UNIVERSITAT POLITÈCNICA DE CATALUNYA

Departament de Llenguatge i Sistemes Informàtics
Ph.D. Programme: Artificial Intelligence

SYMBOLIC AND CONNECTIONIST LEARNING TECHNIQUES FOR GRAMMATICAL INFERENCE

Autor: René Alquézar Mancho
Director: Alberto Sanfeliu Cortés

March 1997

Chapter 5

Regular grammatical inference from positive and negative data using unbiased finite state automata

In this chapter, a type of (possibly non-deterministic) finite-state machines, called *Unbiased Finite State Automata* (or UFSAs), is defined which allows a symmetric representation and processing of positive and negative information and permits to handle the language of strings of uncertain classification. The concept can be extended to cope with C-classes recognition problems giving rise to a type of (possibly non-deterministic) FSMs called *C-classes Unbiased Finite State Automata* (or C-UFSAs).

Both the basic theory and the search space for the RGI problem from positive and negative examples is reformulated in terms of UFSAs. A well-known RGI non-incremental algorithm due to Oncina and Garcia, the RPNI algorithm [OnGa:92b], is generalized using UFSAs, and some variations are presented, each with a particular inductive bias. Then, two new pseudo-incremental methods for RGI from positive and negative examples using UFSAs are described, which are based on the previous algorithm and can take any of the former biases. These methods, which work in polynomial time and improve the average time complexity of the non-incremental algorithm, always obtain a deterministic UFSA that is consistent with the given examples. The solution is updated for each new sample string, but the prefix tree of all the previous strings must be stored (hence the term pseudo-incremental). Indeed, it is derived that a fully incremental approach cannot preserve data consistency, except for trivial solutions.

A new feature of the presented methods is the capability of generalizing negative information, in the same way that is usually done for positive data. The ultimate aim is to reduce the number of negative examples that may be required to obtain a proper solution or to avoid a possible over-generalization caused by limiting the consistency constrain to a given finite set of negative examples. Some experiments have been carried out to show the behaviour of the methods.

The proposed representation and methods were already reported in a communication to the SSPR'94 workshop held in Nahariya (Israel) [AlSa:95a], and part of the results of the experimental study have been included in a recently submitted paper [AlSa:97b].

5.1 Unbiased Finite State Automata

Definition 5.1. An *unbiased finite state automaton*[1] (or UFSA for short) is defined as a six-tuple $U = (Q, \Sigma, \delta, q_0, F_P, F_N)$, where Q is a finite set of states, Σ is a finite set of input symbols (alphabet), $q_0 \epsilon Q$ is the initial state, $F_P \subseteq Q$ and $F_N \subseteq Q$ are sets of positive and negative final states respectively, and $\delta : Q \times \Sigma \rightarrow 2^Q$ is a state transition (possibly partial) function.

Definition 5.2. The language *accepted* by an UFSA $U = (Q, \Sigma, \delta, q_0, F_P, F_N)$ is defined as $L_A(U) = \{\alpha \epsilon \Sigma^* \mid \hat{\delta}(q_0, \alpha) \cap F_P \neq \emptyset\}$, where $\hat{\delta} : Q \times \Sigma^* \rightarrow 2^Q$ is the transition function extended to strings of symbols. Symmetrically, the language *rejected* by U is defined as $L_R(U) = \{\beta \epsilon \Sigma^* \mid \hat{\delta}(q_0, \beta) \cap F_N \neq \emptyset\}$. The language of strings from Σ^* that are neither included in $L_A(U)$ or $L_R(U)$ is said to be *ignored* by U, and it can be defined formally as $L_I(U) = \{\omega \epsilon \Sigma^* \mid \hat{\delta}(q_0, \omega)$ is undefined $\vee$ $(\hat{\delta}(q_0, \omega) \cap (F_P \cup F_N) = \emptyset)\}$. The languages $L_S(U) = L_A(U) \cup L_R(U)$ and Σ^* are called the *scope* and the *domain* of U, respectively.

Definition 5.3. An UFSA $U = (Q, \Sigma, \delta, q_0, F_P, F_N)$ is *deterministic* if and only if the range of the transition function δ is restricted to singletons in Q, i.e. $\delta : Q \times \Sigma \rightarrow Q$.

Deterministic UFSAs will also be denoted DUFAs, while non-deterministic UFSAs will also be denoted NUFAs.

Definition 5.4. An UFSA U is *consistent* if and only if $L_A(U) \cap L_R(U) = \emptyset$.

[1] The term "unbiased" is adopted to emphasize that the representation is symmetric with respect to the positive and negative information, this being inspired by other types of well-known representations such as predicate logic.

Definition 5.5. An UFSA U is *complete* if and only if U is consistent and its scope coincides with its domain (i.e. $L_S(U) = L_A(U) \cup L_R(U) = \Sigma^*$).

A link between "classical" FSAs and UFSAs is established by the following algorithm, which maps DFAs into equivalent complete DUFAs:

ALGORITHM 5.1. *Maps a DFA into an equivalent complete DUFA.*
Input: A given DFA $A = (Q, \Sigma, \delta, q_0, F)$.
Output: A *complete* DUFA $U = (Q', \Sigma, \delta', q_0', F_P, F_N)$.
begin
if δ is fully defined for all $q \in Q$ and $a \in \Sigma$ **then**
$Q' := Q$; $q_0' := q_0$; $F_P := F$; $F_N := Q - F$
$\forall q \in Q, a \in \Sigma:\ \delta'(q,a) := \delta(q,a)$
else
$Q' := Q \cup \{q_G\}$; $q_0' := q_0$; $F_P := F$; $F_N := (Q - F) \cup \{q_G\}$
$\forall q \in Q, a \in \Sigma:$
$\delta(q,a)$ is defined $\Rightarrow\ \delta'(q,a) := \delta(q,a)$
$\delta(q,a)$ is undefined $\Rightarrow\ \delta'(q,a) := q_G$
$\forall a \in \Sigma:\ \delta'(q_G,a) := q_G$
end_if
end_algorithm

Given any regular grammar G, there exists a DFA that accepts the language $L(G)$ generated by G (recall Chapter 2); therefore, by the above algorithm, there also exists a *complete* DUFA U for which $L_A(U) = L(G)$ and $L_R(U) = \Sigma^* - L(G)$. It should be noted that all the *consistent* UFSA U such that $L_R(U) \cup L_I(U) = \Sigma^* - L(G)$ will also (positively) recognize $L(G)$.

It is quite obvious that "classical" FSAs and complete UFSAs have the same classification power, that of recognizing the class of regular languages. However, a complete UFSA is unbiased with respect to positive and negative information, since both types of data are explicitly and symmetrically represented. Furthermore, the class of consistent UFSAs actually extends the expressive capacity of FSAs, since it tackles the existence of strings that cannot be classified with certainty using the available knowledge. In my opinion, these simple and nice properties make UFSAs well-suited (as hypothesis space) for the problem of regular grammatical inference from both positive and negative examples, as I will try to show in following sections. Previously, it is mandatory to establish some basic theory about UFSAs. The next theorems are easily derived from well-known automata theory [HoUl:79].

Theorem 5.1. *Given any NUFA U, there exists an equivalent DUFA U_D, that satisfies $L_A(U_D) = L_A(U) \ \wedge\ L_R(U_D) = L_R(U)$.*

Proof. Let ϕ be the algorithm which transforms an NFA A into an equivalent DFA A_D [HoUl:79]. Now let ϕ' be the algorithm (working on UFSAs) that results from adding to ϕ the following rule: mark as negative final state any state of the deterministic automaton that is built from a set of states of the initial automaton that includes at least one negative final state. Then ϕ' can be applied to obtain U_D from U, since it can be derived symmetrycally both that $L_A(U_D) = L_A(U)$ (considering U as an FSA A in which $F = F_P$ and the negative final state labels are ignored) and $L_R(U_D) = L_R(U)$ (considering U as an FSA A in which $F = F_N$ and the positive final state labels are ignored). □

Theorem 5.2. *There exists an algorithm to determine whether a given UFSA U is consistent.*

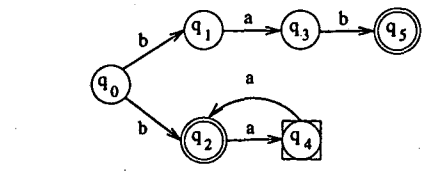
Proof. If U is a DUFA, then it is obvious that U will be consistent if and only if $F_P \cap F_N = \emptyset$. If U is an NUFA, the consistency test algorithm is closely related to algorithm ϕ' of Theorem 5.1. If in the process of applying ϕ' to transform U into a deterministic U_D a state is ever created and marked both as positive and negative final state, then U is inconsistent, and otherwise, U can be declared consistent once the algorithm ϕ' halts. This method is correct because each state of U_D represent a subset of states of U for which a same string leads to, and hence, if a state were labeled both as positive and negative final state, this would mean that at least one string would had led both to a positive and a negative final state in U, so U would be inconsistent. Note that the case of an NUFA having $F_P \cap F_N \neq \emptyset$ is also covered by this method. □

Corollary 1. *There exists an algorithm to determine whether a given UFSA U is complete.*

Proof. Firstly, it may be tested whether U is consistent following the method given in the proof of Theorem 5.2. If U is inconsistent, it is also incomplete, by definition. Otherwise, U is complete if and only if the transition function δ of the corresponding DUFA $U_D = (Q, \Sigma, \delta, q_0, F_P, F_N)$ is fully-defined for the pairs $(q \epsilon Q, a \epsilon \Sigma)$ and every state of U_D is either a positive or a negative final state (i.e. $\forall q \epsilon Q, q \epsilon F_P \oplus q \epsilon F_N$). □

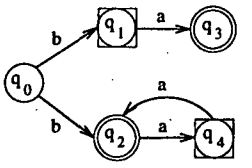
The transition diagrams corresponding to several examples of UFSAs are displayed in Fig. 5.1. Note that, to discriminate graphically between positive and negative final states, the former are marked as encircled vertices whereas the latter are marked as squared vertices.

