

Efficient ambiguity detection in C -NFA

A step towards the inference of non deterministic automata

François Coste, Daniel Fredouille

IRISA/INRIA Rennes
Campus Universitaire de Beaulieu, 35042 RENNES Cedex, France
Phone : +33 2 99 84 71 00
Fax: +33 2 99 84 71 71
{Francois.Coste|Daniel.Fredouille}@irisa.fr

Abstract This work addresses the problem of the inference of non deterministic automata (NFA) from given positive and negative samples. We propose here to consider this problem as a particular case of the inference of unambiguous finite state classifier. We are then able to present an efficient incompatibility NFA detection framework for state merging inference process.

key words : regular inference, non deterministic automata, finite state classifier, sequence discrimination

Introduction

This work addresses the problem of the inference of non deterministic automata (NFA) from given positive and negative samples. This problem has been extensively studied for the inference of deterministic automata (DFA), for which state merging algorithms have been proven efficient [OG92, Lan92, CN97, LPP98, OS98]. Whereas DFA are polynomially identifiable from given data [Gol78, dlH97], this result does not hold for NFA [dlH97]. In contrast, it is well known that there exist languages such that their representation by DFA requires an exponential number of states with respect to the NFA representation. Considering the inference of NFA instead of DFA allows therefore to obtain smaller solution which we expect to require less samples to be characterized.

Few studies have been made on the inference of NFA. Yokomori [Yok94] has proposed an algorithm that needs an oracle and can infer NFA that determinizes polynomially in polynomial time. We propose here to consider the inference of compatible NFA as a particular case of the inference of unambiguous finite state classifier presented in section 1. A first algorithm for checking unambiguousness of a C -NFA is given in this section. The second section proposes an incremental version of this algorithm for a state merging inference process, ensuring the compatibility of the corresponding NFA without parsing the sample. We conclude with a first experimentation comparing minimum sized NFA and DFA inference with respect to the size of the training sample.

1 Inference of Unambiguous Finite State Classifier

The purpose of this section is to introduce the inference of finite state classifier by means of state merging algorithms. Using this representation allows unbiased inference [AS95, Alq97]. We propose here to take advantage of the simultaneous representation of a set of languages for the inference of unambiguous automata.

1.1 Definitions and notations

Definition 1. A C -classes non deterministic finite state automata (C -NFA) is defined by a 6-tuple $(Q, Q_0, \Sigma, \Gamma, \delta, \rho)$ where: Q is a finite set of states; $Q_0 \subseteq Q$ is the set of initial states; Σ is a finite alphabet of input symbols; Γ is a finite alphabet of C output symbols; δ is the next-state function mapping $Q \times \Sigma$ to 2^Q (if δ maps $Q \times \Sigma$ to Q , the automaton is said deterministic and is denoted by C -DFA); ρ is the output function mapping Q to 2^Γ . The function realized by a C -NFA is the classification of sequences.

The classification function γ mapping $\Sigma^* \times Q$ to 2^Γ is defined by:

$$\gamma(q, w) = \bigcup_{q' \in \delta(q, w)} \rho(q')$$

where δ has been extended to sequences following the classical way by:
 $\forall q \in Q, \forall w \in \Sigma^*, \forall a \in \Sigma \cup \{\epsilon\}, \delta(q, \epsilon) = \{q\}, \delta(q, wa) = \bigcup_{q' \in \delta(q, w)} \delta(q', a)$

The classification of a sequence w by a C -NFA may then be defined as the set of classifications obtained from the initial states. We also denote by γ this function mapping Σ^* to 2^Γ :

$$\gamma(w) = \bigcup_{q \in Q_0} \gamma(q, w)$$

Given a C -NFA M , a sequence w is said *classified* if its classification is defined (ie: $\gamma(w) \neq \emptyset$). The set of classified sequences is named the *domain* of M . The classification over this domain defines a C -tuple of regular languages denoted $\mathcal{L}(M)$:

$$\mathcal{L}(M) = \langle L_c(M) \rangle_{c \in \Gamma} \text{ where } \forall c \in \Gamma, L_c(M) = \{w \in \Sigma^* | c \in \gamma(w)\}.$$

A C -NFA allows to handle simultaneously a set of languages. In this paper, we focus on *unambiguous* C -NFA:

Definition 2. A C -NFA is said *unambiguous* if each sequence is classified in at most one class.

From the definition, it follows that a C -NFA M is unambiguous iff the C -tuple of languages represented by M are mutually disjoint, i.e.: $\forall i, j \in \Gamma, L_i(M) \cap L_j(M) = \emptyset$.

The unambiguousness property is important for the search of compatible automata from positive and negative samples and other applications dealing with discrimination of sequences by finite state machines. The choice of a C -NFA representation of a set of languages, instead of the classical automata representation, allows to efficiently characterize the disjunction of the recognized languages. We propose to take advantage of this property in the next sections devoted to the inference of unambiguous C -NFA.

