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

[*] philipp@haselwarter.org

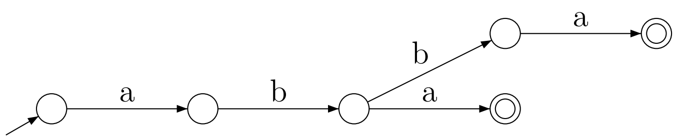

Figure 1: A prefix tree acceptor $PTA(\{aba, abba\})$

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 $\mathcal{L}(PTA(I)) = I$.

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

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

$$\begin{aligned}\mathcal{A}/\pi = \langle Q' = \pi,\\ &q'_0 = B \in \pi, q_0 \in B,\\ &F/\pi = \{B \in Q' | B \cap F \neq \emptyset\},\\ &\delta' : Q' \times \Sigma \to 2^{Q'}\\ &\delta' = \{B' \in \delta'(B, \alpha) \iff \exists q \in B, \exists q' \in B', \delta(q, \alpha) = q'\}\rangle\end{aligned}$$

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 π' *derives from* π if $\forall B' \in \pi', \exists B \in \pi, B' \subseteq B \wedge \pi' \neq \pi$, ie. π is a strict refinement of π' 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 $\mathcal{A}$ can be (partially) ordered by $\ll$ to form a lattice $Lat(\mathcal{A})$. By construction of the quotient automaton, $\mathcal{A} \ll \mathcal{A}/\pi \implies \mathcal{L}(\mathcal{A}) \subset \mathcal{L}(\mathcal{A}/\pi)$ [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 $\alpha \in \mathbb{A}$ for $\mathbb{A}$ finite and $d \in \mathbb{D}$ 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 $(a,5)(b,7)(a,5) \cong (a,3)(b,8)(a,3) \ncong (a,4)(a,4)(a,4)$.

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

Definition 3 (FMA)**.** A finite-memory automaton $\mathcal{A}$ defined over a domain $(\mathbb{A}, \mathbb{D})$ 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
- $\delta : Q \times \mathbb{A} \times \mathbb{D} \to Q \times \{r,\ w\} \times [0, k-1]$ **:** 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 $(\alpha, d) \in (\mathbb{A}, \mathbb{D})$, 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 $\mathcal{A}_2$ over $(\mathbb{A} \supseteq \{a\}, \mathbb{D} = \mathbb{N})$. It accepts the language $\mathcal{L}(\mathcal{A}_2)$ 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.

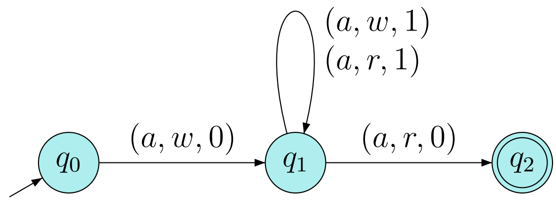

Figure 2: A 2-register FMA

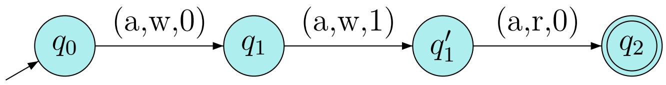

Figure 3: A linear FMA

Definition 4. The transition word associated to a series of transitions is defined as:

$$TW(\langle\delta(q,\alpha,d) = (q',\sigma,i)\rangle\ \langle\delta(q',\alpha',d') = (q'',\sigma',i')\rangle\ \dots) = (\alpha,\sigma,i)(\alpha',\sigma',i')\dots$$

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

By extension, we define the language of a transition word w_δ to be exactly the language $\mathcal{L}(w_\delta)$ 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_δ^+. This eliminates the data coming from $\mathbb{D}$ and provides us with a finite representation of I_D^+. If a sample I_δ^+ 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: $\{w_\delta^k$: normalized k-register transition word $\mid w_D \in \mathcal{L}(w_\delta^k)\}$

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
$I_\delta^+ \leftarrow \emptyset$
foreach $w_D \in I_D^+$ **do**
  $W_\delta \leftarrow \bigcup_{1 \le i \le k}$ `TW_of_dataword`$(i,\ w_D)$
  $W_\delta \leftarrow W_\delta \setminus \{w_\delta \in W_\delta \mid \exists w_D \in I_D^-,\ w_D \in \mathcal{L}(w_\delta)\}$
  if $W_\delta = \emptyset$ **then**
    return $\emptyset$
  end
  $I_\delta^+ \leftarrow I_\delta^+ \cup W_\delta$
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 Σ, 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: $\mathcal{A}$: a NFA
Result: $\{Q_{nd} \subset Q_{\mathcal{A}} \mid \exists u \in \Sigma^*, Q_{nd} = \delta^*_{\mathcal{A}}(u) \wedge |Q_{nd}| > 1\}$

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: $\mathcal{L}(\mathcal{A}) \subset \mathcal{L}(\texttt{det\_merge}(\mathcal{A}))$.

