Chapter 4
On the Inference of Finite State Automata from Positive and Negative Data

Damián López and Pedro García

Abstract The works by Thrakhtenbrot–Barzdin and Gold can be considered to be the first works on the identification of Finite Automata from given data. The main drawback of their results is that they may obtain hypotheses that may be inconsistent with the provided data. This drawback was solved by the *RPNI* and Lang algorithms. Aside from these works, other works have introduced more efficient algorithms with respect to the training data. The direct consequence of this improvement has lead to algorithms that have lower error rates. Recently, some works have tackled the identification of NFAs instead of using the traditional DFA model. In this line of research, the inference of Residual Finite State Automata (RFSA) provides a canonical non-deterministic model. Other works consider the inference of teams of NFAs to be a method that is suitable to solve the grammatical inference of finite automata. We review the main approaches that solve the inference of finite automata by using positive and negative data from the target language. In this review, we will describe the above-mentioned formalisms and induction techniques.

4.1 Introduction

The problem of language identification from positive and negative information was proposed in the 1970s. This task is related to the search for a minimal DFA that is compatible with a given positive and negative set of data. Automata synthesis from data is, in fact, a classic problem of Automata Theory. Nowadays this problem is incorporated within the Grammatical Inference (GI) topics.

One common approach to GI is based on the merging of states of an initial representation that strictly recognizes the training set. In this approach, two states are merged when there is no evidence that the associated language is different, and,

D. López (✉) · P. García
Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia, Valencia, Spain
e-mail: dlopez@dsic.upv.es

P. García
e-mail: pgarcia@dsic.upv.es

J. Heinz and J.M. Sempere (eds.), *Topics in Grammatical Inference*,
DOI 10.1007/978-3-662-48395-4_4

therefore, one of them can be considered to be redundant. The above-mentioned evidence comes from the existence or absence of a positive or negative sample in the training set. This means that mergible states in a given situation might not be merged with additional extra information. Such *inconvenient* merges usually imply a larger size of the automata output and a lower recognition rate.

The work of Gold in the 1960s and 1970s led to several relevant contributions in the field of regular GI. First, he proved that the problem of finding a minimal deterministic finite automaton (DFA) that is consistent with some input positive (D_+) and negative (D_-) data is NP-hard [32]. He also proposed the *identification in the limit* model [31], which is a valuable tool for proving the correction of new GI algorithms. Furthermore, Gold proposed a GI algorithm [32] that outputs the minimal deterministic automaton for the language in polynomial time when a representative enough set of data is provided. This algorithm is of great interest because several subsequent algorithms can be viewed as modifications of it.

In the same decades, Trakhtenbrot and Barzdin [44] proposed a DFA synthesis algorithm that considers all the strings of the language whose length is bounded by a given integer. Even though this work does not present a grammatical inference algorithm it is known that the Gold and the Trakhtenbrot–Barzdin algorithms work in a similar way, despite the fact that they use very different representations of the same information [24].

In the 1980s, some new negative results confirmed the theoretical difficulty of the regular language GI problem. Angluin proved that, for a given set of data, the problem of finding the minimal DFA is NP-hard even for target automata of two states. She also proved that the training set must be complete and that whenever a few samples are removed from the training set, the problem also becomes NP-hard [4]. Another result by Angluin is the proof that the task is also hard when the algorithm is allowed to question an oracle about the membership to the target language of certain samples [6] or to question an oracle about the equivalence of the current hypothesis and the target language [8].

Since those negative theoretical results led to obvious pessimism, in order to design new GI algorithms for applied tasks, several approaches were considered. Some works focused on the inference of stochastic DFA (e.g. [9]). Other papers studied active learning, that is, the possibility to question an oracle whenever it is considered necessary [6–8]. There were also works devoted to the characterization of language classes learnable from positive information [5, 25, 30, 48]. Other works studied the inference of transducers [42]. In our review, we do not consider these methods and techniques, which were reviewed recently by de la Higuera [17], instead we focus on those methods that output (deterministic or non-deterministic) automata using positive and negative information from the target language.

In 1992, two works appeared independently of each other. Oncina and García proposed the *RPNI* algorithm [41] and Lang proposed a modification of the Trakhten–Barzdin algorithm known as the Traxbar algorithm [38]. In *RPNI*, the output depends on the *quality* of the training set, and it has been proved that whenever this training set is *characteristic*, the algorithm converges to the minimal DFA for the target language [41].

In the quest to avoid incorrect mergings as much as possible, de la Higuera et al. [18] proposed a modification of the canonical order in order to favor those pairs of states that have high evidence of equivalence. Therefore, they proposed attaching a score to each pair of states, and subsequently merging the pairs of states with the highest score. There are multiple criteria for attaching this score, but, generally speaking, high (low) score values are expected whenever the training set contains great (little) evidence that the languages of both states are the same. Preliminary experimental results were encouraging but not conclusive.

Price revisited this same idea and proposed the EDSM (Evidence Driven State Merging) algorithm in the context of the Abbadingo One contest [37]. In this competition, which was organized by Lang in 1998, several GI algorithms tried to solve a set of regular inference problems. There were two winners: the EDSM algorithm and a parallel algorithm based on non-deterministic search. The fact that the winners were based on a strategy of merging states encouraged the community to again take up the design of identification algorithms. In fact, taking into account the contest results, Lang proposed the Blue-Fringe algorithm [11, 37]. Several other works were also based on the EDSM algorithm [1, 11]. In [29], García et al. revisited these works and the inference of automata teams was proposed.

Eventually, the GI community also studied the inference of non-deterministic finite automata (NFA). In 1994, Yokomori published a work in this field proposing a method that is based on counterexamples and questions to an oracle [16, 47]. Coste and Fredouille also studied this issue [13–15]; they proposed their NFA inference method taking into account results on the inference of unambiguous finite automata.

In this field of NFA inference, Denis et al. [19, 20, 22] have focused their work on the inference of a subset of NFA called RFSA (residual finite state automata). The interest in this subclass of NFA comes from the fact that there is a canonical RFSA for each regular language.

Currently, even though few research groups are working on the inference of NFA, there is some research on the minimalization of NFAs that are attempting to obtain a canonical form [43]; there are works that present algorithms that converge to RFSAs [2]; algorithms that infer UFAs (Unambiguous Finite Automata) [1, 14, 15]; other works that extend the *RPNI* strategy but consider non-deterministic merging [2]; and other works that study the inference of NFAs using subautomata juxtaposition [46]. In [28], a new general method named *OIL* (Order Independent Learning) is proposed. This method constructs the maximal automaton for the training set (instead of the prefix tree acceptor) and then considers any order in the merging of the states.

This chapter is organized as follows. First, in Sect. 4.2 we summarize the basic notation and definitions used. In Sect. 4.3, we review the inference methods that output deterministic automata (beginning with the seminal results of Gold and Trakhtenbrot–Barzdin, and including more recent works that consider heuristics or teams of automata). Section 4.4 is devoted to reviewing those methods that output non-deterministic automata. Section 4.5 experimentially compares the quality of the reviewed methods taking into account the results obtained when a dataset of languages proposed in the literature is used. Some conclusions and suggestions for continuing this study are presented at the end of the chapter.

4.2 Definitions and Notation

Definitions that are not contained in this section can be found in [34]. Definitions and previous works concerning RFSA can be found in [19, 20, 22]. For the reader who is interested in the Universal Automaton, we suggest [39, 43].

Let Σ be a finite alphabet and let Σ^* be the free monoid generated by Σ with concatenation as the internal operation and λ as the neutral element. A *language* L over Σ is a subset of Σ^*. The elements of L are called *words* or *strings*. Given $x \in \Sigma^*$, if $x = uv$ with $u, v \in \Sigma^*$, then u (resp. v) is called the *prefix* (resp. *suffix*) of x. Let us denote the set of prefixes (suffixes) of x by $Pr(x)$ (resp. $Suf(x)$), and the natural extension of the prefixes (suffixes) operation to a language L by $Pr(L)$ (resp. $Suf(L)$). Let us also recall here the definition of the *canonical order* over Σ^* as being the order that first classifies the shorter strings and considers the alphabetic order for those strings of the same length.

A *finite automaton* (*NFA*) is a 5-tuple $A = (Q, \Sigma, \delta, I, F)$, where Q is a finite set of states, Σ is an alphabet, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states and $\delta : Q \times \Sigma \to 2^Q$ is the transition function, which will also be seen as $\delta \subseteq Q \times \Sigma \times Q$. Given an automaton A and a state $q \in Q$, we denote the right language of q in A as R_q^A, that is, the language accepted by the automaton $A = (Q, \Sigma, \delta, \{q\}, F)$. The language accepted by the automaton A will be denoted as $L(A)$ and can be defined as $L(A) = \bigcup_{q \in I} R_q^A$. Two automata are *equivalent* if they recognize the same language. Given any two states p, q of the automaton, let $\prec$ be the relation defined as $p \prec q$ if and only if $R_p^A \subseteq R_q^A$.

Given any two automata $A_1 = (Q_1, \Sigma_1, \delta_1, I_1, F_1)$ and $A = (Q_2, \Sigma_2, \delta_2, I_2, F_2)$, the *disjoint union of automata* is defined as the automaton $A_1 \uplus A_2 = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta_1 \cup \delta_2, I_1 \cup I_2, F_1 \cup F_2)$.

Let $S \subset \Sigma^*$ be finite. The *maximal automaton* for S is the NFA $MA(S) = (Q, A, \delta, I, F)$ where: $Q = \bigcup_{x \in S} \{(u, v) \in \Sigma^* \times \Sigma^* \ : \ uv = x\}$, $I = \{(\lambda, x) \ : \ x \in S\}$, $F = \{(x, \lambda) \ : \ x \in S\}$, and where $\delta((u, av), a) = (ua, v)$ for $(u, av) \in Q$. Note that, when so defined, $L(MA(S)) = S$.

A *deterministic* finite automaton (*DFA*) is an automaton such that $card(I) = 1$ and, for every state q and every symbol a, the number of transitions $\delta(q, a)$ is at most one.

Given a language L over Σ, the (left) quotient of L by a string u is defined as the language $u^{-1}L = \{v \in \Sigma^* \ : \ uv \in L\}$. Let us also define the set $S_L = \{u^{-1}L : u \in \Sigma^*\}$ and the set $U_L = \{u_1^{-1}L \cap \cdots \cap u_k^{-1}L \ : \ k \geq 0, \ u_1, \ldots, u_k \in \Sigma^*\}$. Note that whenever the language L is regular, the sets S_L and U_L are finite.

The minimal DFA for a language L is $A = (S_L, \Sigma, \delta, \{q_0\}, F)$, where: $q_0 = \lambda^{-1}L = L$; $F = \{u^{-1}L \ : \ u \in L\}$; and $\delta(u^{-1}L, a) = (ua)^{-1}L$ for any state in S_L and any symbol a of the alphabet. The *universal automaton* [12] for a language L is $\mathcal{U}_L = (U_L, \Sigma, \delta, I, F)$, where $I = \{p \in U_L \ : \ p \subseteq L\}$, $F = \{p \in U_L \ : \ \lambda \in p\}$ and the set of transitions is defined as $\delta(u^{-1}L, a) = \{p \in U_L \ : \ p \subseteq (ua)^{-1}L\}$. We note here that, given a regular language L, for every automaton A that recognizes L, there is a morphism ψ such that $\psi(A)$ is a subautomaton of the universal automaton for L [39, 43].

A Moore machine is a 6-tuple $M = (Q, \Sigma, \Delta, \delta, q_0, \Phi)$, where Σ (resp. Δ) is the input (resp. output) alphabet, δ is a partial function that maps $Q \times \Sigma$ in Q, and Φ is a function that maps Q in Δ called *output function*. The behavior of M is given by the partial function $t_M : \Sigma^* \to \Delta$ defined as $t_M(x) = \Phi(\delta(q_0, x))$, for every $x \in \Sigma^*$ such that $\delta(q_0, x)$ is defined.

Given two disjoint finite sets of words D_+ and D_-, we define the (D_+, D_-)-*prefix Moore machine* ($PTMM(D_+, D_-)$) as the Moore machine having $\Delta = \{0, 1, ?\}$, $Q = \text{Pr}(D_+ \cup D_-)$, $q_0 = \lambda$, and $\delta(u, a) = ua$ if $u, ua \in Q$ and $a \in \Sigma$. For every state u, the value of the output function associated to u is 1, 0 or ? (undefined) depending on whether u belongs to D_+, to D_-, or to $Q - (D_+ \cup D_-)$.

