Active automata learning

Based on:
Bernhard Steffen, Falk Howar und Maik Merten: Introduction to Active Automata Learning from a Practical Perspective. SFM 2011: 256-296.

Motivation

- We want to apply model-based techniques.
- But: often there are no models.
- Or: existing models are unrelated to implementations.

- Produce models from implementations.

Example (River crossing game)

Mealy machines

Notation

- Σ, Ω: sets of symbols, ranged over by α_i (o_i resp.)
- $w, w', \ldots$: words, sequences of symbols, e.g., $w = \alpha_1\alpha_2\alpha_1$.
- $u, u', \ldots$: prefixes; $v, v', \ldots$: suffixes.

Concatenation:

- Let $u = \alpha_{u,1} \ldots \alpha_{u,n}$ and $v = \alpha_{v,1} \ldots \alpha_{v,m}$.
- Then: $w = uv = \alpha_{u,1} \ldots \alpha_{u,n}\alpha_{v,1} \ldots \alpha_{v,m}$

Mealy machines

Definition (Mealy machine)

A Mealy machine is defined as a tuple $\mathcal{M} = \langle S, s_0, \Sigma, \Omega, \delta, \lambda \rangle$ where

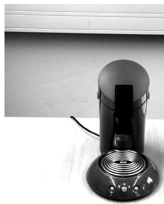
- S is a finite nonempty set of states (be $n = |S|$ the size of the Mealy machine),
- $s_0 \in S$ is the initial state,
- Σ is a finite input alphabet,
- Ω is a finite output alphabet,
- $\delta : S \times \Sigma \to S$ is the transition function, and
- $\lambda : S \times \Sigma \to \Omega$ is the output function.

The unforgiving coffee machine (UCM)

(a) empty

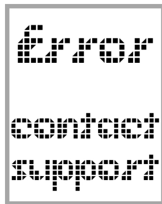
(b) with pod

(c) with water



(d) with pod and water

(e) success



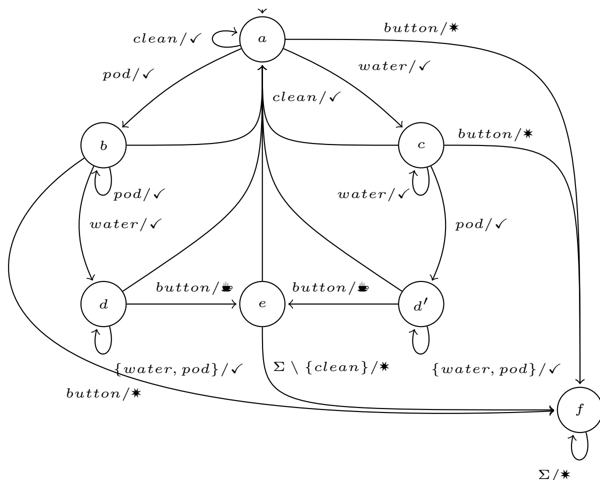
(f) error

Mealy machine example

Example (A coffee machine)

$\mathcal{M}_{cm} = \langle S, s_0, \Sigma, \Omega, \delta, \lambda \rangle$, where

- $S = \{a, b, c, d, d', e, f\}$
- $s_0 = a$
- $\Sigma = \{water, pod, button, clean\}$
- $\Omega = \{\checkmark, ☕, ✷\}$



δ and λ for words

$\delta^* : S \times \Sigma^* \to S$:

- $\delta^*(s, \epsilon) = s$
- $\delta^*(s, \alpha w) = \delta^*(\delta(s, \alpha), w)$

$\lambda^* : S \times \Sigma^* \to \Omega$:

- $\lambda^*(s, \epsilon) = \varnothing$
- $\lambda^*(s, w\alpha) = \lambda(\delta^*(s, w), \alpha)$

Runs of Mealy machines

- Mealy machine processing $\alpha_1 \alpha_2 \dots \alpha_n$
- will produce sequence of outputs $o_1 o_2 \dots o_n$

We will abstract from the complete traces!

