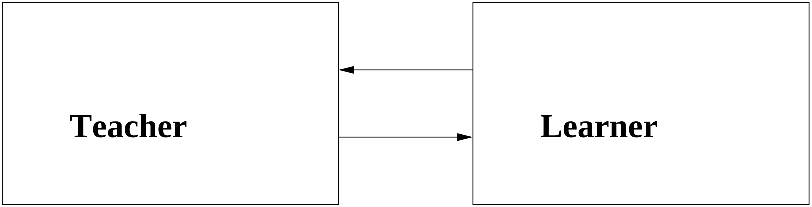
Learning Nondeterministic Register Automata Using Mappers

Fides Aarts Paul Fiterău-Broştean Harco Kuppens
Frits Vaandrager

ICIS, Radboud Universiteit Nijmegen

MBSD Colloquium, 19 February 2015

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.

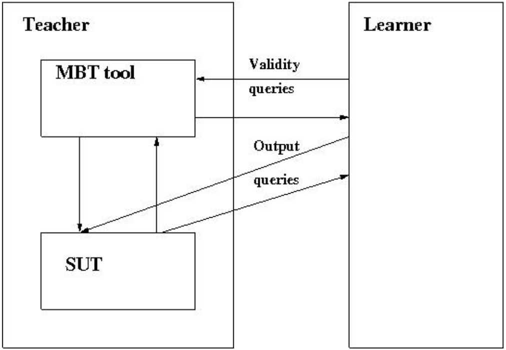
- Membership queries are replaced by output queries: which output is generated in response to a sequence of inputs?

Learning reactive systems

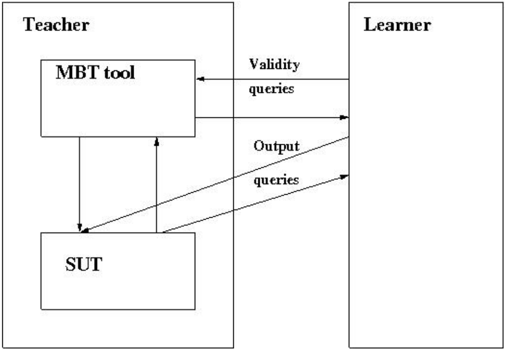
Angluin's algorithm has been extended to Mealy machines by Niese and implemented in the LearnLib tool.

- 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

Research challenges:

Challenges

Research challenges:

- Handle data and large state spaces

Challenges

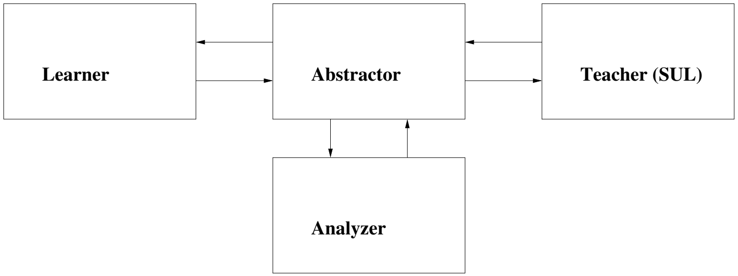
Research challenges:

- Handle data and large state spaces
- Handle nondeterministic systems

Use mappers to handle data

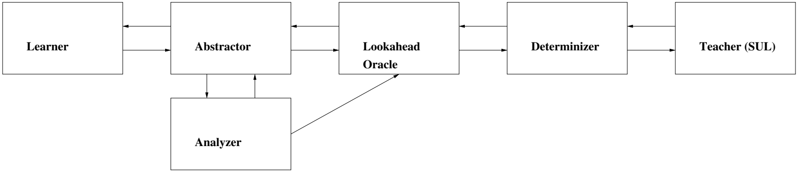
ICTSS'10, FMSD 2015

Use CEGAR to construct mappers automatically



FM'12: automatic construction of abstractions for SUL that can be modeled by (restricted) register automaton; implemented in Tomte