1.2 State Merging Inference

The problem of inferring a C -NFA may be seen as a C -regular inference problem [Cos99]. We assume that a training sample $\mathcal{S} = \langle S_c \rangle_{c \in \Gamma}$ is given such that each S_c is a sample from the target language $L_c(M)$, i.e. a finite subset of $L_c(M)$.

One classical assumption made in grammatical inference is that the sample is structurally complete with respect to the target machine. Under this assumption, the inference of C -NFA may then be done by means of state merging algorithm, which proceeds by merging states of the *Maximal Canonical Automaton*, denoted by $MCA(\mathcal{S})$, which is the automaton resulting from the union of the canonical C -NFA for each sequence of \mathcal{S} (figure 1 and algorithm 1). When looking for unambiguous C -NFA, the search is pruned as soon as the current automaton is detected ambiguous, since all automata obtained by merging states of an ambiguous automaton are ambiguous.

Detecting ambiguity is simple in the deterministic case. It can be done by checking that no states of different classes have been merged, or even by parsing the automaton with the training set. In the non-deterministic case, parsing may

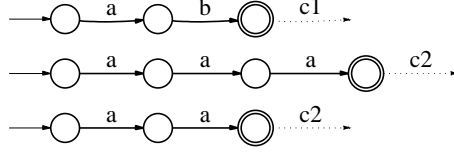


Figure 1. MCA(\mathcal{S}) for $\mathcal{S} = \langle \{ab\}, \{aaa, aa\} \rangle$.

Algorithm 1 Greedy state merging algorithm

```

1: Greedy_SMA( $\mathcal{S}$ )
2: /* Input: training sample  $\mathcal{S}$  */
3: /* Output: a  $C$ -NFA compatible with  $\mathcal{S}$  */
4:  $\mathcal{A} \leftarrow$  Maximal_Canonical_Automaton( $\mathcal{S}$ )
5: while Choose_States_To_Merge( $q_1, q_2$ ) do
6:    $\mathcal{A}' \leftarrow$  Merge( $\mathcal{A}, q_1, q_2$ )
7:   if  $\mathcal{A}'$  is not ambiguous then
8:      $\mathcal{A} \leftarrow \mathcal{A}'$ 

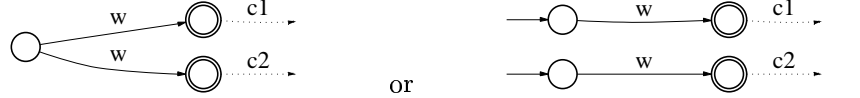
```

be done by a viterbi-like procedure. For classical automata parsing the negative sample is sufficient to ensure compatibility. For non deterministic C -NFA, compatibility with samples and unambiguousness should not be confused: whenever all the samples are correctly labeled by the automaton, sequences outside the training set may have more than one classification. We propose in the next section a first algorithm to detect the ambiguousness of C -NFA.

1.3 Ambiguity detection

Only two cases of ambiguity exist. A C -NFA is ambiguous if:

- There exists a state such that its output function returns two different classifications. For C -DFA, it is the unique case of ambiguity.
- There exist paths labeled by the same sequence w leading to states with **defined and different** classifications.



We introduce the notation $\gamma_1 \not\sim \gamma_2$ (γ_1 incompatible with γ_2) for two different and defined classifications γ_1 and γ_2 :

$$\gamma_1 \not\sim \gamma_2 \Leftrightarrow ((\gamma_1 \neq \gamma_2) \wedge (\gamma_1 \neq \emptyset) \wedge (\gamma_2 \neq \emptyset)).$$

Otherwise, the classifications are said compatible (denoted $\gamma_1 \sim \gamma_2$).

It is easy to detect whether the first case holds. For the second case, we need to introduce the definition of incompatible pair of states. Two states q_1 and q_2 are incompatible, (denoted $q_1 \not\sim q_2$), if there exists a word whose classifications from these states are incompatible:

$$q_1 \not\sim q_2 \Leftrightarrow \exists w \in \Sigma^*, \exists (s_1, s_2) \in \delta(q_1, w) \times \delta(q_2, w), \rho(s_1) \not\sim \rho(s_2)$$

Otherwise, the states are said compatible (denoted $q_1 \sim q_2$).

Then, ambiguity detection for a C -NFA reduces to checking if a state is incompatible with itself or if two initial states are incompatible.

To mark incompatible states, we propose an algorithm (algorithm 2) inspired by the algorithm of Hopcroft and Ullman designed to mark non equivalent states for automaton minimization [HU80]¹. Since the automata we consider are not necessarily deterministic, the original algorithm has been changed by inverting the propagation direction of the marking process, which results in $O(n^2)$ time complexity for tree-like automata. This algorithm may be used to construct the set of incompatible pairs of states $\mathcal{E}_{\not\sim}$ and to raise an exception if it detects ambiguity.