Definition (Runs)

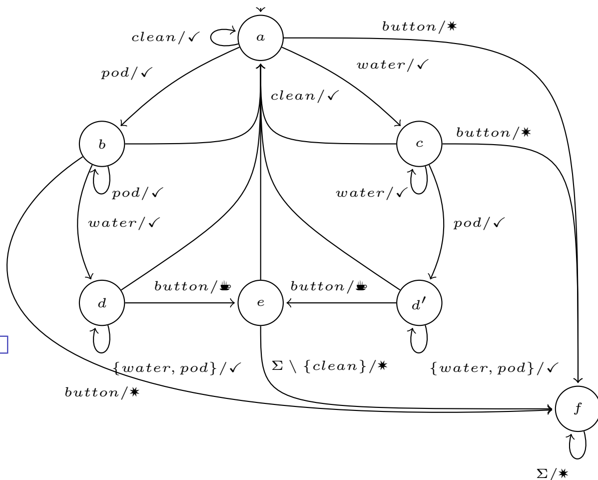
Let $\llbracket M \rrbracket : \Sigma^* \to \Omega$ defined by $\llbracket M \rrbracket(w) = \lambda^*(s_0, w)$.

Runs of Mealy machines (contd.)

Example (Runs)

The run $\langle water\ pod\ button\ clean\ button,\ ✷\rangle$ is in $[\![\mathcal{M}_{cm}]\!]$,

while the run $\langle water\ button\ clean,\ ✓\rangle$ is not. □



Equivalence of Mealy machines

Let $\mathcal{M} \equiv \mathcal{M}'$, iff $[\![\mathcal{M}]\!] = [\![\mathcal{M}']\!]$.

- But: we do not have $[\![\mathcal{M}]\!]$ (and its domain, Σ^*, is infinite)
- Need other means of comparing models

- Construct unique canonical Mealy machine for $[\![\mathcal{M}]\!]$ and $[\![\mathcal{M}']\!]$
- Test for isomorphism (e.g., by synchronized BFS)

Regularity

Equivalence of words

Let $P : \Sigma^* \to \Omega$

Definition (Equivalence of words wrt. P)

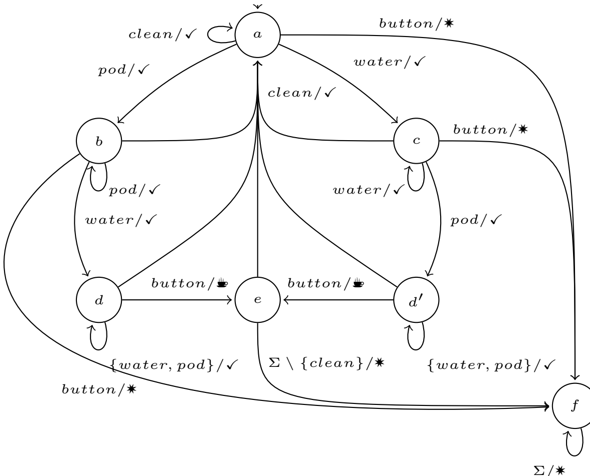
Two words $u, u' \in \Sigma^*$ are equivalent wrt. $\equiv_P$, iff for all continuations $v \in \Sigma^*$, the concatenated words uv and $u'v$ are mapped to the same output by P:

$$u \equiv_P u' \;\Leftrightarrow\; (\forall v \in \Sigma^*.\; P(uv) \;=\; P(u'v)).$$

We write $[u]$ to denote the equivalence class of u wrt. $\equiv_P$.

Equivalence of words (example)

$$
\begin{aligned}
& water\ pod \\
\equiv_{[\![\mathcal{M}_{cm}]\!]}\ & water\ water\ pod \\
\equiv_{[\![\mathcal{M}_{cm}]\!]}\ & pod\ pod\ water
\end{aligned}
$$



Myhill-Nerode for Mealy machines

Theorem (Characterization Theorem)

A mapping $P: \Sigma^* \to \Omega$ is a semantic functional for some Mealy machine iff $\equiv_P$ has only finitely many equivalence classes (finite index).

Proof $(\Rightarrow)$:

- We show: $\mathcal{M}$ is an arbitrary Mealy machine $\Rightarrow$ $\equiv_{[\![M]\!]}$ has finite index
- All words leading to the same state in $\mathcal{M}$ are equivalent wrt. $\equiv_{[\![M]\!]}$.

Myhill-Nerode for Mealy machines (contd.)

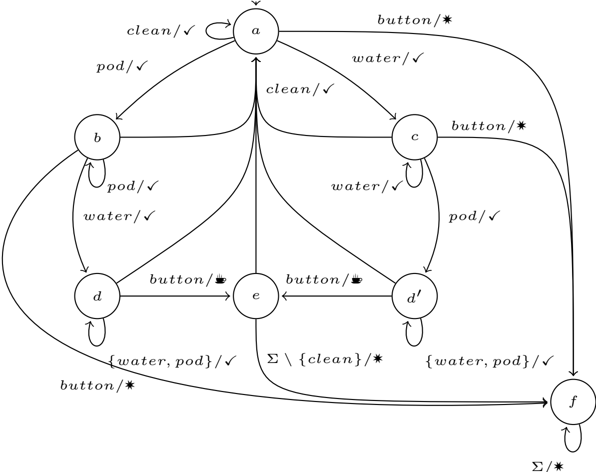
Proof ($\Leftarrow$):
We will construct a Mealy machine for P.

