Inference Improvement by Enlarging the Training Set While Learning DFAs*

Pedro García[1], José Ruiz[1], Antonio Cano[1], and Gloria Alvarez[2]

[1] Universidad Politécnica de Valencia,
Departamento de Sistemas Informáticos y Computación,
Camino de Vera s/n, 46022 Valencia, Spain
{pgarcia, jruiz, acano}@dsic.upv.es
http://www.dsic.upv.es/users/tlcc/tlcc.html
[2] Pontificia Universidad Javeriana - Seccional Cali,
Grupo de Investigación DESTINO, Calle 18 118-250,
Cali, Colombia
galvarez@dsic.upv.es

Abstract. A new version of the *RPNI* algorithm, called *RPNI2*, is presented. The main difference between them is the capability of the new one to extend the training set during the inference process. The effect of this new feature is specially notorious in the inference of languages generated from regular expressions and Non-deterministic Finite Automata (NFA). A first experimental comparison is done between *RPNI2* and *DeLeTe2*, other algorithm that behaves well with the same sort of training data. [1]

1 Introduction

One of the best known algorithms for regular language identification, *RPNI* (Regular Positive and Negative Inference) [9], converges to the minimal Deterministic Finite Automaton (DFA) of the target language. It finds equivalence relations in the data from the prefix tree acceptor of the sample.

Recently an algorithm called *DeLeTe2* [3] that outputs Non-deterministic Finite Automata (NFA) instead of DFAs has been proposed. *DeLeTe2* looks for a special type of NFA called RFSA (Residual Finite State Automata), whose states represent residuals of the target language. Every regular language is recognized by a unique minimal RFSA, called the canonical RFSA. Canonical RFSA consists only of prime residual states (i.e. states that can not be obtained as union of other residuals).

The basis of *DeLeTe2* algorithm is to obtain RFSAs by looking for inclusion relations in the residuals of the target language using the prefix tree acceptor of the data. Its authors have shown that when the target automaton is a randomly generated DFA [8], the probability of occurrence of inclusion relation between

* Work partially supported by Spanish CICYT under TIC2003-09319-C03-02

[1] A two pages abstract presented in the Tenth International Conference on Implementation and Application of Automata [5] gives a shallow description of this work.

M. Lazo and A. Sanfeliu (Eds.): CIARP 2005, LNCS 3773, pp. 59–70, 2005.

states is very small and then, the size of the canonical RFSA and the minimal DFA of a language are the same. Hence, in this case, *DeLeTe2* behaves worse than *RPNI*. On the other hand, when the target languages are generated using random regular expressions or NFAs, the experiments in [3] show that *DeLeTe2* performs better than *RPNI*.

In this work we propose a modification of *RPNI* algorithm, called *RPNI2*. It extends *RPNI* by finding inclusion relations among residuals aiming to predict whether the prefixes of the data belong to the target language or to its complement.

RPNI2 outputs a DFA and converges to the minimal DFA of the target language. When the source of the learning data is a non-deterministic model, its performance is very similar to *DeLeTe2* performance. However, the average descriptive complexity of the hypothesis that *RPNI2* obtains is substantially smaller than the one obtained by *DeLeTe2*.

Next sections have the following structure: section 2 reminds useful definitions, notation and algorithms. Section 3 presents the *RPNI2* algorithm, a brief example is shown in section 4. The experimental results are in section 5 and finally, section 6 contains the conclusions.

2 Definitions and Notation

Definitions not contained in this section can be found in [7,10]. Definitions and previous works concerning RFSAs can be found in [1,2,3].