Algorithm 2 Incompatible states and C -NFA ambiguity.

```

1: Incompatible_States( $A = (\Sigma, \Gamma, Q, Q_0, \delta, \rho)$ ):
2: /* Search of the set of incompatible states of  $A$  */
3: /* and ambiguity detection of  $A$  */
4:  $\mathcal{E}_{\not\sim} \leftarrow \emptyset$  /* set of incompatible states */
5: for all  $\{q_i, q_j\} \in Q \times Q, \rho(q_i) \not\sim \rho(q_j)$  do
6:   if  $\{q_i, q_j\} \notin \mathcal{E}_{\not\sim}$  then
7:     Set_Incompatible_And_Propagate( $q_i, q_j$ )
8: return  $\mathcal{E}_{\not\sim}$ 

9: Set_Incompatible_And_Propagate( $q_1, q_2$ ):
10: /* ambiguity detection */
11: if  $(q_1 = q_2) \vee (q_1 \in Q_0 \wedge q_2 \in Q_0)$  then
12:   throw exception("ambiguous  $C$ -NFA")
13: /* Incompatibility memorization */
14:  $\mathcal{E}_{\not\sim} \leftarrow \mathcal{E}_{\not\sim} \cup \{q_1, q_2\}$ 
15: /* Propagation */
16: for all  $a \in \Sigma, \{p_1, p_2\} \in \delta^{-1}(q_1, a) \times \delta^{-1}(q_2, a)$  do
17:   if  $\{p_1, p_2\} \notin \mathcal{E}_{\not\sim}$  then
18:     Set_Incompatible_And_Propagate( $p_1, p_2$ )

```

¹ The partition refinement algorithm to minimize automata may not be used here since the state equivalence relation is transitive whereas the compatibility relation is not.

In the worst case, the complexity of algorithm 2 is $O(|\Sigma|n^4) : O(n^2)$ calls of the function *Set_Incompatible_And_Propagate*, whose body needs $O(|\Sigma|n^2)$ steps. However, if we denote by t_a the maximal number of incoming transitions with the same symbol in a state, one can refine the complexity result. The complexity of *Set_Incompatible_And_Propagate* body with respect to t_a is $O(|\Sigma|t_a^2)$ which leads to a global complexity of $O(|\Sigma|t_a^2n^2)$. Therefore the complexity lies more practically between $O(|\Sigma|n^2)$ and $O(|\Sigma|n^4)$ according to the value of t_a .

In an inference process, this algorithm may be used to determine whether each candidate is unambiguous. In the next section, we propose an incremental version of this algorithm to detect ambiguity in an extension of the classical state merging framework.

2 Considering unmergeable states during inference

We propose here to extend the classical state merging algorithm to consider pairs of *unmergeable* states (denoted for two states q_1 and q_2 of a C -NFA by $q_1 \not\sim q_2$). At each step of the inference, instead of always merging the chosen pair of states, the algorithm will be allowed to set this pair of states unmergeable. This may be used to guide the search or to prune an entire part of the search space: either because it has already been explored or either because it is known that no solution may be found in it.

2.1 Detection of unmergeable states due to ambiguity

During the inference of unambiguous automata, some pairs of states may be detected to have no other choice than being set unmergeable to ensure unambiguity. The first relation that can be used is that two incompatible states are also unmergeable:

$$\forall q_1, q_2 \in Q \times Q, q_1 \not\sim q_2 \Rightarrow q_1 \not\sim q_2.$$

We can detect more unmergeable states by considering the counterpart of merging for determinization used in the deterministic framework [OG92], that is considering pairs of states that are reachable by a common word from the initial states.

Definition 3. *Two states q_1 and q_2 are said to be in relation \parallel , denoted by $q_1 \parallel q_2$, if they are reachable by a common word from initial states. More formally, we have $q_1 \parallel q_2 \Leftrightarrow \exists w \in \Sigma^*, q_1, q_2 \in \bigcup_{q_0 \in Q_0} \delta(q_0, w)$.*

The algorithm computing relation \parallel is very similar to algorithm 2 for incompatible states. The loop in line 5 is replaced by a loop on pairs of initial states, and backward propagation in line 17 is replaced by forward propagation (using δ instead of δ^{-1}). This algorithm can also detect ambiguity since it tries to put in relation \parallel two states with incompatible output.