Let $\mathcal{M}_P = \langle S, s_0, \Sigma, \Omega, \delta, \lambda \rangle$:

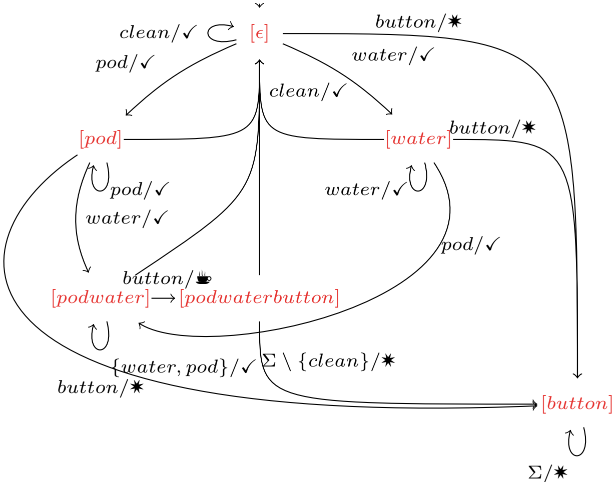
- S is given by the classes of $\equiv_P$.
- s_0 is given by $[\epsilon]$.
- the transition function is defined by $\delta([w], \alpha) = [w\alpha]$.
- the output function can be defined as $\lambda([w], \alpha) = o$, where $P(w\alpha) = o$.

$\mathcal{M}_P$ is well-defined and has semantic functional P.

Example



Example



Consequences (I)

We will call P regular if $\equiv_P$ has finite index, i.e.,
if there is a corresponding Mealy machine $\mathcal{M}_P$.

Corollary (Minimality)

For regular P, $\mathcal{M}_P$ is minimal.

Proof:

- Every state of a Mealy machine $\mathcal{M}$ belongs to one class of $\equiv_{[\![\mathcal{M}]\!]}$.
- $\mathcal{M}_{[\![\mathcal{M}]\!]}$ has exactly one state per class of $\equiv_{[\![\mathcal{M}]\!]}$.

Consequences (II)

Corollary (Bounded reachability)

For regular P, every transition of $\mathcal{M}_P$ can be covered by a sequence of length at most n from the initial state.

(n: number of states of $\mathcal{M}_P$)

Especially: For every Mealy machine with N states, all transitions of its mininmal equivalent Mealy machine can be covered by sequences of length at most N.

Canonical Mealy machines

Minimizing automata

Question: How can we minimize non-minimal automata / proof minimality?

First idea: merge states based on equivalence of words:

$$s \equiv s' \;\Leftrightarrow\; \forall v \in \Sigma^+ \;.\; \lambda^*(s, v) = \lambda^*(s', v)$$

But: it may be hard to argue about all future behaviors.
$\Rightarrow$ exchange quantifier!

k-distinguishability

Definition (k-distinguishability)

Two states $s, s' \in S$ of some Mealy machine $\mathcal{M}$ are k-distinguishable, iff there is a word $w \in \Sigma^*$ of length k or shorter, for which $\lambda^*(s, w) \neq \lambda^*(s', w)$. □

How big can k get?

Proposition

In a Mealy machine of size n, states $s \not\equiv s' \in S$ are n-distinguishable. □

Naive minimization

1. Compute spanning tree for $\mathcal{M}$, (size n)
2. Associate every state with path to it in spanning tree (from root), its access sequence
3. For every state construct $T_s : \Sigma^{\leq n} \mapsto \Omega$ by

$$T_s(v) =_{def} [\![\mathcal{M}]\!](uv)$$

   where u is access sequence to s and $v \in \Sigma^{\leq n}$
4. Merge states s, s', whenever $T_s = T_{s'}$

Complexity for comparing states: $O(n^2 \cdot |\Sigma|^n)$

Partition refinement

λ valuation

Definition

For a Mealy machine $\mathcal{M}$ and some state s in $\mathcal{M}$, the λ valuation of s is a mapping $\lambda_s : \Sigma \mapsto \Omega$, with

$$\lambda_s(\alpha) =_{def} \lambda(s, \alpha)$$

Partition refinement (algorithm)

