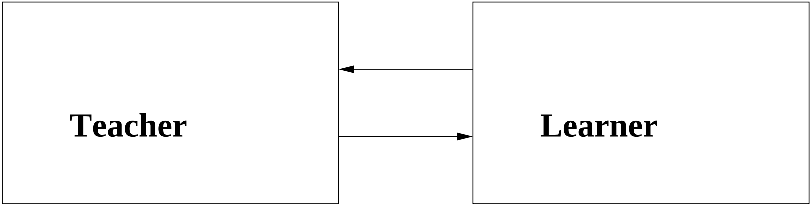
Learning I/O Automata

Fides Aarts and Frits Vaandrager

ICIS, Radboud Universiteit Nijmegen

CONCUR 2010, Paris

Background: Grammatical Inference



Angluin's L^* algorithm for active learning deterministic FSMs:
Learner poses membership and equivalence queries

Learning Reactive Systems

Angluin's algorithm has been extended to Mealy machines by Niese and implemented in the LearnLib tool.

Learning Reactive Systems

Angluin's algorithm has been extended to Mealy machines by Niese and implemented in the LearnLib tool.

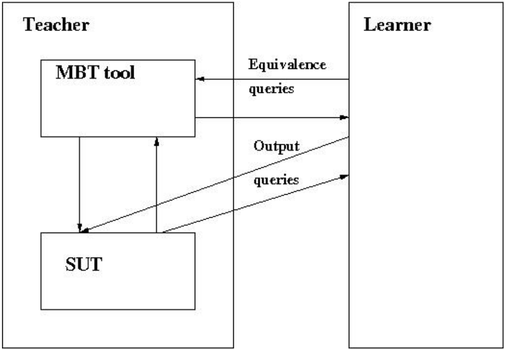
- Membership queries are replaced by output queries: which output is generated in response to a sequence of inputs?

Learning Reactive Systems

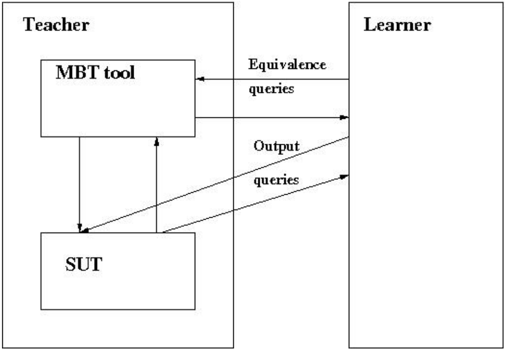
Angluin's algorithm has been extended to Mealy machines by Niese and implemented in the LearnLib tool.

- Membership queries are replaced by output queries: which output is generated in response to a sequence of inputs?
- Equivalence queries are approximated by test sequences generated using algorithms for model based testing.

Learning Reactive Systems (cnt)



Learning Reactive Systems (cnt)



Active learning may provide models of reactive systems in situations where we have no access to the code (black box) and not even a specification, e.g. to learn reference implementations.