The concept of UFSA can be generalized naturally to cope with C-classes recognition problems giving rise to a type of FSMs called *C-classes Unbiased Finite State Automata*:



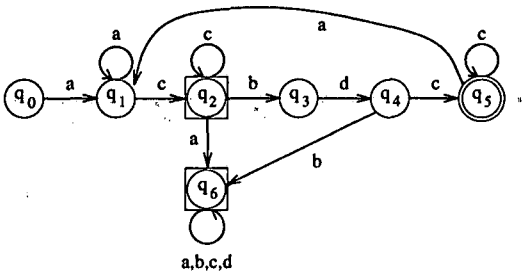
$U_{N1} = (Q, \Sigma, \delta, q_0, F_P, F_N)$ where $\Sigma = \{a, b\}$, $F_P = \{q_2, q_5\}$, $F_N = \{q_4\}$

(a)



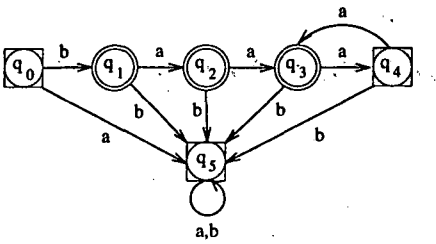
$U_{N2} = (Q, \Sigma, \delta, q_0, F_P, F_N)$ where $\Sigma = \{a, b\}$, $F_P = \{q_2, q_3\}$, $F_N = \{q_1, q_4\}$

(b)



$U_{D1} = (Q, \Sigma, \delta, q_0, F_P, F_N)$ where $\Sigma = \{a, b, c, d\}$, $F_P = \{q_5\}$, $F_N = \{q_2, q_6\}$

(c)



$U_{D2} = (Q, \Sigma, \delta, q_0, F_P, F_N)$ where $\Sigma = \{a, b\}$, $F_P = \{q_1, q_2, q_3\}$, $F_N = \{q_0, q_4, q_5\}$

(d)

Fig. 5.1 *Several examples of unbiased finite state automata (UFSAs).* (a) *A consistent NUFA* U_{N1} (b) *An inconsistent NUFA* U_{N2} (c) *A consistent DUFA* U_{D1} (d) *A complete DUFA* U_{D2}

Definition 5.6. A *C-classes unbiased finite state automaton* (or C-UFSA for short) is defined as a five-tuple $M = (Q, \Sigma, \delta, q_0, \mathcal{F})$, where Q, Σ, δ and q_0 are as in an FSA, and $\mathcal{F}$ is a collection of C subsets of final states, i.e. $\mathcal{F} = (F_1, ..., F_C)$ where $\forall c \in [1, C] : F_c \subseteq Q$.

Definition 5.7. A C-UFSA $M = (Q, \Sigma, \delta, q_0, \mathcal{F})$ *accepts* a collection of C languages $(L_1(M), ..., L_C(M))$, that are defined as $\forall c \in [1, C] : L_c(M) = \{\alpha \epsilon \Sigma^* \mid \hat{\delta}(q_0, \alpha) \cap F_c \neq \emptyset\}$. The language of strings $L_I(M) = \Sigma^* - \bigcup_{c=1}^{C} L_c(M)$ is said to be *ignored* by U, and it can also be defined as $L_I(M) = \{\omega \epsilon \Sigma^* \mid \hat{\delta}(q_0, \omega)$ is undefined $\vee$ $(\hat{\delta}(q_0, \omega) \cap \bigcup_{c=1}^{C} F_c = \emptyset)\}$. The languages $L_S(M) = \bigcup_{c=1}^{C} L_c(M)$ and Σ^* are called the *scope* and the *domain* of M, respectively.

Definition 5.8. A C-UFSA $M = (Q, \Sigma, \delta, q_0, \mathcal{F})$ is *deterministic* if and only if the range of the transition function δ is restricted to singletons in Q, i.e. $\delta : Q \times \Sigma \rightarrow Q$.

Theorem 5.3. *Given any non-deterministic C-UFSA M, there exists an equivalent deterministic C-UFSA M_D, that satisfies* $\forall c \in [1, C] : L_c(M_D) = L_c(M)$.

Proof. It is a generalization of the proof of Theorem 5.1. Let ϕ'_C be the algorithm (working on C-UFSAs) that results from modifying the NFA-to-DFA algorithm ϕ using the following rule: for each $c \in [1, C]$, any state of the deterministic automaton that is built from a set of states of the initial automaton that includes at least one final state in F_c of M must belong to F_c of M_D. Then ϕ'_C can be applied to obtain M_D from M, and this is demonstrated by the same kind of argument used in the proof of Theorem 5.1. □

Definition 5.9. A C-UFSA M is *consistent* if and only if $\forall i, j \in [1, C] :$ $i \neq j \Rightarrow L_i(M) \cap L_j(M) = \emptyset$.

Definition 5.10. A C-UFSA M is *complete* if and only if M is consistent and $L_S(M) = \bigcup_{c=1}^{C} L_c(M) = \Sigma^*$.

Theorem 5.4. *There exists an algorithm to determine whether a given C-UFSA M is consistent.*

Proof. If M is a deterministic C-UFSA, then it is obvious that M will be consistent if and only if $\forall i, j \in [1, C] : i \neq j \Rightarrow F_i \cap F_j = \emptyset$. If M is a non-deterministic C-UFSA, the consistency test algorithm is based on the algorithm ϕ'_C of Theorem 5.3: whenever a state is ever created and marked to belong to more than one subset F_c of final states, then M is inconsistent, and otherwise, M can be declared consistent once the algorithm ϕ'_C halts. □

Corollary 1. *There exists an algorithm to determine whether a given C-UFSA M is complete.*

Proof. Firstly, it may be tested whether M is consistent following the method given in the proof of Theorem 5.4. If M is inconsistent, it is also incomplete, by definition. Otherwise, M is complete if and only if the transition function δ of the corresponding deterministic C-UFSA $M_D = (\Sigma, Q, \delta, q_0, \mathcal{F})$ is fully-defined for the pairs $(q \epsilon Q, a \epsilon \Sigma)$ and every state of M_D is a final state (i.e. $Q = \bigcup_{c=1}^{C} F_c$). □

It can be seen that UFSAs, as defined in the beginning of this section, correspond to C-UFSAs for the particular case of $C = 2$. In the rest of the chapter, I will only deal with UFSAs and their relationship with the RGI problem, although many of the concepts and algorithms that will be introduced may be reformulated for the general case of C-UFSAs as well (e.g. the predicates *consistently-extends*(M_2, M_1) and *consistently-covers*(M_2, M_1)). However, the problem of learning C-UFSAs will not be discussed here and it is left for future research.

5.2 Basic theory for regular grammatical inference using UFSAs

5.2.1 Further definitions and theory about UFSAs

5.2.1.1 Canonical UFSA of a regular language and universal UFSAs

Definition 5.11. Let $A(L)$ be the (minimal-size DFA that is the) canonical automaton of a regular language L. The *canonical UFSA of* L, denoted $U(L)$, is the complete DUFA that results from applying Algorithm 5.1 (DFA to complete DUFA transformation) to $A(L)$.

It is obvious that $U(L)$ is the minimal size complete DUFA U such that $L_A(U) = L$ and $L_R(U) = \Sigma^* - L$. Furthermore, the number of states of $U(L)$ is bounded by $|A(L)| \leq |U(L)| \leq |A(L)| + 1$.

Definition 5.12. A *universal UFSA* over an alphabet Σ is a DUFA $UU = (Q, \Sigma, \delta, q_0, F_P, F_N)$ where $Q = \{q_0\}$ and $\forall a \in \Sigma : \delta(q_0, a) = q_0$. There are four universal UFSAs: the *positive universal UFSA* ($F_P = Q \wedge F_N = \emptyset$), the *negative universal UFSA* ($F_P = \emptyset \wedge F_N = Q$), the *inconsistent universal UFSA* ($F_P = F_N = Q$), and the *empty universal UFSA* ($F_P = F_N = \emptyset$).

5.2.1.2 UFSAs related to a sample

Definition 5.13. Let $S = (S^+, S^-)$ be a sample of a language L. A *canonical UFSA with respect to* S is any UFSA U_C such that $L_A(U_C) = S^+$ and $L_R(U_C) = S^-$.

Definition 5.14. Let $S = (S^+, S^-) = (\{u_1, ..., u_M\}, \{u_{M+1}, ..., u_{M+N}\})$ (containing M positive examples and N negative examples), where $u_i = a_{i,1}...a_{i,|u_i|}$, $1 \leq i \leq M + N$. The *maximal canonical UFSA with respect to* S is the UFSA $MCU(S) = (Q, \Sigma, \delta, q_0, F_P, F_N)$, where

Σ is composed of all the symbols that appear in S,

$Q = \{q_{i,j} \mid 1 \leq i \leq M + N,\ 1 \leq j \leq |u_i|,\ q_{i,j} = a_{i,1}...a_{i,j}\} \cup \{q_0 \mid q_0 = \lambda\}$,

$F_P = S^+ = \{q_{i,|u_i|} \mid 1 \leq i \leq M\}$, $F_N = S^- = \{q_{i,|u_i|} \mid M + 1 \leq i \leq M + N\}$,

$\forall a \in \Sigma : \delta(\lambda, a) = \{q_{i,1} \mid q_{i,1} = a,\ 1 \leq i \leq M + N\}$, and

$\delta(q_{i,j}, a) = \{q_{i,j+1} \mid q_{i,j+1} = q_{i,j}a_{i,j+1} \wedge a_{i,j+1} = a\}$ for $1 \leq i \leq M + N$ and $1 \leq j \leq |u_i| - 1$.

The $MCU(S)$ is the UFSA having the largest number of states, with respect to which S is structurally complete. The $MCU(S)$ is generally an NUFA, where the only possible nondeterministic transitions are from the start state q_0.

Definition 5.15. The (canonical) *prefix tree* UFSA of a sample $S = (S^+, S^-)$ is defined as $PTU(S) = (Pr(S), \Sigma, \delta, \lambda, S^+, S^-)$, where $Pr(S)$ is the set of prefix over the finite language $S^+ \cup S^-$, and δ is given by: $\forall u \in Pr(S), \forall a \in \Sigma : \; ua \in Pr(S) \Rightarrow \delta(u, a) = ua$ (otherwise, $\delta(u, a)$ is undefined). The $PTU(S)$ is always a DUFA.

