LEARNING NOMINAL AUTOMATA

Joshua Moerman
Joint work with
Alexandra Silva, Matteo Sammartino, Bartek Klin and Michał Szynwelski
$S^3 =$
OUTLINE

Context: Register automata
Notion of nominal automata
Learning of nominal automata
Conclusion
REGISTER AUTOMATA
DETERMINISTIC FINITE REGISTER AUTOMATA (RA)

Like finite automata, but add:

1. Registers, i.e. state
2. Guards on transitions

Used in verification of protocols / reactive systems / …
Deterministic Finite Register Automata (RA)

Semantics: languages

\[ L_0 = \{ li(a)s(b_1, m_1) \ldots s(b_k, m_k)lo \mid b_i \neq a \} \]
\[ L = L_0^* \]
LEMMA

Let \( \mathcal{D} \) be some data domain
And \( g \in Aut(\mathcal{D}) \) a symmetry over this domain
Then:
1. \( g \) also acts on the set of symbols and words
2. \( w \in L \iff g(w) \in L \)

Example: \( \mathcal{D} = \mathbb{N} \) and previous automaton

\( g \) swap 3 and 5:
\[(\text{li}(5)s(3,37)lo) \in L \]
\[(\text{li}(3)s(5,37)lo) \in L \]

\( g \) swap 1 and 2:
\[(\text{li}(1)s(1,2)lo) \notin L \]
\[(\text{li}(2)s(2,1)lo) \notin L \]
IDEA

Forget about registers and guards
Take symmetries to be first class
NOMINAL AUTOMATA
GROUP ACTIONS

Let $G$ be a group

Let $X$ be a set

A **$G$-action** is a map $\cdot : G \times X \to X$ such that

- $1 \cdot x = x$
- $g \cdot (h \cdot x) = (g h) \cdot x$

In a way, a $G$-action tells us that $X$ has some symmetry of $G$

Recall (from your maths education?):

A **group** is a set $G$ together with:

- A product $G \times G \to G$
- A unit $1 \in G$

Such that

- Associative: $(g h) i = g (h i)$
- Unital: $1 g = g = g 1$
- There are inverses: $g g^{-1} = 1$
GROUP ACTIONS

In our case, we take:

\( G = Aut(\mathbb{N}) \) the group of all bijections on \( \mathbb{N} \) (product is composition and unit is identity map)

The alphabet \( \Sigma = \{ \text{logout} \} \cup \{ \text{login}(u) \mid u \in \mathbb{N} \} \cup \{ \text{send}(u,m) \mid u,m \in \mathbb{N} \} \) has a natural \( G \)-action:

\[
\begin{align*}
g \cdot \text{logout} &= \text{logout} \\
g \cdot \text{login}(u) &= \text{login}(g(u)) \\
g \cdot \text{send}(u,m) &= \text{send}(g(u),g(m))
\end{align*}
\]

And it gives an action on the set of words \( \Sigma^* \):

\[
\begin{align*}
g \cdot \epsilon &= \epsilon \\
g \cdot (aw) &= (g \cdot a)(g \cdot w)
\end{align*}
\]
Sometimes symmetry is preserved.

Let \((X, \cdot)\) be a set with a \(G\)-action.

Then a subset \(Y \subseteq X\) is **equivariant** if \(y \in Y \iff g \cdot y \in Y\) for all \(g \in G\).

Let \((X, \cdot)\) and \((Y, \cdot)\) be a sets with a \(G\)-action.

Then a function \(f: X \to Y\) is **equivariant** if \(f(g \cdot x) = g \cdot f(x)\) for all \(g \in G\).
Let \((X, \cdot)\) be a set with a \(G\)-action.

Take \(x \in X\), then its **orbit** is the set \(\text{orb}(x) = \{g \cdot x \mid g \in G\}\).

Orbits partition the set.

The alphabet \(\Sigma = \{\text{logout}\} \cup \{\text{login}(u) \mid u \in \mathbb{N}\} \cup \{\text{send}(u,m) \mid u,m \in \mathbb{N}\}\) has four orbits:

\[
\begin{align*}
\{\text{logout}\} \\
\{\text{login}(u) \mid u \in \mathbb{N}\} \\
\{\text{send}(u,m) \mid u,m \in \mathbb{N}, u \neq m\} \\
\{\text{send}(u,u) \mid u \in \mathbb{N}\}
\end{align*}
\]
FROM REGISTERS TO SYMMETRIES
From the register automata we can define an ordinary (infinite) automaton, by expanding the state space.

Note:
Symmetry on states
Transitions are equivariant
IDEA

Forget about registers and guards
Take symmetries to be first class
WHAT’S THE POINT?

**Register automaton** (over a theory) is:
- Set of states with registers
- Acceptance of states
- Transitions with guards and assignments

**G-automata** consist of:
- Initial state: $x_0 \in X$
- Acceptance of states: $X \rightarrow 2$
- A transition function: $X \times \Sigma \rightarrow X$
- Both maps equivariant

**DFA** consist of:
- Initial state: $x_0 \in X$
- Acceptance of states: $X \rightarrow 2$
- A transition function: $X \times \Sigma \rightarrow X$

We hope that definitions and algorithms for DFAs generalize to $G$-automata.