Challenges

Challenges

- Handle data and large state spaces

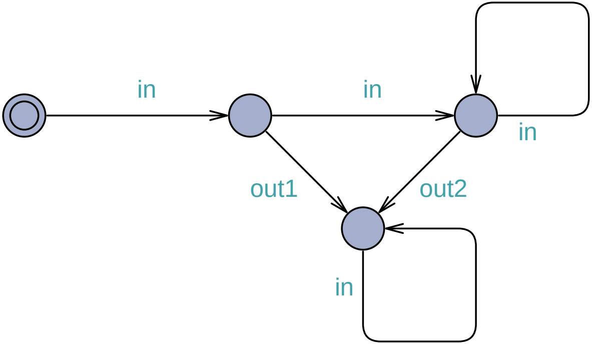
Challenges

- Handle data and large state spaces
- Handle nondeterministic systems

Challenges

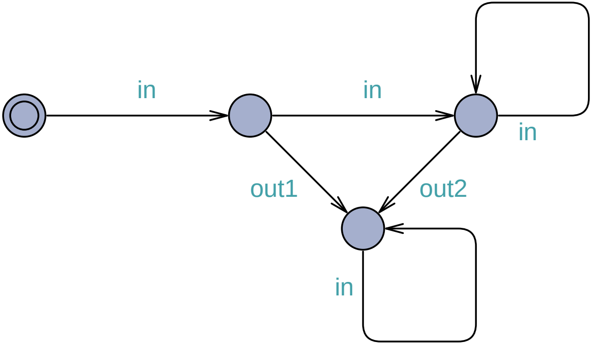
- Handle data and large state spaces
- Handle nondeterministic systems
- *For many applications, requirement that each input corresponds to exactly one output is overly restrictive*

I/O Automata



Proposed independently by Lynch & Tuttle and Jonsson (1987).

The I/O Behavior of I/O Automata?



What output is generated in response to input sequence *in in* ?
Common answer: either *out1* δ or *out2* δ (here δ = quiescence)

Determinism

Engineers want deterministic systems: for every stream of inputs that is offered to the system, the stream of outputs that is generated should be unique. Deterministic systems are predictable.

A deterministic FSM is behavior deterministic: for each string the output (accepted or not accepted) is unique.

A deterministic Mealy machine is also behavior deterministic.

Determinism

Engineers want deterministic systems: for every stream of inputs that is offered to the system, the stream of outputs that is generated should be unique. Deterministic systems are predictable.

A deterministic FSM is behavior deterministic: for each string the output (accepted or not accepted) is unique.

A deterministic Mealy machine is also behavior deterministic.

How about I/O automata?

Determinism

An IOA is input deterministic if

$$q \xrightarrow{i} q_1 \wedge q \xrightarrow{i} q_2 \wedge i \in I \quad \Rightarrow \quad q_1 = q_2$$

Determinism

An IOA is input deterministic if

$$q \xrightarrow{i} q_1 \wedge q \xrightarrow{i} q_2 \wedge i \in I \quad \Rightarrow \quad q_1 = q_2$$

It is output deterministic if

$$q \xrightarrow{o} q_1 \wedge q \xrightarrow{o} q_2 \wedge o \in O \quad \Rightarrow \quad q_1 = q_2$$

Determinism

An IOA is input deterministic if

$$q \xrightarrow{i} q_1 \wedge q \xrightarrow{i} q_2 \wedge i \in I \quad \Rightarrow \quad q_1 = q_2$$

It is output deterministic if

$$q \xrightarrow{o} q_1 \wedge q \xrightarrow{o} q_2 \wedge o \in O \quad \Rightarrow \quad q_1 = q_2$$

It is output determined if:

$$q \xrightarrow{o_1} q_1 \wedge q \xrightarrow{o_2} q_2 \wedge \{o_1, o_2\} \subseteq O \quad \Rightarrow \quad o_1 = o_2 \wedge q_1 = q_2$$

Determinism

An IOA is input deterministic if

$$q \xrightarrow{i} q_1 \wedge q \xrightarrow{i} q_2 \wedge i \in I \quad \Rightarrow \quad q_1 = q_2$$

It is output deterministic if

$$q \xrightarrow{o} q_1 \wedge q \xrightarrow{o} q_2 \wedge o \in O \quad \Rightarrow \quad q_1 = q_2$$

It is output determined if:

$$q \xrightarrow{o_1} q_1 \wedge q \xrightarrow{o_2} q_2 \wedge \{o_1, o_2\} \subseteq O \quad \Rightarrow \quad o_1 = o_2 \wedge q_1 = q_2$$

For usual notion of I/O behavior, input deterministic, output determined IOAs are not behavior deterministic!

Avoiding Race Conditions

Nondeterminism due to race conditions between IUT and tester.