Let A be a finite alphabet and let A^* be the free monoid generated by A with concatenation as the internal operation and ε as neutral element. A *language* L over A is a subset of A^*. The elements of L are called *words*. The length of a word $w \in A^*$ is noted $|w|$. Given $x \in A^*$, if $x = uv$ with $u, v \in A^*$, then u (resp. v) is called *prefix* (resp. *suffix*) of x. $\Pr(L)$ (resp. $\mathrm{Suf}(L)$) denotes the set of prefixes (resp. suffixes) of L. The product of two languages $L_1, L_2 \subseteq A^*$ is defined as: $L_1 \cdot L_2 = \{u_1u_2 | u_1 \in L_1 \wedge u_2 \in L_2\}$. Sometimes $L_1 \cdot L_2$ will be notated simply as L_1L_2. Throughout the paper, the *lexicographical order* in A^* will be denoted as $\ll$. Assuming that A is totally ordered by $<$ and given $u, v \in A^*$ with $u = u_1 \dots u_m$ and $v = v_1 \dots v_n$, $u \ll v$ if and only if ($|u| < |v|$) or ($|u| = |v|$ and $\exists j$, $1 \leq j \leq n, m$ such that $u_1 \dots u_j = v_1 \dots v_j$ and $u_{j+1} < v_{j+1}$).

A *Non-deterministic Finite Automaton* (NFA) is a 5-tuple $\mathcal{A} = (Q, A, \delta, Q_0, F)$ where Q is the (finite) set of states, A is a finite alphabet, $Q_0 \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states and δ is a partial function that maps $Q \times A$ in 2^Q. The extension of this function to words is also denoted δ. A word x is accepted by $\mathcal{A}$ if $\delta(Q_0, x) \cap F \neq \emptyset$. The set of words accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

Given a finite set of words D_+, the *prefix tree acceptor of* D_+ is defined as the automaton $\mathcal{A} = (Q, A, \delta, q_0, F)$ where $Q = Pr(D_+)$, $q_0 = \varepsilon$, $F = D_+$ and $\delta(u, a) = ua$, $\forall u, ua \in Q$.

A Moore machine is a 6-tuple $M = (Q, A, B, \delta, q_0, \Phi)$, where A (resp. B) is the input (resp. output) alphabet, δ is a partial function that maps $Q \times A$ in

Q and Φ is a function that maps Q in B called *output function*. Throughout this paper $B = \{0, 1, ?\}$. A nondeterministic Moore machine is defined in a similar way except for the fact that δ maps $Q \times A$ in 2^Q and the set of initial states is I. The automaton related to a Moore machine $M = (Q, A, B, \delta, I, \Phi)$ is $\mathcal{A} = (Q, A, \delta, I, F)$ where $F = \{q \in Q : \Phi(q) = 1\}$. The *restriction* of M to $P \subseteq Q$ is the machine M_P defined as in the case of automata.

The behavior of M is given by the partial function $t_M : A^* \to B$ defined as $t_M(x) = \Phi(\delta(q_0, x))$, for every $x \in A^*$ such that $\delta(q_0, x)$ is defined.

Given two disjoint finite sets of words D_+ and D_-, we define the (D_+, D_-)-*Prefix Tree Moore Machine* ($PTMM(D_+, D_-)$) as the Moore machine having $B = \{0, 1, ?\}$, $Q = \Pr(D_+ \cup D_-)$, $q_0 = \varepsilon$ and $\delta(u, a) = ua$ if $u, ua \in Q$ and $a \in A$. For every state u, the value of the output function associated to u is 1, 0 or ? (undefined) depending whether u belongs to D_+, to D_- or to $Q - (D_+ \cup D_-)$ respectively. The size of the sample (D_+, D_-) is $\sum_{w \in D_+ \cup D_-} |w|$.

A Moore machine $M = (Q, A, \{0, 1, ?\}, \delta, q_0, \Phi)$ is *consistent* with (D_+, D_-) if $\forall x \in D_+$ we have $\Phi(x) = 1$ and $\forall x \in D_-$ we have $\Phi(x) = 0$.