Input: A Mealy machine $\mathcal{M} = \langle S, s_0, \Sigma, \Omega, \delta, \lambda \rangle$
Output: A partition P on S, the set of states of the Mealy machine

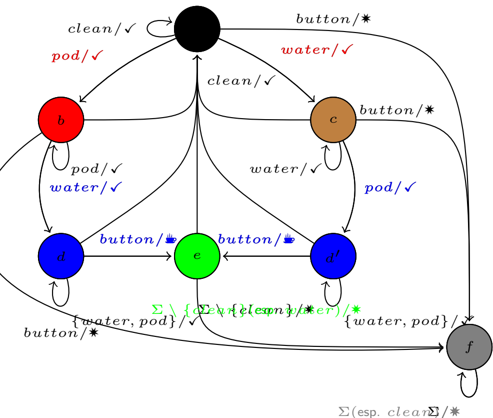
```
1:  i := 1
2:  put all s ∈ S with the same λ valuation into the same class p of partition P_i
3:  loop
4:      for all p ∈ P_i do
5:          for all s ∈ p do
6:              construct mapping sig : Σ → P_i:
7:                      sig(α) = p' such that δ(s, α) ∈ p'
8:              S^i_sig := S^i_sig ∪ s
9:          end for
10:         P_{i+1} := ⋃_sig S^i_sig
11:     end for
12:     if P_i = P_{i+1} then
13:         return P_i
14:     end if
15:     i := i + 1
16: end loop
```

Partition refinement (complexity)

- Initial computation of λ valuations: $(n \cdot |\Sigma|)$
- Computation of sig objects per rounds: $(n \cdot |\Sigma|)$
- Maximum number of rounds n.

Overall complexity: $O(n^2 \cdot |\Sigma|)$

Partition refinement (example)

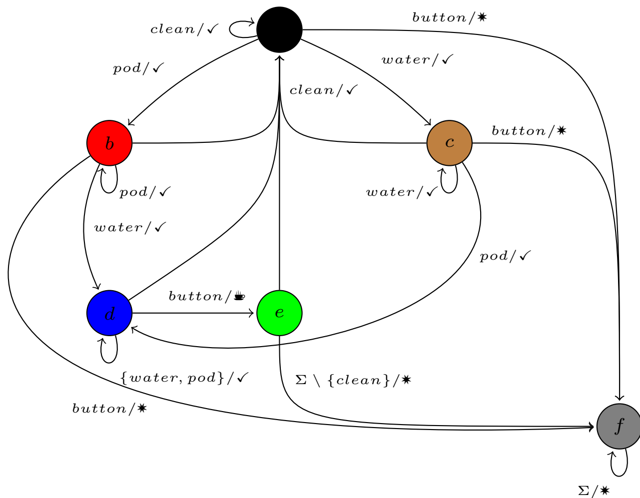


$P_1 = \{a, b, c\}, \{d, d'\}, \{e\}, \{f\}$

$P_2 = \{a\}, \{b\}, \{c\}, \{d, d'\}, \{e\}, \{f\}$

$P_3 = \{a\}, \{b\}, \{c\}, \{d, d'\}, \{e\}, \{f\}$

Minimized Mealy machine



Extension to black-boxes

Scenario

- System under learning (SUL) is a black-box
- No knowledge about (number of) internal states
- Input alphabet given
- Tests can be executed on SUL, output can be recorded

Try to produce complete and correct behavioral model of (interface-)behavior

Naive learning

Bad news: in general impossible! The possibility that one has not tested enough will always remain.

- Assume maximum number of states to be N.
- Modify naive minimization

Naive minimization

1. Compute prefix tree for depth N.
2. Associate every state with path to it in prefix tree (from root)
3. For every state construct $T_s : \Sigma^{\leq N} \mapsto \Omega$ by

$$T_s(v) =_{def} [\![\mathcal{M}]\!](uv)$$

   where u is access sequence to s and $v \in \Sigma^{\leq N}$
4. Merge states s, s', whenever $T_s = T_{s'}$

Complexity for comparing states: $O(|\Sigma|^N \cdot |\Sigma|^N) = O(|\Sigma|^{2N})$

Summarizing

- States can be represented by access sequences
- States can be distinguished by suffixes

Open questions:

- Can we extend the partition refinement pattern to produce suffixes?
- Can the partition refinement pattern be applied in the black box scenario?