Common assumption: there is T such that IUT never produces output if more than T time has elapsed since previous event. Allows tester to observe quiescence.

Dual assumption: there is t such that IUT never produces output if less than t time has elapsed since previous event. Allows tester/learner to avoid race conditions (if it is fast enough)!

I/O Behavior of IOAs

By action Δ we give IOA opportunity to produce next output.
Sequences $e \in (I_\Delta)^*$ are called environment sequences.
By action δ, IOA indicates that it reached quiescent state:
$q \stackrel{\delta}{\rightarrow} q$ iff q enables no output.
Define $\stackrel{e/u}{\Rightarrow}$ to be least relation s.t.

$$
\begin{array}{rcl}
 & & q \stackrel{\epsilon/\epsilon}{\Rightarrow} q \\
q \stackrel{e/u}{\Rightarrow} q' \wedge q' \stackrel{i}{\rightarrow} q'' \wedge i \in I & \Rightarrow & q \stackrel{e\,i/u\,i}{\Rightarrow} q'' \\
q \stackrel{e/u}{\Rightarrow} q' \wedge q' \stackrel{o}{\rightarrow} q'' \wedge o \in O_\delta & \Rightarrow & q \stackrel{e\,\Delta/u\,o}{\Rightarrow} q''
\end{array}
$$

For $\mathcal{A}$ an IOA, $obs_{\mathcal{A}}(e) = \{u \in (I \cup O_\delta)^* \mid \exists q \in Q : q^0 \stackrel{e/u}{\Rightarrow} q\}$.

Observation Equivalence and ioco

Fact (Output deterministic IOAs are behavior deterministic)

If IOA $\mathcal{A}$ is input deterministic and output determined then $obs_{\mathcal{A}}(e)$ is a singleton set, for all $e \in (I_\Delta)^*$.

Observation Equivalence and ioco

Fact (Output deterministic IOAs are behavior deterministic)

If IOA $\mathcal{A}$ is input deterministic and output determined then $obs_{\mathcal{A}}(e)$ is a singleton set, for all $e \in (I_\Delta)^*$.

Theorem

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be IOAs. Then $\mathcal{A}_1$ **ioco** $\mathcal{A}_2$ iff, for all $e \in (I_\Delta)^*$, $obs_{\mathcal{A}_1}(e) \subseteq obs_{\mathcal{A}_2}(e)$.

Interface Automata (De Alfaro & Henzinger)

An interface automaton (IA) is a tuple $\mathcal{A} = (I, O, Q, q^0, \rightarrow)$, where I, O and Q are finite, nonempty sets of input actions, output actions, and states, respectively, with I and O disjoint,

- $q^0 \in Q$ the initial state, and
- $\rightarrow \subseteq Q \times (I \cup O) \times Q$ the transition relation.

An I/O automaton (IOA) is an interface automaton in which each input action is enabled in each state.

XY-simulations

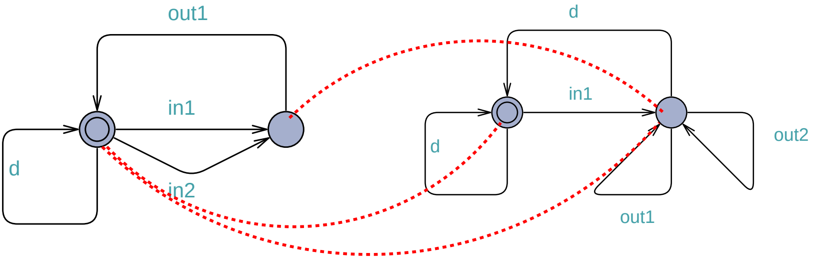
Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be IAs with actions A, and let $X, Y \subseteq A$.
Relation $R \subseteq Q_1 \times Q_2$ is an XY-simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$ iff, for all $(q, r) \in R$ and $a \in A$

$$q \xrightarrow{a}_1 q' \ \wedge \ a \in X \quad \Rightarrow \quad \exists r' \in Q_2 : r \xrightarrow{a}_2 r' \ \wedge \ (q', r') \in R,$$
$$r \xrightarrow{a}_2 r' \ \wedge \ a \in Y \quad \Rightarrow \quad \exists q' \in Q_1 : q \xrightarrow{a}_1 q' \ \wedge \ (q', r') \in R.$$

$\mathcal{A}_1 \leq_{XY} \mathcal{A}_2$ if there exists an XY-simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$ that contains (q_1^0, q_2^0).

