Passive Inference of Register Automata

Philipp Haselwarter *

research internship under the supervision of Peter Habermehl

June 27, 2013

1 Introduction

The passive inference of a formal language is the construction of a grammar from a positive and a negative sample. For regular languages, polynomial-time algorithms for this problem have been proposed by Lang [5] and Oncia and García [6] and refined by Dupont [1].

The class of register automata has been explored to a lesser extent. The case of active learning, where an oracle can be queried for membership of a string, and equality has been investigated by Howar et al. [3]. We extend Dupont’s presentation of the RPNI algorithm [1] to provide a passive inference method for register automata: infRA.

2 Theoretical Framework

In this section we present the data language, the automata model and the auxiliary definitions used in the description of the learning algorithm.

2.1 Language Inference Primer

Let \( \mathcal{A} = (Q, q_0, F, \delta) \) be a deterministic finite automaton (DFA) over an alphabet \( \Sigma \). Its language is denoted by \( \mathcal{L}(\mathcal{A}) \); the empty string is written as \( \epsilon \). In passive inference, the input is under the form of a positive sample \( I^+ \) and a negative sample \( I^- \), such that \( I^+ \subseteq \mathcal{L}(\mathcal{A}) \land I^- \cap \mathcal{L}(\mathcal{A}) = \emptyset \).

*philipp@haselwarter.org
Definition 1. The prefix tree acceptor associated to a set of words $I$ is defined as the tree-shaped deterministic finite automaton obtained by creating, for every word $w \in I$, a new state after each letter of $w$.

For example, in the case of $I = \{aba, abba\}$, the associated PTA($I$) is shown in fig. 1. We observe that the language accepted by PTA($I$) is the smallest (in the sense of inclusion) language containing $I$, in fact $L(PTA(I)) = I$.

Let $P(Q)$ be the set of partitions of $Q$ and $\pi, \pi' \in P(Q)$.

Definition 2. The quotient automaton $A/\pi$ of an automaton $A$ with regards to a partition of its states $\pi \in P(Q_A)$ is defined as:

$A/\pi = (Q' = \pi,$
$q_0' = B \in \pi, q_0 \in B,$
$F/\pi = \{B \in Q' | B \cap F \neq \emptyset\},$
$\delta': Q' \times \Sigma \rightarrow 2^{Q'}$
$\delta' = \{B' \in \delta'(B, \alpha) \iff \exists q \in B, \exists q' \in B', \delta(q, \alpha) = q'\}$

Intuitively, this corresponds to merging some of the states of the automaton while maintaining their transitions, with initial and final states transmitting their properties to the block they get merged into.

A partition $\pi'$ derives from $\pi$ if $\forall B' \in \pi', \exists B \in \pi, B' \subseteq B \land \pi' \neq \pi$, i.e. $\pi$ is a strict refinement of $\pi'$ and we write $\pi \ll \pi'$. As the relation $\ll$ defines a partial order with least upper bound and greatest lower bound for any state partition, the set of quotient automata obtained from one automaton $A$ can be (partially) ordered by $\ll$, to form a lattice $Lat(A)$. By construction of the quotient automaton, $A \ll A/\pi \implies L(A) \subset L(A/\pi)$ [2], so successively deriving quotient automata generalizes the initial automaton's language.

2.2 Finite-Memory Automata and Data Languages

Our aim is to learn languages of data words of the form $(\alpha, d)^*$ where $\alpha \in A$ for $A$ finite and $d \in D$ for some unbounded data domain over which we can
test for equality. Equivalence of data words is defined modulo permutation over \( \mathbb{D} \), so for example \((a, 5)(b, 7)(a, 5) \cong (a, 3)(b, 8)(a, 3) \not\cong (a, 4)(a, 4)(a, 4)\).

To model these languages we use finite-memory automata (FMA) as proposed by Kaminski and Francez [4].

**Definition 3 (FMA).** A finite-memory automaton \( A \) defined over a domain \((A, \mathbb{D})\) is a tuple \( \langle Q, q_0, \delta, F, R \rangle \).