The ID learning algorithm

Identifying states

Definition (Output-signature)

For a SUL and some word u in Σ^*, the output-signature of u in SUL is a mapping $[\![SUL]\!]_u^{\mathcal{D}} : \mathcal{D} \mapsto \Omega$ with $\mathcal{D} \subset \Sigma^*$, and

$$[\![SUL]\!]_u(v) =_{def} [\![SUL]\!](uv)$$

ID learning algorithm

Input: A SUL with inputs Σ, a set $U \subset \Sigma^*$ of access sequences to all states of SUL
Output: A Mealy machine $\mathcal{M} = \langle S, s_0, \Sigma, \Omega, \delta, \lambda \rangle$ for SUL

1: $i := 1$, $\mathcal{D} := \Sigma$, $\mathcal{S} := U \cup U \times \Sigma$
2: start:
3: compute $[\![SUL]\!]_u^{\mathcal{D}}$ for all $u \in \mathcal{S}$
4: put $u, u' \in \mathcal{S}$ into the same class p of partition P_i if $[\![SUL]\!]_u^{\mathcal{D}} = [\![SUL]\!]_{u'}^{\mathcal{D}}$
5: **for all** $p \in P_i$ **do**
6:     **for all** $\alpha \in \Sigma$ **do**
7:         **for all** $u, u' \in p \cap U$ **do**
8:             **if** $u\alpha \in p' \wedge u'\alpha \notin p'$ **then**
9:                 Let $v \in \mathcal{D}$ be s.t. $[\![SUL]\!]_{u\alpha}^{\mathcal{D}}(v) \neq [\![SUL]\!]_{u'\alpha}^{\mathcal{D}}(v)$
10:                 $\mathcal{D} := \mathcal{D} \cup \{\alpha v\}$, $i := i + 1$
11:                 goto start
12:             **end if**
13:         **end for**
14:     **end for**
15: **end for**
16: construct $\mathcal{M}_{[\![SUL]\!]}$ from P_i

Model construction from final P_i

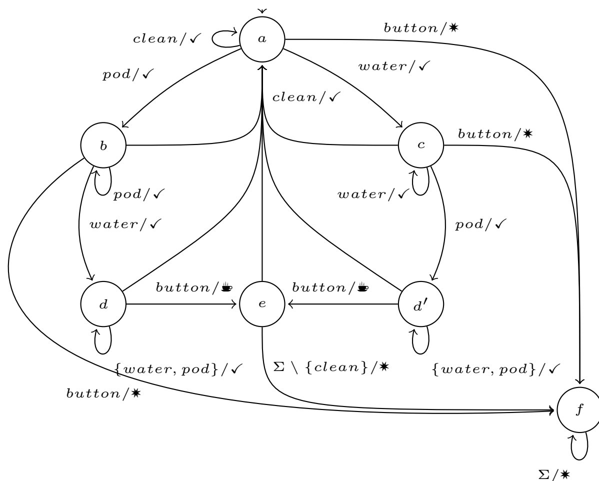
We will construct a Mealy machine from final P_i by:

Let $\mathcal{M}_{[\![SUL]\!]} = \langle S, s_0, \Sigma, \Omega, \delta, \lambda \rangle$:

- S is given by the classes of P_i.
- s_0 is given by p that contains ϵ.
- the transition function is defined by $\delta(p, \alpha) = p'$ if exists $u \in p \cap U$ with $u\alpha \in p'$
- the output function can be defined as $\lambda(p, \alpha) = o$ if exists $u \in p \cap U$ with $[\![SUL]\!]_u^{\mathcal{D}}(\alpha) = o$

$\mathcal{M}_{[\![SUL]\!]}$ is well-defined and has semantic functional $[\![SUL]\!]$ if U contains access sequences to all classes of $[\![SUL]\!]$. $\square$

The coffee machine



ID learning algorithm (example)

	pod	*water*	*button*	*clean*
λ	✓	✓	✷	✓
pod	✓	✓	✷	✓
water	✓	✓	✷	✓
button	✷	✷	✷	✷
pod water	✓	✓	☕	✓
water pod	✓	✓	☕	✓
pod water button	✷	✷	✷	✓
...	...	...	...	...

ID learning algorithm (example)

	pod	*water*	*button*	*clean*
λ	✓	✓	✷	✓
pod	✓	✓	✷	✓
water	✓	✓	✷	✓
button	✷	✷	✷	✷
pod water	✓	✓	☕	✓
water pod	✓	✓	☕	✓
pod water button	✷	✷	✷	✓
...	...	...	...	...

ID learning algorithm (example)

	pod	*water*	*button*	*clean*
λ	✓	✓	✷	✓
pod	✓	✓	✷	✓
water	✓	✓	✷	✓
button	✷	✷	✷	✷
pod water	✓	✓	☕	✓
water pod	✓	✓	☕	✓
pod water button	✷	✷	✷	✓
...	...	...	...	...