2.1 Residual Finite State Automata (RFSA)

The *derivative* of a language L by a word u, also called *residual language* of L associated to u is $u^{-1}L = \{v \in A^* : uv \in L\}$. A residual language $u^{-1}L$ is *composite* if $u^{-1}L = \cup_{v^{-1}L \subsetneq u^{-1}L} v^{-1}L$. A residual language is *prime* if it is not composite.

If $\mathcal{A} = (Q, A, \delta, I, F)$ is an NFA and $q \in Q$, we define the language accepted in automaton $\mathcal{A}$ from state q as $L(\mathcal{A}, q) = \{v \in A^* : \delta(q, v) \cap F \neq \emptyset\}$.

A Residual Finite State Automata RFSA [2] is an automaton $\mathcal{A} = \langle Q, A, \delta, I, F\rangle$ such that, for each $q \in Q$, $L(\mathcal{A}, q)$ is a residual language of the language L recognized by $\mathcal{A}$. So $\forall q \in Q, \exists u \in A^*$ such that $L(\mathcal{A}, q) = u^{-1}L$. In other words, a *Residual Finite State Automaton* (RFSA) $\mathcal{A}$ is an NFA such that every state defines a residual language of $L(\mathcal{A})$.

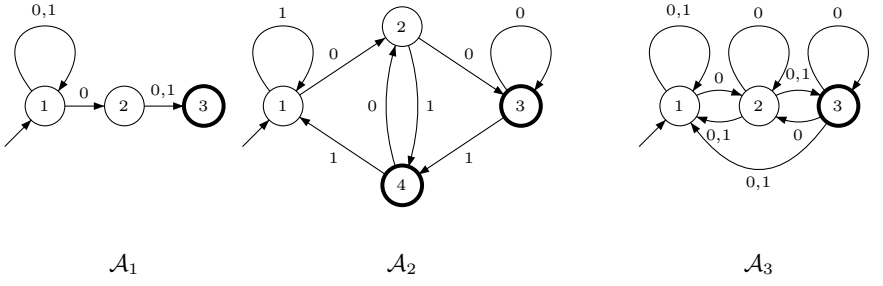
Two operators are defined [2] on RFSAs that preserve equivalence. The saturation and reduction operators. Given $\mathcal{A} = (Q, A, \delta, I, F)$ the *saturated automaton* of $\mathcal{A}$ is the automaton $\mathcal{A}^s = (Q, A, \delta^s, I^s, F)$, where $I^s = \{q \in Q : L(\mathcal{A}, q) \subseteq L(\mathcal{A})\}$ and $\forall q \in Q$, $\forall a \in A$, $\delta^s(q, a) = \{q' \in Q : L(\mathcal{A}, q') \subseteq a^{-1}L(\mathcal{A}, q)\}$. If in automaton $\mathcal{A}$ all the residual languages (not only the prime ones) are considered as states, the new automata is known as saturated RFSA of the minimal DFA for L. The *reduction* operator allows to eliminate from an automaton $\mathcal{A}^s$ the composite states and the transitions related to them. Both operations are useful to get the *canonical RFSA* associated with $\mathcal{A}$: first the saturation operator is applied to $\mathcal{A}$, later the reduction operator is applied to the result. It is known [2] that every regular language L is recognized by a unique reduced saturated RFSA, the *canonical RFSA* of L.

Formally, given a language $L \subseteq A^*$ the *canonical RFSA* of L is the automaton $\mathcal{A} = (Q, A, \delta, I, F)$ where:

- $Q = \{u^{-1}L : u^{-1}L \text{ is prime}, u \in A^*\}$
- A is the alphabet of L
- $\delta(u^{-1}L, a) = \{v^{-1}L \in Q : v^{-1}L \subseteq (ua)^{-1}L\}$
- $I = \{u^{-1}L \in Q : u^{-1}L \subseteq L\}$
- $F = \{u^{-1}L \in Q : \varepsilon \in u^{-1}L\}$

Two relations defined in the set of states of an automaton link RFSAs with grammatical inference. Let $D = (D_+, D_-)$ be a sample, let $u, v \in Pr(D_+)$. We say that $u \prec v$ if no word w exists such that $uw \in D_+$ and $vw \in D_-$. We say that $u \simeq v$ [2] if $u \prec v$ and $v \prec u$.