One immediate problem is:
The sets $G$, $X$, $L$, etc are infinite…
Let $X$ be any set with $G$-action

Condition 1) $X$ is **orbit-finite**: \{$\text{orb}(x) \mid x \in X\}$ is finite

Condition 2) $X$ is **nominal**: every $x \in X$ has finite support, meaning that there is a finite $C \subseteq D$ such that $g \cdot x = x$ if $g \cdot C = C$.

Intuitively: every element $x$ contains only finitely many data values.

Then $X$ is **finitely representable** (i.e. we can do algorithms)

Only for certain type of symmetries. In particular it holds for $G = \text{Aut}(\mathbb{N})$. 

**THEOREM (RYLL-NARDZEWSKI)**

Let $T$ be a complete theory and $M$ a countable model (for example the theory of $(\mathbb{N},=)$). Then TFAE:

1. $T$ is countably categorical (I’ll skip the definition)

2. $T$ has finitely many $n$-types (a notion from logic: an $n$-type is an equivalence class of formulas with $n$ free variables)

3. $M^n$ has finitely many orbits (a notion from group theory, as we just saw)

**Consequence:** For suitable theories: register automata are equi-expressive to nominal automata!
Register automaton (over a theory) is:
Set of states with registers
Acceptance of states
Transitions with guards and assignments

\( G \)-automata consist of:
Initial state: \( x_0 \in X \)
Acceptance of states: \( X \to 2 \)
A transition function: \( X \times \Sigma \to X \)
Both maps equivariant

For the theories \( (\mathbb{N}, =), (\mathbb{Q}, <), (\mathcal{R}, ajd), \ldots \)
register automata and nominal automata are equivalent.

Register automata defined with logic.
Nominal automata defined with symmetries.

Pick your favourite!

Implementations:
Mihda (OCaml, based on symmetries)
Nlambda (Haskell, based on logic)
LEARNING
WHAT IS LEARNING?

There are many models for learning

1. Given a fixed data set. Algorithm finds a (small) model which best explains the data. This is the most common type of “machine learning”.

2. There is no data set, but there is a teacher which provides answers. Algorithm can then query specific samples. This is called query learning. (This is more similar to how people learn, by asking.)

And all kinds of mixes of these models.
Algorithm may pose two types of queries:

1. Membership queries: “Is \( w \in L \)?”

2. Equivalence queries: “Does \( H \) exactly accept \( L \)?”. Here the teacher may answer with a counter example.

Of course, we seek for an efficient algorithm: the number of queries should be bounded polynomially (in something…)

For the application of software verification, query learning is quite natural: queries can be answered by the software itself. The equivalence queries can be approximated by testing. This gives an efficient PAC algorithm.
**L* ALGORITHM FOR FINITE ALPHABETS**

Dana Angluin devised the first such learning algorithm for DFAs. It combines ideas from reachability and **minimisation** of automata.

Fix a language $L$. Two words $x, y \in \Sigma^*$ are **Myhill-Nerode equivalent** if

$$xz \in L \iff yz \in L \quad \text{for all } z \in \Sigma^*$$

Let’s look at an example (with finite alphabet)!
<table>
<thead>
<tr>
<th></th>
<th>_</th>
<th>a</th>
<th>b</th>
<th>aa</th>
<th>ab</th>
<th>ba</th>
<th>bb</th>
<th>aaa</th>
<th>aab</th>
<th>aba</th>
</tr>
</thead>
<tbody>
<tr>
<td>_</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>aa</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ab</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ba</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bb</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>aaa</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>aab</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>aba</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Hankel matrix of language**

$L = \{ \text{awb} \mid w \in \Sigma^* \}$

Infinite in both directions!
<table>
<thead>
<tr>
<th></th>
<th>_</th>
<th>a</th>
<th>b</th>
<th>aa</th>
<th>ab</th>
<th>ba</th>
<th>bb</th>
<th>aaa</th>
<th>aab</th>
<th>aba</th>
</tr>
</thead>
<tbody>
<tr>
<td>_</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>aa</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ab</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ba</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bb</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>aaa</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>aab</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>aba</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>_</td>
<td>a</td>
<td>b</td>
<td>aa</td>
<td>ab</td>
<td>ba</td>
<td>bb</td>
<td>aaa</td>
<td>aab</td>
<td>aba</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>_</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>aa</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ab</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ba</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bb</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>aaa</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>aab</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>aba</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>_</td>
<td>a</td>
<td>b</td>
<td>aa</td>
<td>ab</td>
<td>ba</td>
<td>bb</td>
<td>aaa</td>
<td>aab</td>
<td>aba</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>_</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>aa</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ab</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ba</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bb</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>aaa</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>aab</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>aba</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

x equivalent to y
iff row x and row y have the same color
<table>
<thead>
<tr>
<th></th>
<th>_</th>
<th>a</th>
<th>b</th>
<th>aa</th>
<th>ab</th>
<th>ba</th>
<th>bb</th>
<th>aaa</th>
<th>aab</th>
<th>aba</th>
</tr>
</thead>
<tbody>
<tr>
<td>_</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>aa</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>ab</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>ba</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>bb</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>aaa</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>aab</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>aba</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Each color represents a state in the minimal automaton.
L* ALGORITHM

The main idea of the L* algorithm is as follows:

We can use these colours on a finite segment of the matrix.