Thanks to relation \parallel we can detect new unmergeable states with the following equation which is illustrated in figure 2:

$$q_1 \not\sim q_2 \wedge q_2 \parallel q_3 \Rightarrow q_1 \not\sim q_3$$

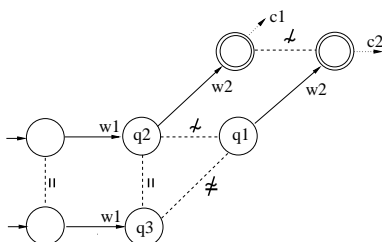


Figure2. Illustration of the equation $q_1 \not\sim q_2 \wedge q_2 \parallel q_3 \Rightarrow q_1 \not\sim q_3$: given a relation $q_2 \parallel q_3$ involved by a word w_1 , and an incompatibility $q_1 \not\sim q_2$ involved by a word w_2 , the merging of q_1 and q_3 is not possible since it entails the acceptance of the word $w_1 w_2$ in two different classes. We can also notice that the relation $q_1 \not\sim q_2 \Rightarrow q_1 \not\sim q_2$ is due to a particular case of $q_1 \not\sim q_2 \wedge q_2 \parallel q_3 \Rightarrow q_1 \not\sim q_3$ with $q_2 = q_3$ thus thanks to the fact that every state is in relation \parallel with itself.

Relation \parallel enables also earlier ambiguity detection: to detect ambiguity, we can check that no incompatible states have to be set in relation \parallel (or that no states in relation \parallel have to be set incompatible). This property comes from the fact that if two states are in relation \parallel due to a word w_1 and that they are incompatible due to a word w_2 , then the word $w_1 w_2$ has an ambiguous classification.

Notice also that this detection can replace the one given in section 1.3 (algorithm 2 line 11) since all initial states are in relation \parallel , and every state is in relation \parallel with itself.

To summarize, before computing a merge we can check in some cases if it will lead to ambiguity, but this checking is not always possible (we do not detect all mergings leading to ambiguity, see figure 3). In this case, ambiguity is detected during the merge thanks to the addition of new relation \parallel and $\not\sim$.

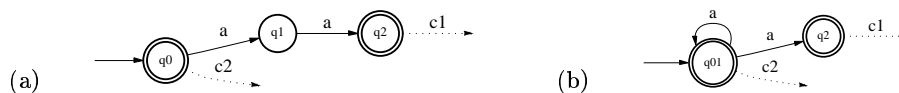


Figure3. part a. States q_0 and q_1 are unmergeable but not detected with our equations (the automaton resulting from the merge, **figure3. part b**, is ambiguous; for example, in this automaton the word aa is both classified c_1 and c_2).

We dispose of various relations between states which are useful not only to detect ambiguity, but also to prevent merging of states that leads to ambiguity. We now propose to maintain these relations after each merge during an inference algorithm.

2.2 Incremental maintenance of relations

Let $\mathcal{E}_{\not\sim}(q)$ (resp. $\mathcal{E}_{\not\sim}(q)$, $\mathcal{E}_{\parallel}(q)$) denote the set of states unmergeable (resp. incompatible, in relation \parallel) with state q .

At the beginning of an inference algorithm, $\mathcal{E}_{\not\sim}(q)$, $\mathcal{E}_{\not\sim}(q)$ and $\mathcal{E}_{\parallel}(q)$ have to be initialized. $\mathcal{E}_{\not\sim}(q)$ and $\mathcal{E}_{\parallel}(q)$ can be computed with algorithm 2 and its counterpart for states in relation \parallel , but update of $\mathcal{E}_{\not\sim}(q)$ must also be done ; for that reason we use the function *Initialize* (algorithm 3).

Algorithm 3 Initialization of $\mathcal{E}_{\not\sim}$, \mathcal{E}_{\parallel} and $\mathcal{E}_{\not\sim}$

```

1: Initialize(A=<  $\Sigma, \Gamma, Q, Q_0, \delta, \gamma$  >)
2:  $\forall q \in Q, \mathcal{E}_{\not\sim}(q) = \emptyset; \mathcal{E}_{\parallel}(q) = \emptyset; \mathcal{E}_{\not\sim}(q) = \emptyset$ 
3: for all  $\{q_1, q_2\} \in Q_0 \times Q_0$  do
4:   SetCP1( $q_1, q_2$ ) /* maintain  $\mathcal{E}_{\not\sim}$ , add  $\parallel$  relation and propagate */
5: for all  $\{q_1, q_2\} \in Q \times Q, \gamma(q_1) \not\sim \gamma(q_2)$  do
6:   SetIncompatible( $q_1, q_2$ ) /* maintain  $\mathcal{E}_{\not\sim}$ , add incompatibility and propagate */
```

Function *Merge'* (algorithm 4) realizes the merging of two states and update sets \mathcal{E}_{\parallel} , $\mathcal{E}_{\not\sim}$ and $\mathcal{E}_{\not\sim}$. This update is realized by propagating existing relations incompatible and \parallel on the state created by the merging (functions *PropagateIncompatibility* and *PropagateCP¹*, algorithm 5).