5.2.1.3 *The* extends *and* consistently-extends *predicates*

Definition 5.16. Given two UFSAs U_1 and U_2, U_2 *extends* U_1 iff $L_A(U_2) \supseteq L_A(U_1)$ and $L_R(U_2) \supseteq L_R(U_1)$. If, in addition, U_2 is consistent, then U_2 *consistently-extends* U_1. Finally, if U_2 is complete and *consistently-extends* U_1, then U_2 *completely-extends* U_1.

Theorem 5.5. *All the UFSAs that extend an inconsistent UFSA are inconsistent.*

Proof. It is immediate from the definitions of a *consistent* UFSA (Def.5.4) and the predicate *extends* (Def.5.16). □

Likewise, it may be shown that the relation *consistently-extends*(U_2, U_1) provides a partial order in the class of consistent UFSAs.

Theorem 5.6. *There exists an algorithm to determine whether, given two UFSAs U_1 and U_2, U_2 extends U_1.*

Proof. In the same manner that it is decidable if a regular grammar (or FSA) is *more-general-than* other regular grammar (or FSA) based on the effective closure properties of regular sets under union, complementation and intersection, it is likewise decidable if an UFSA *extends* other UFSA. The method simply consists of testing twice the regular language superset property (*more-general-than* predicate), once for the accepted language ($L_A(U_2) \supseteq L_A(U_1)$) and once for the rejected language ($L_R(U_2) \supseteq L_R(U_1)$). In order to perform the test, four FSAs are initially defined, two that respectively accept the positive and negative languages of U_1, and two that respectively accept the complements of the positive and negative languages of U_2. Then, two further FSAs can be built that accept $L_A(U_1) \cap \overline{L_A(U_2)}$ and $L_R(U_1) \cap \overline{L_R(U_2)}$ respectively; if both only accept the empty language then U_2 *extends* U_1 and otherwise not. □

Corollary 1. *There exists an algorithm to determine whether, given two UFSAs U_1 and U_2, U_2 consistently-extends U_1.*

Proof. A simple procedure is to test whether U_2 is consistent (Theorem 5.2) and whether U_2 *extends* U_1 (Theorem 5.6). However, whenever U_1 is known to be inconsistent, it directly follows (by Theorem 5.5) that U_2 *consistently-extends* U_1 is false for any UFSA U_2. □

Although the above procedure may be used to test the *consistently-extends*(U_2, U_1) predicate, it is not a good tool for the grammatical inference problem, because it says nothing about how to obtain a "proper" U_2 from an UFSA U_1 canonical with respect to a sample S. Instead, another partial order within the class of consistent UFSAs will be used, which is given by the predicate *consistently-covers*(U_2, U_1), to be defined next.

5.2.1.4 Derived UFSAs and lattices of UFSAs

Definition 5.17. Given an UFSA $U = (Q, \Sigma, \delta, q_0, F_P, F_N)$ and a partition π of Q, let $B(q, \pi)$ denote the only block that contains q and let Q/π denote the quotient set $\{B(q, \pi) \mid q \in Q\}$; then, the quotient UFSA U/π is defined as

$$U/\pi = (Q/\pi, \Sigma, \delta', B(q_0, \pi), \{B \in Q/\pi \mid B \cap F_P \neq \emptyset\}, \{B \in Q/\pi \mid B \cap F_N \neq \emptyset\})$$

where δ' is given by

$$\forall B, B' \in Q/\pi, \forall a \in \Sigma,\ B' \in \delta'(B, a) \text{ if } \exists q, q' \in Q :\ q \in B \ \wedge\ q' \in B' \ \wedge\ q' \in \delta(q, a)$$

The UFSA U/π is also said to be *derived from U with respect to the partition π* of Q.

Definition 5.18. Given two UFSAs U_1 and U_2, U_2 *covers* U_1, also denoted $U_1 \ll U_2$, if and only if $U_2 = U_1/\pi$ for some partition π of the states of U_1. If, in addition, U_2 is consistent, then U_2 *consistently-covers* U_1 (and we say that π is a consistent partition of the states of U_1).

Theorem 5.7. *Given two UFSAs U_1 and U_2, the two following rules are always true:*

i) covers$(U_2, U_1) \Rightarrow$ extends(U_2, U_1)

ii) consistently-covers$(U_2, U_1) \Rightarrow$ consistently-extends(U_2, U_1).

Proof. The first rule is based on Eq.2.17 [FuBo:75], this is, the language accepted by a derived FSA is a superset of the language accepted by the FSA from which is derived. Using this property twice, it follows both that $L_A(U_1) \subseteq L_A(U_2)$ (considering U_1, U_2 as FSAs A_1, A_2, where positive final states are taken as final states) and $L_R(U_1) \subseteq L_R(U_2)$

(considering U_1, U_2 as FSAs A'_1, A'_2, where negative final states are taken as final states). The second rule immediately follows by adding the consistency condition on U_2 to both sides of the first rule. □

Corollary 1. *All the UFSAs that cover an inconsistent UFSA are inconsistent.*

Proof. It is immediate from Theorem 5.5 and the first rule of Theorem 5.7. □

Definition 5.19. Given an UFSA $U = (Q, \Sigma, \delta, q_0, F_P, F_N)$ and two states $q_i, q_j \in Q$, a *merge operation* on U is defined as $U_{Mij} = \text{merge}(U, q_i, q_j) = U/\pi_{ij}$, where the partition $\pi_{ij} = \{\{q \in Q\} \mid q \neq q_i, q \neq q_j\} \cup \{q_i, q_j\}$. We also say that U_{Mij} *is directly derived from* U, or $U \preceq U_{Mij}$.

It is clear that U_2 *covers* U_1 (or $U_1 \ll U_2$) iff U_2 results from zero or more successive merge operations starting on the states of U_1. This is, $\ll$ corresponds to the reflexive and transitive closure of $\preceq$.

Definition 5.20. A merge operation on an UFSA U is said to be consistent if its result U_{Mij} is a consistent UFSA.

It is easy to show that U_2 *consistently-covers* U_1 iff U_2 results from zero or more successive consistent merge operations starting on the states of U_1, since once an inconsistent derived UFSA is produced any further merge will not remove the inconsistency.

The set of UFSAs derived from a given UFSA $U = (Q, \Sigma, \delta, q_0, F_P, F_N)$ is partially ordered by the relation $\preceq$, and furthermore, it is a lattice, denoted $Lat(U)$, of which U and a universal UFSA UU are respectively the null and universal elements[2]. The *depth* of an UFSA U/π in $Lat(U)$ is given by $n - |\pi|$, where n is the number of states of U. Therefore, the depths of U and UU in $Lat(U)$ are 0 and $n - 1$, respectively.

5.2.1.5 Structural completeness of a sample with respect to an UFSA

Definition 5.21. An *acceptance* (respectively *rejection*) of a string $s = a_1...a_l$ by an UFSA $U = (Q, \Sigma, \delta, q_0, F_P, F_N)$, denoted $\mathcal{AC}(s, U)$ (respectively $\mathcal{RE}(s, U)$), is a sequence of $l+1$ states $(q^0, ..., q^l)$ such that $q^0 = q_0$, $q^{i+1} \in \delta(q^i, a_{i+1})$, for $0 \leq i \leq (l-1)$, and $q^l \in F_P$ (respectively $q^l \in F_N$). The $l+1$ states in the sequence (which form a path in the transition diagram of U) are said to be *reached* for $\mathcal{AC}(s, U)$ (respectively

[2]The particular universal UFSA which is the universal element of $Lat(U)$ will depend on whether either F_P, or F_N, or both, are empty or not.

$\mathcal{RE}(s,U)$) and the state q^l is said to be *used as accepting* (respectively *rejecting*) *state.* The l transitions traversed in the path are said to be *exercized* by $\mathcal{AC}(s,U)$ (respectively $\mathcal{RE}(s,U)$).

Definition 5.22. Given a finite set of strings S_s, $\mathcal{AC}(S_s,U)$ (respectively $\mathcal{RE}(S_s,U)$) is a set of acceptances (respectively rejections) of the strings $s \in S_s$ by U, such that there is only one acceptance (rejection) for each string.

Definition 5.23. A sample $S = (S^+,S^-)$ is said to be *structurally complete* with respect to an UFSA $U = (Q,\Sigma,\delta,q_0,F_P,F_N)$, if there exists an acceptance $\mathcal{AC}(S^+,U)$ of S^+ and a rejection $\mathcal{RE}(S^-,U)$ of S^- by U such that

i) every transition in δ is exercized by $\mathcal{AC}(S^+,U)$ or $\mathcal{RE}(S^-,U)$,

ii) every state in F_P is used as accepting state for $\mathcal{AC}(S^+,U)$, and

iii) every state in F_N is used as rejecting state for $\mathcal{RE}(S^-,U)$.

Note that the existence of $\mathcal{AC}(S^+,U)$ and $\mathcal{RE}(S^-,U)$ implies $L_A(U) \supseteq S^+$ and $L_R(U) \supseteq S^-$.

5.2.2 Restatement of the RGI problem

In terms of UFSAs, the problem of regular grammatical inference (RGI) can now be stated as follows: given a sample $S = (S^+,S^-)$ of a language L, such that $S^+ \neq \emptyset \vee S^- \neq \emptyset$, discover the unknown consistent UFSA U_T, called the *target* UFSA, from which S^+ and S^- are supposed to have been (positively and negatively) generated (respectively). Hence, U_T must satisfy $L_A(U_T) \supseteq S^+ \wedge L_R(U_T) \supseteq S^-$ and $L_A(U_T) \cap L_R(U_T) = \emptyset$.

At first, it is clear that all the UFSAs U that *consistently-extend* a canonical UFSA U_C with respect to S are possible candidates to be the target UFSA. Again, a reasonable assumption, when the size of the sample is large enough, is to consider that S is *structurally complete* for the unknown target UFSA U_T. We will see in the next subsection that, under this assumption, U_T *consistently-covers* the $MCU(S)$, so the search of U_T can be constrained to $Lat(MCU(S))$. Moreover, if the target UFSA is restricted to be a DUFA, then U_T *consistently-covers* the $PTU(S)$, and only the lattice $Lat(PTU(S))$ has to be explored. In practice, the corollary of Theorem 5.7 can be used to filter this set, and one or more criteria (inductive biases) must be imposed to select just one or a few plausible solutions among the set of U_T candidates.