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 $\Phi(x) = 0$. Note that a *DFA* $A = (Q, \Sigma, \delta, q_0, F)$ can be simulated by a Moore machine $M = (Q, \Sigma, \{0, 1\}, \delta, q_0, \Phi)$, where $\Phi(q) = 1$ if $q \in F$ and $\Phi(q) = 0$ otherwise. Thus, the language defined by M is $L(M) = \{x \in \Sigma^* : \Phi(\delta(q_0, x)) = 1\}$.

A Mealy machine is a 6-tuple $M = (Q, \Sigma, \Delta, \delta, q_0, \Phi)$, where all the elements are defined as they are in the Moore machine except Φ. In this machine, the function Φ maps $Q \times \Sigma$ in Δ. The behavior of M is given by the partial function $t_M : \Sigma^* \to \Delta$ that, given $x = a_1, a_2, \ldots, a_n \in \Sigma^*$, is defined as $t_M(x) = \Phi(q_0, a_1)\Phi(q_1, a_2) \ldots \Phi(q_{n-1}, a_n)$, where $q_0, q_1, \ldots, q_n$ is such that $\delta(q_{i-1}, a_i) = q_i$ for $1 < i \leq n$.

4.3 Inference of Deterministic Automata

In this section we review the most important contributions to the inference of DFAs. First, we present the seminal works by Gold [32] and Trakhtenbrot–Barzdin [44]. We then analyze the *RPNI* algorithm [41] and the algorithm proposed by Lang in [38]. Both of these are extensions of the works by Gold and Trakhtenbrot–Barzdin. This section also reviews the works that guide the merging of states by heuristics, based on the general scheme of the *RPNI* and Lang algorithms.

4.3.1 Non-merging Algorithms: Trakhtenbrot–Barzdin and Gold

The algorithm we describe below can be ascribed to Trakhtenbrot and Barzdin [44] and also to Gold [32]. The algorithm proposed by Trakhtenbrot and Barzdin was published in a book on automata theory and in a context that is different from the GI context. Trakhtenbrot and Barzdin tackled the automata synthesis task using a uniform and complete set of strings from the language. This set is represented using a finite tree that is complete up to a certain string length, and it contains the information of membership to the language for each string in the set. This set of data concerning positive and negative information is then represented using the *PTMM* that considers the sets D_+ and D_- (which is the input of the algorithm).

In an independent way (and in the context of the GI field), Gold proposes an algorithm that converges to the minimal DFA of a regular language using complete presentation samples. Gold uses an evidence table to represent the available data, and his algorithm outputs a Mealy machine. The variety of formalisms that are used may hide the equivalence of the two algorithms from the reader. Here we use a prepresentation that is close to the Trakhtenbrot–Barzdin representation. Algorithm 4.3.1 shows this GI method, which hereafter we will call *TBG*.

Algorithm 4.3.1 Trakhtenbrot–Barzdin and Gold algorithm

```
1: Input: Two disjoint finite sets $(D_+, D_-)$
2: Output: A consistent Moore Machine
3: Method
4: $M_0 = PTMM(D_+, D_-) = (Q_0, \Sigma, \{0, 1, ?\}, \delta, q_0, \Phi_0)$;
5: $R = \{\lambda\}$;
6: while $\exists s' \in R\Sigma - R \ : \ \forall s \in R,\ od(s, s', M_0) = True$ do
7:    choose $s'$;
8:    $R = R \cup \{s'\}$;
9: end while
10: $Q = R$;
11: $q_0 = \lambda$;
12: for $s \in R$ do
13:    $\Phi(s) = \Phi_0(s)$;
14:    for $a \in \Sigma$ do
15:       if $sa \in R$ then $\delta(s, a) = sa$
16:       else $\delta(s, a) =$ any $s' \in R$ such that $od(sa, s', M_0) = False$
17:       end if
18:    end for
19: end for
20: $M = (Q, \Sigma, \{0, 1, ?\}, \delta, q_0, \Phi)$;
21: if $M$ is consistent with $(D_+, D_-)$ then
22:    Return($M$)
23: else
24:    Return($M_0$);
25: end if
26: End Method.
```

The *TBG* algorithm calls a function to determine whether or not two states are *obviously distinguishable*. Taking M_0 as the initial representation of the training set and two states, s and s', this function is defined as:

$$od(s, s', M_0) = True \Leftrightarrow \exists x \in \Sigma^* \ : \ \begin{cases} \Phi(\delta(s, x)),\ \Phi(\delta(s', x)) \in \{0, 1\} \\ \Phi(\delta(s, x)) \neq \Phi(\delta(s', x)) \end{cases}$$

It is worth noting here that this algorithm does not guarantee the consistency of the output with respect to the input data. The result of the inference process is first checked for consistency (line 21) and when the obtained machine is not consistent the $PTMM(D_+, D_-)$ is output (line 24). Also note that the algorithm is not deterministic. In line 7, the algorithm chooses any state in $R\Sigma - R$ that is obviously different from the states in R. The usual way to deterministically implement this command is to choose s' as the first state in canonical order. In the same way, line 16 is usually implemented to choose the first indistinguishable state in canonical order to sa as the state reached by s using symbol a. Example 4.1 illustrates the behaviour of Algorithm 4.3.1.

Example 4.1 Let us take the finite sample $D_+ = \{\lambda, 00, 10, 11, 010\}$ and $D_- = \{0, 1, 001\}$. Algorithm 4.3.1 first constructs the Moore machine $PTMM(D_+, D_-)$ (shown in Fig. 4.1) where the positive, negative and undefined states are represented with double circles, single circles, and dashed circles respectively. Note that the numbering of the states is consistent with the canonical order of the strings that reach each state. As mentioned above, we will follow this canonical order in order to select the states to be analyzed.

Initially $R = \{1\}$ and $R\Sigma - R = \{2, 3\}$. The empty string allows us to say that $od(1, 2, M_0) = True$. Therefore, $R = \{1, 2\}$ and $R\Sigma - R = \{3, 4, 5\}$. The string λ also shows that $od(1, 3, M_0) = True$, but $od(2, 3, M_0) = False$. This is followed by an analysis of states 1 and 4, which concludes that $od(1, 4, M_0) = False$. State 5 is then considered and it is detected that $od(1, 5, M_0) = True$ (due to the string λ), but that $od(2, 5, M_0) = False$. The output of the algorithm is shown in Fig. 4.2. Note that the output automaton is non-consistent with the input data.

Note that a slight modification of the training data is enough for the algorithm to obtain consistency. Let us consider a new training set where $D_+ = \{\lambda, 00, 10, 11, 010\}$ and $D_- = \{0, 1, 01, 001\}$. Note that the difference lies in string 01, which is now in the negative sample.

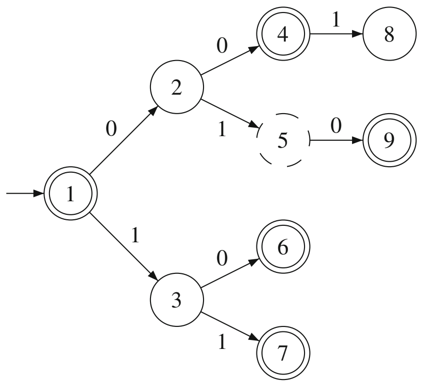

Fig. 4.1 PTMM example when the sets $D_+ = \{\lambda, 00, 10, 11, 010\}$ and $D_- = \{0, 1, 001\}$ are considered

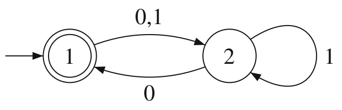

Fig. 4.2 The DFA obtained by the *TBG* algorithm using $D_+ = \{\lambda, 00, 10, 11, 010\}$ and $D_- = \{0, 1, 001\}$. Note that string 11 should be accepted and it is not

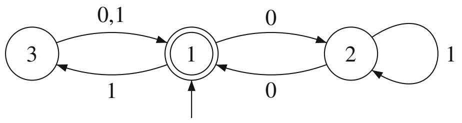

Fig. 4.3 The DFA obtained by the *TBG* algorithm using $D_+ = \{\lambda, 00, 10, 11, 010\}$ and $D_- = \{0, 1, 01, 001\}$ as the training set

In this case, when the finite sample $D_+ = \{\lambda, 00, 10, 11, 010\}$ and $D_- = \{0, 1, 01, 001\}$ is taken into account, Algorithm 4.3.1 first constructs the Moore machine $PTMM(D_+, D_-)$, which is essentially the same as in Fig. 4.1 but where state 5 is a negative state. Initially $R = \{1\}$ and $R\Sigma - R = \{2, 3\}$. The empty string allows us to say that $od(1, 2, M_0) = True$. Therefore, $R = \{1, 2\}$ and $R\Sigma - R = \{3, 4, 5\}$. The string λ also leads to detecting that $od(1, 3, M_0) = True$ and string 1 leads to detecting that $od(2, 3, M_0) = True$. Thus, $R = \{1, 2, 3\}$ and $R\Sigma - R = \{4, 5, 6, 7\}$. This is followed by the analysis of states 1 and 4, and $od(1, 4, M_0) = False$. State 5 is then considered and it is detected that $od(1, 5, M_0) = True$ (due to λ), but $od(2, 5, M_0) = False$. Finally, states 6 and 7 are indistinguishable to state 1, that is, $od(1, 6, M_0) = False$ and $od(1, 7, M_0) = False$. The output of the algorithm is shown in Fig. 4.3. Note that the output automaton is consistent with the input data.

The reason for the inconsistency of the *TBG* algorithm is due to the fact that it carries out all the comparison among states on M_0. The initial automaton (the *PTMM* of the input data) is not modified as a result of these comparisons. In other words, the algorithm does not merge states while it checks consistency. Therefore, for any state whose output is not determined by the training set, it is possible to relate different outputs.

4.3.2 Merging Algorithms: RPNI *and Lang*

As already mentioned, the main drawback of the *TBG* algorithm is that consistence with respect to the input data is not guaranteed. In these cases, the algorithm gives up any generalization and outputs a strict representation of the input data.

In the early 1990s, two algorithms that were proposed independently but were essentially identical solved this problem. These algorithms are: the *RPNI* algorithm (Regular Positive and Negative Inference) [41] and the *Traxbar* algorithm [38]. The main difference of these methods with respect to Algorithm 4.3.1 consists in the

Algorithm 4.3.2 Function for the deterministic merge of states.

```
1: **Input:** A Moore Machine $M = (Q_M, \Sigma, \{0, 1, ?\}, \delta_M, q_0, \Phi_M)$
2: **Input:** Two states $p$ and $q$
3: **Output:** A Moore Machine with $p$ and $q$ deterministically merged (if possible)
4: **Output:** False if the merge of states is not possible
5: **Method**
6: **if** $\{\Phi_M(p), \Phi_M(q)\} = \{0, 1\}$ **then** Return(*False*);
   **end if**
7: $M' = M$;
8: **if** $\Phi_{M'}(p) =?$ **then** $\Phi_{M'}(p) = \Phi_M(q)$;
   **end if**
9: Substitute with $p$ any reference to $q$ in $M'$
10: **for all** $a \in \Sigma$ **do**
11:    **if** both $\delta_M(p, a)$ and $\delta_M(q, a)$ are defined **then**
12:       $M' = detmerge(M', \delta_M(p, a), \delta_M(q, a)$
13:       **if** $M' = False$ **then**
14:          Return(*False*);
15:       **end if**
16:    **end if**
17: **end for**
18: Return($M'$)
19: **End Method.**
```

merging of those indistinguishable states that are detected. Once one of these merges has been carried out, the algorithm considers this new hypothesis and rejects the previous one. As mentioned above, the *TBG* algorithm always compares each pair of states taking into account the initial *PTMM*.

The *RPNI* algorithm uses a function to merge states while assuring that the automaton retains determinism. This operation is carried out by the *detmerge* function that recursively triggers another merge of states whenever the merge of two states implies non-determinism. Note that a given merge may not be possible because it would imply the merge of a positive state and a negative state. In that case, the function returns *False*.

Algorithms 4.3.3 and 4.3.2 show this method, which can be summarized as follows: Let the states of $PTMM(D_+, D_-)$ be ordered canonically; the deterministic merge of each state (based on this order) with all the previous states is checked. In order to do so, the outputs associated to these states must be compatible (the outputs cannot be simultaneously in $\{0, 1\}$ and also to be different); whenever the outputs are compatible, the states can be merged. Nevertheless, the merge can lead to non-determinism and the algorithm tries new merges to handle this situation. These new merges can lead to some incompatibility, in which case the algorithm returns to the previous situation that triggered the process; the algorithm ends when there is no state to be considered.

The first classification of GI algorithms that is possible to make takes into account whether or not the algorithms modify the initial *PTMM* by merging states each time two *compatible* states are found. In order to compare the behaviour of the two approaches, Example 4.2 depicts a run of the *RPNI* algorithm.

