

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

Combined Mealy Machine
Whenever we have
\[ \frac{a}{b} \]
we have
\[ \alpha_1(r,a) / \alpha_\delta(\delta^R(r,a),b) \]
\[ <q,r> \xrightarrow{\frac{a}{b}} <q',\delta^R(\delta^R(r,a),b)> \]

Removing Effect of Mapper
Whenever we have
\[ \frac{a}{b} \]
we have
\[ \alpha_1(r,a) / \alpha_\delta(\delta^R(r,a),b) \]
\[ <q^A,r> \xrightarrow{\frac{a}{b}} <q^A',\delta^R(\delta^R(r,a),b)> \]

Can be Nondeterministic
Application to XMPP Example

- **Register**: 
  - `register(u,p)` / `usr := u ; pwd := p ; ok`

- **Login**: 
  - `login(u,p)` / `u = usr ∧ p = pwd` / `ok`
  - `login(u,p)` / `u ≠ usr ∨ p ≠ pwd` / `nok`

- **Logout**: 
  - `logout()` / `ok`

- **Delete**: 
  - `delete()` / `ok`

- **Transition States**:
  - **I₀**
  - **I₁**
  - **I₂**
Definition of Mapper

State Variables:
- `usr`, `pwd`
- Updated after `register(u,p), pw(p)`

Abstractions of symbols:
- `login(u,p)`
  \[\text{mapped to } login(OK) \text{ or } login(NOK)\]
- All other symbols:
  \[\text{mapped by suppressing parameters}\]
Abstract Model

- The model is Finite-state and deterministic

Diagram:

- Initial state: $I_0$
  - Transition: $delete / ok$
  - Transition: $register / ok$

- State $I_1$
  - Transition: $logout / ok$
  - Transition: $login(OK) / ok$

- State $I_2$
  - Transition: $pw /; ok$
  - Transition: $login(NOK) / nok$
Reverse effect of Abstraction

- \( \text{pw}(p) / \text{pwd} := p ; \text{ok} \)
- \( \text{login}(u,p) [u = \text{usr} \land p = \text{pwd}] / \text{ok} \)
- \( \text{logout}() / \text{ok} \)
- \( \text{delete}() / \text{ok} \)
- \( \text{register}(u,p) / \text{usr} := u ; \text{pwd} := p ; \text{ok} \)
- \( \text{login}(u,p) [u \neq \text{usr} \lor p \neq \text{pwd}] / \text{nok} \)
Systematic Construction of Abstractions

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 $\mathcal{M}$ 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 \textit{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 $y_1 \ldots y_j p_1 \ldots p_n$
Assume:

any complete guard over $y_1, \ldots, y_j$ determines
for each input symbol $a(p_1, \ldots, p_n)$
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 $\mathcal{M}^A$ finite-state and deterministic
Why it works

These assumptions make $M^A$ finite-state and deterministic because in state of combined model

$<$<$l, x_1=d_1, \ldots x_k = d_k>$ , $y_1 = d_1', \ldots y_j = d_j'>$

- 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
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 $pw(d)$

$pw(p) \ [p = \text{usr} = \text{pwd}]$

$pw(p) \ [p = \text{usr} \neq \text{pwd}]$

$pw(p) \ [p \neq \text{usr} = \text{pwd}]$

$pw(p) \ [p = \text{pwd} \neq \text{usr}]$

$pw(p) \ [p \neq \text{pwd} \neq \text{usr} \land p \neq \text{usr}]$
Inferring Information to Store

- **Principle:**
  - A parameter is **memorable** if it influences future behavior

- **First case:**
  - Parameter appears in output
    
    \[ \text{register('Mary', '145#u') / ok ... askpwd('Mary') / reply('145#u')} \]

- **Second case:**
  - Parameter influences decision
    
    \[ \text{register('Mary', '145#u') / ok ... login('145#u') / ok} \]
    \[ \text{register('Mary', '145#u') / ok ... login('fresh') / nok} \]
Inferring Guards

Alphabet Abstraction Refinement:

- Start without guards
- Add guards whenever nondeterminism appears.

```
register/ ok          ...          login/
k
```

```
ok
```
Counter Examples and Witnesses
Counter Examples and Witnesses

\[ \gamma(\alpha(c_1)) \rightarrow \gamma(\alpha(c_2)) \rightarrow \gamma(\alpha(c_3)) \rightarrow \gamma(\alpha(c_4)) \rightarrow c_5 \rightarrow c_6 \rightarrow d \]

Separating Pattern

- p: state
- c_4: representation
- d: future
Abstraction Refinement

\[ \alpha_{\text{new}}(x) = \begin{cases} 
\alpha_{\text{old}}(x) & \text{if } \alpha_{\text{old}}(x) \leftrightarrow \alpha_{\text{old}}(c) \\
\text{a}_c & \text{if } \alpha_{\text{old}}(x) = \alpha_{\text{old}}(c) \text{ and } \gamma(\alpha(p)) x \ d \in F \iff \gamma(\alpha(p)) c \ d \in F \\
\alpha_{\text{old}}(c) & \text{else}
\end{cases} \]

where \( \text{a}_c \) is a new abstract alphabet symbol.

\[ \gamma_{\text{new}}(a) = \begin{cases} 
\gamma_{\text{old}}(a) & \text{if } a \neq \alpha_{\text{old}}(c) \\
\text{c} & \text{if } a = \text{a}_c \\
\gamma_{\text{old}}(a) & \text{else}
\end{cases} \]
Inferring Guards

Alphabet Abstraction Refinement:

- Start without guards
- Add guards whenever nondeterminism appears.

\[
\text{register/ ok} \quad \ldots \quad \text{login/ ok} \\
\text{register(‘Mary’, ’145#u’) / ok} \quad \ldots \quad \text{login(‘Mary’, ’145#u’) / ok}
\]

\[
\text{register(‘Mary’, ’145#u’) / ok} \quad \ldots \quad \text{login(‘Mary’, ’fresh’) / nok}
\]
Inferring Guards

Alphabet Abstraction Refinement:

- Start without guards
- Add guards whenever nondeterminism appears.

\[ \text{register/ ok} \quad \cdots \quad \text{login/} \]

- Split \textit{login} into
  
  - \( \text{login}(u,p) \ [u = \text{usr} \land p = \text{pwd}] \)
  
  - \( \text{login}(u,p) \ [u \neq \text{usr} \lor p \neq \text{pwd}] \)
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]
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
Variables: $\text{From, CurId, CurSeq}$
Constants: $\text{Me}$

$INVITE(from,to,cid,cseq) [to == Me]/$

$From = from ; CurId = cid ; CurSeq = cseq; 100(From,to,CurId,CurSeq)$

$PRACK(from,to,cid,cseq) [from == From$

$\land to == Me \land cid == CurId$

$\land cseq == CurSeq+1] / 200(From,to,CurId,CurSeq+1)$

$ACK(from,to,cid,cseq) [from == From$

$\land to == Me \land cid == CurId$

$\land cseq == CurSeq] / \epsilon$
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