5.2.3 The search space of the RGI problem using UFSAs

The two following theorems establish the basis to approach the RGI problem using UFSAs. They are straightforward extensions of Theorems 2.4 and 2.5, respectively [DuMV:94].

Theorem 5.8. *Let $S = (S^+, S^-)$ be a sample of any regular language L and let U_T be any consistent UFSA accepting exactly L and rejecting a language L' such that $S^- \subseteq L' \subseteq \Sigma^* - L$. If S is structurally complete with respect to U_T then U_T belongs to $Lat(MCU(S))$.*

Proof. Let $S = (S^+, S^-) = (\{u_1, ..., u_M\}, \{u_{M+1}, ..., u_{M+N}\})$, where $u_i = a_{i,1}...a_{i,|u_i|}$, $1 \leq i \leq M+N$. The maximal canonical UFSA with respect to S, $MCU(S) = (Q, \Sigma, \delta, q_0, F_P, F_N)$, is constructed as described in Def.5.14. Let $U_T = (Q', \Sigma, \delta', q'_0, F'_P, F'_N)$. A partition π will be defined such that U_T is isomorphic to $MCU(S)/\pi$.

Firstly, let us define $M+N$ sequences of states from an acceptance $\mathcal{AC}(S^+, U_T)$ and a rejection $\mathcal{RE}(S^-, U_T)$ as follows: for each string u_i a sequence $(q'_{i,0}, ..., q'_{i,|u_i|})$ of $|u_i|+1$ states is defined, where $q'_{i,0} = q'_0$ and $q'_{i,j+1} \in \delta'(q'_{i,j}, a_{i,j+1})$, $1 \leq i \leq M+N$, $0 \leq j \leq |u_i|-1$; furthermore, $q'_{i,|u_i|} \in F'_P$, $1 \leq i \leq M$, and $q'_{i,|u_i|} \in F'_N$, $M+1 \leq i \leq M+N$. Next, a function $\varphi : Q \to Q'$ is defined as

i) $\varphi(q_0) = q'_0$, and
ii) $\varphi(q_{i,j}) = q'$, whenever $q' = q'_{i,j}$, $1 \leq i \leq M+N$, $1 \leq j \leq |u_i|$.

Let the partition π be given by $\forall q_k, q_l \in Q : \; B(q_k, \pi) = B(q_l, \pi) \Leftrightarrow \varphi(q_k) = \varphi(q_l)$. Then, U_T is isomorphic to $MCU(S)/\pi$, since the structural completeness of S with respect to U_T implies the three following equalities:

i) δ of $MCU(S)/\pi$ exactly corresponds to δ' (since all transitions in δ' are exercized),
ii) F_P of $MCU(S)/\pi$ exactly corresponds to F'_P (since $\forall q' \in F'_P : \; \exists i$, $1 \leq i \leq M$ such that $q'_{i,|u_i|} = q'$),
iii) F_N of $MCU(S)/\pi$ exactly corresponds to F'_N (since $\forall q' \in F'_N : \; \exists i$, $M+1 \leq i \leq M+N$ such that $q'_{i,|u_i|} = q'$).

Hence, U_T belongs to $Lat(MCU(S))$. □

Theorem 5.9. *Let $S = (S^+, S^-)$ be a sample of any regular language L and let $U(L)$ be the canonical UFSA of L. If S is structurally complete with respect to $U(L)$ then $U(L)$ belongs to $Lat(PTU(S))$.*

Proof. A similar argument than in the previous theorem holds, except that, now, since $U(L)$ is deterministic, there is a unique acceptance $\mathcal{AC}(S^+, U(L))$ and a unique rejection $\mathcal{RE}(S^-, U(L))$, from which a tree of states of $U(L)$ can be built that has the same structure than $PTU(S)$. A function φ can be defined which maps states of the $PTU(S)$ into states of $U(L)$ located in the corresponding nodes of the tree. Again, the partition π is given by $B(q_k, \pi) = B(q_l, \pi) \Leftrightarrow \varphi(q_k) = \varphi(q_l)$, and it follows that $U(L)$ is isomorphic to $PTU(S)/\pi$, because of the structural completeness of S with respect to $U(L)$. □

Similarly, a straightforward extension of Theorem 2.6 [DuMV:94] yields the following theorem:

Theorem 5.10. *Let $S = (S^+, S^-)$ be a sample. Let $\mathcal{U}'_S$ be the set of UFSAs such that S is structurally complete with respect to any UFSA belonging to $\mathcal{U}'_S$, and let $\mathcal{U}_S \subseteq \mathcal{U}'_S$ be the subset of its consistent UFSAs. Then, $\mathcal{U}'_S$ is equal to $Lat(MCU(S))$, and consequently, $\mathcal{U}_S \subseteq Lat(MCU(S))$.*

Proof.

i) If S is structurally complete with respect to an UFSA U, then U belongs to $Lat(MCU(S))$. This can be proved by removing the consistency requirement on U_T in Theorem 5.8 and realizing that its proof is still valid.

ii) If an UFSA U belongs to $Lat(MCU(S))$, then S is structurally complete with respect to U. It is clear that S is structurally complete with respect to $MCU(S)$ and, from the definition of a derived UFSA, it is also structurally complete with respect to any UFSA $U \in Lat(MCU(S))$. □

Whenever $S^+ \neq \emptyset$ and $S^- \neq \emptyset$, $\mathcal{U}_S \subset Lat(MCU(S))$, since at least the inconsistent universal UFSA belongs to $Lat(MCU(S))$. Also, $\mathcal{U}_S \neq \emptyset$, since at least $MCU(S) \in \mathcal{U}_S$ and $PTU(S) \in \mathcal{U}_S$ always.

Given a sample $S = (S^+, S^-)$, the *minimal complete DUFA problem* consists of finding the complete DUFA U such that S is structurally complete with respect to U and U has the minimal number of states. This problem is NP-hard, since it is always possible to obtain the minimal consistent DFA from the minimal complete DUFA, by removing any *garbage* negative final state (at most one), together with its incoming and outgoing transitions, and ignoring the rest of negative final state labels. It is obvious that the minimal complete DUFA U belongs to $\mathcal{U}_S$; indeed, the search space for this problem can be further delimited using a reformulation of the concept of *deterministic border set*, which is given next.

Definition 5.24. An *antistring* $\overline{as}$ in a lattice of UFSAs is a set of UFSAs such that any element of $\overline{as}$ is not covered by any other element of $\overline{as}$.

Definition 5.25. An UFSA U is *at a maximal depth* in a lattice of UFSAs, if U is a consistent UFSA and there is no other consistent UFSA U' which may be derived from U.

Definition 5.26. Given a sample $S = (S^+, S^-)$, the *border set* $BS_{MCU}(S)$, respectively $BS_{PTU}(S)$, is the antistring in $Lat(MCU(S))$, respectively $Lat(PTU(S))$, of which each element is at a maximal depth.

Definition 5.27. Given a sample $S = (S^+, S^-)$, the *deterministic border set* $DBS_{PTU}(S)$, is the antistring in $Lat(PTU(S))$, of which each element is a consistent DUFA U such that there is no other consistent DUFA U' which may be derived from U.

Some interesting properties of the search space of the RGI problem using UFSAs, which may be derived from the related properties in the FSA case [DuMV:94], are the following:

1. $Lat(PTU(S)) \subseteq Lat(MCU(S))$.
2. $\forall U \in Lat(MCU(S)) :$ U is a DUFA $\Rightarrow$ $U \in Lat(PTU(S))$.
3. $BS_{PTU}(S) \subseteq BS_{MCU}(S)$.
4. There may be several distinct languages accepted (and several distinct languages rejected) by the UFSAs belonging to $BS_{MCU}(S)$.
5. There may exist canonical UFSAs (for some languages) which belong to $Lat(PTU(S))$ but do not belong to $BS_{PTU}(S)$, since other UFSAs in $BS_{PTU}(S)$ may be derived from them.
6. The minimal complete DUFA could belong or not to $BS_{PTU}(S)$ (by the previous property), but it must belong to the deterministic border set $DBS_{PTU}(S)$; therefore, the minimal complete DUFA problem can be considered as the discovery of the smallest complete DUFA in $DBS_{PTU}(S)$.

5.3 Basic method for non-incremental RGI from positive and negative examples

In order to design a method for RGI from positive and negative examples using UFSAs, we should choose whether to search the target UFSA U_T in $Lat(MCU(S))$ or in $Lat(PTU(S))$. The latter is adequate if we look for a DUFA U_T, such as the canonical UFSA $U(L)$ of the target regular language, and we assume that the sample S is structurally complete with respect to it. In such a case, Theorem 5.9 guarantees that there exists a partition π of the states of $PTU(S)$ which yields a derived UFSA isomorphic to U_T, i.e. $PTU(S)/\pi = U_T$. Hence, we can restrict ourselves to select a solution among the DUFAs that *consistently-cover* $PTU(S)$ (the prefix tree of S).

However, there is a technical problem that must be considered. It concerns the process of testing the consistency of the UFSAs generated during the search in $Lat(PTU(S))$. Whereas the computational cost of this test is negligible for a derived DUFA, since it reduces to check whether a positive and a negative final state have been merged, the cost of the test for a derived NUFA may be considerable in the worst case, since an NUFA-to-DUFA transformation is involved (recall Theorem 5.2). To avoid this possible source of inefficiency, we can constrain the hypothesis space to DUFAs, by defining and using a *deterministic merge operation* which takes a DUFA and always returns another DUFA (after performing possibly several simple merge operations). This restriction is perfectly applicable, since we are looking for a DUFA U_T; note that an identical restriction is used in the RGI method by Oncina and García that returns a DFA [OnGa:92b].

Definition 5.28. Given a DUFA $U = (Q, \Sigma, \delta, q_0, F_P, F_N)$ and two states $q_i, q_j \in Q$, a *deterministic merge operation* on U is defined as $U_{DMij} = \text{Dmerge}(U, q_i, q_j) = \text{D}(U_{Mij})$, where the operation $D : UFSA \rightarrow DUFA$ is defined recursively as follows

$$D(U) = \begin{cases} D(\text{merge}(U, q_l, q_m)) & \text{if } \exists a \in \Sigma, \exists q_k, q_l, q_m \text{ in } U\text{: } q_l \in \delta(q_k, a) \wedge q_m \in \delta(q_k, a) \\ U & \text{otherwise} \end{cases}$$

This is, $Dmerge(U, q_i, q_j)$ starts by running $merge(U, q_i, q_j)$ and continues by merging any two states that are destination of a non-deterministic transition until a DUFA U_{DMij} is obtained. If U is consistent, then U_{DMij} will be consistent if no new state is created from a merge of a positive and a negative final state. It is also clear that U_{DMij} is *derived from* U, although it is not necessarily *directly derived from* U.