Algorithm 4.3.3 *RPNI* algorithm

```
1: **Input:** Two disjoint finite sets $(D_+, D_-)$
2: **Output:** A consistent Moore Machine
3: **Method**
4: $M = PTMM(D_+, D_-)$;
5: //$\{u_0, u_1, ..., u_r\}$ states of $M$ in canonical order, $u_0 = \lambda$//;
6: $R = \{u_0\}$;
7: $B = R\Sigma - R$;
8: while $B$ not empty do
9:     $q =$ canonical order first state in $B$;
10:    $B = B - \{q\}$
11:    $merged = False$;
12:    for all $p$ in $R$ traversed in canonical order do
13:        if $detmerge(M, p, q) \neq False$ then
14:            $merged = True$;
15:            $M = detmerge(M, p, q)$;
16:            BreakFor
17:        end if
18:    end for
19:    if $not\ merged$ then
20:        $R = Append(R, q)$;
21:    end if
22:    $B = R\Sigma - R$;
23: end while
24: Return($M$)
25: End Method.
```

Example 4.2 Let us consider the following sets of samples:

$$D_+ = \{a, aba, abba, abbba\}$$
$$D_- = \{\lambda, b, aa, ab, ba, bb, aaa, abb, baa, bba\}$$

A run of the *RPNI* algorithm first constructs the *PTMM* for D_+ and D_-. Figure 4.4 shows the resulting *PTMM*.

When the first merging $detmerge(M, 1, 2)$ is tried, it fails due to the fact that $\Phi(1) = 0$ and $\Phi(2) = 1$. The next merge to try is $detmerge(M, 1, 3)$, which also fails. Note that both $detmerge(M, 1, 4)$ and $detmerge(M, 2, 4)$ are not possible, but $detmerge(M, 3, 4)$ is possible. This merge triggers the merge of state 6 and state 8. The next state to consider is state 5, which cannot be deterministically merged and is therefore promoted to the set R. State 6 is then considered and deterministically merged with state 3, which triggers the merge of state 3 and state 11. The machine obtained at this point is shown in Fig. 4.5.

The consideration of state 7 leads to the merge of states 3 and 7, which also triggers the merge of states 3 and 12. Now, the states in the set B are 9 and 10. The first state to consider is state 9 which can be deterministically merged with state 2. State 10 can be deterministically merged with state 5. This merge triggers the following merges:

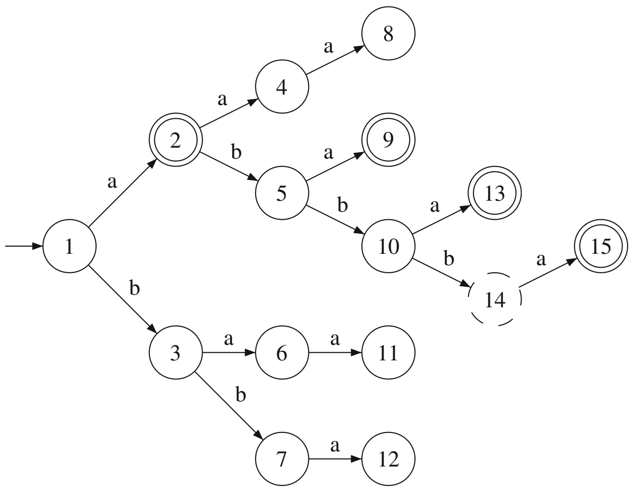

Fig. 4.4 The *PTMM* for $D_+ = \{a, aba, abba, abbba\}$ and $D_- = \{\lambda, b, aa, ab, ba, bb, aaa, abb, baa, bba\}$

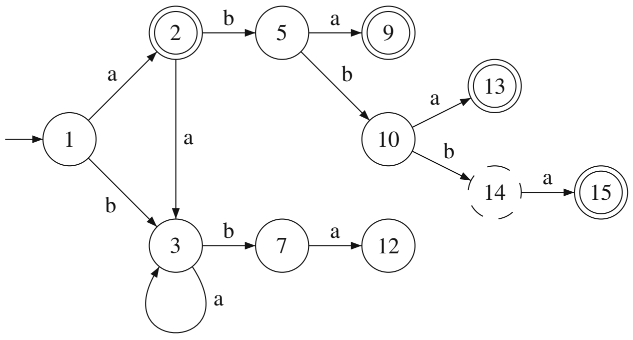

Fig. 4.5 The DFA obtained when the pairs of states $\langle 3, 4\rangle$, $\langle 3, 6\rangle$ of the initial *PTMM* (Fig. 4.4) are deterministically merged

states 2 and 13; states 5 and 14; and, finally, states 2 and 15. These merges end the process because set *B* is now empty. Thus the output of the algorithm is the machine shown in Fig. 4.6.

4.3.3 Algorithms Guided by Heuristics: **EDSM** *and* **Blue-Fringe**

In a run, *RPNI* merges those equivalent states (i.e., the states for which there is no sample that contradicts this equivalence).

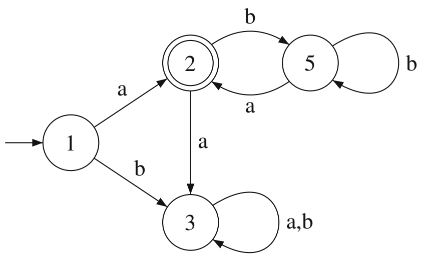

Fig. 4.6 The output of *RPNI* algorithm with input $D_+ = \{a, aba, abba, abbba\}$ and $D_- = \{\lambda, b, aa, ab, ba, bb, aaa, abb, baa, bba\}$

When the training set is not sufficiently representative of the language, it may imply that some *inconvenient* merges could be done. Those merges produce a negative effect on the inference process, leading to lower recognition rates.

In order to avoid as many of these merges as possible, de la Higuera et al. [18] proposed modifying the canonical order used by *RPNI* to favor the merge of states for which there exists great evidence of equivalence. This evidence is measured by a score to be computed for each pair of states. It is expected that the higher the score value, the greater the evidence of equivalence.

The initial results were not conclusive. Nevertheless, this approach was revisited by Price [37] who proposed the *EDSM* (Evidence-Driven State Merging) algorithm. This algorithm had very good behaviour in the Abbadingo contest and was one of the two winners. The *EDSM* strategy is summarized in Algorithm 4.3.4. In this algorithm, we denote by $\Phi(p) \approx \Phi(q)$ that states p and q are compatible, where states p and q are compatible if and only if $\Phi(p) = 1$ (resp. $\Phi(p) = 0$) implies that $\Phi(q) \neq 0$ (resp. $\Phi(q) \neq 1$).

As stated in line 6, the algorithm iterates while mergible states in the current automaton, let's say $A = (Q, \Sigma, \delta, q_0, F)$, are found. Each pair of mergible states p and q is evaluated by the function *FindScore* that, briefly speaking, takes into account the number of coincidences of states with defined output in the automata $A = (Q, \Sigma, \delta, p, F)$ and $A = (Q, \Sigma, \delta, q, F)$. Once all the mergible pairs of states are evaluated, the algorithm merges the pair of states with the highest score, that is, the pair of states with greatest evidence of equivalence. The algorithm ends when all mergible states have been considered.

The main drawback of the *EDSM* strategy is due to the cost of evaluating each pair of mergible states for each merge carried out. The first modification that was proposed to avoid this was to consider only those pairs of states at a given distance W from the initial state. This version is known as *W-EDSM* [11]. A better strategy for selecting the pairs of states to merge is known as *red-blue* [11, 37], which led to the *Blue-Fringe* method which is described in Algorithm 4.3.5. This algorithm is considered to be the *state of the art* with respect to the inference of DFA by merging of states.

Algorithm 4.3.5 uses the *PTMM* of the training sample. First, the *Blue-Fringe* algorithm considers the *red* set that contains only states of the hypothesis. Thus, the algorithm initializes the *red* set using the initial state of the machine. The *blue* set is

Algorithm 4.3.4 *EDSM* algorithm.

```
1: Input: Two disjoint finite sets $(D_+, D_-)$
2: Output: A consistent Moore Machine
3: Method
4: $A = PTMM(D_+, D_-)$;
5: $ok = True$;
6: while $ok$ do
7:    $score = \{\}$;
8:    for $(p, q) \in Q \times Q$ do
9:       if $\Phi(p) \approx \Phi(q) \wedge p \neq q$ then
10:          $score = score \cup \{((p, q), FindScore(M, p, q))\}$;
11:      end if
12:   end for
13:   if $score = \{\}$ then
14:      $ok = False$;
15:   else
16:      $(p, q) = MaximumScorePair(score)$;
17:      $A = detmerge(M, p, q)$;
18:   end if
19: end while
20: Return($A$);
21: End Method.
```

obtained by taking into account the *red* set, which contains those non-*red* states of the hypothesis that are reachable from any state in the *red* set. The algorithm ends when the *blue* set is empty.

In each iteration, the algorithm searches for a *blue* state that is non-mergible with any *red* state. The first of such states detected is promoted to the *red* set and the *blue* set is recalculated. Note that any state in the *blue* set that is not mergible should belong to the hypothesis in order to maintain consistence with the supplied data.

If this is not the case, then all the *blue* states are mergible with (at least) one *red* state. At this stage, the algorithm merges the pair of states with the greatest evidence of equivalence. The quantification of the evidence of equivalence takes into account the coincidences of the output values of the states in the prefix machine and also takes into account the depth of the *red* state (minimal number of transitions to reach the red state from the initial state). When the algorithm ends, it returns the resulting Moore machine.

It is worth noting here that the *RPNI* algorithm can be considered to be a *red-blue* method. In fact, note that if canonical order is considered (which is the usual order considered in the *Blue-Fringe* implementations) and the score computation is not carried out in the the *Blue-Fringe* algorithm, then the algorithms do not differ from each other.

The next example shows the different experimental behaviours of the *Blue-Fringe* and *RPNI* algorithms. Note that the guided merging leads to a more efficient use of the available data.

Algorithm 4.3.5 *Blue-Fringe* algorithm

```
1: Input: Two disjoint finite sets $(D_+, D_-)$
2: Output: A consistent Moore Machine
3: Method:
4: $M = PTMM(D_+, D_-) = (Q, \Sigma, \{0, 1, ?\}, \delta, q_0, \Phi)$;
5: $red = \{\lambda\}$
6: $score = \emptyset$
7: $blue = \{q \in Q : q = \delta(p, a), p \in red \wedge a \in \Sigma\} - red$;
8: while $blue \neq \emptyset$ do
9:   for $q \in blue$ do
10:    $merged = False$;
11:    for $p \in red$ do
12:      if $(p, q)$ have a score in $score$ then
13:        $merged = True$
14:      else
15:        if $p$ and $q$ are mergible $\wedge\ p \neq q$ then
16:          $score = score \cup \{((p, q), 100 * FindScore(M, p, q) + 99 - depth(p))\}$;
17:          $merged = True$
18:        end if
19:      end if
20:    end for
21:    if $not\ merged$ then
22:      $red = red \cup \{q\}$
23:      BreakFor
24:    end if
25:  end for
26:  if $merged$ then
27:    $(p, q) = MaximumScorePair(score)$;
28:    $M = detmerge(M, p, q)$;
29:    $score = \emptyset$
30:  end if
31:  $blue = \{q \in Q | q = \delta(p, a)$ for $p \in red \wedge a \in \Sigma\} - red$
32: end while
33: Return($M$);
34: End Method.
```

Example 4.3 Let us consider the following training sets:

$$D_+ = \{b, bb, aab, aba, bbb\}$$
$$D_- = \{bba\}$$

Figure 4.7 shows $PTMM(D_+, D_-)$. If the *RPNI* algorithm is run with this input, the first merge, $detmerge(M, 1, 2)$, is successful: the merging of the pair of states $(1, 2)$ triggers the merging of the pairs $(1, 4)$, $(3, 5)$, $(3, 7)$. The output of the *RPNI* algorithm is shown in Fig. 4.8.