For example, the ambiguity of the automaton figure 3 part b, may be detected during the merging thanks to addition of new relations: the incompatibility $q_0 \not\sim q_2$ is transformed into $q_{01} \not\sim q_2$ by the merging, then this relation is propagated to $q_{01} \not\sim q_{01}$ by the function *PropagateIncompatibility*. At this step an exception is thrown since it would imply a \parallel relation *and* an incompatibility between the same states.

Every time an incompatibility or a relation \parallel between two states has to be added (functions *SetIncompatible* and *SetCP¹*, algorithm 5), two actions are to be taken: (1) we check that the new relation does not mean ambiguity of the C-NFA (algorithm 5, line 3), (2) we compute new unmergeable states using the relation $q_1 \parallel q_2 \wedge q_2 \not\sim q_3 \Rightarrow q_1 \not\sim q_2$ (algorithm 5, line 11-14, and algorithm 6).

Thanks to this new algorithm, we are able to infer efficiently non deterministic and non ambiguous C-NFAs. This algorithm can also be directly applied to the inference of classical NFA by inferring a 2-NFA and enabling only merges

¹ CP stands for Common Prefix, and correspond to the \parallel relation. We do not detail the functions *SetCP* and *PropagateCP* which are the counterpart for relation \parallel of functions *SetIncompatible* and *PropagateIncompatibility* shown in algorithm 5.

Algorithm 4 Merge two states and update \mathcal{E}_{\neq} , \mathcal{E}_{\parallel} and \mathcal{E}_{\neq}

```

1: Merge'(A,q1,q2)
2: /* detection of unmergeable states */
3: if  $q_1 \in \mathcal{E}_{\neq}(q_2)$  then
4:   throw exception
5: else
6:    $A \leftarrow \text{Merge}(A,q_1,q_2)$  /* substitute  $q_2$  by  $q_1$  in  $A$  and  $\mathcal{E}_{\neq}$ ,  $\mathcal{E}_{\neq}$ ,  $\mathcal{E}_{\parallel}$  */
7:   for all  $q' \in \mathcal{E}_{\parallel}(q_1)$  do
8:     PropagateCP( $q',q_1$ )
9:   for all  $q' \in \mathcal{E}_{\neq}(q_1)$  do
10:    PropagateIncompatibility( $q',q_1$ )
11:  return A

```

Algorithm 5 Add a new incompatibility in \mathcal{E}_{\neq} and propagates its effects

```

1: SetIncompatible( $q_1,q_2$ )
2: if  $q_1 \notin \mathcal{E}_{\neq}(q_2)$  then
3:   if  $q_1 \in \mathcal{E}_{\parallel}(q_2)$  then
4:     throw exception
5:   else
6:     /* add  $q_1$  to  $\mathcal{E}_{\neq}(q_2)$  and  $q_2$  to  $\mathcal{E}_{\neq}(q_1)$  */
7:      $\mathcal{E}_{\neq}(q_1) \leftarrow \mathcal{E}_{\neq}(q_1) \cup \{q_2\}$  ;  $\mathcal{E}_{\neq}(q_2) \leftarrow \mathcal{E}_{\neq}(q_2) \cup \{q_1\}$ 
8:     /* Propagation */
9:     PropagateIncompatibility( $q_1,q_2$ )
10:    /* Update the blocs in relation  $\neq$  */
11:    for all  $q \in \mathcal{E}_{\parallel}(q_1)$  do
12:      SetUnmergeable( $q_2,q$ )
13:    for all  $q \in \mathcal{E}_{\parallel}(q_2)$  do
14:      SetUnmergeable( $q_1,q$ )
15:    PropagateIncompatibility( $q_1,q_2$ )
16:    for all  $a \in \Sigma, \{p_1,p_2\} \in \delta^{-1}(q_1,a) \times \delta^{-1}(q_2,a)$  do
17:      SetIncompatible( $p_1,p_2$ )

```

between states of $MCA(\langle S_+, S_- \rangle)$ created by the positive sample S_+ . In this framework, branches created by the negative sample are only used to check the ambiguity of the current C -NFA. To find the corresponding NFA, we suppress in the C -NFA the part of $MCA(\langle S_+, S_- \rangle)$ corresponding to the negative sample.

We present in the next section experiments applying this approach for the inference of classical NFAs.

3 Experiments

We have implemented our algorithm to carry out first experiments in order to test the validity of our approach.

Our idea is to compare the information needed to correctly infer a NFA versus its determinized version. We first present the benchmark designed for

Algorithm 6 Add unmergeable states in \mathcal{E}_{\neq}

```

1: SetUnmergeable( $q_1, q_2$ )
2: if  $q_1 = q_2$  then
3:   throw exception
4: else
5:   if  $q_1 \notin \mathcal{E}_{\neq}(q_2)$  then
6:      $\mathcal{E}_{\neq}(q_1) \leftarrow \mathcal{E}_{\neq}(q_1) \cup \{q_2\}$ ;  $\mathcal{E}_{\neq}(q_2) \leftarrow \mathcal{E}_{\neq}(q_2) \cup \{q_1\}$ 

```

this experiment and the state merging algorithm we have used before giving the experimental results.