On the other hand, the set of DUFAs that *consistently-cover* the prefix tree $PTU(S)$ may be quite large, and thus, any practical RGI scheme has to impose some additional conditions to select just one or a few U_T candidates. The three following inductive biases should yield meaningful solutions:

1. Try to maximize the generalization of *positive* data.
2. Try to maximize the generalization of *negative* data.
3. Try to maximize the generalization of both *positive and negative* data.

The method proposed by Oncina and García [OnGa:92b] (using DFAs), that has been described in Chapter 2 (see Algorithm 2.2), is a good representative for the first bias above. Next, let us explain how to reformulate this algorithm in terms of UFSAs and how to derive similar procedures for the two other biases. To this end, some more definitions are required.

Definition 5.29. Let q_i, q_j be two distinct states in a prefix tree $PTU(S)$, and let $s_i, s_j \in Pr(S)$ be the corresponding unique prefixes that lead from the start state to q_i, q_j in $PTU(S)$, respectively. We say that $q_i < q_j$ iff $s_i < s_j$ according to the lexicographic order in Σ^*. Moreover, let us think of s_i as an identifier for the state q_i, so that, for notational purposes, we can use either s_i or q_i to denote the same state in $PTU(S)$.

Definition 5.30. Let π be a partition of the states of a prefix tree $PTU(S)$. If q is a state in the quotient UFSA $PTU(S)/\pi$, let B_q denote the block of the partition associated with q (i.e. the set of states of $PTU(S)$ that have been merged together into q). Let q_i, q_j be two distinct states in $PTU(S)/\pi$, and let $B_{q_i}, B_{q_j} \in \pi$ be their corresponding blocks. We say that $q_i < q_j$ (and also $B_{q_i} < B_{q_j}$) iff $\exists u \in B_{q_i} : \forall v \in B_{q_j} : u < v$.

Consequently, for every state q in $PTU(S)/\pi$, we may choose the lowest prefix (in lexicographic order) in B_q as an identifier for q. This enables to establish a total order in the set of states. It should also be remarked that the operation $merge(U, q_i, q_j)$ updates the block information by creating $B_{q_{ij}} = B_{q_i} \cup B_{q_j}$ and removing both B_{q_i} and B_{q_j}.

Definition 5.31. Given a state $q \epsilon Q$ in an UFSA $U = (Q, \Sigma, \delta, q_0, F_P, F_N)$, the following regular languages are defined:

$$Heads(q) = \{\alpha \in \Sigma^* \mid q \in \hat{\delta}(q_0, \alpha)\};$$
$$Pos\_tails(q) = \{\psi \in \Sigma^* \mid \hat{\delta}(q, \psi) \cap F_P \neq \emptyset\}; \text{ and}$$
$$Neg\_tails(q) = \{\omega \in \Sigma^* \mid \hat{\delta}(q, \omega) \cap F_N \neq \emptyset\}.$$

Any state q contained in U can be labeled as *uncertain*, *positive*, *negative*, or *mixed*, according to Table 5.1.

	$Pos\_tails(q) = \emptyset$	$Pos\_tails(q) \neq \emptyset$
$Neg\_tails(q) = \emptyset$	uncertain	positive
$Neg\_tails(q) \neq \emptyset$	negative	mixed

Table 5.1. Labeling of a state q in an UFSA.

The labels of the states in any prefix tree $PTU(S)$ are easily computable. For example, in a constructive manner: for each new string $s \in S^+$, the states in the acceptance $\mathcal{AC}(s, PTU(S))$ are labeled *mixed* if they were previously labeled as *negative* or *mixed*, and they are labeled *positive* otherwise; a symmetric procedure applies to the strings $s \in S^-$.

After any merge operation $merge(U, q_i, q_j)$, the label of the new state q_{ij} can be assigned from the labels of the merged states q_i, q_j as shown in Table 5.2. Likewise, the labels of the new states resulting from any deterministic merge operation $Dmerge(U, q_i, q_j)$ are readily computed according to the sequence of merge operations that are performed.

		q_j			
		uncertain	*positive*	*negative*	*mixed*
q_i	*uncertain*	uncertain	positive	negative	mixed
	positive	positive	positive	mixed	mixed
	negative	negative	mixed	negative	mixed
	mixed	mixed	mixed	mixed	mixed

Table 5.2: Labeling of the new state resulting from a merge operation.

Given two states q, q' of an UFSA U (in particular, U can be a quotient UFSA $PTU(S)/\pi$), a boolean function $mergeable(q, q', bias)$ may be defined to determine whether the states q and q' are allowed to be merged depending on a certain parameter *bias*. If we take the three inductive biases aforementioned, we can define an instance of this function, which is depicted hereinafter. More sophisticated functions $mergeable(q, q', bias)$ could be defined by introducing additional biases, for example, by establishing equivalence relationships among the states of U based on partial similarities [KuSh:88].

function *mergeable*$(q, q', bias)$ **returns** boolean
var b: boolean
begin
case
bias = *maximize_positive_generalization* :
    $b :=$ q is labeled *positive* or *mixed* **and** q' is labeled *positive* or *mixed*
bias = *maximize_negative_generalization* :
    $b :=$ q is labeled *negative* or *mixed* **and** q' is labeled *negative* or *mixed*
bias = *maximize_positive_and_negative_generalization* :
    if q is labeled *positive* and q' is labeled *negative* **or**
    q' is labeled *positive* and q is labeled *negative*
    then $b := FALSE$
    else $b := TRUE$
    end_if
end_case
return b
end_function

Now, we have all the elements required to write formally a general non-incremental RGI method using UFSAs (Algorithm 5.2), that permits to introduce inductive biases through the user-defined boolean function *mergeable*$(q, q', bias)$. The worst-time complexity of Algorithm 5.2 is $O(||S||^3)$, or more precisely, $O(|PTU(S)|^3)$; hence, it remains in cubic polynomial order as in Algorithm 2.2 (Oncina and García's). Both algorithms return exactly the same automaton when the bias "maximize positive generalization" is selected and the DUFA U_m is pruned by removing all the negative labeled states together with their incoming and outgoing transitions.

Therefore, Algorithm 5.2 with the "maximize positive generalization" bias (plus the post-processing step) is guaranteed to identify in the limit any regular language, since for each target automaton a representative sample can be built which (possibly augmented with further examples) leads to the solution [OnGa:92a]. By symmetry, it follows that Algorithm 5.2 with the "maximize negative generalization" bias and a post-processing step of pruning the positive labeled states (and associated transitions) of U_m is guaranteed to identify in the limit the complement of the target regular language. Furthermore, it can be proved that Algorithm 5.2 with the "maximize positive and negative generalization" bias (and no post-processing step) also identifies in the limit any regular language, as stated in Theorem 5.11.

ALGORITHM 5.2: *Non-incremental RGI using DUFAs*
Inputs: A sample $S = (S^+, S^-)$ of an unknown language L.
A parameter *bias* that determines the condition to be met to allow the merge of two states.
Outputs: A consistent DUFA $U_m \in Lat(PTU(S))$.

begin
$m := |PTU(S)|$ { let m be the number of states of the prefix tree UFSA of S }
$U_1 := PTU(S)$ { let take the $PTU(S)$ as initial hypothesis }
Arrange the set of prefixes $Pr(S)$ in lexicographic order: $\{s_1, ..., s_m\}$, where $s_1 = \lambda$.
for j:= 2 **to** m **do**
  if $\exists q \in Q$ of U_{j-1} : $s_j \in B_q \wedge q < s_j$ { the state identified by the prefix s_j has already been merged in a previous iteration }
  then $U_j := U_{j-1}$ { do not change the hypothesis }
  else
    Find within the set $\{q \in Q$ of $U_{j-1} \mid q < s_j\}$, the lowest state q_i in lexicographic order such that $mergeable(q_i, s_j, bias)$ is $TRUE$ and $Dmerge(U_{j-1}, q_i, s_j)$ is a consistent DUFA.
    if found such q_i
    then $U_j := DMerge(U_{j-1}, q_i, s_j)$ { update the current hypothesis }
    else $U_j := U_{j-1}$ { do not change the hypothesis }
    end_if
  end_if
end_for
{ U_m is the selected solution }
end_algorithm

Theorem 5.11. *Let L be a regular language. Let $A(L)$ and $A(\overline{L})$ be the canonical DFAs of L and $\Sigma^* - L$, respectively, and let $U(L)$ be the canonical UFSA of L. Let $S_{pos} = (S^+_{pos}, S^-_{pos})$ and $S_{neg} = (S^+_{neg}, S^-_{neg})$ be the representative samples that ensure the convergence of Oncina-García's RGI method (Algorithm 2.2) to $A(L)$ and $A(\overline{L})$, respectively. If the sample $S = (S^+, S^-)$ of L is such that $S^+ \supseteq S^+_{pos} \cup S^-_{neg}$ and $S^- \supseteq S^+_{neg} \cup S^-_{pos}$ then Algorithm 5.2 with the "maximize positive and negative generalization" bias infers the $U(L)$.*

Proof. Firstly, it is clear that $U(L)$ only contains states and transitions that are present either in $A(L)$ or $A(\overline{L})$ or both. On one hand, the condition $S^+ \supseteq S^+_{pos}$ and $S^- \supseteq S^+_{neg}$ guarantees that S is structurally complete with respect to $U(L)$. On the other hand, imposing additionally $S^- \supseteq S^-_{pos}$ ensures that any pair of the states in $A(L)$ will not be merged together [OnGa:92a]. Symmetrically, imposing the condition $S^+ \supseteq S^-_{neg}$ ensures that any pair of the states in $A(\overline{L})$ will not be

merged together. Finally, the process of merging the states of the prefix tree of a structurally complete sample in lexicographic order according to the "maximize positive and negative generalization" bias guarantees that, in the inferred UFSA, both there is no state labeled positive or mixed which does not belong to $A(L)$ and there is no state labeled negative or mixed which does not belong to $A(\overline{L})$ (since otherwise, further state merges could have been done). Therefore, it follows necessarily that when $S^+ \supseteq S^+_{pos} \cup S^-_{neg}$ and $S^- \supseteq S^+_{neg} \cup S^-_{pos}$, the inferred UFSA is always isomorphic to the canonical UFSA $U(L)$. □