When the *Blue-Fringe* algorithm is run, initially $red = \{1\}$ and $blue = \{2, 3\}$. The score of the pair $(1, 2)$ is 0; therefore, an evidence of $0 \cdot 100 + 99 - 0 = 99$ is assigned to the pair of states $(1, 2)$. The pair of states $(1, 3)$ lead to the comparison

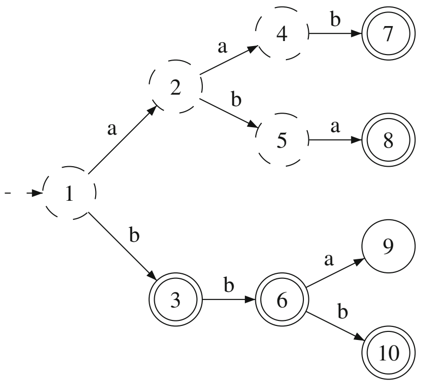

Fig. 4.7 The *PTMM* for $D_+ = \{b, bb, aab, aba, bbb\}$ and $D_- = \{bba\}$

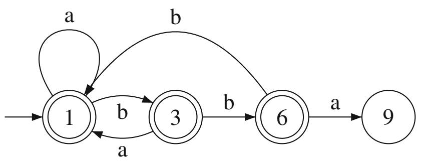

Fig. 4.8 The *RPNI* output when $D_+ = \{b, bb, aab, aba, bbb\}$ and $D_- = \{bba\}$ are considered

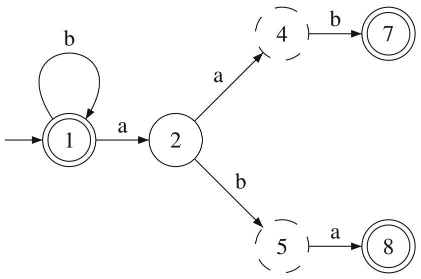

Fig. 4.9 The DFA obtained by the deterministic merging of states 1 and 3 of the machine in Fig. 4.7

of the output values of the pairs (3, 6) and (6, 10), with two coincidences ($\Phi(3) = \Phi(6) = 1$ and $\Phi(6) = \Phi(10) = 1$); therefore, an evidence of $2 \cdot 100 + 99 - 0 = 299$ is assigned. The hypothesis that is obtained by the deterministic merging of states 1 and 3 is shown in Fig. 4.9.

From *red* $= \{1\}$ and *blue* $= \{2\}$, it follows that state 2 cannot be merged with any *red* state, and, therefore, state 2 is promoted to the *red* set. Thus, *red* $= \{1, 2\}$ and *blue* $= \{4, 5\}$. The evidence computed for the pairs of states (1, 4), (2, 4), and (2, 5) are 199, 98, and 98, respectively (note that state 1 and 5 are not mergible). This leads to the merge of states 1 and 4. The new *red* and *blue* sets are *red* $= \{1, 2\}$ and *blue* $= \{5\}$. State 5 can only be merged with state 2, which gives the final automaton shown in Fig. 4.10.

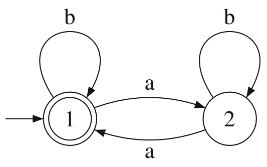

Fig. 4.10 The *Blue-Fringe* output when $D_+ = \{b, bb, aab, aba, bbb\}$ and $D_- = \{bba\}$ are considered

4.3.4 Inference of Teams of Automata

The last method we review in this section is the inference of teams of automata proposed by García et al. in [29]. This work takes into account that there is usually not enough information available to infer the automaton that accepts the target language (the characteristic sample is not available). The approach considers that different orders in the state merging process are able to capture some features of the target language while neglecting others. Therefore, it should be possible to improve the classification rates by considering a set (team) of automata and a vote scheme that weights the result obtained by each individual automaton in the set.

This approach is based on the proof that, under certain conditions, a red-blue algorithm that consistently but arbitrarily merges states in the prefix tree acceptor of the sample converges.

Similar to any red-blue algorithm, the approach described in [29] considers the merging of states but arbitrarily chooses a blue state and tries to merge it with a red one. Only when all merges are proved to not be possible is the state promoted to the red set. This scheme improves the computational efficiency with respect to other methods that give priority to promotion (e.g. the *Blue-Fringe* method).

It is worth noting here that the time complexity of the method is ($\mathcal{O}(k \times n^2)$, where k stands for the number of automata in the team and n stands for the size of the training set), which is better than the best algorithm so far.

Once the team of automata is obtained, several methods can be used to carry out the recognition of test strings. In [29], the authors consider a *fair vote* scheme together with several weighted vote schemes. The experimentation shows that the method obtains better results when the bigger automata (those bigger than the average size in the inferred team) are discarded and the vote of each remaining automaton is inversely proportional to its size.

The authors claim that the use of automata teams increases the probability of good results since the inclusion of the characteristic sample in the input data is not usually assured. This is corroborated by the experimental results shown in [29]. These results are summarized in Sect. 4.5. The authors compare their approach with current state-of-the-art algorithms. It can be seen that their method outperforms previous research approaches.

4.3.5 Identification in the Limit

Despite the differences between the algorithms reviewed hitherto, all three algorithms follow a red-blue scheme. These algorithms identify the class of regular languages in the limit. The respective proofs follow the same approach, that is, for every algorithm and any regular language, there exists a *c*haracteristic set (or sample) of positive and negative data. Whenever any of the three algorithms is run with such a characteristic set as input, the algorithm outputs the minimal DFA for the language. Furthermore, any training set that contains the characteristic set also obtains the same output.

Recently, García et al. studied the convergence of data-driven, red-blue algorithms [23]. In their work, the authors propose a *g*eneral scheme to implement red-blue algorithms. In this general algorithm, any order can be used to traverse the states of the automata. Therefore, it is possible to implement as many red-blue algorithms as orders among the states can be defined. Please, note that *Blue-Fringe* is just an instance of this family of algorithms.

In their work, García et al. prove the existence of a *c*haracteristic sample for any red-blue inference algorithm (whether or not it is data-driven) that uses an *a* priori fixed order to traverse the states, thus proving the convergence of any such algorithms. This result is enunciated in Theorem 4.1.

Theorem 4.1 (García et al. [23]) *Any DFA GI algorithm, such that the promotion of states is independent from the input set, has a polynomial characteristic set, no matter the order followed to carry out the merge of states.*

In order to compute the characteristic set for any given language L, the authors present a method that considers a *minimal set of test states* of the minimal DFA $A = (Q, \Sigma, \delta, q_0, F)$ for L. This set of test states consists of any set of strings such that, if for every $q \in Q$, there exists only one string in the set such that $\delta(q_0, x) = q$. Example 4.4 illustrates this concept.

Example 4.4 Let us consider the automaton in Fig. 4.11. Note that it is possible to obtain several sets of test states by taking into account different orders to traverse the states. For instance, according to the alphabetic order, the set $S = \{\lambda, a, aa\}$ would be a minimal set of test states, and, according the canonical order, $S' = \{\lambda, a, b\}$ would be the minimal test set. Note that, for each state q, each one of the minimal test sets contains the first string that reaches q based on the chosen order (alphabetic or canonic).

Briefly speaking, for any language L, the proof of Theorem 4.1 takes into account a prefix-closed minimal set of test states obtained according to a defined order. The authors prove that Algorithm 4.3.6 outputs two sets $D_+(S)$ and $D_-(S)$ that are a characteristic sample for a red-blue algorithm using this defined order to identify the language L. Example 4.5 depicts how the algorithm works.

Example 4.5 Let us again consider the automaton in Fig. 4.11 and the prefix-closed minimal sets of test states $S = \{\lambda, a, aa\}$. The following table summarizes the process

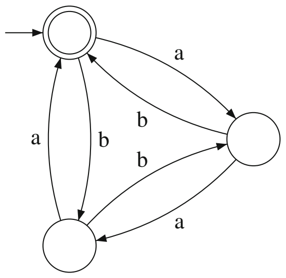

Fig. 4.11 DFA example

Algorithm 4.3.6 Algorithm to obtain the characteristic set for a language L.

```
Require: The minimal DFA $A$ for $L$
Ensure: The polynomial characteristic set for $L$
 1: Method
 2: Let $S$ be the minimal set of test states for $A$
 3: $E = \{\lambda\}$
 4: Let $S' = S\Sigma - S$
 5: Let $T$ be a matrix indexed by the strings $u \in S \cup S'$ and $e \in E$ that stores the membership of
    the string $ue$ to the language
 6: while there exist two undistinguished $u, v \in S$ and a symbol $a \in \Sigma$ such that $T[ua, e] \neq$
    $T[va, e]$ for some $e \in E$ do
 7:   $E = E \cup \{ae\}$
 8: end while
 9: for each row $u$ and column $v$ in $T$ do
10:   if $T[u, v] = 1$ then
11:     Add $uv$ to $D_+$
12:   else
13:     Add $uv$ to $D_-$
14:   end if
15: end for
16: Return($D_+$, $D_-$);
17: End Method.
```

of obtaining the characteristic set. For the sake of clarity, we represent the elements in S and those in $S\Sigma - S$ separately. Initially the only column available is the one with label λ. The 1 and 0 entries in the table represent whether or not the strings obtained by concatenation of the strings that label the row and column belong to the language L.

	λ	b
λ	1	0
a	0	1
aa	0	0
b	0	0
ab	1	0
aaa	1	0
aab	0	1

Note that taking into account just the column labelled λ, the undistinguished elements in S are $\{a, aa\}$. It is possible to distinguish the first element using the suffix b. Once the table is filled in, all the elements in S are distinguished; therefore, the characteristic sample for the language is $D_+(S) = \{\lambda, aaa, aabb, ab\}$ and $D_-(S) = \{a, aa, aaab, aab, abb, b, bb\}$. Figure 4.12 shows the corresponding *PTMM* and the numbering of the states.

When the algorithm is run using the canonical order, then the prefix-closed minimal set of test states to be used is $S = \{\lambda, a, b\}$ and the sets output by the algorithm are $D_+(S) = \{\lambda, ab, ba, bbb\}$ and $D_-(S) = \{a, b, aa, bb, aab, abb, bab\}$. Figure 4.13 shows the corresponding *PTMM* and the order to traverse the states.

Obviously, it is not possible to ensure the inclusion of a characteristic sample in the training set. In any case, in this context, a characteristic sample depends directly on the promotion order of the states that a red-blue algorithm uses. This implies that

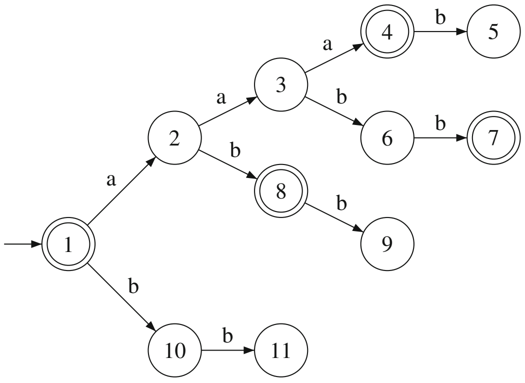

Fig. 4.12 The *PTMM* for $D_+(S) = \{\lambda, aaa, aabb, ab\}$, $D_-(S) = \{a, aa, aaab, aab, abb, b, bb\}$ and an alphabetic numbering of the states

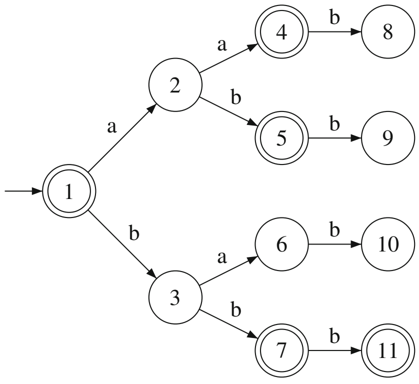

Fig. 4.13 The *PTMM* for $D_+(S) = \{\lambda, ab, ba, bbb\}$, $D_-(S) = \{a, b, aa, bb, aab, abb, bab\}$ and a canonical numbering of the states

the same training set may or may not be representative depending on the promotion order implemented.

Thus, the consideration of several arbitrary orders (and therefore a team of automata) is convenient and motivates the approach presented in [29]; here we note that the experimental results confirm the authors' hypothesis.

Let us finally note the similarity between the inference of teams of automata and the *Blue-Fringe* algorithm. In essence, *Blue-Fringe* looks for evidence of the best order to merge the states. The team approach uses different orders in order to obtain a team of automata and classifies any test string by combining those inferred automata.

4.4 Inference of Non-deterministic Automata

Recently, a new line of work has proposed the inference of non-deterministic automata (NFA) instead of the usual deterministic model. In 1994, Yokomori published a paper that proposes a method that considers queries to an oracle and counterexamples [16, 47]. Coste and Fredouille also studied the inference of NFAs [13–15] proposing a method to infer unambiguous finite automata.

Later, Denis et al. [19, 21] focused their work on the inference of a subclass of NFA, the *Residual Finite State Automata* (RFSA). The consideration of this subclass of NFAs is interesting because it has been proved that there is a unique canonical RFSA for each regular language. In [19], Denis et al. propose an algorithm (known as the *DeLeTe2* algorithm) that converges to a RFSA. Unfortunately, the hypothesis that the algorithm outputs is not always consistent with the supplied training data. To solve this drawback, the authors propose the *DeLeTe2* program [21], which always returns a consistent machine. In the same paper, the authors presented an experimental comparison with respect to other well-known methods such as *RPNI* and *Blue-Fringe*. The experimentation considers data from languages represented by NFAs and regular expressions, and the results obtained show that *DeLeTe2* performs better than the *RPNI* and *Blue-Fringe* methods.