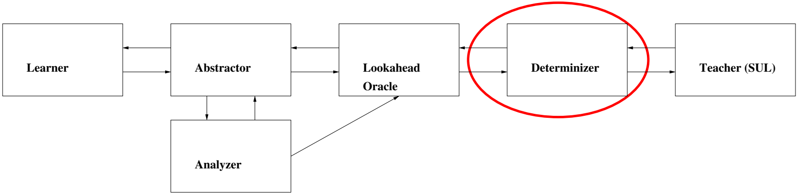
Learning with four "action heroes"



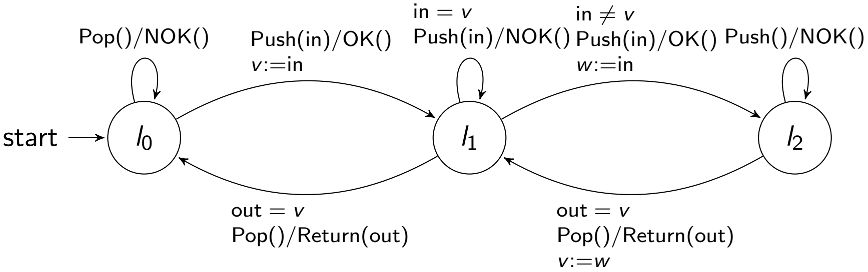
PhD Thesis Fides and ISOLA'14: deterministic register automata
This work: class of nondeterministic register automata


TOMTE
EATURING FIDES AARTS AS LOOKAHEAD ORACLE, PAUL FITERAU AS DETERMINIZER,
HARCO KUPPENS AS ABSTRACTOR AND FRITS VAANDRAGER AS ANALYZER

Today: the determinizer



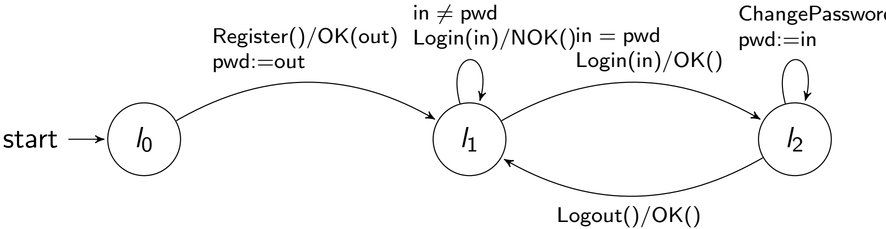
Register automaton for FIFO-set



Each action carries a single data parameter (in or out)

Example trace (data values that do not matter omitted):
Push(12) OK() Push(12) NOK() Push(4) OK Pop() Return(12) Pop() Retu

Nondeterministic register automaton for login procedure



Example traces:
Register() OK(2345) Login(543) NOK() Login(2345) OK()
Register() OK(9345)

Formulas

We assume an infinite set $\mathcal{V}$ of variables.

An atomic formula is an expression $x = y$ or $x \neq y$, with $x, y \in \mathcal{V}$.

A formula φ is a conjunction of atomic formulas. Write $\Phi(X)$ for set of formulas with variables taken from X.

Register automata

A register automaton (RA) is a tuple $\mathcal{R} = \langle I, O, V, L, l_0, \Gamma \rangle$, where

- I and O are finite sets of input symbols and output symbols, respectively
- $V \subseteq \mathcal{V}$ is a finite set of state variables; we assume special variables in and out not contained in V; we write $V_{i/o} = V \cup \{\text{in}, \text{out}\}$
- L is finite set of locations
- $l_0 \in L$ is initial location
- $\Gamma \subseteq L \times I \times \Phi(V_{i/o}) \times (V \to V_{i/o}) \times O \times L$ is a finite set of transitions. We require that out does not occur negatively in guards, that is, not in subformulae $x \neq y$.

Mealy machines

A Mealy machine is a tuple $\mathcal{M} = \langle I, O, Q, q^0, \rightarrow \rangle$, where

- I and O are nonempty sets of input and output actions, respectively,
- Q is a set of states,
- $q^0 \in Q$ is the initial state, and
- $\rightarrow \subseteq Q \times I \times O \times Q$ is the transition relation.