Alternating Simulations

OI-simulations are known as alternating simulations:



Alternating Simulations and IOCO

The δ-extension of an IA $\mathcal{A}$ is obtained by adding a loop with output action δ to all quiescent states.
We write $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$ iff $\mathcal{A}_1^\delta \leq_{OI} \mathcal{A}_2^\delta$.

Alternating Simulations and IOCO

The δ-extension of an IA $\mathcal{A}$ is obtained by adding a loop with output action δ to all quiescent states.
We write $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$ iff $\mathcal{A}_1^\delta \leq_{OI} \mathcal{A}_2^\delta$.

Theorem

Suppose $\mathcal{A}_1$ is input deterministic and $\mathcal{A}_2$ is output deterministic. Then $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$ implies $\mathcal{A}_1$ **ioco** $\mathcal{A}_2$.

Alternating Simulations and IOCO

The δ-extension of an IA $\mathcal{A}$ is obtained by adding a loop with output action δ to all quiescent states.
We write $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$ iff $\mathcal{A}_1^\delta \leq_{OI} \mathcal{A}_2^\delta$.

Theorem

Suppose $\mathcal{A}_1$ is input deterministic and $\mathcal{A}_2$ is output deterministic. Then $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$ implies $\mathcal{A}_1$ **ioco** $\mathcal{A}_2$.

Theorem

Suppose $\mathcal{A}_1$ is input enabled and $\mathcal{A}_2$ is deterministic. Then $\mathcal{A}_1$ **ioco** $\mathcal{A}_2$ implies $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$.

Learning Purpose

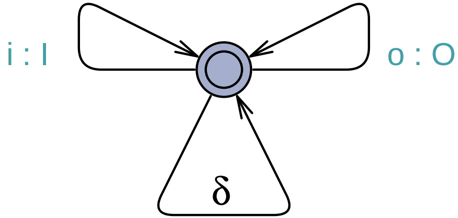
In practice, we often cannot (or do not want to) learn full IOA. We may not be able to offer inputs at the right times, or IOA may be too big, or we are only interested to learn a part.

A learning purpose is a deterministic IA $\mathcal{P}$ that specifies which part of an IOA $\mathcal{A}$ we want to learn. Our goal is to find IA $\mathcal{H}$ such that

$$\mathcal{A}^\delta \ \leq_{OI} \ \mathcal{H}^\delta \ \leq_{AI} \ \mathcal{P}$$

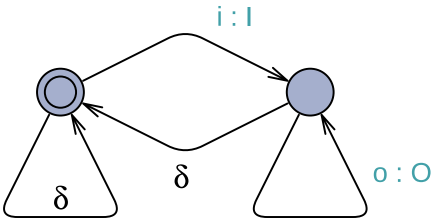
Learning Purpose (cnt)

A trivial learning purpose:



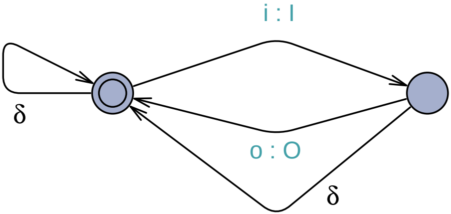
Learning Purpose (cnt)

After an input one has to wait until the system gets into a quiescent state before offering the next input:



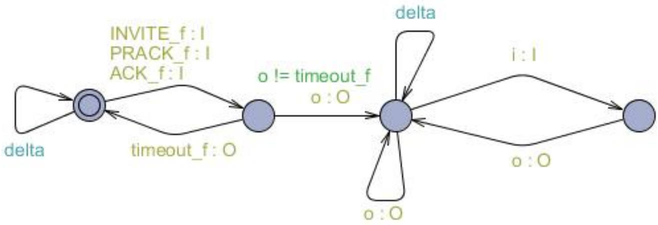
Learning Purpose (cnt)

Each input is followed by at most one output:



Learning Purpose (cnt)

Initially only certain inputs are allowed and two consecutive inputs are not allowed:



Uniqueness!

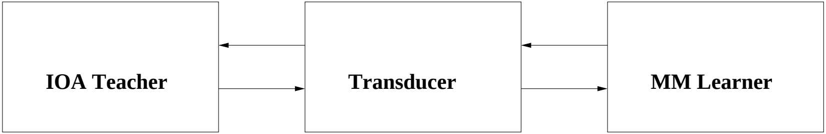
Key Lemma

Suppose $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ are IAs, $\mathcal{A}_1$ is active and input deterministic, $\mathcal{A}_2$ is output determined, $\mathcal{A}_3$ is output deterministic, and $\mathcal{A}_1 \leq_{OI} \mathcal{A}_3 \leq_{AI} \mathcal{A}_2$.
Then $\mathcal{A}_3$ is uniquely determined up to bisimulation equivalence.

Learning IOAs

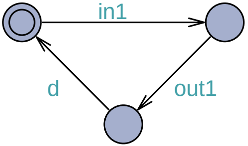
Learner	Teacher
knows deterministic learning purpose $\mathcal{P}$	knows deterministic, output determined IOA $\mathcal{A}$
maintains state of $\mathcal{P}$	maintains state of $\mathcal{A}$
may present $i \in I \cup \{\Delta\}$ to teacher	replies with $o \in O \cup \{\delta\}$
may do reset and jump to initial state of $\mathcal{P}$	upon reset jumps to initial state of $\mathcal{A}$
may pose preorder query $\mathcal{H}$, for $\mathcal{H}^\delta \leq_{AI} \mathcal{P}$	replies with *yes* if $\mathcal{A}$ **ioco** $\mathcal{H}$ or else with *no*+counterexample

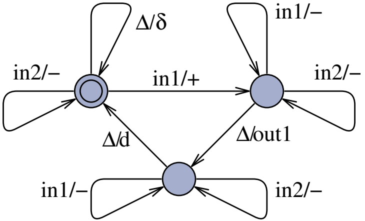
Set Up for Learning IOAs



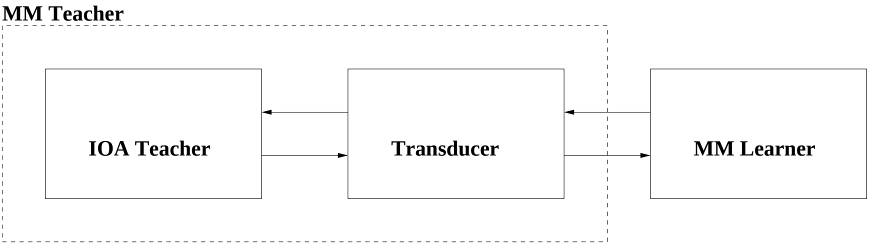
Teacher is teaching IOA $\mathcal{A} = (I, O, Q, q^0, \rightarrow)$
Transducer handles learning purpose $\mathcal{P}$ and translates between IOAs and MMs
Learner learns MM with inputs I_Δ and outputs $O \cup \{\delta, +, -\}$.