Among the recent work on NFA inference, it is worth citing some works in the field. In [10, 33, 35, 36, 40, 43] results are presented on the minimality and reduction of automata. Other papers study methods that converge to a RFSA [3]. In [1, 14, 15], the authors propose methods to infer *Unambiguous Finite Automata* (UFA). There are also works that propose extensions of the *RPNI* scheme that consider language inclusion relations as a way to increase the training data [26]. Finally, in [46], the authors study the inference of NFAs by juxtaposition of subautomata.

Other works that are related to this topic are [27, 28]. These papers study the influence of the merging order on the convergence of the algorithms. It is proved that whenever some more general conditions are fulfilled, convergence is assured, regardless of the order in which the non-deterministic merges have been done.

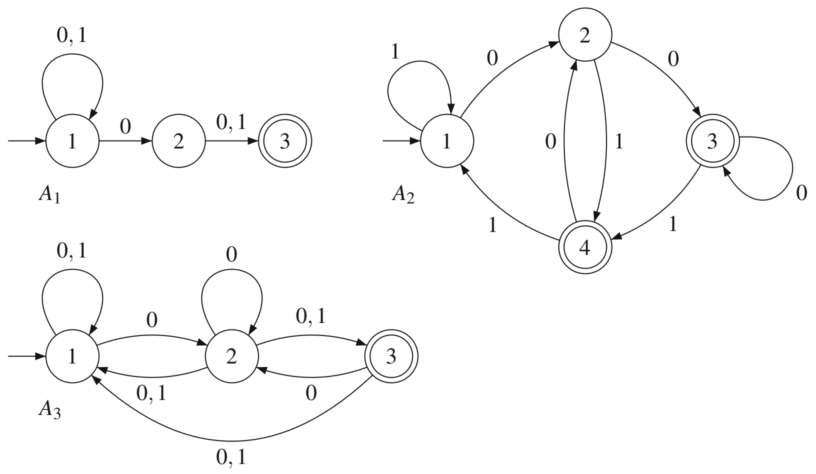

Fig. 4.14 Example from [20]. A_1 is an automaton that recognizes $L = \Sigma^*0\Sigma$, but it is neither a DFA nor a RFSA (note that $\nexists u \in \Sigma^*$ such that $R_3^{A_1} = u^{-1}L$). The automaton A_2 is a DFA that recognizes L (it is also a RFSA by definition). A_3 is the canonical RFSA for L

4.4.1 Inference of Canonical NFAs

Given a language L, a *residual finite state automaton* (RFSA) [20] for the language is a finite automaton A that accepts the language L and, for any state q, there exists a word u such that $R_q^A = u^{-1}L$. The *saturated* RFSA of a minimal DFA A is defined as the subautomaton of the universal automaton $\mathscr{U}_L$ that is induced by the set of states of the automaton A.

In other words, a RFSA A is a non-deterministic automaton such that all its states define a residual language of $L(A)$. Note that every DFA fulfills the conditions for being RFSA. Figure 4.14 shows three automata for the same regular language $L = \Sigma^*0\Sigma$. Note that there is a NFA, a DFA, and the canonical RFSA, all of which identify L.

Two relationships that are defined over the set of states of an automaton link the RFSA with GI. Let $D = (D_+, D_-)$ be a set of samples and let u, v be two strings in $Pr(D_+)$. First, the relation $\prec$ is redefined as $u \prec v$ if there is no string w such that $uw \in D_+$ and $vw \in D_-$. Note that this is important because, in a grammatical inference process, right languages of the states are unknown. Second, if $u \prec v$ and $v \prec u$, the function *obviously different* is defined as $od(u_1, u_2, PTMM(D_+, D_-)) =$ *False* (non-obviously different using Gold's terms). In the sequel, we will denote this condition as $u \simeq v$.

It is well known that every regular language has a finite set of residual languages (see Nerode's theorem in any language theory text, for instance [34]). It is also known that every regular language has a DFA that recognizes the language and that any given DFA is also a RFSA. Therefore, any regular language can be represented using a

RFSA [20]. Among the RFSA that identify a regular language, the smallest one, the *canonical RFSA*, is of special interest. Formally speaking, for a given language $L \subseteq \Sigma^*$, the canonical RFSA of L is the automaton $A = (Q, \Sigma, \delta, I, F)$ where:

- $Q = \{u^{-1}L \ : \ u^{-1}L$ is prime, $u \in \Sigma^*\}$, where prime quotients are such that they cannot be obtained by the union of other quotients.
- $\delta(u^{-1}L, a) = \{p \in Q \ : \ p \subseteq (ua)^{-1}L\}$
- $I = \{p \in Q \ : \ p \subseteq L\}$
- $F = \{p \in Q \ : \lambda \in p\}$

The canonical RFSA for a given language L is unique, and it is the smallest RFSA that recognizes L (taking into account the number of states). If $R_q^A = u^{-1}L$, then the string u is said to be characteristic of state q. The size of the canonical RFSA is upper bounded by the size of the minimal DFA for the language and lower bounded by the minimal equivalent NFAs [20].

Under these conditions, several situations may occur: the canonical RFSA may be exponentially smaller than the minimal equivalent DFA; both automata may be of the same size and there may be an exponentially smaller NFA than both the minimal DFA and the canonical RFSA; there may be a minimal NFA with the same number of states but with few transitions. Taking into account the length of the characteristic strings, it is possible for the smallest characteristic string of a given state to have an exponential length with respect to the size (number of states) of the canonical RFSA. This may occur when the characteristic strings are computed taking into account the minimal DFA and also where this automaton has an exponential size with respect to the canonical RFSA.

All these situations show that any inference algorithm that converges to the canonical RFSA does not ensure the output to be minimal. Despite this, it is possible to improve the results whenever the target language L is characterized by the fact that the DFA for L is made up of many composite residual languages. It remains to be studied how common such languages are. The practical use of RFSAs depends on the results of that study.

Denis et al. studied the inference of RFSAs in [22]. In their work, the authors propose a method to infer residual automata from positive and negative data. The authors claim this method converges to the saturated subautomata of the minimal DFA. The main drawback of the method is that, in some circumstances, the method does not guarantee consistency. The implementation of the method provides a solution to this drawback, but the implementation does not completely correspond to the algorithm. Section 4.4.1.1 reviews the initial method, which will hereafter be referred to as the *DeLeTe2* algorithm. Section 4.4.1.2 reviews the implementation of the method, which will be referred to as the *DeLeTe2* procedure.

4.4.1.1 The *DeLeTe2* Algorithm

The *DeLeTe2* procedure is shown in Algorithm 4.4.1. We have rewritten the algorithm in [22] in order to relate it with the other algorithms mentioned above. We point out that (leaving aside implementation details) Algorithm 4.4.1 and the method described in [22] are equivalent.

As the authors state in [22], this automata can be obtained by saturating the minimal DFA A and reducing all the states that are greater than p, where p is the greatest prime state in A.

The algorithm looks for inclusion relations among residual languages and considers them using a saturation operator. As commented above, this method is of special interest when the target automaton has many non-prime residual languages. In this situation, many inclusion relationships may be found, and this leads to a smaller hypothesis. Whenever the target automaton has many prime residual languages, the size of the hypothesis and the size of the minimal DFA is expected to be the same [22].

Algorithm 4.4.1 *DeLeTe2* algorithm.

```
1: **Input:** Two finite sets of data $D_+ \cup D_-$ over $\Sigma$
2: **Output:** A finite automaton
3: **Method:**
4: Let $Pref$ be the set of prefixes of $D_+ \cup D_-$
5: $R = \{\lambda\}$
6: $B = (R\Sigma - R) \cap Pref$
7: $Q = R$
8: $I = \delta = F = \emptyset$
9: $A = (Q, \Sigma, I, \lambda, F)$
10: while $B \neq \emptyset$ do
11:   $I = \{q \in Q \ : \ q \prec \lambda\}$
12:   $F = Q \cap D_+$
13:   for $p \in R$ do
14:     $\delta = \{(p, a, r) \ : \ r \in R \ \wedge \ r \prec pa\}$
15:   end for
16:   if $A$ is not consistent with respect to $(D_+, D_-)$ then
17:     Let $C = \{q \in B \ : \ \nexists p \in R, \ p \simeq q\}$
18:     if $C = \emptyset$ then
19:       $Return(A)$
20:     end if
21:     choose $q \in C$
22:     $R = R \cup \{q\}$
23:     $B = (R\Sigma - R) \cap Pref$
24:   else
25:     $B = \emptyset$
26:   end if
27: end while
28: $Return(A)$
29: End Method:
```

It is worth noting that the *TBG* method (Algorithm 4.3.1) is closely related to the *DeLeTe2* algorithm. Also note that when the *TBG* algorithm is going to add a transition, say for a state s and a symbol a, the algorithm selects one of the possible states which is not distinguishable from sa (line 16 in Algorithm 4.3.1). When the *DeLeTe2* algorithm builds the set of transitions, it adds a transition (p, a, r) for each

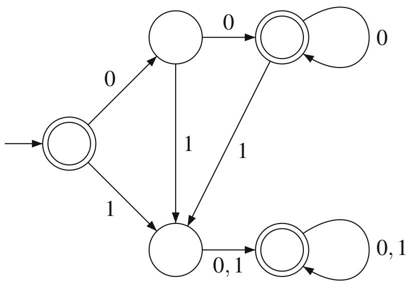

Fig. 4.15 The minimal DFA for $L = \lambda + 00^+ + 0^*1(0+1)^+$

state $r \prec pa$. Thus, if line 16 of the *TBG* algorithm is modified to work the way *DeLeTe2* works, then the *TBG* algorithm would output the minimal saturated DFA for the language (when a characteristic sample is input).

The reason why the *DeLeTe2* algorithm can output smaller automata than the minimal DFA is that this algorithm builds a hypothesis whenever a new state is added. The algorithm ends when one of these hypotheses is consistent with the data. Nevertheless, an important drawback of the *DeLeTe2* algorithm is that, in some cases, it outputs non-consistent automata. The following example illustrates this.

Example 4.6 Let us consider the language $L = \lambda + 00^+ + 0^*1(0+1)^+$. The minimal DFA for L is shown in Fig. 4.15.

When the sets $D_+ = \{\lambda, 00, 10, 11, 010\}$ and $D_- = \{0, 1, 01, 001\}$ are provided, the *RPNI* algorithm outputs the automaton shown in Fig. 4.3. Note that the automaton does not accept the string 10111 that belongs to L. Although not explicitly considered in the *DeLeTe2* algorithm, in order to illustrate the run of the algorithm we will take into account the *PTMM* of the sample shown Fig. 4.1 (if the state 5 is considered to be negative). Thus, the *DeLeTe2* algorithm first considers the state λ. The algorithm sets $Q = \lambda$ and also adds this state to the set of initial states (it fulfills that $\lambda \prec \lambda$) and to the final states (λ is in D_+). The algorithm takes into account the relationships among the state λ and the set of *blue* states $\{0, 1\}$. Since no relationship is detected, no transitions are added. Then, the first hypothesis is the automaton $A = (\{\lambda\}, \{0, 1\}, \{\}, \{\lambda\}, \{\lambda\})$. Since this hypothesis is not consistent with the data, a new state is added.

When the first state that is obviously distinguishable from the those in Q (according a canonical order) is considered, the state 0 is taken into account. The relationships that are to be analyzed consider the sets of states $\{\lambda, 0\}$ and $\{1, 00, 01\}$. Thus, it is detected that $0 \prec 1$, $\lambda \simeq 00$, $0 \prec 00$ and $0 \simeq 01$. The second hypothesis is shown in Fig. 4.16.