Algorithm 4: `determ_merge`

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

Procedure 5: `compatible`

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

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 $Lat(PTA(I_\delta^+))$. 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_δ^+ 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_{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_δ^+ is found and then relies on `RAPNI` to generalize I_δ^+. 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_max ← maximum number of different data values in any word ∈ I_D^+
for (k ← 1, k ≤ k_max, k++) do
    I_δ^+ ← TW_of_sample(k, I_D^+, I_D^-)
    if I_δ^+ ≠ ∅ then
        A ← RAPNI(I_δ^+, I_D^-)
        if A ≠ ⊥ then
            return A
        end
    end
end
return ⊥
```

Algorithm 7: `RAPNI`

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

```
𝒜 ← PTA(I⁺_δ)
π ← Q_𝒜
for i ← 1 to |Q_𝒜| do
    for j ← 0 to i − 1 do
        π′ ← π \ {B_i, B_j} ∪ {B_i ∪ B_j}
        π″ ← determ_merge(𝒜/π′)
        foreach 𝒜′ ∈ split_FMA(𝒜/π″) do
            if compatible(𝒜′, I⁻_D) then
                𝒜 ← 𝒜′
                break
            end
        end
    end
end
if compatible(𝒜, I⁻_D) then
    return 𝒜
else
    return ⊥
end
```

4 Example application

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

4.1 Generating a finite representation

`TW_of_dataword`(k, w_D):

$w_D \in I_D^+$	k 1	2
$(a,5)(a,5)$	$\{(a,w,1)(a,r,1)\}$	$\emptyset$
$(a,3)(a,7)(a,7)(a,3)$	$\{(a,w,1)(a,w,1)(a,r,1)(a,w,1)\}$	$\{(a,w,1)(a,w,2)(a,r,2)(a,r,1)\}$

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

4.2 Learning a compatible FMA

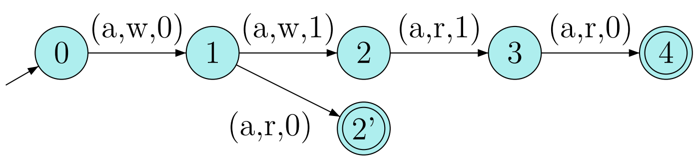

Figure 4: The prefix tree acceptor `PTA`(I_δ^+)

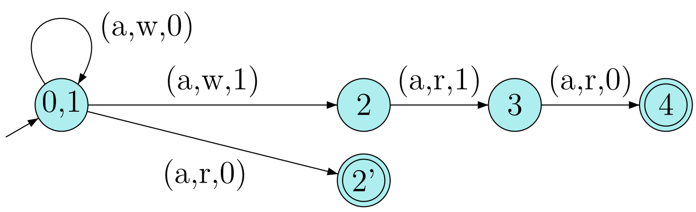

Figure 5: Merging states 0 and 1 creates a semantically incorrect FMA

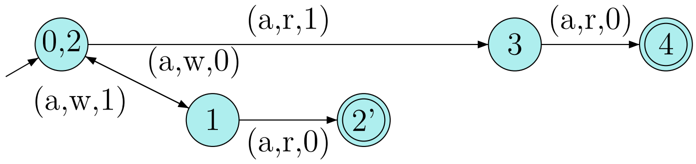

Figure 6: Merging {0, 2} accepts $(a, 5)(a, 7)(a, 8)(a, 8) \in I_D^-$

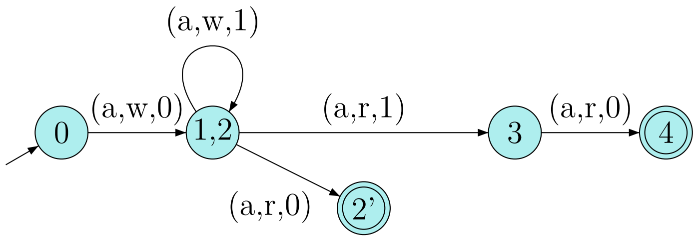

Figure 7: Merging {1, 2} is compatible

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

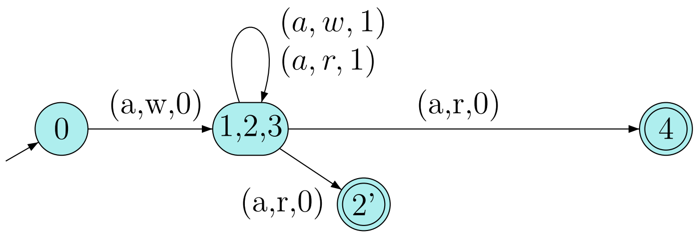

Figure 8: Merging $\{1, 2, 3\}$ is compatible, creating non-determinism on the finitistic level between 4 and 2’

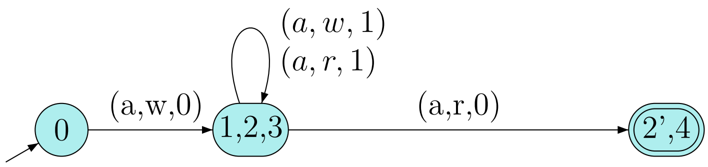

Figure 9: Merging {2’, 4} is compatible

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*, ie. 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

[1] Pierre Dupont. Incremental regular inference. In *Proceedings of the Third ICGI-96*, pages 222–237. Springer, 1996.

[2] K S Fu and T L Booth. Grammatical inference: Introduction and survey-part i. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(3):343–359, March 1986.

[3] Falk Howar, Bernhard Steffen, Bengt Jonsson, and Sofia Cassel. Inferring Canonical Register Automata. In Viktor Kuncak and Andrey Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 7148, pages 251–266. Springer Berlin / Heidelberg, 2012. 10.1007/978-3-642-27940-9_17.

[4] Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

[5] Kevin Lang. Random dfa's can be approximately learned from sparse uniform examples. In *In Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 45–52. ACM Press, 1992.

[6] P. Oncina, J.; García. *Inferring regular languages in polynomial update time*, chapter -. World Scientific Publishing, 1992.