Incremental Regular Inference

Pierre Dupont

Carnegie Mellon University
Pittsburgh, PA 15213, USA
E-mail : pdupont@cs.cmu.edu

Abstract

In this paper, we extend the characterization of the search space of regular inference [DMV94] to sequential presentations of learning data. We propose the RPNI2 algorithm, an incremental extension of the RPNI algorithm. We study the convergence and complexities of both algorithms from a theoretical and practical point of view. These results are assessed on the Feldman task.

1 Introduction

Regular inference is the problem of learning a regular language from a positive sample, that is, a finite set of strings supposed to be drawn from a target language. Whenever a negative sample, that is, a finite set of strings not belonging to the target language, is also available, the problem may be solved by the RPNI algorithm[1] proposed by Oncina and García [OG92] and, independently, by Lang [Lan92].

The RPNI algorithm has been shown to identify in the limit any regular language with polynomial complexity as a function of the positive and negative sample sizes. However, this algorithm requires the positive and negative information to be presented as a whole, that is, it uses a *presentation from given data.* If new learning data become available, the inference process must be restarted from scratch. The objective of the present work is to overcome this limitation by extending the RPNI algorithm to deal with *sequential presentation.* In this case, the learning data are presented one at a time in a random order, each example being labeled either as positive or negative.

In section 2, we recall the theoretical framework of regular inference by specifying its *search space* and its *border set*, that is, the limit of possible generalizations of a positive sample under the control of a negative sample [DMV94]. We also discuss identification in the limit for the two data presentations.

Section 3 is devoted to the description of the RPNI algorithm. We formally define the notion of a characteristic sample that guarantees the correct identification of an unknown language. Two related theorems are presented which characterize the solution produced by this algorithm.

[1] RPNI stands for Regular Positive and Negative Inference.

We pursue the characterization of the search space proposed in [DMV94] for a sequential data presentation. Some interesting properties of an incrementally constructed search space are proved in section 4.

In section 5 we propose the RPNI2 algorithm[2], the incremental extension of the RPNI algorithm. Here, we observe that the asymptotic complexities of both algorithms are equal while a practical complexity reduction may be expected with RPNI2. This expectation is confirmed by experimental results detailed in section 6.

2 Theoretical Framework[DMV94]

The reader familiar with the classical definitions of automata theory and regular inference may omit section 2.1. We introduce these notions here for the sake of completeness.

2.1 Definitions and Notations

Let Σ denote a finite alphabet, let u, v, w denote elements of Σ^*, i.e. strings over Σ, and let λ denote the empty string. Let $|u|$ denote the length of the string u. We say that u is a *prefix* of v if there exists w such that $uw = v$. We say that u is a *suffix* of v if there exists w such that $wu = v$. A language L is any subset of Σ^*. Let $Pr(L) = \{u | \exists v, uv \in L\}$ denote the set of prefixes of L. Let $L/u = \{v | uv \in L\}$ denote the *right-quotient* of L by u. This is also called the *set of tails* of u in L. We have $L/u \neq \phi$ if and only if $u \in Pr(L)$.

A *finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of *states*, Σ is an *alphabet*, δ is a *transition function*, i.e. a mapping from QxΣ to 2^Q, q_0 is the *initial state* and F is a subset of Q identifying the *final* or *accepting states*. If for any q in Q and any a in Σ, $\delta(q, a)$ has at most one member (exactly one member) the automaton A is said to be *deterministic* (respectively, *complete)*. Henceforth we shall denote using DFA (NFA) a deterministic (respectively, a non-deterministic) finite automaton.

If $A = (Q, \Sigma, \delta, q_0, F)$ is an automaton, the automaton A/π *derived from* A *with respect to the partition* π of Q, also called the *quotient automaton* $A/\pi = (Q', \Sigma, \delta', B(q_0, \pi), F')$, is defined as follows:

$Q' = Q/\pi = \{B(q, \pi) | q \in Q\}$, $F' = \{B \in Q' | B \cap F \neq \phi\}$,

$\delta' : Q'$x$\Sigma \rightarrow 2^{Q'} : \forall B, B' \in Q', \forall a \in \Sigma, B' \in \delta'(B, a) \Leftrightarrow \exists q, q' \in Q, q \in B, q' \in B'$ and $q' \in \delta(q, a)$.

The states of Q belonging to the same block B of the partition π are said to be *merged* together.

Let I_+ (respectively, I_-) denote some *positive sample (*respectively, *negative sample)* of an unknown language[3] L. Let $A(L)$ denote the *canonical acceptor* of L, i.e. the *minimal DFA* accepting L. We say that an automaton is *compatible* with a positive sample (with a negative sample) if it accepts (respectively, rejects) any string from the sample. Let $PTA(I_+)$ denote the *prefix tree acceptor* built

[2] **RPNI2** stands for *Regular Positive and Negative Incremental Inference.*

[3] By default, "language" refers here to a *regular* language.

from a positive sample I_+ and let $MCA(I_+)$ denote the *maximal canonical acceptor* of I_+.

Let $P(A)$ denote the set of partitions of the state set of an automaton A. Let $r(\pi_i)$, or simply r_i, denote the number of blocks of the partition π_i. Let $\pi_1 = \{B_{11}, \ldots, B_{1r_1}\}$ and π_2 be two partitions of $P(A)$. We say that π_2 *directly derives from* π_1 if the partition π_2 is constructed from π_1 as follows: $\pi_2 = \{B_{1j} \cup B_{1k}\} \cup (\pi_1 \setminus \{B_{1j}, B_{1k}\})$, for some j, k between 1 and r_1, $j \neq k$. Consequently, $r_2 = r_1 - 1$.