3.1 Benchmark

We have chosen for this benchmark different non deterministic automata (figure 4) inspired by different papers [Dup96, SY97, DLT00] or specifically designed for the benchmark.

We have tried to represent the various processes of determinization. The benchmark contains: a DFA such that no smaller NFA that recognizes the same language is expected to exist (L1) ; a NFA such that its determinization is polynomial (L2) ; NFAs with exponential determinization, representing a finite language (L3) or not (L4, L5) ; a simple NFA common in the DFA literature [Dup96], with a transition added in its transition function (L6).

The various properties of these automata are summarized in table 1. A parameter n is set for some of the automata allowing to tune their size, the value chosen for n in the benchmark is indicated in the third column of table 1.

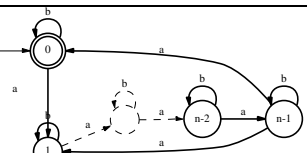
| number | Language with $\Sigma = \{a, b\}$ | n in the benchmark | size of NFA | size of DFA |
|--------|--|--------------------|-------------|---|
| L1 | all word such that the absolute value of its number of a minus its number of b modulo n is 0 | 8 | n | n |
| L2 |  | 4 | n | $(n-1)^2 + 2$ |
| L3 | $\{w \in \Sigma \mid w = uav, w < n \wedge v = \lfloor n/2 \rfloor - 1\}$ | 5 | $n+1$ | $2^{n/2+1} - 1$ if n is even $3 * 2^{\lfloor n/2 \rfloor} - 1$ if n is odd |
| L4 | $\Sigma^* a \Sigma^n$ | 2 | $n+2$ | 2^{n+1} |
| L5 | $\{(b^* a)^{(n-1)}.x^{n-1}.y \mid x \in \mathbb{N}, y \in \mathbb{N}^+\}$ | 3 | n | $2^n - 1$ |
| L6 | see automaton | - | 3 | 7 |

Table1. Characteristics of the benchmark's automata

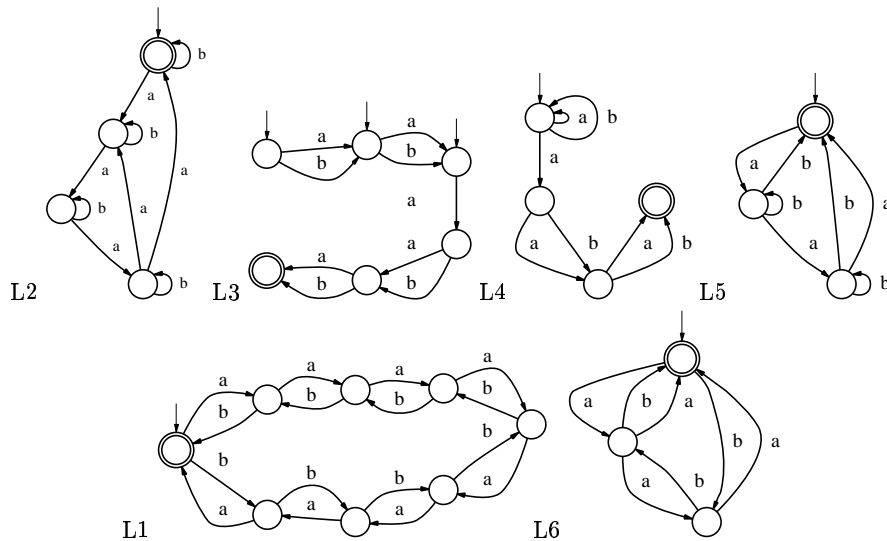


Figure 4. Automata of the benchmark

Samples of training and testing sets were generated following a normal distribution for length and a uniform distribution for words of a given length. Training and testing sets are disjoint.

3.2 Algorithm

In these experiments, we consider the inference of a minimum sized non deterministic automaton. We propose to use the “coloring” scheme which has been proven efficient for this search in the deterministic case [BF72, CN97, OS98].

We briefly describe the algorithm we have used. A set \mathcal{C} of *colored* states (the states of the target automaton) is maintained. Search space exploration is performed by a function choosing at each step a state q of $Q - \mathcal{C}$ and calling itself recursively, first after each successful merging between q and a state of \mathcal{C} , and second, after the promotion of q in \mathcal{C} . Adopting a *Branch & Bound* strategy, the search is pruned if the number of states in \mathcal{C} is greater than in the smallest solution found.