- \( Q \) : a finite set of states
- \( q_0 \in Q \) : the initial state
- \( F \subset Q \) : the set of final states
- \( \delta : Q \times A \times \mathbb{D} \rightarrow Q \times \{ r, w \} \times [0, k-1] \) : the transition function
- \( R = \{ r_0, \ldots, r_{k-1} \} \subset (\mathbb{D} \cup \{\#\})^k \) : a set of registers, initialized to \( \# \notin \mathbb{D} \)

The semantics of the transition function are as follows. Upon reading input \((\alpha, d) \in (A, \mathbb{D})\), if there exists a register \( i \), such that \( r_i = d \), the automaton must perform a reading operation on that register, if there is no such register, it must perform a writing operation into some register. We write \( \delta(q, \alpha, d) = (q', \sigma, i) \) where \( q, q' \in Q \), \( \sigma \in \{r, w\} \), \( i \in [0, k-1] \).

In order to reduce the number of permutations of FMA’s with equivalent languages, we choose to normalize our definition by requiring the automaton to write into the lowest register containing \( \# \), as long as such register exist. This entails that a data value can occur in at most one register at a time.

**Example 1.** The FMA in fig. 2 shows a 2-register automaton \( A_2 \) over \((A \supseteq \{a\}, \mathbb{D} = \mathbb{N})\). It accepts the language \( L(A_2) \) of data words of length at least two that start and end with the same data value, with any different value allowed between them. Such words include \((a, 42)(a, 42)\) or \((a, 0)(a, 1)(a, 2)\ldots(a, 42)(a, 0)\), but not \((a, 7)(a, 5), (a, 0)(a, 0)(a, 0)\) or \((a, 0)(b, 0)\).

If all the zeros and ones on the transitions were swapped, the language would obviously remain unchanged, but the FMA would no longer be normal.

### 2.3 Transition Words

In order to build a FMA from a sample from the data-language, we will require an operational description of a FMA run. This amounts to the list of transitions used. As we cannot know the states of an unknown FMA a priori, we suppose that after every transition a new state is reached, ie. the runs are without loops. The description of the states becomes thus implicit and will be omitted for the sake of a more concise notation.
Definition 4. The transition word associated to a series of transitions is defined as:

\[
TW(\langle \delta(q, \alpha, d) = (q', \sigma, i) \rangle \langle \delta(q', \alpha', d') = (q'', \sigma', i') \rangle \cdots) = (\alpha, \sigma, i)(\alpha', \sigma', i') \cdots
\]

Every transition word \( w_\delta \) defines thus a linear FMA whose transition function is reduced to the operations occurring in \( w_\delta \), each transition effectively reaching a new state. If the FMA associated to a transition word \( w_\delta \) uses \( k \) different registers, we refer to it as a \( k \)-register transition word and note it \( w_\delta^2 \).

By extension, we define the language of a transition word \( w_\delta \) to be exactly the language \( L(w_\delta) \) accepted by its associated linear FMA.

Example 2. The run of the FMA in fig. 2 on the input \( w_D = (a, 42)(a, 7)(a, 42) \) is described operationally by \( w_\delta^2 = (a, w, 0)(a, w, 1)(a, r, 0) \). Its associated linear FMA fig. 3 looks like the original automaton with the loop over \( q_1 \) unrolled and transitions that have not been used in the run removed.

3 The infRA Algorithm

The learning of a FMA is divided into two phases. First, the positive sample \( I_D^+ \) is transformed into a set of transition words \( I_\delta^+ \). This eliminates the data coming from \( D \) and provides us with a finite representation of \( I_\delta^+ \). If a sample \( I_\delta^+ \) compatible with \( I_D \) is found, it can be learned using a modified technique from regular inference.
3.1 Towards a Finite Representation

In order for us to be able to treat the input, we need to transform the positive sample into building blocks for a FMA. For each data word in $I^+_D$, several k-register transition words may exist.

**Procedure 1: TW_of_dataword**

**Data:** $k, w_D$

**Result:** \{ $w^k_δ$ : normalized k-register transition word $|$ $w_D \in \mathcal{L}(w^k_δ)$ \}

As a FMA with $k$ registers can always use less than $k$, in TW_of_sample, for every data word, the transition words using up to $k$ different register words are collected. Those transition words that would lead to accepting words from the negative sample must be pruned from the transition-sample. But if for any word $w_D \in I^+_D$, this means that all its k-TWs are pruned, then it won’t be possible to construct a k-transition sample $I^+_δ$ that accepting $w_D$. In that case the empty set, ie. failure, is returned.