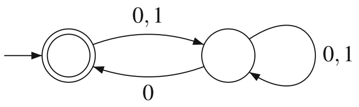

Fig. 4.16 The second hypothesis of the *DeLeTe2* algorithm. Note that the automaton does not accept 001

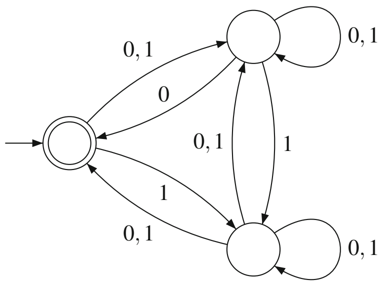

Fig. 4.17 The RFSA output by the *DeLeTe2* algorithm for $D_+ = \{\lambda, 00, 10, 11, 010\}$ and $D_- = \{0, 1, 01, 001\}$

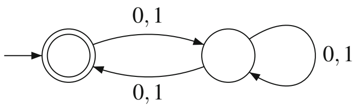

Fig. 4.18 The automaton output by the *DeLeTe2* algorithm using. $D_+ = \{\lambda, 00, 10, 11, 010\}$ and $D_- = \{0, 1, 001\}$. Note that the automaton is not consistent with the data

This hypothesis is also non-consistent with respect to the data, and the next added state is 1. The relationships among the sets of states $\{\lambda, 0, 1\}$ and $\{00, 01, 10, 11\}$ are to be taken into account. Thus, it is detected that $1 \simeq 01$, $\lambda \simeq 10$, $0 \prec 10$, $1 \prec 10$, $\lambda \simeq 11$, $0 \prec 11$ and $1 \prec 11$. The third hypothesis, which is shown in Fig. 4.17, is consistent with the data and therefore is output. Please note that this automaton accepts L and that it is smaller than the minimal DFA.

Let us consider the same example but where the string 01 does not belong to D_-. Figure 4.18 shows the automaton output in this case. This automaton is not consistent with the data. It accepts the string 001 which is in D_-.

Convergence of the Inference of RFSAs To prove the convergence of *DeLeTe2*, it is possible to use an argument that is similar to the one used to prove the *RPNI* convergence. Therefore, it is possible to build a characteristic sample that ensures accessibility to all of the states of the target automaton and that contains information of the possible relations of inclusion.

In [22], the authors prove that the number of strings in the characteristic sample is polynomial with respect to the size of the minimal DFA for the language. Note that, for a given language L, the minimal DFA for L can be exponentially bigger than the canonical RFSA for L.

4.4.1.2 An Improved Implementation

As shown above, the *DeLeTe2* algorithm does not guarantee consistence. In order to solve this drawback, the authors propose an improved implementation of the algorithm, which is usually referred to as the *DeLeTe2* procedure. The authors do not study the theoretical properties of the implementation, and experimentation cannot guarantee the convergence of the *DeLeTe2* procedure to the saturated subautomata of the minimal DFA.

The *DeLeTe2* procedure also represents the hypothesis obtained by RFSAs. Its source code is available on the home web page of the authors. To our knowledge, there is only one work that has studied the behaviour of this implementation [2]. In that work, the author takes into account the source code and detects that both (the *DeLeTe2* algorithm and the *DeLeTe2* procedure) behave differently. The main features of the *DeLeTe2* procedure are summarized below:

- It is a regular language GI algorithm based on state merging.
- The algorithm considers the *PTMM* of the training set and outputs a RFSA. The algorithm takes advantage of several properties of the residual languages associated to each state. Specifically, the transitivity of the inclusion relationship is very useful.
- The hypotheses that the algorithm obtains are always consistent with respect to the supplied data.

The *DeLeTe2* procedure strategy considers each pair of states of the *PTMM*, in canonical order, in order to determine the relationship of the states, that is, the *inclusion*, *no inclusion*, or *unknown* relation of the residual languagest that are associated to the states of the pair.

When an *unknown* relationship is going to be considered, it is temporarily set to *inclusion* and propagated. If this assumption generates inconsistency, the initial assumption is discarded, along with any other relationship found by the triggered one. We note that this kind of assumption may imply several consequences, such as the definition of unknown outputs of the *PTMM* or the discovery of new *inclusion* or *no inclusion* relationships (among others). The time complexity of the *DeLeTe2* procedure is bounded by $\mathcal{O}(n^4)$.

4.4.2 Inference by Juxtaposition of Automata

One advantage of using non-deterministic models to infer regular languages is that this model is more concise than the deterministic one. It is worth noting that, in this sense, the use of RFSA does not ensure that we obtain good (small) representations of the target languages. This is because, for a given language, the size of the minimal RFSA can be exponentially bigger than the size of a minimal NFA for the same language. As an example, consider the following family of languages indexed by n:

$$\{0^i : i \text{ has a divisor greater than 1 and lower than n}\}$$

In this family, the number of states of the NFAs grows polynomially with respect to n, while the number of states of the RFSAs grows exponentially with respect to n.

Algorithm 4.4.2 Word Associated Subautomata Regular Inference (*WASRI*) general scheme.

```
1: **Input:** Two finite sets $D_+ = \{x_1, x_2, \ldots, x_n\}$ and $D_-$
2: **Output:** A finite automaton consistent with the input sets
3: **Method:**
4: $A = (Q, \Sigma, \delta, I, F)$ where $Q = \delta = I = F = \emptyset$
5: for $i = 1$ to $n$ do
6:   if $x_i \notin L(A)$ then
7:     for at least $j = 1$ do
8:       Infer an automaton $A_i^j$ consistent with $D_+ = \{x_i\}$ and $D_-$
9:       $A = A \uplus A_i^j$
10:      //let us recall that $\uplus$ stands for disjoint union of the automata//
11:    end for
12:  end if
13: end for
14: for all component (automaton) $A_i \in A$ do
15:   if Component $A_i$ is not necessary to accept $D_+$ then
16:     Remove the component $A_i$ from $A$
17:   end if
18: end for
19: Return $A$
20: End Method.
```

In [46], the authors propose a family of algorithms that infer the regular language class in the limit. The general scheme of this family is shown in Algorithm 4.4.2. Briefly speaking, for each string of the positive sample u, the method obtains at least one irreducible consistent automaton (i.e., an automaton such that the merging of any two states in it makes the resulting automaton accept negative strings). In order to infer such an automaton, the method takes into account the automaton that only recognizes the positive string u and merges states while it is possible. The automaton to be output is obtained by disjoint union of the inferred automata.

This method is quite flexible because the input parameters of the method are the number of automata to infer for each word as well as the order of state merging. The authors prove that convergence is ensured even if the these parameters change.

The method also includes an option to filter the collection of automata obtained (the loop in line 14 is just one possibility to do so). For example, if several automata for each word have been obtained (using different merging order criteria), it is possible to consider a criterion to select one (or some) of them (for instance, the smallest in size).

Algorithm 4.4.2 implements this filter by the deletion of some of the automata in the output collection (of course, as long as the resulting automaton still accepts D_+).

The complexity of the *WASRI* algorithm is $\mathscr{O}(kn^2|D_-|)$, where k is an integer, n is the length of the longest word of D_+, and $|D_-|$ is the sum of the lengths of the negative words of the sample. It is worth mentioning that the time complexity depends on the length of each word and not on the sum of the lengths of the input (i.e., the size of the prefix tree acceptor of the sample). This fact makes *WASRI* a very fast algorithm.

An interesting example to illustrate the behaviour of *WASRI* is the language of strings in 0^* whose length is a multiple of either 2, 3, or 5. The minimal DFA for L has the same number of states as the canonical RFSA (30 states). Taking into account the sets $D_+ = \{0^2, 0^3, 0^5\}$ and $D_- = \{0, 0^{11}\}$, Algorithm 4.4.2 returns an automaton with 10 states that identifies the language. We also note that *RPNI* and *DeLeTe2* (running with the same input) return automata that are far from convergence.

Convergence of the *WASRI* Algorithm To prove the convergence of any *WASRI* implementation, for any given target language L, the authors consider the universal automaton for the language $\mathscr{U}_L$. This automaton allows a *universal sample* D_+ to be computed. It is proved that, from $MA(D_+)$ every irreducible automaton in $\overline{L}$ that can be obtained accepts the target language. The authors also show that this theoretical condition can be replaced by the construction of a (finite) set of negative samples D_-, which contains at least one string in $merge(p, q, MA(D_+)) - L$ for any pair of states p and q in $MA(D_+)$.

In other words, when D_- is taken into account, for any string in L, it is possible to obtain an irreducible automaton that accepts a sublanguage of the target language. This process can be iterated as many times as necessary using the strings in D_+. Note that convergence is ensured because any automaton that recognizes a sublanguage of L can be projected into the universal automaton for L and there exists a finite set of subautomata of the universal automaton.

4.4.3 Order-Independent Merging Inference

In [28], the authors propose a state-merging algorithm that, given a *universal sample* as input, converges to a nondeterministic finite automaton that recognizes the target language independently of the order in which the states are merged.

The definition of universal sample is closely related to the notion of *automata irreducibility*, where an automaton A is said to be irreducible if the automaton is such that the merge of any pair of states leads to an automaton that accepts a super-language of the target language. The authors extend this notion and define an automaton A as being *irreducible in a regular language* L if and only if $L(A) \subseteq L$ and the merge of any pair of states leads to an automaton that accepts a super-language of L.

Taking this into account, for every regular language L, a universal sample for L is a finite set $D_+ \subseteq L$ with the property that any partition over the maximal automaton for D_+ that produces an irreducible automaton in L produces an automaton that identifies L.

It is worth noting that, while the irreducibility of an automaton in a regular target language L needs $\overline{L}$ to be known, it is possible to prove that it is enough to achieve irreducibility using only a finite set of negative samples.

In [28], the authors use some theoretical tools (especially the universal automaton for the language) that help to prove the convergence of the method. These tools also help to clarify and simplify ideas about the convergence of other inference algorithms, including those that may be proposed in the future. Even though different orders of merging states may lead to different hypotheses (automata), note that, when convergence is achieved, the language accepted by those automata will be the target language.

In the same work, the authors use the theoretical results to propose the *Order-Independent Learning* (*OIL*) family of algorithms. This scheme is described in Algorithm 4.4.3. The authors prove that this algorithm identifies the family of regular languages in the limit.

When a set of blocks of positive and negative samples for the target language L is given to the algorithm, an automaton that recognizes L in the limit is obtained. Note that a block may contain just a single word, so the algorithm is presented here in a very general way.

Briefly, the method first builds the maximal automaton for D^1_+ and merges the states in a random order until the algorithm obtains an irreducible automaton in D^1_-. Then the algorithm performs the following steps for every new block:

1. If the existing automaton is consistent with the new block, nothing has to be done.
2. If it is consistent with the new set of negative samples, the algorithm considers only the positive words that were not accepted by the previous hypothesis. Then the algorithm builds the maximal automaton for the new set of positive words, adds the new negative words to D_-, and finds a partition of the states of the automaton until an irreducible automaton in D_- is obtained.
3. Otherwise, the algorithm runs a recursive call taking the following into account in each step: the corresponding set of positive samples and the whole set of negative samples. This part of the algorithm (line 19) overcomes the fact that even though we may have a universal sample, the negative samples may not lead to consistency.

Algorithm 4.4.3 Order-Independent Learning (*OIL*) general scheme.

```
1: Input: A sequence of finite sets $\langle (D^1_+, D^1_-), (D^2_+, D^2_-), \ldots, (D^n_+, D^n_-) \rangle$
2: Output: A finite automaton consistent with the input sets
3: Method:
4: $A = MA(D^1_+)$
5: $D_- = D^1_-$
6: Find a partition $\pi$ of the states of $A$ irreducible in $\overline{D_-}$
7: $A = A/\pi$
8: for $i = 2$ to $n$ do
9:     $D_- = D_- \cup D^i_-$
10:    if $A$ is consistent with $(D^i_+, D^i_-)$ then Continue end if
11:    if $A$ is consistent with $D^i_-$ then
12:        $S_+ = D^i_+ - L(A)$
13:        $A' = (Q', \Sigma, \delta', I', F') = MA(S_+)$
14:        $A = (Q \uplus Q, \Sigma, \delta \uplus \delta', I \uplus I', F \uplus F')$
15:        //where $\uplus$ stands for disjoint union //
16:        Find a partition $\pi$ of the states of $A$ irreducible in $\overline{D_-}$
17:        $A = A/\pi$
18:    else
19:        $A = OIL(D^1_+, D_-), (D^2_+, D_-), \ldots, (D^i_+, D_-))$
20:    end if
21: end for
22: Return $A$
23: End Method.
```

Note that Algorithm 4.4.3 is presented in a very general way and there are many possible ways to obtain a partition of the set of states (lines 6 and 16 of the algorithm).

One possible implementation of the algorithm could consider a random ordering of the states in $MA(D^1_+)$. Thus, the algorithm would analyze the merging of the states in the chosen order in order to obtain an irreducible automaton in D_-.

At every step, the algorithm considers only those words of the new block (e.g. D^i_+) that are not accepted by the hypothesis at that moment. The algorithm then proceeds to analyze the merging of the states (according to a random ordering) of the maximal automaton obtained from the remaining words.

Whenever the current hypothesis is not consistent with the new block of negative samples, the previous hypothesis is discarded. A recursive call takes into account the whole set of negative samples seen up to that point. The following example illustrates this implementation.