This derivation operation defines a partial order relation on $P(A)$, which we shall denote $\preceq$. In particular, we have $\pi_1 \preceq \pi_2$. Let $\ll$ denote its transitive closure. In other words, $\pi_i \ll \pi_j$ if and only if π_i is finer than π_j. By extension, we say that A/π_i is *finer than* A/π_j and that A/π_j *derives from* A/π_i. The set of automata partially ordered by the relation $\preceq$ is a lattice, which we shall denote $Lat(A)$. In particular, we shall consider $Lat(PTA(I_+))$, the lattice of automata derived from the $PTA(I_+)$.

By construction of a quotient automaton, we have the language inclusion property which may be stated as follows [FB75]: $A/\pi_i \ll A/\pi_j$ if $L(A/\pi_i) \subseteq L(A/\pi_j)$. In other words, the derivation operation gives rise to a *language generalization* in a lattice of automata.

We say that I_+ is *structurally complete* with respect to an automaton A accepting a language L if there exists an acceptance of the strings of I_+ by A such that every transition of A is exercised and every element of the final state set of A is used as accepting state.

The search space of possible solutions to the regular inference problem is defined by the following theorem [Ang82, DMV94].

Theorem 2.1 *Let I_+ be a positive sample of any regular language L and let $A(L)$ be the canonical automaton accepting L. If I_+ is structurally complete with respect to $A(L)$ then $A(L)$ belongs to $Lat(PTA(I_+))$.*

In other words, if we assume the structural completeness of the positive sample with respect to an unknown canonical automaton, this automaton may be found by searching the lattice $Lat(PTA(I_+))$.

The *border set* $BS_{PTA}(I_+, I_-)$ is the set of automata which are at a *maximal depth* in the lattice [DMV94], that is, the set of automata which are compatible with I_+ and I_-, and from which there are no compatible automata that can be derived. The border set of $Lat(PTA(I_+))$ is the set of automata which correspond to the limit of generalization, by merging states of the $PTA(I_+)$, under the control of the negative sample. The *minimal DFA consistency problem*, i.e. finding the smallest DFA compatible with I_+ and I_-, can then be viewed as the discovery of the smallest DFA in $BS_{PTA}(I_+, I_-)$.

The minimal DFA consistency problem has been shown to be NP-hard [Ang78, Gol78] in the general case. However, as detailed in section 3, this problem may be solved by the RPNI algorithm if the sample is supposed to be *characteristic*. Note that the RPNI algorithm also offers a good generalization of the learning data even if the sample is not characteristic. This last point will be illustrated experimentally in section 6.

2.2 Language Identification in the Limit and Characteristic Samples

The language identification in the limit paradigm was introduced by Gold [Gol67]. For a sequential presentation of data, the identification in the limit is guaranteed if an inference method is guaranteed to converge to a correct representation of the language to be identified. Gold also proposed a related identification paradigm for fixed learning data [Gol78] called *identification from given data.* In this case, the learning set D is made of a pair of samples respectively positive and negative, that is, $D = (I_+, I_-)$. We restate Gold's definition to introduce the notion of *characteristic sample* $D^c = (I_+^c, I_-^c)$.

Definition 2.1 An inference method M *identifies* a language L *from given data* if there exists a sample $D^c = (I_+^c, I_-^c)$ such that for any data set $D = (I_+, I_-)$, with $I_+ \supseteq I_+^c$ and $I_- \supseteq I_-^c$, the hypothesis $H(D)$ proposed by the method M is correct, that is $L(H(D)) = L$. The sample D^c is called *characteristic* for the language L and the inference method M.

This definition gives a general characterization of an inference method. It does not prescribe, for a given method M and a language L, how to specify D^c. Since the canonical automaton $A(L)$ of a regular language is unique, the specification of a characteristic sample is usually based on this automaton. However, an inference method which would identify a regular language represented by an NFA might require another specification.

The following theorem, proved for instance in [Dup96], states the equivalence of the two related identification paradigms.

Theorem 2.2 *An inference method M identifies a language L from given data if and only if it identifies the language L from a sequential presentation.*

3 The RPNI Algorithm

3.1 Characteristic Sample Specification

Given a regular language L, represented by its canonical automaton $A(L)$, we shall detail the conditions that the learning data must satisfy to be characteristic for the language L and the RPNI algorithm. To this end, Oncina and García define the *short prefixes* and *kernel* of a language [OG92]. Henceforth, we will denote the standard order[4] on the strings of an alphabet Σ by the symbol "$<$ ".

3.1.1 The Set of Short Prefixes and the Kernel

Definition 3.1 The set of *short prefixes* $Sp(L)$ of a language L is defined as

$$Sp(L) = \{x \in \Pr(L) \ \mid \nexists u \in \Sigma^* \ \textit{with} \ L/u = L/x \ \textit{and} \ u < x\}$$

[4] According to the standard order the first strings on the alphabet $\Sigma = \{a, b\}$ are $\lambda, a, b, aa, ab, ba, bb, aaa, \ldots$

We know that in the canonical automaton $A(L)$ of the language L, there are as many states as the number of tails L/x of strings x belonging to the prefixes of L. The set of short prefixes is the set of the first strings in standard order, each of which leads to a particular state of the canonical automaton. Consequently, there are as many short prefixes as states in $A(L)$.