ID learning algorithm (example)

	pod	*water*	*button*	*clean*	*pod button*
λ	✓	✓	✷	✓	✷
pod	✓	✓	✷	✓	✷
water	✓	✓	✷	✓	☕
button	✷	✷	✷	✷	✷
pod water	✓	✓	☕	✓	☕
water pod	✓	✓	☕	✓	☕
pod water button	✷	✷	✷	✓	✷
...	...	...	...	...	...

ID learning algorithm (example)

	pod	*water*	*button*	*clean*	*pod button*
λ	✓	✓	✷	✓	✷
pod	✓	✓	✷	✓	✷
water	✓	✓	✷	✓	☕
button	✷	✷	✷	✷	✷
pod water	✓	✓	☕	✓	☕
water pod	✓	✓	☕	✓	☕
pod water button	✷	✷	✷	✓	✷
...	...	...	...	...	...

ID learning algorithm (example)

	pod	*water*	*button*	*clean*	*pod button*
λ	✓	✓	✷	✓	✷
pod	✓	✓	✷	✓	✷
water	✓	✓	✷	✓	☕
button	✷	✷	✷	✷	✷
pod water	✓	✓	☕	✓	☕
water pod	✓	✓	☕	✓	☕
pod water button	✷	✷	✷	✓	✷
...	...	...	...	...	...

ID learning algorithm (example)

	pod	*water*	*button*	*clean*	*pod button*	*water button*
λ	✓	✓	✷	✓	✷	✷
pod	✓	✓	✷	✓	✷	☕
water	✓	✓	✷	✓	☕	✷
button	✷	✷	✷	✷	✷	✷
pod water	✓	✓	☕	✓	☕	☕
water pod	✓	✓	☕	✓	☕	☕
pod water button	✷	✷	✷	✓	✷	✷
...	...	...	...	...	...	...

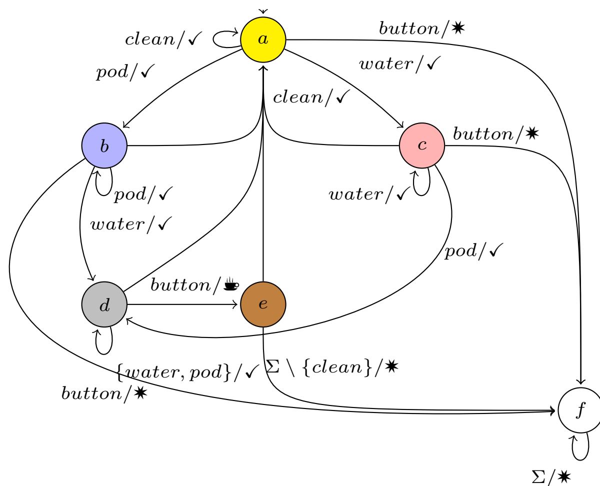
ID learning algorithm (example)

	pod	*water*	*button*	*clean*	*pod button*	*water button*
λ	✓	✓	✷	✓	✷	✷
pod	✓	✓	✷	✓	✷	☕
water	✓	✓	✷	✓	☕	✷
button	✷	✷	✷	✷	✷	✷
pod water	✓	✓	☕	✓	☕	☕
water pod	✓	✓	☕	✓	☕	☕
pod water button	✷	✷	✷	✓	✷	✷
...	...	...	...	...	...	...

ID learning algorithm (example)

	pod	*water*	*button*	*clean*	*pod button*	*water button*
λ	✓	✓	✷	✓	✷	✷
pod	✓	✓	✷	✓	✷	☕
water	✓	✓	✷	✓	☕	✷
button	✷	✷	✷	✷	✷	✷
pod water	✓	✓	☕	✓	☕	☕
water pod	✓	✓	☕	✓	☕	☕
pod water button	✷	✷	✷	✓	✷	✷
water pod button	✷	✷	✷	✓	✷	✷
pod water water	✓	✓	☕	✓	☕	☕
water pod water	✓	✓	☕	✓	☕	☕
pod water pod	✓	✓	☕	✓	☕	☕
water pod pod	✓	✓	☕	✓	☕	☕
pod water clean	✓	✓	✷	✓	✷	✷
water pod clean	✓	✓	✷	✓	✷	✷
...	...	...	...	...	...	...