Example of RFSAs. The following example has been taken from [2]. Let $A = \{0, 1\}$ and let $L = A^*0A$. L can be recognized by the three automata of Figure 1. States that output 1 (resp. 0) are drawn using thick (resp. thin) lines.



$\mathcal{A}_1$ $\mathcal{A}_2$ $\mathcal{A}_3$

Fig. 1. $\mathcal{A}_1$ is an automaton recognizing $L = A^*0A$ which is neither a NFA nor a RFSA. $\mathcal{A}_2$ is a DFA recognizing L which is also RFSA. $\mathcal{A}_3$ is the canonical RFSA for L.

- The first one $\mathcal{A}_1$ is neither DFA nor RFSA. The languages associated with states are: $L(\mathcal{A}_1, 1) = A^*0A$, $L(\mathcal{A}_1, 2) = A$ and $L(\mathcal{A}_1, 3) = \varepsilon$. One can see that $L(\mathcal{A}_1, 3) = \varepsilon$ and $\nexists u \in A^*$ such that $L(\mathcal{A}_1, 3) = u^{-1}L$.
- Automaton $\mathcal{A}_2$ is a minimal automaton recognizing L and thus, is a RFSA, in this case $L(\mathcal{A}_2, 1) = A^*0A$, $L(\mathcal{A}_2, 2) = A^*0A+A$, $L(\mathcal{A}_2, 3) = A^*0A+A+\varepsilon$, $L(\mathcal{A}_2, 4) = A^*0A + \varepsilon$.
- Automaton $\mathcal{A}_3$ is the L's canonical RFSA, which is not a DFA. The languages associated with states are: $L(\mathcal{A}_3, 1) = \varepsilon^{-1}L$, $L(\mathcal{A}_3, 2) = 0^{-1}L$ and $L(\mathcal{A}_3, 3) = 01^{-1}L$.

2.2 The *RPNI* Algorithm

The aim of grammatical inference is to obtain a description of a language L by means of a sample (a set of words labelled as belonging to L or to its complement). Throughout this work we will assume that the target language L is regular; then, the description we will look for is an automaton.

[2] This relation is known in the terminology set up by Gold as *not obviously different states*. Other authors call it *compatible states*.

We will use the convergence criterion called *identification in the limit*, introduced by Gold [6].

The *RPNI* algorithm [9] is used for inference of regular languages. It receives as input a sample of the target language and outputs, in polynomial time, a DFA consistent with the input. *RPNI* converges to the minimal automaton of the target language in the limit.

Algorithm *RPNI* (D_+, D_-) starts from the $PTMM(D_+, D_-)$, and recursively merges every state with the previous ones to keep a deterministic automaton under the condition that it does not accept a negative sample. State merging is done by the function `detmerge`(M, p, q) shown in Algorithm 1, which merges states p and q in M if they are compatible. If one of the merging states is undefined and the other is not, the merged state takes the value of the latter state.

Algorithm 1 Function detmerge

```
detmerge(M, p, q) //p ≪ q in lexicographical order//
    M' := M
    list := {(p, q)}
    while list' ≠ ∅
        (r, s) := first(list)
        M1 := merge(M', r, s)
        if M1 = M'
            Return M
        else
            M' := M1
            for a ∈ A
                if δ(p, a) and δ(q, a) are defined
                    list := append(list,(δ(p, a), δ(q, a)))
                endif
            endfor
        endif
    endwhile
    Return M'
```

The merging process is recursively repeated with the successors of the merged states until either the nondeterminism disappears or the algorithm tries to merge incompatible states. In the former case, the output of the algorithm is the deterministic automaton resulting of merging states, whereas in the latter the output of `detmerge`(M, p, q) is M.