From IAs to MMs



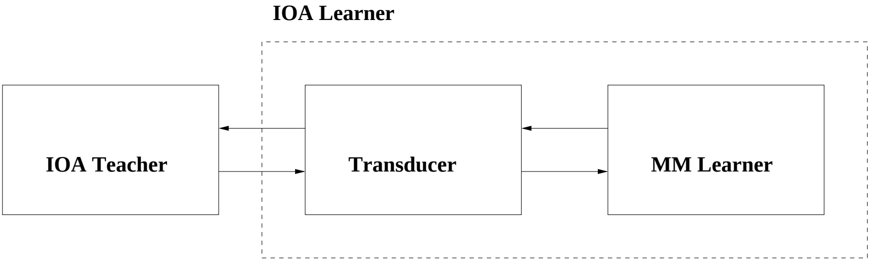


Lemma



IOA teacher and transducer together act as teacher for MM $T(\mathrm{AS}(\mathcal{A}^{\delta}, \mathcal{P}))$.

Theorem



MM learner and transducer together act as learner of IOA $\rho(\mathsf{AS}(\mathcal{A}^{\delta}, \mathcal{P}))$

Applications

We have implemented our approach and applied it to learn parts of

Applications

We have implemented our approach and applied it to learn parts of

- TCP

  F. Aarts, *MSc Thesis*, 2009

Applications

We have implemented our approach and applied it to learn parts of

- TCP

  F. Aarts, *MSc Thesis*, 2009

- SIP

  F. Aarts, B. Jonsson & J. Uijen, *Generating Models of Infinite-State Communication Protocols using Regular Inference with Abstraction*, ICTSS 2010

Applications

We have implemented our approach and applied it to learn parts of

- TCP

  F. Aarts, *MSc Thesis*, 2009

- SIP

  F. Aarts, B. Jonsson & J. Uijen, *Generating Models of Infinite-State Communication Protocols using Regular Inference with Abstraction*, ICTSS 2010

- Biometric Passport

  F. Aarts, *Inference and Abstraction of the Biometric Passport*, YR-CONCUR 2010

Conclusions

Conclusions

- Learning of reactive systems generalized to setting in which inputs and outputs don't have to alternate

Conclusions

- Learning of reactive systems generalized to setting in which inputs and outputs don't have to alternate
- Interesting links between three widely used models: interface automata, ioco theory and Mealy machines.

Conclusions

- Learning of reactive systems generalized to setting in which inputs and outputs don't have to alternate
- Interesting links between three widely used models: interface automata, ioco theory and Mealy machines.
- Using concept of transducer, learning of IOAs has been reduced to problem of learning MMs.

Conclusions

- Learning of reactive systems generalized to setting in which inputs and outputs don't have to alternate
- Interesting links between three widely used models: interface automata, ioco theory and Mealy machines.
- Using concept of transducer, learning of IOAs has been reduced to problem of learning MMs.
- Initial experiments indicate work may become quite useful in practice.

Future Work

Future Work

1. Improve treatment of data (using abstraction techniques of Aarts, Jonsson & Uijen)

Future Work

1. Improve treatment of data (using abstraction techniques of Aarts, Jonsson & Uijen)
2. Extend to nondeterministic automata

Future Work

1. Improve treatment of data (using abstraction techniques of Aarts, Jonsson & Uijen)
2. Extend to nondeterministic automata
3. Extend to probabilistic and timed automata

Future Work

1. Improve treatment of data (using abstraction techniques of Aarts, Jonsson & Uijen)
2. Extend to nondeterministic automata
3. Extend to probabilistic and timed automata
4. Extend to other learning algoritms

Future Work

1. Improve treatment of data (using abstraction techniques of Aarts, Jonsson & Uijen)
2. Extend to nondeterministic automata
3. Extend to probabilistic and timed automata
4. Extend to other learning algoritms
5. Integrate verification, testing and learning

Future Work

1. Improve treatment of data (using abstraction techniques of Aarts, Jonsson & Uijen)
2. Extend to nondeterministic automata
3. Extend to probabilistic and timed automata
4. Extend to other learning algoritms
5. Integrate verification, testing and learning

Many nice challenges for CONCUR community!