The same heuristic than in [CN97] has been used both for the deterministic and the non deterministic automaton inference: at each step the state, having the maximum number of colored states unmergeable with it, is chosen to be colored. This algorithm has been used in the upper bound framework [BF72, OS98], which means that it tries to find a solution of size one and increments the size until a solution is found. Within this framework, we guarantee that the solution found is of minimum size and structurally complete with the samples.

3.3 Results

The result of algorithm’s runs are given in figure 5 and table 2. We can verify that for all the experiments except one, identification of the NFA requires a smaller sample than identification of its deterministic version. The only exception is L1 which has been constructed so as to be hard to identify in the non deterministic approach. For all other languages non deterministic approach seems clearly better suited to this task as sparse training data are available.

We may interpret this result by applying Occam’s razor principle: smaller automaton compatible with positive and negative samples are more likely to identify the target language.

This results may also be explained by the amount of data needed to ensure structural completeness with respect to the target automaton.

| language | number of samples needed to reach “stable” 100% of recognition | |
|----------|--|------------------------|
| | deterministic case | non deterministic case |
| L1 | 166 | 278 |
| L2 | 372 | 23 |
| L3 | > 500 | 79 |
| L4 | 65 | 22 |
| L5 | 100 | 32 |
| L6 | 190 | 27 |

Table2. convergence observed

Conclusion

We have proposed an algorithm to detect whether a C -NFA is ambiguous. This algorithm may be used incrementally in a state merging inference process, taking in account not only the possible state merging but also the impossible ones.

We have applied this approach for the exact search of minimal NFA with a saturation strategy. Experimental results are promising and tend to show that less data may be needed to identify the non deterministic automata representation of a language than its deterministic representation.

However the main problem for the inference of non deterministic automata remains the lack of canonical form. Denis & al. [DLT00] have presented very recently a first response to this problem by constructing a subclass of NFA for which a canonical form can be defined. Their results may probably be integrated in the state merging framework in order to reduce the search space and to obtain identification results.

Acknowledgements: The authors wish to thank Jacques Nicolas for helpful discussions about this work and Tallur Basavanneppa for valuable comments on the manuscript.