Semantics of register automata

Let $\mathcal{R} = \langle I, O, V, L, l_0, \Gamma \rangle$ be a register automaton.
The semantics of $\mathcal{R}$, denoted $[\![\mathcal{R}]\!]$, is the Mealy machine $\langle I \times (\mathbb{Z} \setminus \{0\}), O \times (\mathbb{Z} \setminus \{0\}), L \times \mathrm{Val}(V), (l_0, \xi_0), \rightarrow \rangle$, where $\xi_0(v) = 0$ for all $v \in V$, and relation $\rightarrow$ is given by the rule

$$\frac{\langle l, i, g, \varrho, o, l' \rangle \in \Gamma \qquad \iota = \xi \cup \{(\mathrm{in}, d), (\mathrm{out}, e)\} \quad \iota \models g \quad \xi' = \iota \circ \varrho}{(l, \xi) \xrightarrow{i(d)/o(e)} (l', \xi')}$$

Behavior determinism

A partial run of Mealy machine $\mathcal{M}$ is a finite sequence

$$\alpha \quad = \quad q_0 \; i_0 \; o_0 \; q_1 \; i_1 \; o_1 \; q_2 \cdots i_{n-1} \; o_{n-1} \; q_n,$$

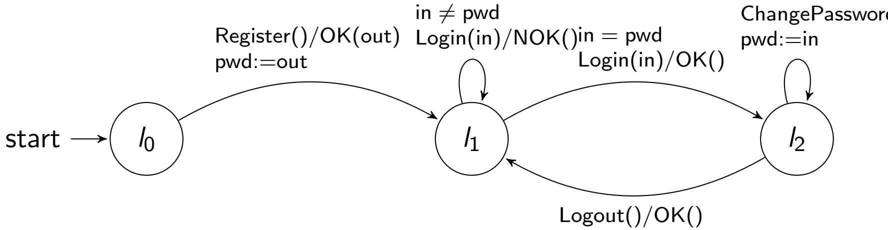
beginning and ending with a state, s.t. for all $j < n$, $q_j \xrightarrow{i_j/o_j} q_{j+1}$.
A run of $\mathcal{M}$ is a partial run that starts with initial state q^0.
A trace of $\mathcal{M}$ is a finite sequence $\beta = i_0 \; o_0 \; i_1 \; o_1 \cdots i_{n-1} \; o_{n-1}$ obtained by erasing all states from a run of $\mathcal{M}$.
A set S of traces is behavior deterministic if, for all traces $\beta \; i \; o \in S$ and $\beta \; i \; o' \in S$, we have $o = o'$.
$\mathcal{M}$ is behavior deterministic if its set of traces is so.

Input determinism versus behavior determinism

Register automaton $\mathcal{R}$ is behavior deterministic if $[\![\mathcal{R}]\!]$ is behavior deterministic.

$\mathcal{R}$ is input deterministic if for each state and for each input action at most one transition may fire.

In our work we only consider input deterministic register automata.



Automorphisms

A zero respecting automorphism is a bijection $h : \mathbb{Z} \to \mathbb{Z}$ such that $h(0) = 0$.

Zero respecting automorphisms can be lifted to the valuations, states, actions, runs and traces of a register automaton.

Neat traces

Consider a trace β of register automaton $\mathcal{R}$:

$$\beta \quad = \quad i_0(d_0)\ o_0(e_0)\ i_1(d_1)\ o_1(e_1)\ \cdots\ i_{n-1}(d_{n-1})\ o_{n-1}(e_{n-1})$$

We say that β has neat inputs if each input value is either equal to a previous value, or equal to the largest preceding value plus one. Similarly, we say that β has neat outputs if each output value is either equal to a previous value, or equal to the smallest preceding value minus one.
A trace is neat if it has neat inputs and neat outputs, and a run is neat if its trace is neat.

We only need to study neat traces!

Proposition. For every run α there exists a zero respecting automorphism h such that $h(\alpha)$ is neat.