For example, the set of short prefixes of the language accepted by the automaton presented in figure 1 is $Sp(L) = \{\lambda, b\}$. Note that λ and b are the two first strings in standard order leading respectively to state 0 and state 1.

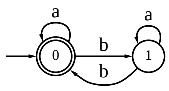

Figure 1: A canonical automaton.

Definition 3.2 The *kernel* $N(L)$ of the language L is defined as

$$N(L) = \{\lambda\} \cup \{xa \mid x \in Sp(L), a \in \Sigma, xa \in \Pr(L)\}$$

The kernel is made of the string λ and of the prefixes of the language corresponding to the short prefixes extended by one letter. By construction $Sp(L) \subseteq N(L)$. The kernel elements represent the transitions of the canonical automaton $A(L)$ since they are obtained by adding one letter to the short prefixes which represent the states of $A(L)$. Moreover, all elements of the kernel that also belong to the language represent the accepting states of $A(L)$. The kernel of the language of our example is $N(L) = \{\lambda, a, b, ba, bb\}$. The strings λ, a and bb belong to L ; they lead to an accepting state in the automaton presented in figure 1.

Note that if $|Q|$ denotes the number of states of the canonical automaton $A(L)$, the kernel has at most $1 + |Q| \cdot |\Sigma|$ elements.

Definition 3.3 A sample $D^c = (I_+^c, I_-^c)$ is *characteristic* for the language L and the algorithm RPNI if it satisfies the following conditions

1. *$\forall x \in N(L)$, **if** $x \in L$ **then** $x \in I_+^c$ **else** $\exists u \in \Sigma^*$ such that $xu \in I_+^c$.*
2. *$\forall x \in Sp(L), \forall y \in N(L)$ **if** $L/x \neq L/y$ **then** $\exists u \in \Sigma^*$ such that $(xu \in I_+^c$ and $yu \in I_-^c)$ or $(xu \in I_-^c$ and $yu \in I_+^c)$.*

Condition 1 guarantees that each element of the kernel belongs to I_+^c if it also belongs to the language or, otherwise, is prefix of a string of I_+^c. One can easily check that this condition implies the structural completeness of the sample I_+^c with respect to $A(L)$. In this case, theorem 2.1 assures that the automaton $A(L)$ belongs to the lattice $Lat(PTA(I_+^c))$. When an element x of the short prefixes and an element y of the kernel do not have the same set of tails $(L/x \neq L/y)$, they necessarily correspond to distinct states in the canonical automaton. In this case, condition 2 guarantees that a suffix u would distinguish them. In other words, the merging of a state corresponding to a short prefix x in the $PTA(I_+^c)$ with another state corresponding to an element y of the kernel is made incompatible by the existence of xu in I_+^c and yu in I_-^c or the converse.

There may exist several distinct characteristic samples for a given language L as several suffixes u may satisfy condition 1 or 2. Since the set of short prefixes contains $|Q|$ elements and the kernel has $\mathcal{O}(|Q| \cdot |\Sigma|)$ elements, the size of a characteristic sample is given by

$$|I_+^c| = \mathcal{O}(|Q|^2 \cdot |\Sigma|) \text{ and } |I_-^c| = \mathcal{O}(|Q|^2 \cdot |\Sigma|).$$

For a fixed alphabet Σ, this size only depends on the number of states of the canonical automaton to be identified.

One can verify that $D^c = (I_+^c, I_-^c)$, with $I_+^c = \{\lambda, a, bb, bba, baab, baaaba\}$ and $I_-^c = \{b, ab, aba\}$, forms a characteristic sample for the language accepted by the canonical automaton of figure 1.

3.2 Outline of the RPNI Algorithm

Algorithm RPNI

```
input I+, I-
output A DFA compatible with I+, I-

begin

// N is the number of states of PTA(I+)

π ← {{0}, {1}, ..., {N − 1}}             // One block for each prefix in the order <
A ← PTA(I+)

for i = 1 to |π| − 1                      // Loop on the blocks of partition π
   for j = 0 to i − 1                     // Loop on the blocks of lower rank
      π' ← π\{Bj, Bi}U{BiUBj}             // Merging of block Bi and block Bj
      A/π' ← derive (A, π')
      π'' ← determ_merge (A/π')
      if compatible (A/π'', I-) then      // Deterministic parsing of I-
        A ← A/π''
        π ← π''
        break                             // Break j loop
      end if
   end for                                // End j loop
end for                                   // End i loop
return A
end RPNI
```

The RPNI algorithm performs an ordered search in the lattice $Lat(PTA(I_+))$. By construction of the $PTA(I_+)$, each of its states corresponds to a unique prefix. The prefixes may be sorted according to the standard order. This order also applies on the prefix tree states. A partition in the state set of the $PTA(I_+)$ is made of an ordered block set, each block receiving the rank of its state of minimal rank. The RPNI algorithm proceeds in $N - 1$ steps where $N = \mathcal{O}(||I_+||)$ is the number of states[5] of $PTA(I_+)$. The partition $\pi(i)$ at step i is obtained by

[5] $||I_+||$ denotes the sum of the lengths of the strings belonging to I_+.

merging the two first blocks, according to the standard order, of the partition $\pi(i-1)$ at step $i-1$ such that $PTA(I_+)/\pi(i)$ is a compatible automaton.