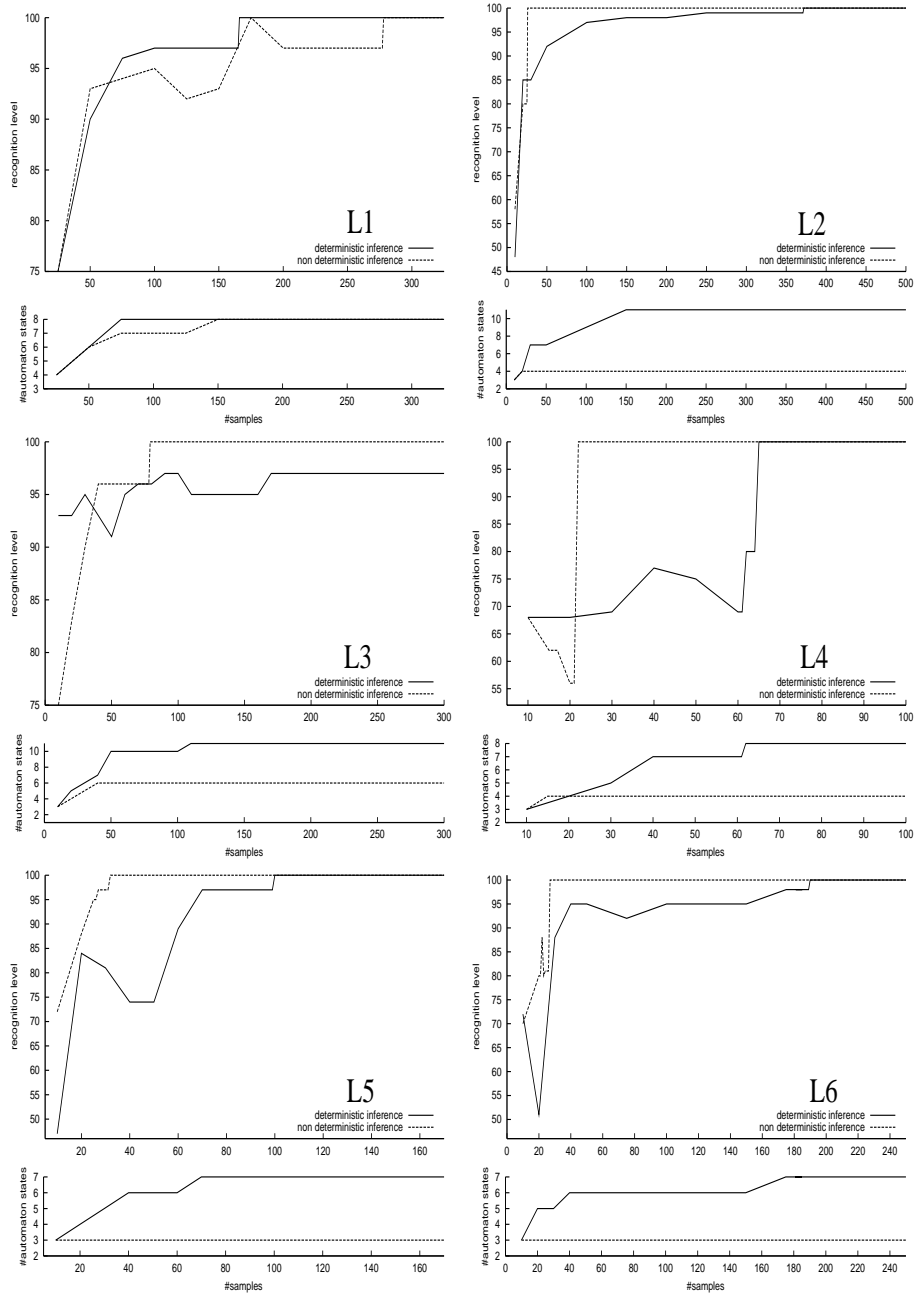


Figure 5. Graphs giving recognition level on testing set and size of automaton found (ordinate) compared to the number of samples in training set (abscissa). Inference of DFAs and NFAs is given on same graphs for each language.

References

- [Alq97] Alquézar (R.). – *Symbolic and connectionist learning techniques for grammatical inference*. – Thèse de PhD, Universitat Politècnica de Catalunya, mars 1997.
- [AS95] Alquézar (R.) et Sanfeliu (A.). – Incremental grammatical inference from positive and negative data using unbiased finite state automata. *In: Shape, Structure and Pattern Recognition, Proc. Int. Workshop on Structural and Syntactic Pattern Recognition, SSPR '94, Nahariya (Israel)*, pp. 291–300. – 1995.
- [BBP75] Biermann (Alan W.), Baum (Richard I.) et Petry (Frederick E.). – Speeding up the synthesis of programs from traces. *IEEE Transactions on Computers*, vol. C-24, 1975, pp. 122–136.
- [BF72] Biermann (A. W.) et Feldmann (J. A.). – On the synthesis of finite-state machines from samples of their behaviour. *IEEE Transactions on Computers C 21*, 1972, pp. 592 – 597.
- [Bré79] Brélaz (D.). – New methods to color the vertices of a graph. *Communications of the ACM*, vol. 22, 1979, pp. 251–256.
- [CN97] Coste (F.) et Nicolas (J.). – Regular inference as a graph coloring problem. *In: Workshop on Grammar Inference, Automata Induction, and Language Acquisition (ICML' 97)*. – Nashville, TN., USA, juillet 1997.
- [Cos99] Coste (F.). – *State merging inference of finite state classifiers*. – Rapport technique n° INRIA/RR-3695, IRISA, septembre 1999.
- [dlH97] de la Higuera (C.). – Characteristic sets for polynomial grammatical inference. *Machine Learning*, vol. 27, 1997, pp. 125–138.
- [DLT00] Denis (F.), Lemay (A.) et Terlutte (A.). – Apprentissage de langages réguliers à l'aide d'automates non déterministes. *In: Conférence d'apprentissage CAp'00*. – 2000.
- [Dup96] Dupont (P.). – *Utilisation et apprentissage de modèles de langages pour la reconnaissance de la parole continue*. – Thèse de PhD, Ecole Nationale Supérieure des Télécommunications, 1996.
- [Gol78] Gold (E. M.). – Complexity of automaton identification from given data. *Information and Control*, vol. 37, 1978, pp. 302 – 320.
- [HU80] Hopcroft (J.) et Ullman (J.). – *Introduction to Automata Theory, Languages, and Computation*. – N. Reading, MA, Addison-Wesley, 1980.
- [Lan92] Lang (K. J.). – Random dfa's can be approximately learned from sparse uniform examples. *5th ACM workshop on Computation Learning Theorie*, 1992, pp. 45 – 52.
- [LPP98] Lang (K. J.), Pearlmutter (B. A.) et Price (R. A.). – Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. *Lecture Notes in Computer Science*, vol. 1433, 1998, pp. 1–12.
- [OG92] Oncina (J.) et Garcia (P.). – Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis*, 1992, pp. 49 – 61.
- [OS98] Oliveira (A. L.) et Silva (J. P. M.). – Efficient search techniques for the inference of minimum size finite automata. *In: South American Symposium on String Processing and Information Retrieval*. – 1998.
- [SY97] Salomaa (K.) et Yu (S.). – Nfa to dfa transformation for finite languages. *In: First international workshop on implementing automata, WIA '96*, p. 188. – 1997.
- [Yok94] Yokomori (T.). – Learning non-deterministic finite automata from queries and counterexamples. *Machine Intelligence*, vol. 13, 1994, pp. 169–189.