2.3 *DeLeTe2* Algorithm

The *DeLeTe* and *DeLeTe2* algorithms output residual nondeterministic finite automata (RFSA). The algorithms look for inclusion relations between the residual

languages and reflect those situations in the automaton using the saturation operator. As it can be expected, the method becomes more interesting when the target automaton contains many composite residual languages, because then may exist many inclusion relations between states that make the size of the hypothesis to decrease. Otherwise, if most of the residual languages of the target automaton are prime the output automaton would have a size similar to the size of the minimal DFA [3].

It is known [3] that *DeLeTe2* is an improvement to *DeLeTe* algorithm (algorithm 2) it solves the eventual lack of consistency with the sample of *DeLeTe*, unfortunately the details of this improvement have not been published yet.

Algorithm 2 Algorithm *DeLeTe*

```
DeLeTe(D+, D−)
    let Pref be the set of prefixes of D+ in lexicographical order
    Q := ∅; I := ∅; F := ∅; δ := ∅; u := ε
    stop := false
    while not stop
        if ∃u′|u ≃ u′
            delete uA* from Pref
        else
            Q := Q ∪ {u}
            if u ≺ u′
                I := I ∪ {u}
            endif
            if u ∈ D+
                F := F ∪ {u}
            endif
            δ := δ ∪ {(u′, x, u)|u′ ∈ Q, u′x ∈ Pref, u ≺ u′x}∪
                    {(u, x, u′)|u′ ∈ Q, ux ∈ Pref, u′ ≺ ux}
        endif
        if u is the last word of Pref or
        A = ⟨Q, A, I, F, δ⟩ is consistent with D+, D−
            stop := true
        else
            u := next word in Pref
        endif
    endwhile
    Return A = ⟨Q, A, I, F, δ⟩
```

3 The *RPNI2* Algorithm

The idea behind *RPNI2* is to try to establish the possible inclusion relation between states that can not be merged. Sometimes this will allow us to define the output associated to states that were previously undefined.

The following definitions are useful to understand functions `tryInclusion` and `defineStates` shown in Algorithms 4 and 5 respectively. These functions are used in *RPNI2* to represent the new ideas stated above. Algorithm *RPNI2* is shown in Algorithm 3.

Algorithm 3 Algorithm *RPNI2*

```
RPNI2(D+, D−)
    M := PTMM(D+, D−)
    list := {u0, u1, ..., ur} //states of M in lexicographical order, u0 = λ//
    list' := {u1, ..., ur}
    q := u1
    while list' ≠ ∅
        for p ∈ list and p ≪ q (in lexicographical order)
            if detmerge(M, p, q) = M
                defineStates(M,p,q)
            else
                M := detmerge(M, p, q)
                exit for
            endif
        endfor
        list := Delete from list the states which are not in M
        list' := Delete from list' the states which are not in M
        q := first(list')
    endwhile
    Return M
```

Definition 1. *States p and q are non-comparable (for inclusion relations) in a Moore machine if there exist $u, v \in A^*$ such that $\Phi(\delta(p,u)) = 1 \wedge \Phi(\delta(q,u)) = 0$ and $\Phi(\delta(p,v)) = 0 \wedge \Phi(\delta(q,v)) = 1$.*

Definition 2. *Given $p, q \in Q$, we say that state p is potentially lesser than state q if they are not non-comparable and there does not exist $u \in A^*$ such that $\Phi(\delta(p,u)) = 1 \wedge \Phi(\delta(q,u)) = 0$.*

When states p and q can not be merged while running *RPNI*, that is, when $\texttt{detmerge}(M,p,q) = M$, the new algorithm tries to define the output for the undefined states using the function `defineStates` shown in Algorithm 5.