The function *derive* (A, π') returns the quotient automaton A with respect to partition π'. The automaton A/π' may be non-deterministic. The function *determ_merge* (A/π') returns the partition π'' obtained by recursively merging all blocks of π', which creates a non-determinism. If A/π' is deterministic, the partition π'' is equal to partition π'. This is the "determinization by merging" operation. The function *compatible* $(A/\pi'', I_-)$ returns TRUE if the quotient automaton is compatible with the negative sample, FALSE otherwise.

Given that $PTA(I_+)$ has $\mathcal{O}(\|I_+\|)$ states, the time complexity of *determ_merge* is $\mathcal{O}(\|I_+\|)$. Moreover, since the quotient automaton A/π'' is necessarily deterministic, the compatibility checking with I_- has a $\mathcal{O}(\|I_-\|)$ complexity. Since the inside loop (on index j) is performed $\mathcal{O}(\|I_+\|^2)$ times, the global complexity of the RPNI algorithm is $\mathcal{O}((\|I_+\| + \|I_-\|) \cdot \|I_+\|^2)$.

3.3 Identification in the Limit

Theorem 3.1 *The RPNI algorithm identifies in the limit the class of regular languages.*

Two distinct proofs of this theorem may be found in [OG92] and in [Dup96].

Theorem 3.2 *When the learning data contains a characteristic sample for the language L, the canonical automaton $A(L)$ is the DFA having the least number of states and belonging to $BS_{PTA}(I_+, I_-)$. Consequently, $A(L)$ is the solution to the minimal DFA consistency problem.*

A proof of this theorem may be found in [Dup96].

This theorem characterizes the search space independently of the inference algorithm. As a result the definition of a characteristic sample becomes more general. In short, theorem 3.2 guarantees the existence of a notable element of the border set while theorem 3.1 guarantees that this element may be found by applying the RPNI algorithm.

The polynomial complexity of the RPNI algorithm and theorem 3.2 do not contradict the NP-hardness of the minimal DFA consistency problem. This is because in order to guarantee the identification, the RPNI algorithm must be provided with learning data that contain a characteristic sample.

4 Incremental Search Space

In this section, we study the structure of the search space for a sequential presentation of the learning data.

4.1 Induction Hypothesis

We suppose that the current solution $A(k)$ at step k is a DFA compatible with the first k data received. More precisely, the current positive sample $I_+(p)$ is defined by the first p examples, which may be stored as a prefix tree $PTA(I_+(p))$, and the current negative sample $I_-(q)$ is defined by the first q negative examples, with $k = p + q$. The solution $A(k)$ is supposed to belong to the border set $BS_{PTA}(I_+(p), I_-(q))$.

We can initialize this induction by applying the RPNI algorithm on the first example. If the first example is positive, the solution will be the universal automaton, otherwise the solution will be the empty automaton.

4.2 Incremental Lattice Construction

Property 4.1 defines the relation between successive lattices when a new positive example is received[6].

Property 4.1 $Lat(PTA(I_+(p))) \not\subseteq Lat(PTA(I_+(p+1)))$.

Proof : Both lattices could be completely distinct, i.e. $Lat(PTA(I_+(p))) \cap Lat(PTA(I_+(p+1))) = \phi$. When the $(p+1)^{th}$ example introduces a new symbol, the universal automaton on the new alphabet is not in $Lat(PTA(I_+(p)))$. Since the universal automaton may be derived from any automaton belonging to $Lat(PTA(I_+(p+1)))$, none of them belong to $Lat(PTA(I_+(p)))$. When, on the contrary, the $(p+1)^{th}$ example does not introduce a new symbol, the universal automaton belongs to both lattices, but $Lat(PTA(I_+(p))) \not\subseteq Lat(PTA(I_+(p+1)))$ because $PTA(I_+(p)) \notin Lat(PTA(I_+(p+1)))$. Indeed, $I_+(p+1)$ is not included in the language accepted by $PTA(I_+(p))$. Since the language accepted by any automaton in $Lat(PTA(I_+(p+1)))$ includes $I_+(p+1)$, $PTA(I_+(p))$ does not belong to this lattice. □

4.2.1 New Example Acceptance

Given a solution $A(k)$, the membership of a new example x to $L(A(k))$ may be checked and $A(k)$ may be extended by *the shortest suffix* of x. This operation consists in finding the longest prefix u of x accepted by the automaton $A(k)$. This acceptance defines a state q such that $q \in \delta^*(q_0, u)$. Since $A(k)$ is deterministic, the state q is unique. From state q, the automaton $A(k)$ is extended by the suffix v to accept $x = uv$.

Let $A^x(k)$ denote the automaton obtained by the shortest suffix extension of x from the automaton $A(k)$. By construction, $A^x(k)$ is deterministic. If x is already accepted by $A(k)$, the shortest suffix v is the empty string and hence $A^x(k) = A(k)$.

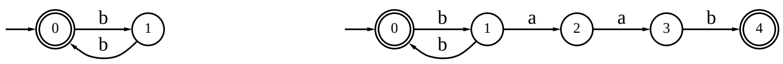

Figure 2: Shortest suffix extension.