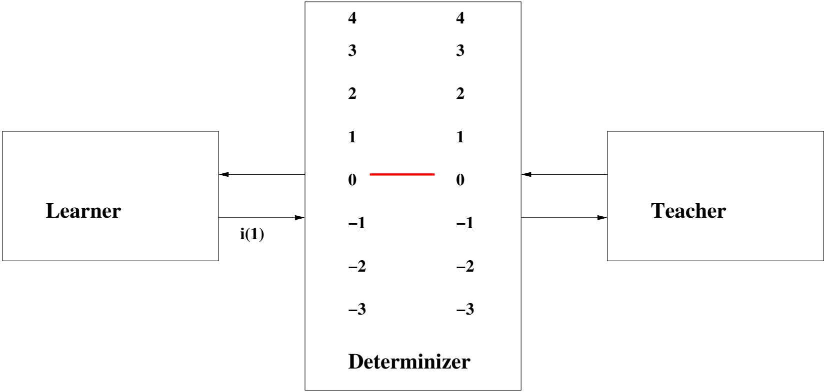
Problem: the learner may generate neat inputs only, but we cannot force the SUL to only generate neat outputs.

Mappers

A mapper for a set of inputs I and a set of outputs O is a deterministic Mealy machine $\mathcal{A} = \langle I \cup O, X \cup Y, R, r_0, \delta, \lambda \rangle$, where

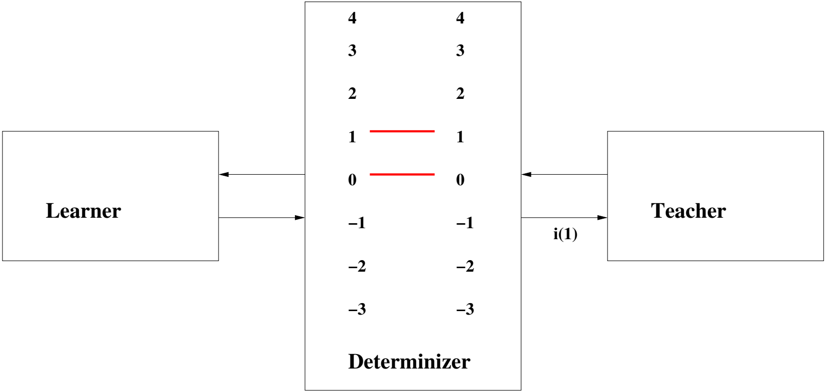
- I and O are disjoint sets of concrete input and output symbols,
- X and Y are finite sets of abstract input and output symbols, and
- $\lambda : R \times (I \cup O) \to (X \cup Y)$, referred to as the abstraction function, respects inputs and outputs, that is, for all $a \in I \cup O$ and $r \in R$, $a \in I \Leftrightarrow \lambda(r, a) \in X$.

Mapper for determinizer



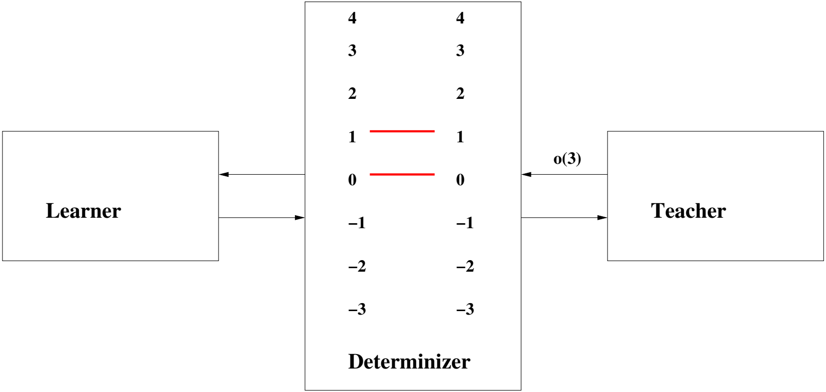
Idea: mapper stores part of automorphism constructed thus far