To explain the behavior of the function `defineStates` we will use the Moore machine M in Fig. 2; in this figure and the next ones the output value of each state q will be represented as a thin circle if $\Phi(q) = 0$, a thick one if $\Phi(q) = 1$ and a dashed one if $\Phi(q) =?$. Following the algorithm it is possible to see that $\texttt{defineStates}(M,1,2) = M$, since otherwise the negative sample 1000010 should be accepted.

Algorithm 4 Function tryInclusion

```
tryInclusion(M, p, q)
    M′ := M
    while p and q are not non-comparable
        for any u common successor of p and q
            if φ(δ(p,u)) = 1 ∧ φ(δ(q,u)) =?
                φ(δ(q,u)) = 1; Update M′
            endif
            if φ(δ(p,u)) =? ∧ φ(δ(q,u)) = 0
                φ(δ(p,u)) = 0; Update M′
            endif
        endfor
    endwhile
    if p and q are non-comparable
        Return M
    else
        Return M'
```

Algorithm 5 Function defineStates

```
defineStates(M, p, q)
    if p and q are non-comparable
        Return M
    else
        if p is potentially lesser than q
            tryInclusion(M,p,q)
        endif
        if q is potentially lesser than p
            tryInclusion(M,q,p)
        endif
    endif
```

In the tree in Fig. 3, the signs preceding the state name represent its output value; for example: $(+2, 11)$ means $\Phi(2) = 1$ and $\Phi(11) =?$. Since the states of the same node do not have different labels named "+" and "-", states 1 and 2 are not non-comparable with respect to inclusion.

Executing `tryInclusion`$(M, 1, 2)$ returns M; otherwise the node $(4, -13)$ would imply $\Phi(4) = 0$ and at the same time the node $(+1, 4)$ would imply $\Phi(4) = 1$. However, executing `tryInclusion`$(M, 2, 1)$ changes the output of states 4 and 8 and then the function `defineStates`$(M, 1, 2)$ changes M to $\Phi(4) = 1$ and $\Phi(8) = 1$. Notice that the change of the output value of a state from ? (indefinite)

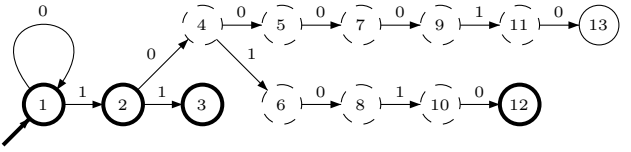


Fig. 2. Initial Moore machine used to describe the behavior of function `defineStates`

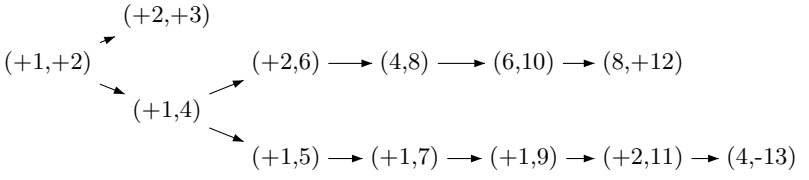


Fig. 3. Scheme used to compare states in function `defineStates`

to 0 or 1 is equivalent to suppose that a new word is present in the input sample. Hence, this process can be seen as an enlargement of the training set.

4 Example

We are going to describe the behavior of algorithm *RPNI2* using the sample $D_+ = \{0, 001, 000011, 0101010\}$ and $D_- = \{01000010\}$ that gives different outputs for the three algorithms. We also show the automata that *RPNI* and *DeLeTe2* output. The Prefix Tree Moore machine is depicted in Fig. 4.