Corollary 1. *The size of the representative sample $S = (S^+, S^-)$ (i.e the number of strings) that guarantees the induction of the canonical UFSA $U(L)$ by Algorithm 5.2 with the "maximize positive and negative generalization" bias is of $O(n^2)$, where n is the number of states of the canonical DFA $A(L)$.*

Proof. It is known that the size of both S_{pos} and S_{neg} are of $O(n^2)$ [OnGa:92a] (the latter because $|A(\overline{L})| \leq n + 1$). Hence, since the size of S is bounded by the sum of the number of examples in S_{pos} and S_{neg}, it follows that the size of S is also $O(n^2)$. □

5.4 Pseudo-incremental methods for RGI using UFSAs

Now let us turn our attention to the problem of incremental RGI. A weak statement of this problem, using UFSAs as hypothesis space, is: given a prior (possibly null) sample S_{old}, a prior compatible hypothesis U_{old}, and a new (positive or negative) example s, obtain an updated consistent UFSA U, such that $L_A(U) \supseteq S^+ \wedge L_R(U) \supseteq S^-$. The weakness resides on the requirement of the storage of S_{old}, and this is the reason for the term "*pseudo-incremental*".

It is not difficult to conclude that a strong (fully-incremental) statement of the problem, in which the prior sample is not stored, constrains the solutions to "uninteresting" UFSAs, with null or partial sample generalization, in order to ensure the data compatibility of the successive hypotheses. One such trivial solution would consist of maintaining a canonical UFSA U_c (i.e. $L_A(U_c) = S^+ \wedge L_R(U_c) = S^-$). Otherwise, once a hypothesis generalize either S^+, S^-, or both, inconsistent new positive (negative) examples may arrive which will oblige to reduce the extension of the set of rejected (accepted) strings by the current UFSA. In that case, and if the prior sample is not stored, some previous positive (negative) examples could be not accepted (rejected) by the resulting UFSA, unless a "trivial" reduction were

carried out. A "trivial" reduction would consist of discriminating the successively found inconsistent strings by putting them in a list of exceptions[3]. This tricky strategy could be even performed keeping the UFSA representation, but probably, it would lead to *bad* solutions, since only some (arbitrary) part of the provided examples would contribute to the inferred generalization.

Therefore, only *pseudo-incremental* methods for RGI will be discussed here. In the methods to be presented next, the whole sample S is stored and updated in the form of a prefix tree UFSA $PTU(S)$. Furthermore, any current hypothesis U *consistently-covers* $PTU(S)$, i.e. $U = PTU(S)/\pi$, and a record of the corresponding partition π is needed. To this end, each state q of U keeps a record of the set of states of $PTU(S)$ that have been merged into q (i.e. the block B_q). Algorithm 5.3 depicts the basic pseudo-incremental procedure that is proposed.

ALGORITHM 5.3: *Pseudo-incremental RGI using DUFAs*

Inputs: A sequence of examples $e_1, ..., e_i, ...$, where each example e_i is a pair formed by a string s_i and a positive or negative label (depending on whether s_i belongs to a target unknown language L or not).

A parameter *bias* that determines the condition to be met to allow the merge of two states.

Outputs: A sequence of consistent DUFAs $U_1, ..., U_i, ...$, where each DUFA U_i is compatible with the sequence of examples up to e_i.

begin

$PTU_0 := (\{\lambda\}, \emptyset, \emptyset, \lambda, \emptyset, \emptyset)$ { set an empty initial prefix tree }

$U_0 := PTU_0$ { set an empty initial hypothesis }

$i := 0$ { i counts the number of examples that have been read }

read_example(*s*,*class*) { where s is a string and *class* is "+" or "-" }

while not end of example sequence **do**

$i := i + 1$

PTU_i := expand_prefix_tree($PTU_{i-1}, s, class, tpath, tplen, plns$) { this operation updates the sample prefix tree, and it also returns *tpath*, the path of states of PTU_i visited by s; its length *tplen*; and *plns*, the position in *tpath* of the lowest new state due to s }

p := recognize($s, U_{i-1}, upath, uplen$) { where p can be *accepted*, *rejected* or *ignored*, and this operation also returns *upath*, the path of states of U_{i-1} visited by s, and its length *uplen* }

[3]Actually, a list of positive exceptions and a list of negative exceptions.

if $(p = accepted \wedge class = \text{"+"}) \vee (p = rejected \wedge class = \text{"-"})$ **then**
{ the new example is *consistent* with current hypothesis U_{i-1} }
U_i := add_new_treestates_to_partition($U_{i-1}, upath, tpath, tplen, plns$)
else
if $(p = accepted \wedge class = \text{"-"}) \vee (p = rejected \wedge class = \text{"+"})$ **then**
{ the new example is *inconsistent* with current hypothesis U_{i-1} }
U_i := split_and_merge($U_{i-1}, PTU_i, upath, tpath, tplen, plns, bias$)
else
{ the new example is *ignored* by the current hypothesis U_{i-1} }
U_i := restricted_remerge($U_{i-1}, upath, uplen, tpath, tplen, plns, bias$)
end_if
end_if
write_UFSA(U_i)
read_example(s,*class*) { where s is a string and *class* is "+" or "-" }
end_while
end_algorithm

The name of the procedure "add_new_treestates_to_partition" is almost self-explanatory. In this procedure, the new states in the prefix tree $PTU_i(S)$ due to the current example (which are stored in *tpath* from *plns* to *tplen* positions) are included in the partition blocks B_q of the corresponding states q of U_{i-1} that have been visited during the recognition and recorded in the path *upath*. Except for this partition update, the UFSA hypothesis itself is not changed when the current example is compatible with it; hence, Algorithm 5.3 is a *conservative* inductive inference method.

The procedure "restricted_remerge" firstly augment the hypothesis U_{i-1} in the simplest way to accept (or reject) the current positive (negative) example: either a non-final state is made final, or a single-path tail is appended to the state of *upath* where an undefined transition occurred for the current example. Then, in order to generalize according to the predefined *bias*, the states in the new tail (if any) are tested for a consistent merge with the rest of states of U_{i-1}, and also with themselves, in lexicographic order. Note that it is not needed to try new merges among the previous states, since they would fail: if any two previous states remained separated was because their merge would cause an inconsistency with a previous example, and this fact is not altered by the addition of an ignored string.

Concerning the procedure "split_and_merge", that is applied when an inconsistency is detected, two possible approaches are described next, which give rise to two distinct *pseudo-incremental* methods:

1. Maximal splitting (*SM_1*)
2. Minimal splitting (*SM_2*)

The first one, *SM_1*, simply gets rid of the current partition and UFSA hypothesis, i.e. maximizes the split step, and builds a new hypothesis from $PTU_i(S)$ through lexicographically-ordered new consistent merges. This causes the same output (UFSA hypothesis) that is yielded by the non-incremental Algorithm 5.2 from the same sample. This behaviour permits to inherit the convergence property of Algorithm 5.2 (as shown in Theorem 5.12), but at a lower average cost in a sequential processing. Both algorithms perform exactly the same steps when an inconsistent string is found, with a worst-case cost of $O(|PTU(S)|^3)$. But when strings that are consistent or ignored by the current hypothesis are supplied, the computational burden is much reduced due to the less costly "add_new_treestates_to_partition" ($O(l)$) and "restricted_remerge" ($O(l \cdot |U|^2)$) operations, respectively, where l refers to the length of the current example string and $|U|$ refers to the number of states of the (extended) UFSA hypothesis.

It must be noted, however, that the output of both algorithms is not necessarily the same at each step, since, for example, the introduction of a compatible example does not change the hypothesis in the pseudo-incremental method, but it could lead to a different UFSA if the partition were computed again through lexicographically-ordered merges from the expanded prefix tree (non-incremental method). In the next section, it will be shown empirically that in the very most part of cases, the non-incremental and the *SM_1* pseudo-incremental methods lead to the same UFSA hypothesis, given a finite sample.

Theorem 5.12. *Let L be a regular language and let $U(L)$ be the canonical UFSA of L. Algorithm 5.3 with split_and_merge procedure* SM_1 *and the "maximize positive and negative generalization" bias identifies in the limit L by converging to $U(L)$ or an equivalent non-minimal DUFA after some finite number of examples.*

Proof. After some finite step t, the representative sample $S = (S^+, S^-)$ of L that ensures the convergence of Algorithm 5.2 with the "maximize positive and negative generalization" bias will be included in the presented sample. If the Algorithm 5.3 with *SM_1* procedure has not converged to a DUFA equivalent to $U(L)$ by then, an inconsistent example will eventually be presented after some more finite number of steps, and, at this point, the *SM_1* procedure with the "maximize positive and negative generalization" bias is guaranteed to yield the $U(L)$, since the representative sample is already included in the presented sample. □

A similar proof can be established to demonstrate that Algorithm 5.3 with *SM_1* procedure and the "maximize positive generalization" bias (followed by negative state pruning) or the "maximize negative generalization" bias (followed by positive state

pruning) is able to identify in the limit any target regular language, based on the corresponding property of Algorithm 5.2 with the same bias and the related post-processing.