Mapper for determinizer



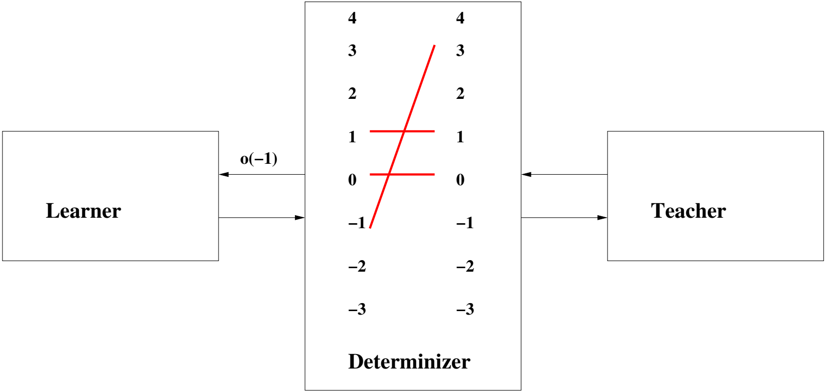
Idea: mapper stores part of automorphism constructed thus far

Mapper for determinizer



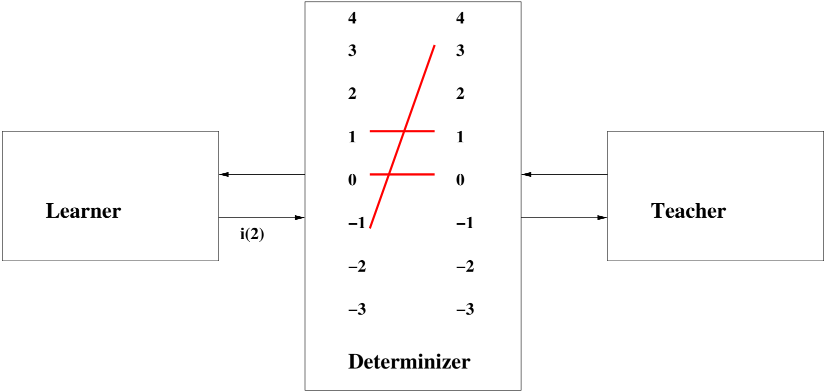
Idea: mapper stores part of automorphism constructed thus far

Mapper for determinizer



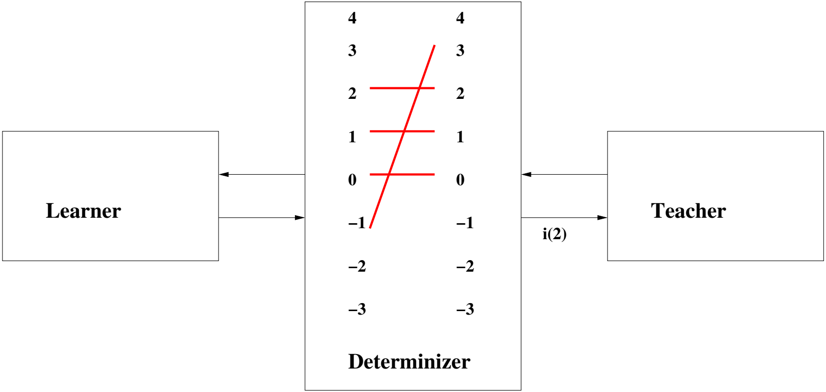
Idea: mapper stores part of automorphism constructed thus far

Mapper for determinizer



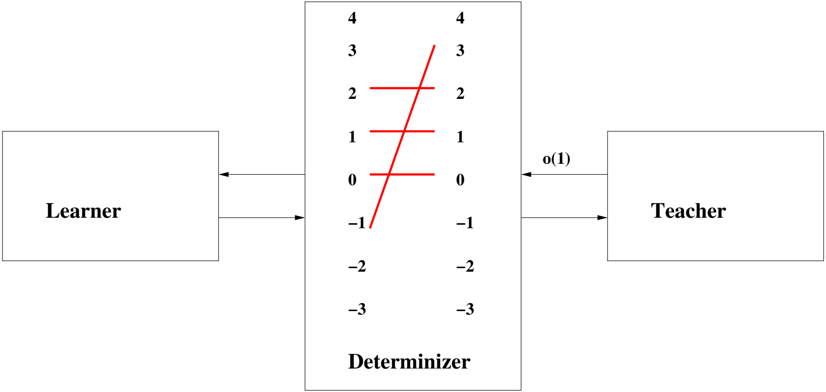
Idea: mapper stores part of automorphism constructed thus far