**Algorithm 2: TW_of_sample**

**Data:** $k, I = (I^+_D, I^-_D)$

**Result:** the set of 1-to-k-register transition words compatible with $I$

$I^+_δ \leftarrow \emptyset$

foreach $w_D \in I^+_D$ do

$W_δ \leftarrow \bigcup_{1 \leq i \leq k} \text{TW_of_dataword}(i, w_D)$

$W_δ \leftarrow W_δ \setminus \{w_δ \in W_δ \mid \exists w_D \in I^-_D, w_D \in \mathcal{L}(w_δ)\}$

if $W_δ = \emptyset$ then

return $\emptyset$

end

$I^+_δ \leftarrow I^+_δ \cup W_δ$

end

return $I^+_δ$

3.2 Learning a FMA from its Operational Behaviour

Given a set of transition words $I^+_δ$, we can adopt a dual view: It can either be seen as description of linear FMAs, as by definition 4, or it can be seen as a set of words over the finite alphabet $\Sigma = \mathbb{A} \times \{r, w\} \times [0, k - 1]$.

In procedure 3, we adopt the latter point of view. For a given non-deterministic finite automaton working over $\Sigma$, it returns the set of states that can be reached non-deterministically, ie. through more than one path.
for a single word.

**Procedure 3: non_det_states**

*Data:* \(A\): a NFA  
*Result:* \(\{Q_{nd} \subseteq Q_A \mid \exists u \in \Sigma^*, Q_{nd} = \delta_A^*(u) \land |Q_{nd}| > 1\}\)

Algorithm 4 builds a deterministic finite automaton from a non-deterministic one by successively merging all of its non-deterministically reachable states. Note that in contrast to the powerset construction, this process does not usually yield a language-equivalent automaton, but rather generalizes the language: \(L(A) \subseteq L(\text{det_merge}(A))\).

**Algorithm 4: detem_merge**

*Data:* \(A\): a NFA  
*Result:* a DFA  
\(\text{non\_det\_stack} \leftarrow \text{non\_det\_states}(A)\)  
\(\pi \leftarrow Q_A\)  
while not empty(\(\text{non\_det\_stack}\)) do  
\(\text{non\_det\_blocks} \leftarrow \text{pop}(\text{non\_det\_stack})\)  
\(\pi \leftarrow \pi \setminus \text{non\_det\_blocks} \cup \bigcup_{B \in \text{non\_det\_blocks}} B\)  
\(A \leftarrow A/\pi\)  
push(\(\text{non\_det\_stack}, \text{non\_det\_states}(A)\))  
end  
return \(A\)

**Procedure 5: compatible**

*Data:* \(A\): a FMA, \(I_D^-\): a set of data words  
*Result:* \(\text{true}\) iff \(A\) is a well-defined FMA and does not accept any word in the sample \(I_D^-\)  
return \(\bigwedge_{w_D \in I_D^-} w_D \notin L(A)\)

The RAPNI\(^1\) algorithm is closely related to the RPNI\(^2\) algorithm [1], as it

---

\(^1\)Register Automaton Positive and Negative Inference  
\(^2\)Regular Positive and Negative Inference
mostly adopts the finitistic view of the sample. It proceeds by incrementally searching \( \text{Lat}(PTA(I_\delta^+)) \). The difference lies in the selection of an automaton respecting the FMA semantics and the call to compatible, which only removes incompatible elements from the search space. Therefore we can rely on it to find the maximum generalization of \( I_\delta^+ \) and identify the target language \([6]\), while complying with \( I_D^- \).

**Procedure 6: split_FMA**

**Data:** \( \mathcal{A} \): a non-deterministic FMA

**Result:** the set of semantically correct FMAs that can be extracted from \( \mathcal{A} \) by picking a choice

### 3.3 Fitting the Pieces together

By the definition of the semantics of FMA, every data value can occur at most in one register at a time. This gives us an upper bound \( k_{\text{max}} \) to the number of registers a FMA can require to identify a language. The infRA algorithm proceeds by incrementally traversing the search space of \( k \)-register automata until a compatible finite representation \( I_\delta^+ \) is found and then relies on RAPNI to generalize \( I_\delta^+ \). If no such representation is found, no FMA can exist that accepts \( I_D^+ \) but rejects all of \( I_D^- \).