Suppose that the current solution $A(k)$ is the automaton presented in figure 2 (left). Assume also that the new example is $bbbaab$. The longest accepted prefix is bbb, which leads to state 1. The resulting automaton, presented at figure 2 (right), is obtained by extension using the suffix aab from state 1.

The shortest suffix extension may cause an incompatibility with the current negative sample even if $A(k)$ is compatible. In our example, $L(A(k)) = (bb)^*$. The extension guarantees the acceptance of the string $bbbaab$ but also of any

[6] If a positive example has already been presented, the prefix tree acceptor remains unchanged, as does the associated lattice.

string of the form $b(bb)^*aab$. If, for instance, the string $baab$ belongs to the current negative sample, the extended solution is incompatible. The solution to this problem is presented in section 5.1.

The property 4.2 states that the shortest suffix extension guarantees that, if the current solution at step k belongs to the lattice built from the first p examples, then the extended automaton necessarily belongs to the lattice built by adding the new example.

Property 4.2 $A(k) \in Lat(PTA(I_+(p))) \Rightarrow A^x(k) \in Lat(PTA(I_+(p+1)))$, *with x being the $(p+1)^{th}$ example.*

Proof : By hypothesis, $A(k)$ is a quotient automaton of $PTA(I_+(p))$. Thus $I_+(p)$ is structurally complete with respect to $A(k)$. Let v be the shortest suffix corresponding to the extension of $A(k)$ to get $A^x(k)$. Let w be the shortest suffix corresponding to the extension of $PTA(I_+(p))$ to get $PTA(I_+(p+1))$. By construction, v is suffix of w and, consequently, $I_+(p+1)$ is structurally complete with respect to $A^x(k)$. In other words, $A^x(k) \in Lat(MCA(I_+(p+1)))$. Finally, since $A^x(k)$ is deterministic, it belongs to $Lat(PTA(I_+(p+1)))$. $\square$

Property 4.3 states that if the automaton $A(k)$, which, by induction hypothesis, belongs to the border set of the lattice obtained at step k, accepts the new example, then it also belongs to the border set of the new lattice.

Property 4.3 $A(k) \in BS_{PTA}(I_+(p), I_-(q))$ *and* $A(k) = A^x(k) \Rightarrow A^x(k) \in BS_{PTA}(I_+(p+1), I_-(q))$, *with x being the $(p+1)^{th}$ example.*

Proof : We have $A(k) = A^x(k)$ if and only if the string x belongs to the language accepted by $A(k)$. By induction hypothesis, $A(k)$ belongs to $BS_{PTA}(I_+(p), I_-(q))$, i.e. it is at a maximal depth in $Lat(PTA(I_+(p)))$. Since the negative sample has not been modified, $A(k)$ is also at a maximal depth in the new lattice $Lat(PTA(I_+(p+1)))$. $\square$

4.2.2 Dealing with a new Negative Example

The reception of a new negative example does not modify the search space since, by theorem 2.1, the solution is to be found in the lattice built from the prefix tree acceptor of the positive sample. On the other hand, the border set is generally modified at the reception of a new negative example. In other words, the current solution $A(k)$ belonging to $BS_{PTA}(I_+(p), I_-(q))$ may not belong to $BS_{PTA}(I_+(p), I_-(q+1))$. The property 4.4 states that if an automaton $A(k)$, which, by induction hypothesis, belongs to the border set at step k, does not accept the new negative example then it also belongs to the border set of the same lattice after treating the additional negative example.

Property 4.4 $A(k) \in BS_{PTA}(I_+(p), I_-(q))$ *and* $x \notin L(A(k)) \Rightarrow$
$A(k) \in BS_{PTA}(I_+(p), I_-(q+1))$, *with x being the $(q+1)^{th}$.*

Proof : Since the string x does not belong to the language accepted by $A(k)$ it does not modify the compatibility of $A(k)$ with respect to the negative sample. Since the automaton $A(k)$ is at a maximal depth after the first q negative examples, it remains at a maximal depth after receiving the new negative example. Consequently, $A(k)$ belongs to $BS_{PTA}(I_+(p), I_-(q+1))$.□

5 The RPNI2 Algorithm

5.1 The Guarantee of Compatibility with I_-

The properties proved in section 4 characterize the search space and its border set when new learning data are received. As we have seen, there are two possible cases of incompatibility with the negative sample. The first case results from the shortest suffix extension. This operation extends the current solution to obtain an element of the new search space. This element may be incompatible with the negative sample as illustrated in section 4. The second case results from the adding of a new negative example that is not rejected by the current solution.

These two incompatibility cases may be dealt in the same way. Let A be a DFA belonging to $Lat(PTA(I_+))$ with $L(A) \cap I_- \neq \phi$, and let π be the partition such that $A = PTA(I_+)/\pi$. This partition may be stored from one step to the next and easily adapted when the shortest suffix extension is performed.

Let us consider the prefix tree and a deterministic quotient automaton A presented in figure 3. The data already received are $I_+ = \{\lambda, a, bb, bba, baab, baaaba\}$ and $I_- = \{abbb\}$.

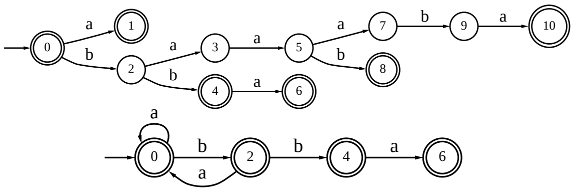

Figure 3: A prefix tree and a deterministic quotient automaton.