Mapper for determinizer



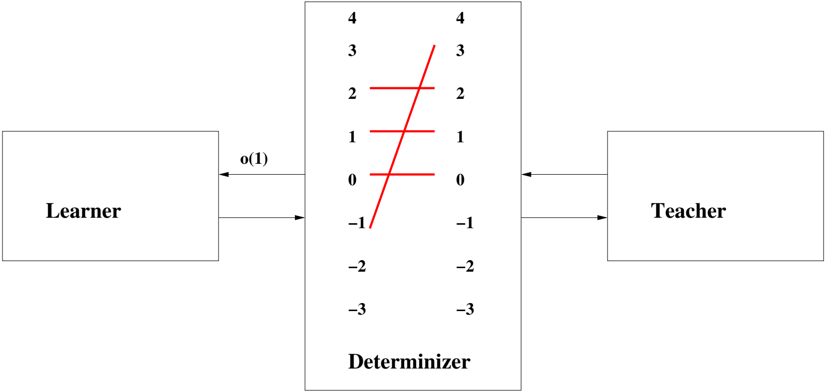
Idea: mapper stores part of automorphism constructed thus far

Mapper for determinizer



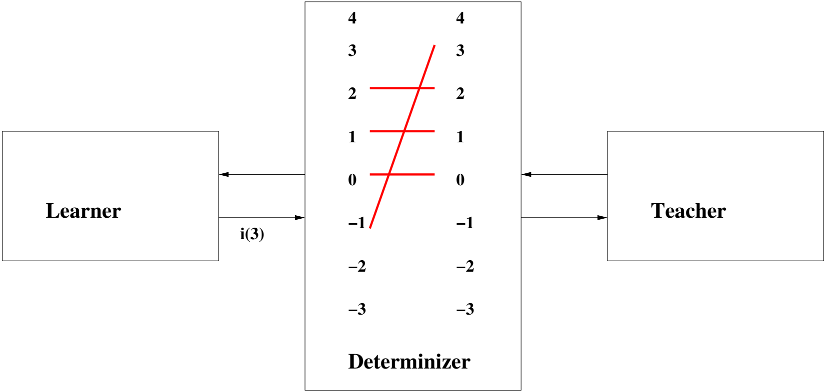
Idea: mapper stores part of automorphism constructed thus far

Mapper for determinizer



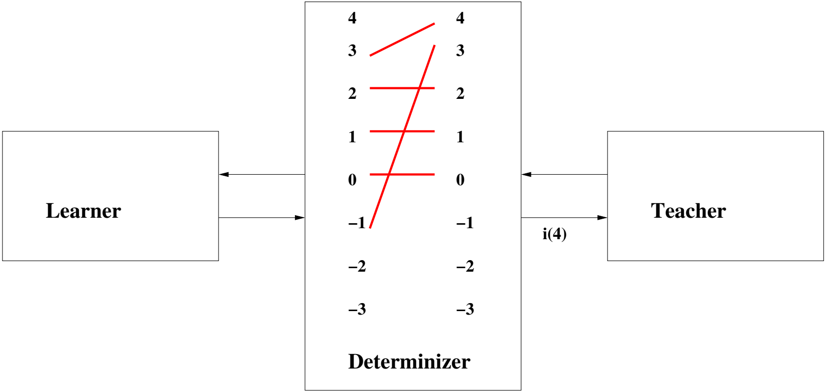
Idea: mapper stores part of automorphism constructed thus far

Mapper for determinizer



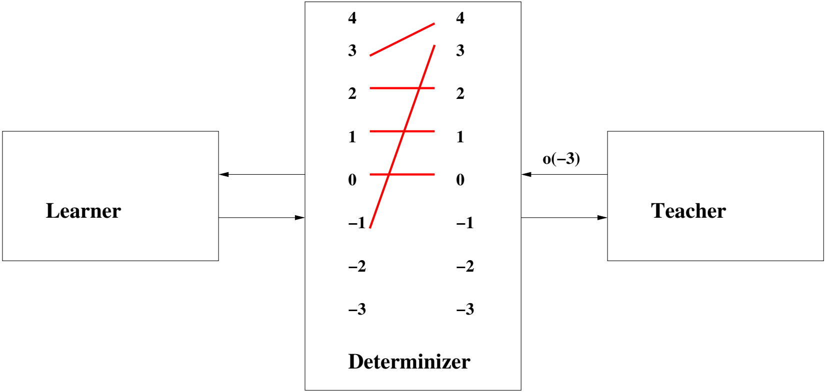
Idea: mapper stores part of automorphism constructed thus far