ID learning algorithm (example)



ID learning algorithm (complexity)

Measure: number of tests to be conducted

- Number of prefixes is in $O(U|\Sigma|)$
- Number of suffixes is in $O(|\Sigma| + n)$

Number of tests is in $O(U|\Sigma|^2 \ + \ U|\Sigma|n)$

In case U in $O(n)$, we get $O(n|\Sigma|^2 + n^2|\Sigma|)$

Intermediate summary

Learning in general incomplete! The possibility that one has not tested enough will always remain.

Solutions for restricted problems so far:

1. Maximum number of states given (naive), $O(|\Sigma|^{2N})$.
2. Prefixes for all states given, $O(U|\Sigma|^2 \; + \; U|\Sigma|n)$
3. Maximum number of states given (partition refinement),

$$O(\;(|\Sigma|^N)|\Sigma|^2 \;+\; (|\Sigma|^N)|\Sigma|n) \;=\; O(|\Sigma|^N n)$$

4. Use equivalence queries, ?.

Is it possible to construct prefixes incrementally?

The L^*_M-algorithm

Handling counterexamples

CE descomposition

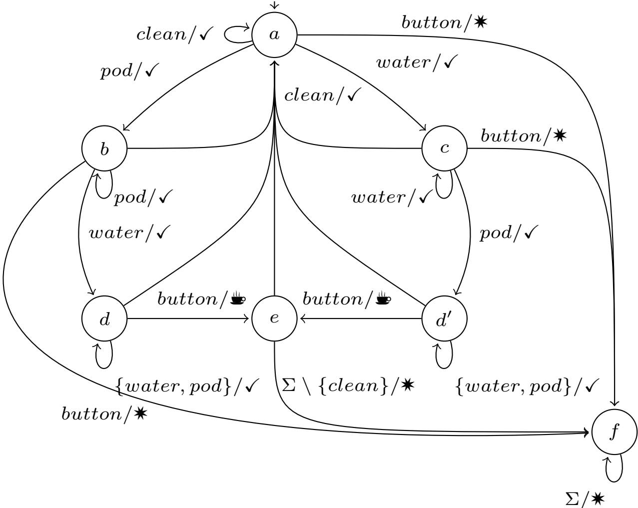
Let $[u]_{\mathcal{H}}$ denote the access sequence of u in hypothesis $\mathcal{H}$.

Theorem (Counterexample Decomposition)

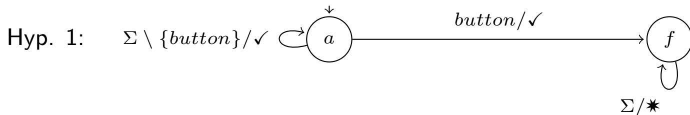
For every counterexample $\bar{c}$ there exists a decomposition $\bar{c} = u\alpha v$ into a prefix u, an action α, and a suffix v such that
mq$([u]_{\mathcal{H}}\alpha v) \neq$ *mq*$([u\alpha]_{\mathcal{H}} v)$.

$[u]_{\mathcal{H}}\alpha$ and $[u\alpha]_{\mathcal{H}}$ lead to same state in $\mathcal{H}$
$\Rightarrow$ use suffix v to produce un-closedness in observation table

The coffe machine



The coffe machine - counterexample



Counterexample: $\bar{c} = pod\ water\ pod\ water\ button$, leads to ☕ in UCM but to ✷ in Hyp.

i	u	$[u]_{\mathcal{H}}$	α	v	$\mathcal{M}_{cm}()$
1	ϵ	ϵ	pod	$water\ pod\ water\ button$	☕
2	pod	ϵ	$water$	$pod\ water\ button$	☕
3	$pod\ water$	ϵ	pod	$water\ button$	☕
4	$pod\ water\ pod$	ϵ	$water$	$button$	✷
5	$pod\ water\ pod\ water$	ϵ	$button$	ϵ	✷

But

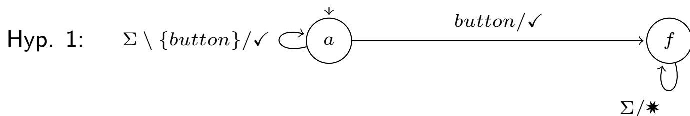
Both strategies (i.e., all prefixes to $\mathcal{S}p$, all suffixes to $\mathcal{D}$) are expensive.
Is there a more efficient strategy?