One can easily check that the partition π such that $A = PTA(I_+)/\pi$ is given by $\pi = \{\{0, 1, 3, 5, 7, 10\}, \{2, 8, 9\}, \{4\}, \{6\}\}$.

Let us assume that the next negative example is b. The automaton A accepts the string b and thus becomes incompatible with $I_- = \{b, abbb\}$. The incompatibility is due to at least one state merging that allows for accepting b. Consequently, we look for the partition π' finer than π such that $PTA(I_+)/\pi'$ is compatible with I_-.

We know that the negative examples aim at distinguishing between kernel elements and short prefixes that do not share the same set of tails. More precisely, let a negative example w be in a characteristic sample. There must exist a prefix $u \in \mathrm{Pr}(L)$ and a string v such that $w = uv$ and $uv \in I_-^c$. Knowing the negative example w and the prefix tree acceptor, one can determine the strings u and v. In our case, the negative example b leads to state 2 in $PTA(I_+)$. We have $u = b$ and $v = \lambda$. Since the negative example is accepted by the quotient automaton, there exists a state q_v of which v is a suffix and which belongs to the block of u. In our example, state q_v has λ as a suffix and corresponds to state 8 in $PTA(I_+)$. This state belongs in partition π to the block of state 2. The definition of state q_v will be called the *acceptance prefix definition*. It also determines the *rank* of state q_v. We say that the block of state q_v is *split* or that state q_v is taken out of its block to create the new block $\{q_v\}$.

5.1.1 Conforming with the Standard Order

We look for a partition π' finer than partition π such that the quotient automaton $PTA(I_+)/\pi'$ is compatible. We know by the proof of convergence of the RPNI algorithm that it is essential to respect the standard order of merging [Dup96]. Therefore if, by the splitting operation, we negate the merging of state q_v with any other state, the merging of any state of higher rank has to be negated as well. In our example, q_v corresponds to state 8, meaning that state 8, 9 and 10 have to be taken out of their respective blocks. We perform these splittings in reverse order, starting with state 10.

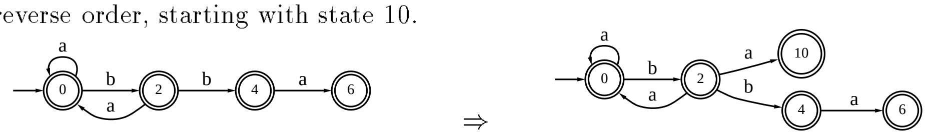

Figure 4: Splitting of a DFA.

5.1.2 Determinization by Splitting

While the original automaton was deterministic, the splitting may create non-determinism. In our case, the splitting of the block of state 10 introduces a non-determinism. We may then apply a *determinization by splitting,* the converse of the determinization by merging quoted in section 3. It is formally defined as follows.

> If $A = PTA(I_+)/\pi$ and if Bq designates the block of state q in partition π, the *splitting* of π *with respect to* q gives rise to the new partition
> $$\pi' = (\pi \setminus Bq) \cup \{Bq \setminus q\} \cup \{q\}$$
> **If** in $PTA(I_+)$ state q has a predecessor q_p on letter a, i.e. $q \in \delta_{PTA}(q_p, a)$, and **if** $\exists Bq_p, B' \in \pi'$ with $B' \neq \{q\}$ and $\delta(Bq_p, a) \supseteq \{\{q\}, B'\}$ in $PTA(I_+)/\pi'$ **then** apply the splitting of π' with respect to q_p.

In other words, the splitting with respect to q introduces a non-determinism on letter a since there exists a state q_p belonging to block Bq_p, predecessor of q, from which there exists another transition on letter a in the quotient automaton $PTA(I_+)/\pi'$. In this case, a splitting of π' with respect to q_p is performed.

Determinization by splitting is a recursive operation. In order to respect the standard order, each time a splitting of a partition π with respect to a state q is performed, the states belonging to the tails of q in $PTA(I_+)$ must also be taken out of their respective blocks.

Property 5.1 *[Dup96] Determinization by splitting necessarily stops at a deterministic automaton.*

The determinization by splitting applied in the previous example yields a compatible DFA presented in figure 5.

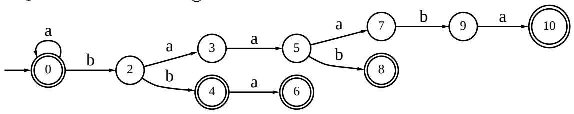

Figure 5: Compatible DFA obtained by recursive splitting.