The second "split_and_merge" procedure, *SM_2*, tends to be more conservative about the current hypothesis U_{i-1}, and it further reduces the computational cost, but at the severe expense of losing the identification in the limit property. In addition, it shows a quite greater sensitivity with respect to the presentation order. Nevertheless, a DUFA U_i that *consistently-covers* $PTU_i(S)$ is guaranteed at every step, and the number of states of U_i tends to be small in comparison with the size of $PTU_i(S)$. The procedure *SM_2* firstly transform U_{i-1} in the simplest way to accept (or reject) the current positive (negative) example, through repeated split of states of *upath* in backward direction until a consistent DUFA U_i' is obtained. Such a DUFA U_i' is always attainable, since, in the worst case, a single-path tail from the start state q_0 is produced for the current string. Afterwards, a remerge of the states of U_i' is carried out, subject to the UFSA consistency restriction, following the lexicographic order.

function *SM_2* ($U_{old}, PTU, upath, tpath, tplen, plns, bias$) **returns** UFSA
begin
if $plns > tplen$ **then** { there are no new state in PTU_{new} }
    $q_1 := upath[tplen]$; $q_2 := tpath[tplen]$
    $U :=$ split_state($U_{old}, q_1, q_2, PTU, deterministic$)
    $k := tplen$
else
    $q_1 := upath[plns-1]$; $q_2 := tpath[plns-1]$
    $U :=$ split_state($U_{old}, q_1, q_2, PTU, deterministic$)
    $U :=$ append_new_tree_tail($U, q_2, tpath, plns, tplen$)
    $k := plns - 1$
end_if
while not deterministic **do**
    { the non-determinism of U is checked in the previous split_state operation }
    $k := k - 1$;
    $q_1 := upath[k]$; $q_2 := tpath[k]$
    $U :=$ split_state($U, q_1, q_2, PTU, deterministic$)
end_while
return remerge($U, bias$) { try all pairwise merges over U states in lexicographic order }
end_function

The operation split_state($U, q_1, q_2, PTU, deterministic$) returns the UFSA that results from splitting the state q_1 of U in two states q_1' and q_2 such that $B_{q_1'} = B_{q_1} - \{q_2\}$

and $B_{q_2} = \{q_2\}$, i.e. the state of the prefix tree identified by q_2 is segregated from the partition block of q_1, and the incoming and outgoing transitions of q_1' and q_2 are recomputed using the information of the current partition and the prefix tree PTU. In addition, it is evaluated whether the returned UFSA is deterministic or not by seeking a non-deterministic transition $\delta(q, a) = \{q_1', q_2\}$, $a \in \Sigma$, in the state q whose block B_q includes the father of q_2 in the prefix tree PTU.

The worst-case time complexity of the operation split_state is $O(|PTU|)$, due to the recomputation of the involved transitions. Since this operation is called at most *tplen* times (the length of the current example) in *SM_2*, and on the other hand, the operation remerge(U, *bias*) has a worst-case cost of $O(|U|^3)$, it is derived that the worst-case time complexity of *SM_2* is $O(l \cdot |PTU(S)| + |U|^3)$, where l is the length of the current example and $|U|$ is the number of states of the extended UFSA hypothesis (after the split operations). Recall that the worst-case time complexity of the split_and_merge procedure *SM_1* was $O(|PTU(S)|^3)$, and that usually, $|U| \ll |PTU(S)|$.

For both the *SM_1* and *SM_2* procedures, a predetermined inductive bias (see the function *mergeable* in the preceding section) must be imposed to obtain a unique and meaningful solution. In addition to the aforementioned three general biases (for which the *SM_1*-based method is guaranteed to identify in the limit the target language), state equivalence relationships based on partial similarities may be defined [KuSh:88], e.g. two states can be considered equivalent, and thus mergeable, if the symbols of the incoming transitions are the same for both (successor method).

Other biases may be specified in the form of constraints to be satisfied for each new merged state and the resulting transition function δ; for example, the following constraints might be imposed on every positive path of any UFSA hypothesis:

- a certain symbol $a_1 \in \Sigma$ must not be followed by other symbol $a_2 \in \Sigma$ (e.g. in smooth chain-coded contours);
- if a certain substring $\alpha_1 \in \Sigma^*$ ends a head leading to a positive state $q \in Q$, then other substring $\alpha_2 \in \Sigma^*$ must not begin a positive tail of q;
- disable loop construction (to avoid substrings of potentially infinite length).

This kind of constraints, which imply some degree of a-priori knowledge about the language being learned, can be used to save the introduction of a large number of negative and positive examples, that could be required to reach the same solution in an unconstrained inductive run. However, it should be emphasized that the use of other biases different from the three general biases included in the function *mergeable* cannot ensure, in principle, the identification of the target language or even the convergence of the *SM_1*-based method. On the other hand, the *SM_2*-based method may not to converge even with any of the three general biases.

5.5 Experimental assessment

In order to evaluate the RGI methods described in the previous sections, two experiments were carried out, the first one aimed at assessing the classification rate of the UFSAs inferred from sparse samples, and the second one aimed at testing the convergence of the algorithms.

The set of fifteen regular languages selected by Dupont as a benchmark for RGI methods [Dupo:94,MiGe:94], and which include the seven languages introduced by Tomita [Tomi:82] (used in several studies about RGI using recurrent neural networks [Poll:91,MiGi:93]), were chosen as test languages. Figure 5.2 displays the minimal-size DFA $A(L)$ and the minimal-size complete DUFA $U(L)$ for each one of them. A compact description of the test languages is given here below:

L_1 : a^*

L_2 : $(ab)^*$

L_3 : any sentence without an odd number of consecutive a's after an odd number of consecutive b's.

L_4 : any sentence without more than two consecutive a's.

L_5 : any sentence with an even number of a's and an even number of b's.

L_6 : any sentence such that the number of a's differs from the number of b's by 0 modulo 3.

L_7 : $a^*b^*a^*b^*$

L_8 : a^*b

L_9 : $(a^* + c^*)b$

L_{10} : $(aa)^*(bbb)^*$

L_{11} : any sentence with an even number of a's and an odd number of b's.

L_{12} : $a(aa)^*b$

L_{13} : any sentence with an even number of a's.

L_{14} : $(aa)^*ba^*$

L_{15} : $bc^*b + ac^*a$

Note that all the test languages are over the binary alphabet $\{a, b\}$, except the languages L_9 and L_{15}, which are over the alphabet $\{a, b, c\}$.

5.5.1 RGI from sparse samples

The first experiment followed the protocol described by Dupont [Dupo:94]; furthermore, the same test data was used for comparison purposes.

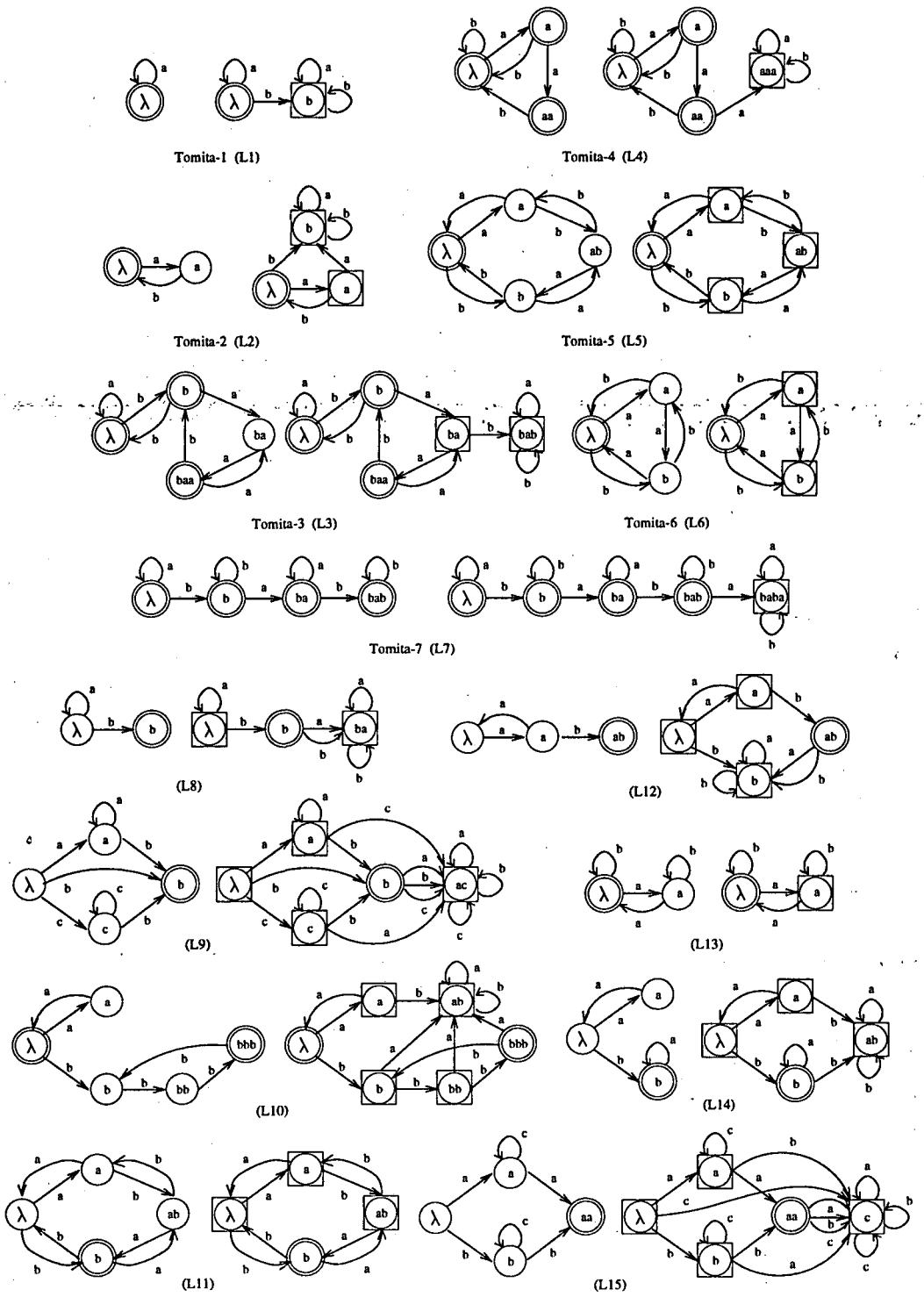


Fig. 5.2 *Minimal-size DFA $A(L)$ (left) and minimal-size complete DUFA $U(L)$ (right) for each one of the fifteen test languages.*

For each test language L, ten learning samples were available. Each learning sample $S = (S^+, S^-)$ had been originally generated in the following manner: (i) S^+ consisted of a structurally complete positive sample randomly generated from the DFA $A(L)$, such that $|S^+| = 3|S^+|_{sc}$, where $|S^+|_{sc}$ was the number of examples initially generated up to yield a structurally complete positive sample; (ii) S^- was constructed similarly using the minimal DFA accepting $\Sigma^* - L$. Both S^+ and S^- could contain repeated strings. The average number of positive, negative, and total examples in the learning samples of each test language can be seen in Table 6.4 (in Chapter 6).