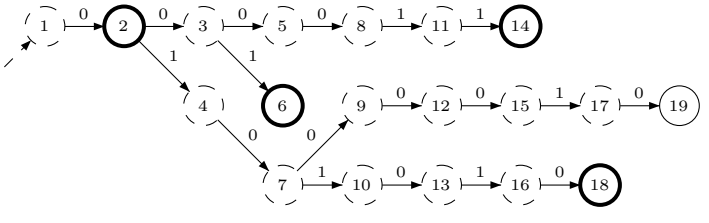


Fig. 4. Prefix Tree Moore machine of the sample

With this sample as input, *RPNI2* first merges states 1 and 2, then tries to merge 1 and 4. These states can not be merged, but the function `defineStates` changes the value of states 7 and 13 to positive. The same happens with states 4 and 7. The function `defineStates` changes now the value of states 9, 10, 12 and 15 to positive. Finally, the algorithm merges states 4 and 9 and then it merges states 1 and 10.

Fig. 5 depicts the outputs given by algorithms *RPNI*, *DeLeTe2* and *RPNI2* when using the above sample input. It should be noted that during the execution of *RPNI2*, states 7, 9, 10, 12, 13 and 15 are labelled as positive.

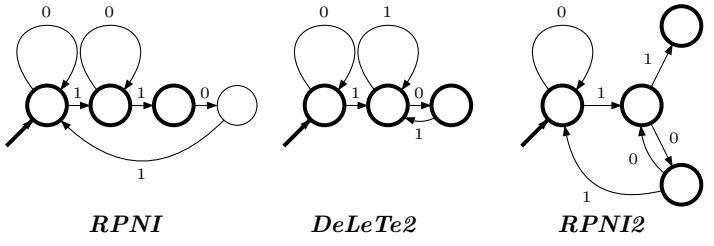


Fig. 5. Output automata given by the three algorithms compared in this work on input $D_+ = \{0, 001, 000011, 0101010\}$ and $D_- = \{01000010\}$

5 Results

The aim of the experiments is to analyze the behaviour of *RPNI2* and to compare it with the *DeLeTe2* algorithm. Both the training/test samples and the *DeLeTe2* program used in this experimentation are provided by their authors and are available in Aurélien Lemay's web page `http://www.grappa.univ-lille3.fr/~lemay/`. The samples have the following features [3]:

- The target language are regular expressions or NFAs.
- The probability distribution of the sample generation methods are different in each case: NFAs or regular expressions.
- The NFAs sample generation method chooses randomly the states number n, the alphabet size $|\Sigma|$, the transitions number per state n_δ and the initial and final state probabilities (p_I and p_F respectively) for each state. Each state has exactly n_δ successors. The symbol and destination state of each transition are chosen randomly. Once the automaton is trimmed some states will have fewer than n_δ transitions. The parameter values used in these experiments were: $n = 10$, $|\Sigma| = 2$, $n_\delta = 2$, $p_I = p_F = 0.5$.
- The regular expressions generation method consider a set of operators $Op = \{\emptyset, 0, 1, *, \cdot, +\}$. An upper bound n_{op} for the number of operators used is chosen and a probability distribution p on Op is defined. The root operator is chosen by means of the distribution p. If the operator is 0-ary the expression ends, if it is 1-ary the procedure is called recursively with parameter $n_{op} - 1$ and if it is binary, it is called twice with parameters $\lceil n_{op}/2 \rceil$ and $\lfloor (n_{op}-1)/2 \rfloor$. The parameter values used in these experiments are: $n_{op} = 100$, $p_\varepsilon = 0.02$, $p_0 = p_1 = 0.05$, $p_* = 0.13$, $p_\cdot = 0.5$ and $p_+ = 0.25$.

Two kinds of experiments are reported in table 1, depending on the source of the training and test samples: er_* if they come from regular expressions and

nfa_* from NFAs. The number in the identifier of the experiment represents the number of training samples. Each experiment consist of 30 different languages to be learned. Each experiment has 1000 test samples. Table 1 reports the recognition rate and the average size of the inferred hypothesis. These results are calculated as follows: each test sample is presented to the inference program, the program tags the sample as belonging to the target language or not, if this classification agrees with the real sample tag, the sample is considered correct and increases a counter; at the end, the number of correct samples is divided by 1000 (the total of test samples) and this value is reported as recognition rate. The average size is computed adding up the number of states of the 30 hypothesis generated in each experiment and dividing by 30. As it can be seen in Table 1, the error rate of the new algorithm *RPNI2* is better than the previous *RPNI* but slightly worse than *DeLeTe2*. The opposite happens with the description complexity (i.e. states number) of the output hypothesis: the results obtained by *RPNI2* are then better than those of *DeLeTe2*.