5.2 Outline of the RPNI2 Algorithm

Each time a new positive example x is received, the prefix tree is extended by the function *extend* $(PTA(I_+(p)), x)$. The function *compatible* (A, I) returns TRUE if the automaton A is compatible with the sample I, otherwise it returns FALSE. When I is a positive sample, the compatibility means the acceptance of all strings in I. Conversely, when I is a negative sample, compatibility means the rejection of all strings in I. If the previous solution remains compatible after adding the new example, this solution is returned. When an incompatibility with a new positive example x is found, the previous solution is extended by the shortest suffix of x. This operation is implemented by the function *suffix_extension* $(A/\pi, x)$. At this stage, the current automaton A belongs to the search space defined by all the data received. However, it may be incompatible with the current negative sample I_-. The function *acceptance_prefix* (A, I_-) returns the minimal rank of the state with respect to which partition π must be split. The function *determ_split* $(A/\pi, j)$ performs the splitting of automaton A/π with respect to state j and seeks, by recursive splitting, to find a compatible DFA. Finally, the RPNI algorithm is applied starting with the automaton obtained after splitting. Since, in the worst case, the automaton obtained after splitting is the prefix tree, the RPNI algorithm is applied without making use of the previously constructed solution. However, in practice, the RPNI algorithm will generally be applied starting with a smaller automaton than the prefix tree. That is, the search will be performed on a sub-lattice of $Lat(PTA(I_+))$.

```
Algorithm RPNI2
input PTA(I+(p))               // Prefix Tree Acceptor of the first p positive examples
      I-(q)                    // Negative sample made of the first q negative examples
      A/π                      // The current solution
      x                        // The new example
output A DFA compatible with (I+(p), I-(q)) and x
begin
if x is a positive example then
  PTA(I+(p + 1)) ←extend (PTA(I+(p)), x)        // Prefix tree extension
  if compatible (A/π,{x}) then                   // Acceptance checking
    return A/π
  else
    A/π' ← suffix_extension (A/π, x)            // Shortest suffix extension
    A ← A/π'
    π ← π'
  end if
end if
```

```
if x is a negative example then
  if compatible (A/π,{x}) then                              // Rejection checking
    return A/π
  else
    A ← A/π'
    π ← π'
  end if
end if

i ← acceptance_prefix (A, I_)                    // Minimal rank of states to be split
// N is the number of states of the PTA
for j = N to i                                   // State splitting in reverse order <
    A/π' ← determ_split (A/π, j)
    π ← π'
end for
A ← A/π

// RPNI algorithm applied from automaton A
for i = 1 to |π| − 1                             // Loop on the blocks of partition π
    for j = 0 to i − 1                           // Loop on the blocks of lower rank
        π' ← π\{B_j, B_i}U{B_iUB_j}                // Merging block B_i and block B_j
        A/π' ← derive (A, π')
        π'' ← determ_merge (A/π')
        if compatible (A/π'',I_) then                     // Deterministic parsing of I_
          A ← A/π''
          π ← π''
          break                                                       // Break j loop
        end if
    end for                                                             // End j loop
end for                                                                 // End i loop
return A
end RPNI2
```

5.3 Identification in the Limit and Complexity

Theorem 5.1 *The RPNI2 algorithm identifies in the limit the class of regular languages.*

Theorem 5.1, proved in [Dup96], guarantees the identification of the correct automaton when the data, received one by one, include a characteristic sample. The identification in the limit may also be obtained with the RPNI algorithm. This requires the storage of all the data received sequentially and the application of the RPNI algorithm as if the data were given as a whole. However, the RPNI algorithm cannot make use of the solution constructed from the previous data set. In this regard, the RPNI2 algorithm offers a possible complexity reduction. We showed that the asymptotic space and time complexities of both algorithms are the same [Dup96]. The advantage of the RPNI2 algorithm lies, then, in a practical complexity reduction which arises because the core of the algorithm, the double loop, is generally applied on a smaller search space. This point is experimentally confirmed in section 6.

6 Experimental Results

6.1 Learning Task and Experimental Protocol

The Feldman task [FLSW90] was proposed for language acquisition studies. It is not meant to study the whole cognitive process of learning a language but it offers a clear framework which may be easily extended. This task has served as a benchmark for research in different domains such as cognitive science [FLSW90], estimation of probabilistic attribute grammars [Sto94] and transducer learning [CGV94].

In its original formulation, the Feldman task consists in building an automatic system capable of classifying as true or false statements about object positions on a screen. Typically, we might have the following statements:

"A circle is to the right of a triangle",
"A triangle and a circle are above a square".

We consider here an extension of the original task including actions on the objects:

"Delete the triangle on the right of the square",
"Add a medium size square above a light circle and a small dark triangle".

Positive learning samples are randomly generated from the canonical automaton of the Feldman task[7]. Neither the structural completeness nor the inclusion of a possible characteristic sample are guaranteed. Negative samples are obtained by randomly corrupting independently generated positive samples according to a Markov process. Each string is corrupted by allowing insertion, substitution and deletion of symbols. The corrupted data are checked to eliminate those strings that are positive. In general 95 % of the corrupted strings generated in this fashion were negative examples of the Feldman language.

We generate 10 independent pairs of positive and negative samples. The size $|I_+|$ of each positive sample is fixed at 2,500 examples. Each example having on the average about 10 symbols, the total number of symbols $||I_+||$ is about 25,000 from an alphabet of 29 symbols. The size of the prefix tree built on a full positive sample is about 10,000 states and 10,000 transitions. The total number of symbols in each negative sample $||I_-||$ is about 20,000 while the size $|I_-|$ of a negative sample is typically 2,250. Following the same procedure, we generate a new pair of positive and negative samples each containing 500 strings. This pair of samples form the test set on which we compute the correct classification rate of positive and negative test strings with respect to the target automaton.

For each pair of positive and negative samples, the data are presented sequentially by picking at random either the next positive or the next negative example. At each step of this sequential presentation the inference algorithm proposes a solution. The RPNI2 algorithm uses the preceeding solution to build the new one. The incremental use of the RPNI algorithm consists in returning the previous solution if it is compatible with the new data, and otherwise in inferring from the prefix tree built on the positive examples already received.