**Algorithm 8: infRA**

**Data:** \( I = (I_D^+, I_D^-) \)

**Result:** FMA compatible with \( I \)

\( k_{\text{max}} \leftarrow \) maximum number of different data values in any word \( \in I_D^+ \)

for \( (k \leftarrow 1, k \leq k_{\text{max}}, k++) \) do

\( I_\delta^+ \leftarrow \text{TW_of_sample}(k, I_D^+, I_D^-) \)

if \( I_\delta^+ \neq \emptyset \) then

\( \mathcal{A} \leftarrow \text{RAPNI}(I_\delta^+, I_D^-) \)

if \( \mathcal{A} \neq \bot \) then

return \( \mathcal{A} \)

end

end

return \( \bot \)
Algorithm 7: RAPNI

Data: $I^+_\delta$, $I^-_D$
Result: a register-minimal compatible FMA

$A \leftarrow$ PTA($I^+_\delta$)
$\pi \leftarrow Q_A$

for $i \leftarrow 1$ to $|Q_A|$ do
  for $j \leftarrow 0$ to $i - 1$ do
    $\pi' \leftarrow \pi \setminus \{B_i, B_j\} \cup \{B_i \cup B_j\}$
    $\pi'' \leftarrow$ determ_merge($A/\pi'$)
    foreach $A' \in$ split_FMA($A/\pi''$) do
      if compatible($A'$, $I^-_D$) then
        $A \leftarrow A'$
        break
      end
    end
  end
if compatible($A$, $I^-_D$) then
  return $A$
else
  return $\bot$
end
4 Example application

Let $A = \{a \ldots z\}$, $D = \mathbb{N}$, $I^+_D = \{(a,5)(a,5), (a,3)(a,7)(a,7)(a,3)\}$, $I^-_D = \{\epsilon, (a,5), (a,5)(a,7)(a,7)(a,8), (a,5)(a,7)(a,8)(a,8)\}$. At most two data values of the positive sample differ, thus $k_{max} = 2$.

4.1 Generating a finite representation

$\text{TW}_D(k, w_D)$:

\[
\begin{array}{c|c|c}
& k & 1 & 2 \\
(a,5)(a,5) & \{(a,w,1)(a,r,1)\} & \emptyset \\
(a,3)(a,7)(a,7)(a,3) & \{(a,w,1)(a,r,1)(a,w,1)\} & \{(a,w,1)(a,w,2)(a,r,2)(a,r,1)\}
\end{array}
\]

Observe that $I^-_D \cap \mathcal{L}((a, w, 1)(a, w, 1)(a, r, 1)(a, w, 1)) = \{(a,5)(a,7)(a,8)\}$. Therefore, $\text{TW}_D(1, I^+_D, I^-_D) = \emptyset$ and $I^+_D \leftarrow \text{TW}_D(2, I^+_D, I^-_D) = \{(a, w, 1)(a, w, 2)(a, r, 2)(a, r, 1)\}$. We can now attempt to learn this sample.

4.2 Learning a compatible FMA

Figure 4: The prefix tree acceptor $\text{PTA}(I^+_D)$

Figure 5: Merging states 0 and 1 creates a semantically incorrect FMA
Merging \{0, 2\} (respectively \{1, 2, 2'\}) would lead to accepting \(\epsilon\) (respectively \((a,5)\)), both of which are in the negative sample.

Figure 6: Merging \{0, 2\} accepts \((a,5)(a,7)(a,8)(a,8) \in I_D^{-}\)

Figure 7: Merging \{1, 2\} is compatible

Merging \{0, 2'\} (respectively \{1, 2, 2'\}) would lead to accepting \(\epsilon\) (respectively \((a,5)\)), both of which are in the negative sample.

Figure 8: Merging \{1, 2, 3\} is compatible, creating non-determinism on the finitistic level between 4 and 2'

Figure 9: Merging \{2', 4\} is compatible

Any further merging would lead to acceptance of part of the negative sample.
5 Conclusion

In this report, we propose a technique for passive learning of Kaminski-Francez style register automata. Like in the case of regular inference, a sufficient sample size allows us to find an interesting generalization of the input, using a minimal number of registers. But for the lack of a widely accepted definition of what constitutes a canonical or even just a minimal acceptor for a data language, this result is not as universal as its equivalent for DFAs. Further investigations could explore how the result relates to different models proposed as canonical. Such an adjustment to the algorithm would probably take place in the selection of a compatible candidate in RAPNI or when treating the FMA-non-determinism in split_FMA.

Defining a canonical FMA is of particular interest, because it would allow to bring into our setting the notion of characteristic sample, i.e., describing the minimal positive and negative sample that is required to identify a given language. The example in section 4 succeeds in reconstructing the FMA shown in fig. 2 from a very small sample size, which is an encouraging result.

References