It should be noted that the results obtained with our implementation of *RPNI* slightly differ from those obtained in [3] with the same data, maybe because of different implementations of the algorithm. To be more specific about the cause of these differences would be required to know the code used by the authors. The results corresponding to *DeLeTe2* execution, are slightly different too, although they were generated with their own program.

Table 1. Inference results with *RPNI*, *RPNI2* and *DeLeTe2* algorithms

	RPNI		*RPNI2*		*DeLeTe2*	
Iden.	Recogn. rate	Avg. size	Recogn. rate	Avg. size	Recogn. rate	Avg. size
er_50	76.36%	9.63	80.03%	16.32	81.68%	32.43
er_100	80.61%	14.16	88.68%	19.24	91.72%	30.73
er_150	84.46%	15.43	90.61%	26.16	92.29%	60.96
er_200	91.06%	13.3	93.38%	27.37	95.71%	47.73
nfa_50	64.8%	14.3	66.43%	30.64	69.80%	71.26
nfa_100	68.25%	21.83	72.79%	53.14	74.82%	149.13
nfa_150	71.21%	28.13	75.69%	71.87	77.14%	218.26
nfa_200	71.74%	33.43	77.25%	88.95	79.42%	271.3

6 Conclusions

The *RPNI2* strategy behaves better than the original *RPNI* when the language to learn comes from a regular expression or a NFA. In this case, because of the inclusion relation between residual automata, the output values assigned to some states, provide significant information to the inference process thus improving the recognition rate with the test samples.

The experiments presented in [3], which we have also reproduced with *RPNI2*, do not seem to obtain decisive conclusions about the usefulness of inferring RFSAs, because the main reason for its use (the smaller size of the hypothesis)

does not hold. Although the experiments are still preliminary, it seems that the slightly better results obtained by *DeLeTe2* with respect to *RPNI2* do not compensate the fact that the size of the representations obtained by *RPNI2* are clearly smaller.

References

1. Denis, F. Lemay, A. and Terlutte, A. *Learning regular languages using nondeterministic finite automata.*A.L. Oliveira (Ed.), ICGI 2000, LNAI 1891, pp 39-50 (2000).
2. Denis, F. Lemay, A. and Terlutte, A. *Residual finite state automata.*In STACS 2001, LNAI 2010, pp 144-157 (2001).
3. Denis, F., Lemay, A., Terlutte, A. Learning regular languages using RFSAs. Theoretical Computer Science 313(2), pp 267-294 (2004).
4. García, P., Cano, A., Ruiz, J. A comparative study of two algorithms for Automata Identification. A.L. Oliveira(Ed.), LNAI 1891, pp 115-126 (2000).
5. García, P., Ruiz, J., Cano, A., Alvarez G. Is learning RFSAs better than learning DFAs. Proceedings of Tenth International Conference on Implementation and Application of Automata. 2005. To be published.
6. Gold, E.M. Language identification in the limit. Information and Control 10, pp 447-474 (1967).
7. Hopcroft, J. and Ullman, J. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley (1979).
8. Nicaud, C. Etude du comportment des automates finis et des langages rationnels. Ph.D. Thesis, Université de Marne la Vallée. 2001.
9. Oncina, J., García, P. Inferring Regular Languages in Polynomial Updated Time. In Pattern Recognition and Image Analysis. Pérez de la Blanca, Sanfeliú and Vidal (Eds.) World Scientific (1992).
10. Trakhtenbrot B., Barzdin Y. Finite Automata: Behavior and Synthesis. North Holland Publishing Company (1973).