Binary search for counterexamples

Input: counterexample $\bar{c} = \bar{c}_1 \ldots \bar{c}_m$, hypothesis $\mathcal{H}$.
Output: new suffix for $\mathcal{D}$

1: $o_{\bar{c}} :=$ mq$(\bar{c})$
2: $lower := 2, \quad upper := m - 1$
3: **loop**
4: $\quad mid := \lfloor (lower + upper) / 2 \rfloor$
5: $\quad s := \bar{c}_1 \ldots \bar{c}_{mid-1}, \quad s' := [s]_{\mathcal{H}}, \quad d := \bar{c}_{mid} \ldots \bar{c}_m$
6: $\quad o_{mid} :=$ mq$(s'd)$
7: $\quad$ **if** $o_{mid} = o_{\bar{c}}$ **then**
8: $\quad\quad lower := mid + 1$ // same as reference output: move right
9: $\quad\quad$ **if** $upper < lower$ **then**
10: $\quad\quad\quad$ **return** $\bar{c}_{mid+1} \ldots \bar{c}_m$
11: $\quad\quad$ **end if**
12: $\quad$ **else**
13: $\quad\quad upper := mid - 1$ // not same as reference output: move left
14: $\quad\quad$ **if** $upper < lower$ **then**
15: $\quad\quad\quad$ **return** $\bar{c}_{mid} \ldots \bar{c}_m$
16: $\quad\quad$ **end if**
17: $\quad$ **end if**
18: **end loop**

Binary search over counterexample



Counterexample: $\bar{c} = pod\ water\ pod\ water\ button$, leads to ☕ in UCM but to ✷ in Hyp.

u	$[u]_{\mathcal{H}}$	lower	mid	upper	1	2	3	4	5
ϵ	ϵ				☕				
$pod\ water\ pod\ water$	ϵ								✷
$pod\ water$	ϵ	2	3	4			☕		
$pod\ water\ pod$	ϵ	3+1	4	4				✷	
-		4	-	4-1					

Summary

Active automata learning / summary

- System under learning (SUL) is a black-box
- Input alphabet given
- Tests can be executed on SUL, output can be recorded
- Membership queries: $[\![SUL]\!] : \Sigma^* \mapsto \Omega$

Active automata learning / summary

Bad news: in general impossible! The possibility that one has not tested enough will always remain.

- States can be represented by access sequences
- States can be distinguished by suffixes

- Partition refinement pattern yields suffixes!
- In general, we cannot construct prefixes incrementally!

Good news: We can introduce some magic that resolves prefix problem.
Equivalence queries test wether $[\![\mathcal{H}]\!] = [\![SUL]\!]$, or not.

Summary (learning w/ MQ)

1. Maximum number of states given (naive): construct perfix tree of depth N trees and trees T_s. MQ complexity: $O(|\Sigma|^{2N})$.
2. Prefixes for all states given (ID algo.): use partition refinement to find suffixes. MQ complexity: $O(U|\Sigma|^2 \; + \; U|\Sigma|n)$
3. Maximum number of states given (partition refinement): combine approaches. MQ complexity:

$$O(\; (|\Sigma|^N)|\Sigma|^2 \; + \; (|\Sigma|^N)|\Sigma|n) \; = \; O(|\Sigma|^N n)$$

Summary (learning w/ MQ and EQ)

Angluins algorithm:

- Use observation tables to organize results from tests
- Partition states according to (growing partial) $\lambda_s : \Sigma^* \mapsto \Omega$.
- Iteratively refine partition:
  - Closedness: Move "new" row from lower to upper part of table. (Allows to drop the assumption from ID algo. that all necessary prefixes are given as input).
  - Consistency: Partition refinement to generate suffixes.

Summary (learning w/ MQ and EQ)

Angluins algorithm (contd.):

- Use equivalence queries to get counterexamples:
  - Counterexample passes additional state (cf. proof in lecture & in paper)
  - $\Rightarrow$ put all prefixes of ce. to upper part of table. (i.e., "incremental" ID algorithm)
- Complexity:
  - MQ: $O(nm|\Sigma| \cdot (n + |\Sigma|)) = O(n^2|\Sigma|m + n|\Sigma|^2 m)$
  - EQ: $O(n)$

Summary (learning w/ MQ and EQ)

Other strategies to handle counterexamples (due to decomposition theorem):

- Add all suffixes of ce. to suffixes:
  - No inconsistencies!
  - Complexity (MQ): $O(n|\Sigma| \cdot (nm + |\Sigma|)) = O(n^2|\Sigma|m + n|\Sigma|^2)$
- Find one suffix by binary search:
  - Complexity (MQ): $O(n|\Sigma| \cdot (n + |\Sigma|) + n \cdot log_2(m)) = O(n^2|\Sigma| + n|\Sigma|^2 + n \cdot log_2(m))$
- Complexity (EQ): $O(n)$