Mapper for determinizer



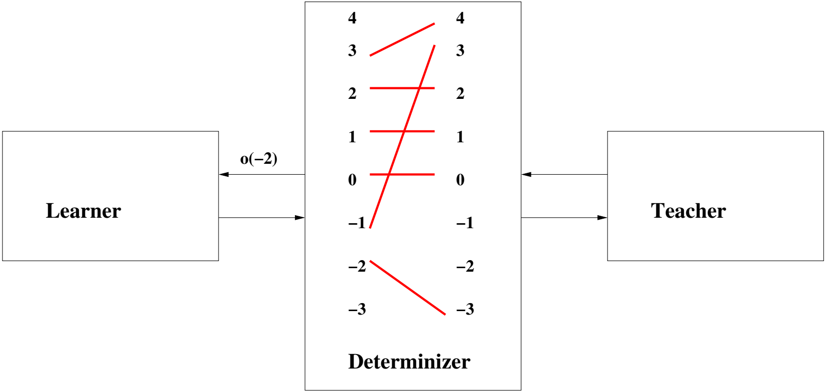
Idea: mapper stores part of automorphism constructed thus far

Mapper for determinizer



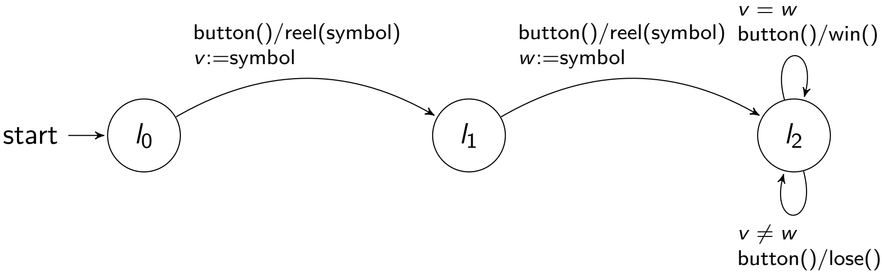
Idea: mapper stores part of automorphism constructed thus far

Mapper for determinizer



Idea: mapper stores part of automorphism constructed thus far

A register automaton with nondeterministic behavior



Collisions

Let β be a trace of $\mathcal{R}$.
Then we say that β ends with a collision if

- the last output value e is not fresh, and
- the sequence obtained by replacing e by any other value (except 0) is also a trace of $\mathcal{R}$.

Trace β has a collision if it has a prefix that ends with a collision.

Handling collisions

- Collisions are typically very rare
- Collisions can be detected by repeating experiments
- We just assume that collisions do not occur!!!
- If collisions are rare one cannot learn behavior anyway
- If they occur frequently one should not use our algorithm, but e.g. algorithm of Volpato & Tretmans

Proposition. The set of collision free neat traces of an input deterministic register automaton is behavior deterministic.

Conclusions

Conclusions

1. We have developed a mapper-based learning algorithm for class of nondeterministic register automata

Conclusions

1. We have developed a mapper-based learning algorithm for class of nondeterministic register automata
2. Algorithm implemented in Tomte

Conclusions

1. We have developed a mapper-based learning algorithm for class of nondeterministic register automata
2. Algorithm implemented in Tomte
3. Tomte outperforms LearnLib on common benchmarks

Conclusions

1. We have developed a mapper-based learning algorithm for class of nondeterministic register automata
2. Algorithm implemented in Tomte
3. Tomte outperforms LearnLib on common benchmarks
4. LearnLib does not handle nondeterministic systems but *does* support operations on data

Conclusions

1. We have developed a mapper-based learning algorithm for class of nondeterministic register automata
2. Algorithm implemented in Tomte
3. Tomte outperforms LearnLib on common benchmarks
4. LearnLib does not handle nondeterministic systems but *does* support operations on data
5. Future: extend Tomte to setting with operations on data