Example 4.7 Let $L = a^* + b^*$, and let the input sample be divided into the following blocks:

$$\begin{array}{ll} D^1_+ = \{a, bb, aa\} & D^1_- = \{ab, bba\} \\ D^2_+ = \{b, aaa\} & D^2_- = \{aaab, aab\} \\ D^3_+ = \{\lambda, bbb, aaaa\} & D^3_- = \{abb, ba\} \end{array}$$

The *OIL* algorithm starts considering the first block (D^1_+, D^1_-). The maximal automaton $MA(D_+)$ for this block is shown in Fig. 4.19, in which the order among

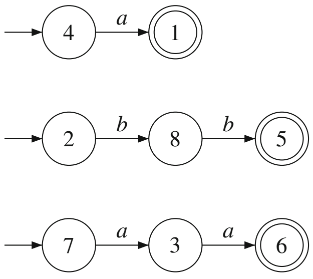

Fig. 4.19 The maximal automaton for the sample $D_{+}^{1} = \{a, bb, aa\}$ with a randomly generated order of the states

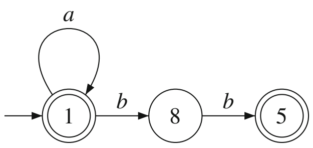

Fig. 4.20 The first hypothesis obtained by *OIL*

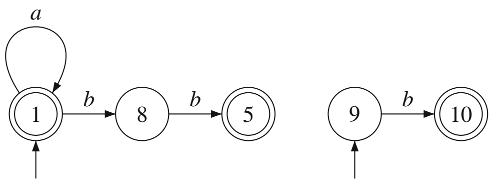

Fig. 4.21 The automaton that considers the relevant information in the second block of samples

the states has been randomly generated. The algorithm follows this order and makes all the possible merges. Thus, states 1 and 2 are (non-deterministically) merged as well as states 1 and 3, states 1 and 4, states 1 and 6, and states 1 and 7.

Note that, in this process, state 1 cannot be merged with state 5 because it would lead to accepting the word $bba \in D_{-}^{1}$. The pair of states 1 and 8 cannot be merged for a similar reason. The algorithm obtains the automaton shown in Fig. 4.20, which is irreducible with respect to D_{-}^{1}.

Once the first hypothesis has been obtained, the algorithm starts processing the second block. First, the algorithm checks for consistency with the current hypothesis. Thus, the automaton shown in Fig. 4.20 is consistent with the negative samples $D_{-}^{2} = \{aaab, aab\}$. Second, the algorithm selects those strings in D_{+}^{2} that are not accepted by the current hypothesis. With the remaining positive strings, a new maximal automaton is built and incorporated into the hypothesis. Note that the ordering of the states has been randomly generated, starting with $N + 1$, where N is the number of states of the maximal automaton for the previous block ($MA(D_{+}^{1})$). The resulting automaton is shown in Fig. 4.21.

Now, the algorithm, which is controlled by the set of negative samples (i.e. $D_{-} = D_{-}^{1} \cup D_{-}^{2} = \{ab, bb, aab, aaab\}$), tries to merge the states in this order. The possible merges at this point are state 1 with state 10, and state 5 with state 9. Note that the merging of state 1 with state 9 would cause the word *ab* to be accepted. The merging of the previous states produces the second hypothesis depicted in Fig. 4.22.

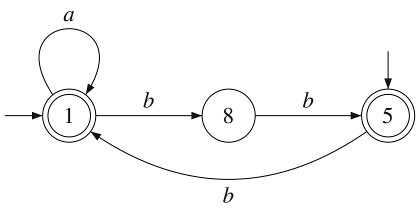

Fig. 4.22 The second hypothesis obtained by *OIL*

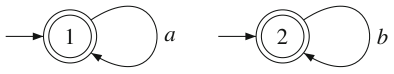

Fig. 4.23 The final automaton output by *OIL*

The algorithm starts the processing of the third block of samples. The algorithm checks the third block of the sample for consistency with the current hypothesis and detects that there are words in the negative sample that are accepted by the hypothesis (for instance *abb*). Therefore, the algorithm processes the positive samples D_+^1, D_+^2, and D_+^3 again. This process is controlled by the whole set of negative samples, that is, $D_- = \{ab, abb, aab, aaab, abb, ba\}$.

If we consider the previous enumeration of the states in the maximal automaton, in this run states 1 and 2 cannot be merged since the resulting automaton would accept *abb*. States 1 and 5 cannot be merged either since *bba* would be accepted. States 1 and 3, states 1 and 4, states 1 and 7, states 2 and 5, and states 2 and 8 are merged. Figure 4.23 shows the output of the algorithm that recognizes the target language.

Convergence of the *OIL* Algorithm Despite the non-deterministic nature of any implementation of *OIL*, the convergence of this algorithm is proved in [28]. The argument is quite similar to the one used to prove the convergence of *WASRI*. Thus, for any given target language L, the authors consider the universal sample D_+ of L.

As described above, the *OIL* algorithm considers a pair of sets of positive D_+^i and negative samples D_-^i in each iteration. Let us denote the sets of positive and negative information available at a given iteration i with $D_+^{\leq i}$ and $D_-^{\leq i}$, respectively.

First, note that the universal sample for the target language L will eventually be available after n iterations, that is $D_+ \subseteq D_+^{\leq n}$. Second, as stated in Sect. 4.4.2, there exists a finite subset of negative samples D_- that avoids any undesired merging of states over the maximal automaton $MA(D_+)$. This set will also be available after $m \geq n$ iterations ($D_- \subseteq D_-^{\leq m}$).

Note that *OIL* can output an automaton that recognizes L whenever enough negative samples are provided. If convergence has not yet been achieved before the m iteration, then the current hypothesis will accept words that are not in L, and, therefore, the hypothesis will not be consistent with $D_-^{\leq m}$. In this case, the algorithm runs a recursive call taking into account $D_{+}^{\leq m}$ and $D_{-}^{\leq m}$, which will return a correct automaton.

4.5 (Experimental) Comparison of Different Approaches

In order to compare the behaviour of their results, in [22], Denis et al. built a corpus of data. This corpus has been subsequently used by several other authors, and, therefore, it is quite useful for comparing a variety of GI algorithms. This corpus contains data for 120 languages represented by NFAs and 120 represented by regular expressions. Each set of 120 languages was processed and distributed into four sets of 30 languages. A training set of 50 strings was generated for each language in the first set, and training sets of 100, 150 and 200 strings were generated for each language in the second, third, and fourth sets. The maximum length of the strings was set to 30. A test set of 1000 strings was also generated for every language in the corpus. Each set was guaranteed to contain, at least, 20 % of positive and negative strings.

Coste and Fredouille [14] extend this dataset to consider languages represented by DFAs and Unambiguous Finite Automata (UFAs). Their dataset is built following the distribution described above. Therefore, this new dataset contains information for 240 target languages (120 represented by DFAs and 120 represented by UFAs). Each one of the test sets for each language also contains 1000 strings.

In order to test the evolution behaviour of the algorithms with respect to the size data available, the training sets should be incremental. Note that neither of these corpora are incremental, and, furthermore, each set may contain several occurrences of the same string. In order to solve this problem, in [2], the author takes into account the distinct languages represented in these corpora and generates new training and test data for each language. The new corpus considers 102 languages represented by regular expressions, 120 languages represented by NFAs, and 119 languages represented by DFAs. For each of these languages, a training set of 500 strings was generated, labelled by the target language and incrementally distributed into five sets of 100, 200, 300, 400, and 500 strings. In addition, a disjoint set of 1000 strings was also generated to be used as the test set. The length of all the strings varied from 0 to 18. This corpus is the one that is used in the experimental comparison below.

In order to evaluate the *learning rate*, we first show the results of a naive baseline algorithm, the *Majority* algorithm. This algorithm considers the number of positive and negative strings in the training set. Thus, if there are more positive than negative strings, the algorithm classifies the whole test set as positive (otherwise, it accordingly classifies the whole test set as negative). Table 4.1 shows the results.

The first comparison we show includes the *RPNI*, *Blue-Fringe* and *DeLeTe2* algorithms (data from [2]). Table 4.2 shows the results. Taking into account the subset of regular expressions, all three algorithms obtained very good recognition rates. Nevertheless, *DeLeTe2* obtained the best results even with the smaller training sets. *DeLeTe2* was also the best algorithm when the NFA subset was considered. Note

Table 4.1 The results of the (naive) baseline algorithm using the experimental dataset

Tr. set ident.	*Majority* Rec. rate (%)
er_100	66.33
er_200	66.35
er_300	66.46
er_400	66.27
er_500	66.16
nfa_100	66.94
nfa_200	67.00
nfa_300	66.97
nfa_400	67.04
nfa_500	67.03
dfa_100	72.13
dfa_200	72.13
dfa_300	72.13
dfa_400	72.13
dfa_500	72.14

Table 4.2 The experimental results and average size of the inferred automata for the *Blue-Fringe*, *RPNI*, and *DeLeTe2* algorithms

	Blue-Fringe		*RPNI*		*DeLeTe2*	
	Rec. rate (%)	av. $\|A\|$	Rec. rate (%)	av. $\|A\|$	Rec. rate (%)	av. $\|A\|$
er_100	87.94	10	83.35	12.39	91.65	30.23
er_200	94.81	9.97	93.91	11.52	96.96	24.48
er_300	96.46	11.05	96,32	11.17	97.80	31.41
er_400	97.74	10.43	97.45	10.98	98.49	27.40
er_500	98.54	10.47	98.11	10.95	98.75	29.85
nfa_100	68.15	18.83	66.50	20.31	73.95	98.80
nfa_200	72.08	28.80	69.27	32.35	77.79	220.93
nfa_300	74.55	36.45	72.90	40.86	80.86	322.13
nfa_400	77.53	42.58	74.59	49.75	82.66	421.30
nfa_500	80.88	47.54	76.75	55.91	84.29	512.55
dfa_100	69.12	18.59	66.29	20.27	62.94	156.89
dfa_200	77.18	25.83	71.01	31.11	64.88	432.88
dfa_300	88.53	25.10	80.61	33.33	66.37	706.64
dfa_400	94.42	21.36	87.39	31.90	69.07	903.32
dfa_500	97.88	18.75	91.67	29.61	72.41	1027.42

that *RPNI* is the algorithm that obtained the lowest recognition rates with this subset. Finally, we considered the DFA subset. With this subset, the algorithm with the best behaviour up to that point becames the worst one and *Blue-Fringe* became the best one.

When the recognition rates obtained with the baseline algorithm are compared, note that the recognition rates of the three algorithms using the subset of regular expressions beat the rates obtained by the baseline algorithm. When the NFA subset is taken into account, on the one hand, both the *RPNI* and the *Blue-Fringe* algorithms had poor learning rates using the smaller training sets of the corpus. On the other hand, *DeLeTe2* notably improved the results of the baseline. Finally, the smaller training sets of the DFA subset were apparently not big enough for any of the algorithms, and all three obtained even lower recognition rates than the baseline. Despite this, *RPNI* and *Blue-Fringe* quickly improved their results and obtained close to 100 % results with the bigger training sets. Nevertheless, *DeLeTe2* was only able to reach the baseline rate with the bigger set.

In Table 4.3, we also show the comparative results of an instance of *WASRI*, *Blue-Fringe*, and *DeLeTe2*. This data is from [45] and considers the regular expression and the NFA subsets of the corpus.