[7] This canonical DFA contains 57 states and 149 transitions.

6.2 RPNI and RPNI2 Comparative Results

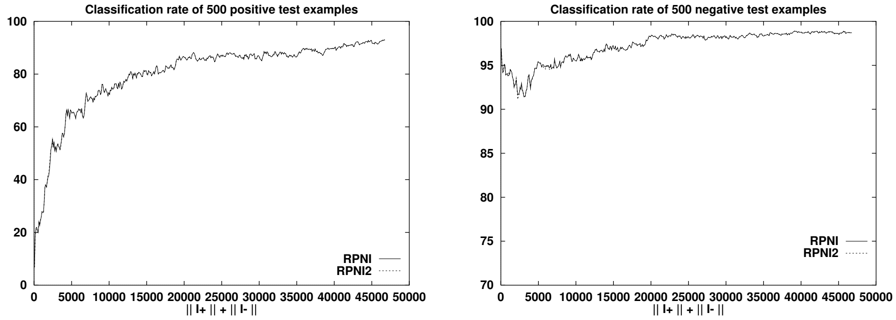

Figure 6: RPNI and RPNI2 : classification rates on the test data.

Figure 6 presents the classification rates of positive and negative test examples as a function of $||I_+|| + ||I_-||$, the number of training symbols received. These curves are averaged over the 10 independent pairs of training samples. On these tests, both algorithms have nearly identitical classification rates as illustrated by their remarkable similarity in figure 6. Let us remark that the identification in the limit only guarantees that both curves reach 100 % correct classification at the same time.

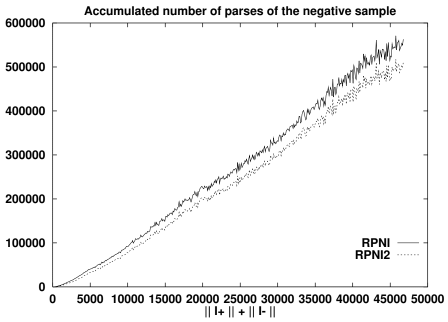

Figure 7: RPNI and RPNI2 : practical complexities.

Figure 7 presents the accumulated number of parses of the negative sample I_- performed during the entire incremental learning process. At each step we measure the number of calls to the compatibility checking function, as it dominates the practical cost of both algorithms.

At the end of the sequential presentation, the RPNI algorithm has performed 10.2 % more calls than the RPNI2 algorithm. The observed gain may depend on the order of presentation which is in this case purely random. An interesting property is that the gain increases monotonically with the size of the learning samples.

7 Conclusion

In this work, we propose a new characterization of the incremental search space of regular inference. The exhibited properties resulted in the design of the RPNI2 algorithm, the incremental extension of the RPNI algorithm. We study the characterization of these algorithms, together with their theoretical and practical complexities. Both algorithms experimentally converge at the same rate. RPNI2 allows for monotonically decreasing the practical complexity as a function of the learning data size.

While the RPNI2 algorithm has been designed in such a way that the identification in the limit may be proved, examples previously received could be forgotten, for instance, by memorizing only the last n examples. It would then offer a natural tradeoff between the space complexity and the convergence rate, the exact identification not being guaranteed.

References

[Ang78] D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, pages 337–350, 1978.

[Ang82] D. Angluin. Inference of reversible languages. *Journal of the Association for Computing Machinery*, 29(3):741–765, 1982.

[CGV94] A. Castellanos, I. Galiano, and E. Vidal. Application of OSTIA to machine translation tasks. In *Grammatical Inference and Applications, ICGI'94*, number 862 in Lecture Notes in Artificial Intelligence, pages 93–105. Springer Verlag, 1994.

[DMV94] P. Dupont, L. Miclet, and E. Vidal. What is the search space of the regular inference ? In *Grammatical Inference and Applications, ICGI'94*, number 862 in Lecture Notes in Artificial Intelligence, pages 25–37. Springer Verlag, 1994.

[Dup96] P. Dupont. *Utilisation et Apprentissage de Modèles de Langage pour la Reconnaissance de la Parole Continue.* Thèse de Doctorat, Ecole Nationale Supérieure des Télécommunications, Paris, France, 1996.

[FB75] K.S. Fu and T.L. Booth. Grammatical inference: Introduction and survey, part 1. *IEEE Transactions on Systems, Man and Cybernetics*, 5:85–111, 1975.

[FLSW90] J.A. Feldman, G. Lakoff, A. Stolcke, and S.H. Weber. Miniature language acquisition: A touchstone for cognitive science. Technical Report TR-90-009, International Computer Science Institute, Berkeley, California, 1990.

[Gol67] E.M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[Gol78] E.M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[Lan92] K.J. Lang. Random DFA's can be approximately learned from sparse uniform examples. In *5th ACM workshop on Computational Learning Theory*, pages 45–52, 1992.

[OG92] J. Oncina and P. García. Inferring regular languages in polynomial update time. In N. Pérez de la Blanca, A. Sanfeliu, and E.Vidal, editors, *Pattern Recognition and Image Analysis*, volume 1 of *Series in Machine Perception and Artificial Intelligence*, pages 49–61. World Scientific, 1992.

[Sto94] A. Stolcke. *Bayesian Learning of Probabilistic Language Models.* Ph. D. dissertation, University of California, 1994.