However, not every finite segment defines an automaton.

In that case, we can add rows and columns (using membership queries) until it does define an automaton.

At some point, the finite segment does define an automaton. Then the algorithm poses an equivalence query.

If it is equivalent, we terminate. Otherwise we extend the matrix with the counter example.
NOMINAL L* ALGORITHM

In the case of nominal automata, we do exactly the same
Only the matrix changes.
<table>
<thead>
<tr>
<th></th>
<th>_</th>
<th>Lo</th>
<th>Li(0)</th>
<th>Li(1)</th>
<th>...</th>
<th>S(0,0)</th>
<th>S(1,1)</th>
<th>...</th>
<th>S(0,0)Lo</th>
</tr>
</thead>
<tbody>
<tr>
<td>_</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>Lo</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>Li(0)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>Li(1)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>S(0,0)</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>S(1,1)</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>S(0,1)</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>S(0,2)</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>LoLo</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
WHAT IS A (NOMINAL) TABLE?

A table is a function $T: R \times C \to 2$
For finite sets: write $R = \{r_1, \ldots, r_n\}, C = \{c_1, \ldots, c_m\}$
Then $T$ is a collection of cells: $T_{ij}: \{r_i\} \times \{c_j\} \to 2$
And each cell is encoded by a single boolean

For nominal sets: write $R = \langle r_1, \ldots, r_n \rangle, C = \langle c_1, \ldots, c_m \rangle$, where all the components are representatives of orbits.
Again a cell is $T_{ij}: \langle r_i \rangle \times \langle c_j \rangle \to 2$
This is not necessarily a single boolean
But still finitely many output, so we can fill it with MQs.
<table>
<thead>
<tr>
<th></th>
<th>_</th>
<th>&lt;Lo&gt;</th>
<th>&lt;Li(0)&gt;</th>
<th>&lt;S(0,0)&gt;</th>
<th>&lt;S(0,1)&gt;</th>
<th>&lt;S(0,0)Lo&gt;</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>_</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&lt;Lo&gt;</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&lt;Li(0)&gt;</td>
<td>0</td>
<td>1</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/1</td>
<td>...</td>
</tr>
<tr>
<td>&lt;S(0,0)&gt;</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>&lt;S(0,1)&gt;</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>&lt;LoLo&gt;</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Lemma 4. Assume that \((S, E, T)\) is a closed, consistent observation table. Suppose the acceptor \(M(S, E, T)\) has \(n\) states. If \(M' = (Q', q'_0, F', \delta')\) is any acceptor consistent with \(T\) that has \(n\) or fewer states, then \(M'\) is isomorphic to \(M(S, E, T)\).

Following Angluin’s original proof:

Lemma 4. Let \(H\) be the automaton associated with a closed and consistent table \((S, E)\). If \(M'\) is an automaton consistent with \((S, E)\) (meaning that \(se \in \mathcal{L}(M') \iff se \in \mathcal{L}(H)\) for all \(s \in S \cup S \cdot A\) and \(e \in E)\) and \(M'\) has at most as many orbits as \(H\), then there is a surjective map \(f : Q_{M'} \to Q_H\). If moreover

- \(M'\)’s dimension is bounded by the dimension of \(H\), i.e. \(\text{supp}(m) \subseteq \text{supp}(f(m))\) for all \(Q'_M\), and
- \(M'\) has no fewer local symmetries than \(H\), i.e. \(\pi \cdot f(m) = f(m)\) implies \(\pi \cdot m = m\) for all \(m \in Q'_M\),

then \(f\) defines an isomorphism \(M' \cong H\) of nominal DFAs.
THE GOOD PARTS

Correctness was easy
Brings almost nothing new to the table
Easy to implement variations
For example, different counter example analysis

Works for any symmetry $Aut(M)$ for any $\omega$-categorical model $M$
For example homogeneous structures: $(\mathbb{N}, =)$, $(\mathbb{Q}, \leq)$, $(\mathbb{R}, adj)$, ...
Needed for products to be orbit-finite
THE BAD PARTS

Implementation was not easy
(Partly because the library for nominal computation was young)

Implementation is very slow

No generalization to $\text{Aut}(\mathcal{M})$ if $\mathcal{M}$ is not $\omega$-categorical
Problem here is that we cannot fill the table in a finite way
THANKS FOR BEING HERE! 😊