Before commenting on the results, in order to properly compare the behaviour of the algorithms, the instance of *WASRI* used in the experiments should be described in detail. In this experimentation, two automata were inferred for each string in the training set. The first inference process considered the states of the maximal automaton for the string in canonical order. The second inference process considered the states in inverse canonical order. The smaller automaton was chosen. It should be noted that *WASRI* was the algorithm that best behaved with small training sets. An overall comparison shows that *WASRI* behaved better than *Blue-Fringe*, and also behaved in a way that is quite similar to *DeLeTe2* with respect to the recognition rates

Table 4.3 The experimental results of an instance of *WASRI* compared with the *Blue-Fringe* and the *DeLeTe2* algorithms

	Blue-Fringe	*WASRI*		*DeLeTe2*
	Rec. rate (%)	Rec. rate (%)	av. $\|A\|$	Rec. rate (%)
er_100	87.94	93.4	25.3	91.65
er_200	94.81	95.7	33.4	96.96
er_300	96.46	96.2	47.0	97.80
er_400	97.74	97.1	51.3	98.49
er_500	98.54	97.6	48.6	98.75
nfa_100	68.15	74.7	82.9	73.95
nfa_200	72.08	77.1	170.4	77.79
nfa_300	74.55	79.7	267.2	80.86
nfa_400	77.53	81.1	350.1	82.66
nfa_500	80.88	83.5	431.2	84.29

Table 4.4 The experimental results of *OIL* compared with the *Blue-Fringe* and *DeLeTe2* algorithms

	Blue-Fringe Rec. rate (%)	*OIL*			*DeLeTe2* Rec. rate
		Smallest $\|A\|$		Fair vote	
		Rec. rate (%)	av. $\|A\|$	Rec. rate (%)	
er_100	87.94	93.79	8.27	93.32	91.65
er_200	94.81	97.83	7.80	97.27	96.96
er_300	96.46	98.77	7.68	98.68	97.80
er_400	97.74	99.20	7.55	99.10	98.49
er_500	98.54	99.66	6.82	99.53	98.75
nfa_100	68.15	75.00	21.46	76.42	73.95
nfa_200	72.08	78.05	35.23	79.94	77.79
nfa_300	74.55	81.27	45.81	82.94	80.86
nfa_400	77.53	83.87	52.40	85.58	82.66
nfa_500	80.88	85.64	58.81	87.06	84.29
dfa_100	69.12	60.17	28.01	60.34	62.94
dfa_200	77.18	63.05	49.63	63.54	64.88
dfa_300	88.53	66.01	65.17	67.41	66.37
dfa_400	94.42	69.12	78.66	70.53	69.07
dfa_500	97.88	72.29	88.30	73.66	72.41

achieved. With regard to the size of the automata inferred, note that this instance of *WASRI* obtained smaller automata than *DeLeTe2*, but bigger than *RPNI* and *Blue-Fringe*.

In [27], the authors carried out a comparative experimentation of *OIL* and other algorithms. Due to the non-deterministic nature of *OIL* five runs of the algorithm were carried out in their work for each language in the corpus. The results of two different approaches are given: the first approach considers the classification rate of the smallest automaton obtained; the second approach carries out the classification using a poll where the weight of the vote of all the automata is the same. Table 4.4 summarizes these results.

Using the regular expression or NFA subsets of the corpus, both approaches behaved in a similar way and improved previous results obtained by *RPNI*, *Blue-Fringe*, *DeLeTe2*, or *WASRI*. Nevertheless, the performance of *OIL* with the DFA subset was quite similar to the performance of *DeLeTe2* and far from other algorithm rates.

We recall the results obtained by the inference of a team of automata. In [29], the authors proposed their method and used the same dataset described above to compare the experimental behaviour of the approach with *Blue-Fringe* and *DeLeTe2*. Several sizes of the team were considered, and the classification was carried out taking into account a weighted vote scheme (inverse to the square of the sizes of the automata). An option to select those automata that were smaller than the average size of the team

Table 4.5 The results obtained by the inference of teams of automata method

Set	Team Inference (81 FA)			Blue-Fringe	DeLeTe
	No select.	Sel. FA size smaller average			
	cl. rate	cl. rate	#*FA*		
er_100	95.19	95.55	50.00	87.94	91.65
er_200	98.32	98.45	58.24	94.81	96.96
er_300	98.78	98.87	60.58	96.46	97.80
er_400	99.24	99.32	63.45	97.74	98.49
er_500	99.42	99.49	65.91	98.54	98.75
nfa_100	77.24	77.45	40.51	68.15	73.95
nfa_200	80.96	81.25	41.28	72.08	77.79
nfa_300	83.46	83.77	41.26	74.55	80.86
nfa_400	85.14	85.50	41.47	77.53	82.66
nfa_500	86.71	86.98	42.25	80.88	84.29
dfa_100	76.68	76.83	40.48	69.12	62.94
dfa_200	83.13	84.04	36.00	77.18	64.88
dfa_300	90.04	91.59	36.00	88.53	66.37
dfa_400	95.24	96.48	36.31	94.42	69.07
dfa_500	98.16	98.68	37.69	97.88	72.41

was also studied. Table 4.5 shows the results obtained using a team of 81 automata. This table shows the size of the resulting team when the bigger automata are discarded (column #*FA*). Note that regardless of whether or not the selection of automata was carried out, the approach improved the classification rates obtained by both *Blue-Fringe* and *DeLeTe2*.

As Table 4.5 shows, comparing the average size of the automata inferred and leaving aside the inference of teams of automata where the size depends on the size of the team, the smallest automata were obtained by the *Blue-Fringe* algorithm followed by the *RPNI* and *OIL* algorithms (which obtained the smallest automata for the regular expression subset). The methods that obtained the largest hypotheses were *WASRI* and *DeLeTe2*.

4.6 Conclusions

The inference of finite deterministic automata has been thoroughly studied since it was proposed in the 1970s. As a result of this work, many methods have been proposed that provide solutions to the problem with good time and space complexity.

The consideration of different orderings in the process of state merging has been proved to be relevant, so these can be applied to practical tasks. Nevertheless, no order has been proved to be the best, and the final selection depends on the nature and characteristics of the practical task. Recently, it has been proved that, no matter what order is considered, convergence is not an issue when some minimum conditions are fulfilled.

Non-deterministic automata can be exponentially more concise than deterministic automata, and this can be of particular interest in some practical contexts. This is the main reason why the inference of non-deterministic finite automata arouses interest nowadays. Some works have proposed algorithms that consider different approaches.

The main drawback for the inference of non-deterministic automata is the lack of a canonical representative for any regular language. The definition and inference of residual automata is an attempt to obtain such a canonical representation. Other approaches are focused on obtaining a set of several automata, with the aim of achieving a team that is able to identify the target language so that each individual automata concisely recognizes a fragment of that language. Another drawback of these methods is that they have worse time complexity than those that infer deterministic automata.

In our opinion, the inference of non-deterministic models deserves to be studied further, and the study of time-efficient NFA inference or the inference of *small* NFAs may provide interesting results.

References

1. J. Abela, F. Coste, and S. Spina. Mutually compatible and incompatible merges for the search of the smallest consistent DFA. *LNAI*, 3264:28–39. 7th International Colloquium, ICGI-04. Springer. 2004.
2. G. I. Álvarez. *Estudio de la Mezcla de Estados Determinista y No Determinista en el Diseño de Algoritmos para Inferencia Gramatical de Lenguajes Regulares*. PhD Thesis in Computer Science, Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, 2007. in Spanish.
3. G. I. Álvarez, J. Ruiz, A. Cano, and P. García. Non-deterministic regular positive negative inference NRPNI. In J.F. Díaz, C. Rueda, and A.A. Buss, editors, *Proc. of the XXXI Conferencia Latinoamericana de Informática (CLEI2005)*, pages 239–249, 2005. ISBN:958-670-422-X.
4. D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39:337–350, 1978.
5. D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.
6. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.
7. D. Angluin. Queries and Concept Learning. *Machine Learning*, 2(4):319–342, 1988.
8. D. Angluin. Negative Results for Equivalence Queries. *Machine Learning*, 5(2):121–150, 1990.
9. R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. *Lecture Notes in Artificial Intelligence*, 862:139–152, 1994.
10. J. M. Champarnaud and F. Coulon. NFA reduction algorithms by means of regular inequalities. *Theoretical Computer Science*, 327(3):241–253, 2004. Erratum in TCS 347:437-440 (2005).
11. O. Cichello and S. C. Kremer. Inducing grammars from sparse data sets: A survey of algorithms and results. *Journal of Machine Learning Research*, 4:603–632, 2003.
12. J. H. Conway. *Regular Algebras and Finite Machines*. Chapman & Hall, London, 1974.
13. F. Coste and D. Fredouille. Efficient ambiguity detection in C-NFA. a step towards the inference of non-deterministic automata. *LNAI*, 1891:25–38. 5th International Colloquium, ICGI-00. Springer. 2000.

14. F. Coste and D. Fredouille. Unambiguous automata inference by means of state-merging methods. *LNCS*, 2837:60–71, 2003. 14th European Conference ECML-03.
15. F. Coste and D. Fredouille. What is the search space for the inference of non-deterministic, unambiguous and deterministic automata? Technical Report 4907, INRIA, 2003. Internal Report Project Symbiose.
16. C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38:1332–1348, 2005.
17. C. de la Higuera. *Grammatical Inference. Learning Automata and Grammars*. Cambridge University Press, 2010.
18. C. de la Higuera, J. Oncina, and E. Vidal. Identification of DFA: data-dependent vs data-independent algorithms. *LNAI*, 1147:313–325. 3rd International Colloquium, ICGI-96. Springer. 1996
19. F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using non-deterministic finite automata. *LNAI*, 1891:39–50. 5th International Colloquium, ICGI-00. Springer. 2000
20. F. Denis, A. Lemay, and A. Terlutte. Residual finite state automata. *Fundamenta Informaticae*, 51(4):339–368, 2002.
21. F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using non-deterministic finite automata. *LNCS*, 1891:39–50, 2004.
22. F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using RFSA. *Theoretical Computer Science*, 313(2):267–294, 2004.
23. P. García, D. López, and M. Vázquez de Parga. Polynomial characteristic sets for DFA identification. *Theoretical Computer Science*, 448:41–46, 2012.
24. P. García, A. Cano, and J. Ruiz. A comparative study of two algorithms for automata identification. *LNAI*, 1891:115–126. 5th International Colloquium, ICGI-00. 2000.
25. P. García and J. Ruiz. Learning in varieties of the form $V * LI$ from positive data. *Theoretical Computer Science*, 362(1–3):100–114, 2006.
26. P. García, J. Ruiz, A. Cano, and G. I. Álvarez. Is learning RFSAs better than learning DFAs? *LNCS*, 3845:343–344, 2005. 10th International Conference on Implementation and Application of Automata (CIAA-05).
27. P. García, M. Vázquez de Parga, G. I. Álvarez, and J. Ruiz. Learning regular languages using nondeterministic finite automata. *LNCS*, 5148:92–102, 13th International Conference on Implementation of Automata (CIAA'08). Springer. 2008.
28. P. García, M. Vázquez de Parga, G. I. Álvarez, and J. Ruiz. Universal automata and NFA learning. *Theoretical Computer Science*, 407(1-3):192–202, 2008.
29. P. García, M. Vázquez de Parga, D. López, and J. Ruiz. Learning automata teams. *LNAI*, 6339:52–65. 10th International Colloquium, ICGI-10. Springer. 2010.
30. P. García and E. Vidal. Inference of k-testable Languages in the Strict Sense and application to Syntactic Pattern Recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence*, pages 920–925, 1990.
31. E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
32. E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.
33. G. Gramlich and G. Schnitger. Minimizing NFA's and regular expressions. *Journal of Computer and System Sciences*, 73(6):908–923, 2007.
34. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
35. L. Illie and S. Yu. Reducing NFAs by invariant equivalences. *Theoretical Computer Science*, 306(1-3):373–390, 2003.
36. T. Kameda and P. Weiner. On the state minimization of nondeterministic finite automata. *IEEE Trans. on Computers*, 19(7):617–627, 1970.
37. K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. *LNAI*, 1433:1–12. 4th International Colloquium, ICGI-98. Springer. 1998.

38. K.J. Lang. Random DFA's can be approximately learned from sparse uniform examples. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 45–52, 1992.
39. S. Lombardy and J. Sakarovitch. The universal automaton. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 457–504. Amsterdam University Press, 2008.
40. O. Matz and A. Potthoff. Computing small nondeterministic finite automata. In *Tools and Algorithms for the Construction and Analysis of Systems - TACAS '95*, volume NS-95-2 of *BRICS Notes Series*, pages 74–88, 1995.
41. J. Oncina and P. García. Inferring regular languages in polynomial updated time. *Pattern recognition and image analysis*, volume 1, pages 49–61. World Scientific, 1992.
42. J. Oncina, P. Garcia, and E. Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):448–458, 1993.
43. L. Polák. Minimalizations of NFA using the universal automaton. *Int. J. Found. Comput. Sci.*, 16(5):999–1010, 2005.
44. B.A. Trakhtenbrot and Ya. M. Barzdin. *Finite automata. Behavior and Synthesis*. North-Holland, 1973.
45. M. Vázquez de Parga. *Autómatas finitos: Irreducibilidad e inferencia*. PhD Thesis in Computer Science, Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, 2008. in Spanish.
46. M. Vázquez de Parga, P. García, and J. Ruiz. A family of algorithms for non-deterministic regular language inference. *LNCS*, 4094:265–274, 2008. 12th International Conference on Implementation of Automata (CIAA'06).
47. T. Yokomori. Learning non-deterministic finite automata from queries and counterexamples. *Machine Learning*, 13:169–189, 1994.
48. T. Yokomori. Polynomial-time identification of very simple grammars from positive data. *Theoretical Computer Science*, 298:179–206, 2003.