To assess the goodness of the inferred hypotheses, the correct classification rate, according to the target automaton, was computed on three sets of strings for each language L and sample S: $T = \Sigma^l - S$, the set of all the strings up to a given length l but the learning examples, $T^+ = \Sigma^l_+ - S^+$, the set of all the language strings up to length l but the positive examples, and $T^- = \Sigma^l_- - S^-$, the set of all the strings up to length l not belonging to L but the negative examples. The length l was equal to 9 for the languages over $\{a, b\}$ and 7 for the languages over $\{a, b, c\}$. In some cases, T^+ or T^- were enlarged conveniently to contain at least 10 strings.

The results that are displayed in Tables 5.3 and 5.4 correspond to five features for each language and RGI method (associated with the five wide columns). The former three are averages over the ten learning samples of the correct positive, negative, and total classification rates, respectively. The fourth one refers to the arithmetic mean of the positive and negative classification rates [Dupo:94]. The fifth one refers to the identification rate, the percentage of times the target automaton was inferred. In the computation of the classification rates, the test strings that were *ignored* by the inferred UFSA were classified as negative strings. The last row of the tables displays the above features averaged over the 15 test languages.

Table 5.3 shows the good quality of the UFSAs inferred by the non-incremental approach (Algorithm 5.2) from the sparse samples, both with the "maximize positive generalization" (left) and the "maximize positive and negative generalization" (right) biases. The first method is equivalent to Algorithm 2.2 (Oncina-García's) if the inferred UFSA is *stripped* by removing the states that are labeled *negative*; this post-processing was carried out in order to emulate the Oncina-García's method. It must be noted that the identification criteria for the two tested methods are therefore distinct, since the target automaton is the minimal-size DFA $A(L)$ for the first method, whereas it is the minimal-size complete DUFA $U(L)$ for the second one. This explains the remarkable difference in identification rate, mainly in the languages for which the $U(L)$ contains much more transitions than the $A(L)$ (e.g. L_{10}, L_{12}, L_{15}). Concerning the classification rates, the former method outperformed the latter, with the exceptions of L_4 and L_5.

	Pos.class		Neg.class		Tot.class		Av.class		Identif.	
	Pos.	P-N	Pos.	P-N	Pos.	P-N	Pos.	P-N	Pos.	P-N
L_1	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
L_2	97.1	85.7	99.9	89.6	99.8	89.5	98.5	87.7	90.0	10.0
L_3	100.0	100.0	100.0	88.6	100.0	93.0	100.0	94.3	100.0	50.0
L_4	90.7	98.3	90.4	88.8	90.5	94.3	90.5	93.6	80.0	80.0
L_5	43.0	47.6	88.8	88.0	81.1	81.2	65.9	67.8	20.0	20.0
L_6	82.4	77.7	97.3	91.7	92.3	87.0	89.8	84.7	80.0	70.0
L_7	91.1	94.5	93.2	89.9	92.5	91.6	92.2	92.2	80.0	80.0
L_8	100.0	100.0	100.0	82.3	100.0	82.5	100.0	91.2	100.0	30.0
L_9	100.0	100.0	98.2	92.3	98.2	92.3	99.1	96.1	0.0	0.0
L_{10}	100.0	100.0	99.9	91.2	99.9	91.3	99.9	95.6	90.0	0.0
L_{11}	93.7	86.8	97.0	95.6	95.9	92.6	95.4	91.2	90.0	80.0
L_{12}	100.0	100.0	100.0	91.6	100.0	91.6	100.0	95.8	100.0	0.0
L_{13}	81.4	89.8	98.6	90.1	90.0	89.9	90.0	89.9	80.0	80.0
L_{14}	97.7	86.1	99.8	83.8	99.8	83.8	98.8	84.9	90.0	10.0
L_{15}	100.0	93.8	99.6	89.7	99.6	89.7	99.8	91.7	70.0	0.0
Mean	91.8	90.7	97.5	90.2	96.0	90.0	94.7	90.5	78.0	40.7

Table 5.3. *Classification results of the non-incremental method, with "maximize positive generalization" and "maximize positive and negative generalization" biases respectively, for sparse samples of the test languages.*

	Pos.class		Neg.class		Tot.class		Av.class		Identif.	
	SM_1	SM_2	SM_1	SM_2	SM_1	SM_2	SM_1	SM_2	SM_1	SM_2
L_1	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
L_2	85.7	74.3	89.6	74.2	89.5	74.2	87.7	74.2	10.0	0.0
L_3	97.7	51.4	87.0	62.9	91.1	58.3	92.3	57.2	40.0	0.0
L_4	98.3	33.5	88.8	76.9	94.3	51.6	93.6	55.2	80.0	0.0
L_5	48.6	57.7	88.9	65.8	82.1	64.4	68.8	61.8	20.0	0.0
L_6	78.2	61.1	91.2	60.0	86.8	60.4	84.7	60.5	70.0	10.0
L_7	94.5	44.4	89.9	76.9	91.6	65.1	92.2	60.7	80.0	0.0
L_8	100.0	86.4	82.3	74.5	82.5	74.6	91.2	80.5	30.0	0.0
L_9	100.0	100.0	92.3	77.7	92.3	77.8	96.1	88.9	0.0	0.0
L_{10}	100.0	65.0	91.2	83.7	91.3	83.4	95.6	74.3	0.0	0.0
L_{11}	86.8	75.2	95.6	62.1	92.6	66.4	91.2	68.7	80.0	0.0
L_{12}	100.0	93.8	91.6	79.9	91.6	80.0	95.8	86.8	0.0	0.0
L_{13}	89.8	81.4	90.1	68.9	89.9	75.2	89.9	75.2	80.0	40.0
L_{14}	86.1	75.8	83.8	76.1	83.8	76.0	84.9	75.9	10.0	0.0
L_{15}	93.8	88.7	89.7	80.0	89.7	80.1	91.7	84.4	0.0	0.0
Mean	90.6	72.6	90.1	74.6	89.9	72.5	90.4	73.6	40.0	10.0

Table 5.4. *Classification results of the pseudo-incremental methods SM_1 and SM_2, with "maximize positive and negative generalization" bias, for the test languages.*

Table 5.4 displays the features of the UFSAs inferred by the pseudo-incremental approach (Algorithm 5.3) using the split-and-merge procedure *SM_1* (left) and the *SM_2* (right), with the "maximize positive and negative generalization" bias in both cases. The presentation order was given by the random generation of the examples, alternating a positive and a negative string. The *SM_1*-based method (maximal splitting) returned the same UFSAs than the non-incremental method with the same bias in 145 of the 150 runs; this caused just slight differences in the figures computed for the languages L_3, L_5 and L_6, while the figures were identical for the rest of test languages. On the other hand, the performance of the *SM_2*-based method (minimal splitting) was considerably worse, including a poor identification ability, although the global classification rates were still above the 70%.

ALGORITHM	Pos.class	Neg.class	Tot.class	Av.class	Identif.
Non-incremental Max.Pos.	91.8	97.5	96.0	94.7	78.0
Non-incremental Max.P-N	90.7	90.2	90.0	90.5	40.7
Incremental SM_1 Max.P-N	90.6	90.1	89.9	90.4	40.0
Incremental SM_2 Max.P-N	72.6	74.6	72.5	73.6	10.0

Table 5.5. *Summary of results for the experiment using sparse samples.*

Table 5.5 shows the summary of the experiment results for the four RGI methods tested. The Oncina-García's method (non-incremental max.pos.) provided the best results. Both the non-incremental and the pseudo-incremental *SM_1*-based method, with "maximize positive and negative generalization" bias, also showed a high performance with classification rates around 90%. For comparison, Dupont reported figures corresponding to the "Av.class." column of 85.4% and 94.4% for his non-incremental and semi-incremental genetic RGI methods, respectively [Dupo:94].

5.5.2 RGI from complete samples

The second experiment consisted of testing the RGI algorithms when a *complete* sample is provided. This complete sample $\Sigma^l = (\Sigma^l_+, \Sigma^l_-)$ was made of all the strings over the involved alphabet Σ up to a given length l, each of them being declared either as positive or negative, according to the target language.

Firstly, the two non-incremental methods, corresponding to the "maximize positive generalization" (Oncina-García's) and "maximize positive and negative generalization" biases, were applied to learn the test languages from a complete sample Σ^l, where for each language L the length l was initially set to 1 and iteratively increased until the $A(L)$ and the $U(L)$ were inferred by the first and the second method, respectively. Note that once the target DFA or DUFA is inferred, supplying more examples will

not modify the output of the algorithm, and thus, we can say that the method has converged.

Table 5.6 shows the length values l for which convergence occurred for each test language and method. Let n be the number of states of the target automaton (either $A(L)$ or $U(L)$); in all cases, the empirical convergence length l was $\leq n+1$. Recall that the Trakhtenbrot and Barzdin method converges to the $A(L)$ from a complete sample with $l = 2n-1$ in the worst case. Likewise, it is interesting to comment that in the best results reported with (second-order) recurrent neural networks [MiGi:93], the presentation of a complete sample with $l = 9$ did not yield the convergence to a perfect classifier for Tomita's languages ($L_3 - L_7$), even using FSA extraction and minimization techniques.

It can be observed that in the languages where the first non-incremental method (Oncina-García's) converged earlier, the $U(L)$ contains one more state than the $A(L)$. On the other hand, the second method converged earlier in languages L_{11} and L_{13}; this was due to the fact that, in these languages, the complete sample for which the second method converged was structurally complete with respect to $U(L)$, while the associated positive sample was not structurally complete with respect to $A(L)$. Hence, this is a typical case when the symmetric generalization of positive and negative data causes an inference improvement.

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}	L_{11}	L_{12}	L_{13}	L_{14}	L_{15}
Max.Pos.	1	2	5	3	4	3	5	2	4	6	5	4	3	4	3
Max.P-N	2	3	5	4	4	3	5	4	4	6	4	5	2	5	5

Table 5.6. *Convergence length of the two non-incremental methods, corresponding to the "maximize positive generalization" and "maximize positive and negative generalization" biases, for complete samples of the test languages.*

Next, the pseudo-incremental *SM_1*-based method with "maximize positive and negative generalization" bias was tested following the same protocol, except that now, each complete sample was given to the algorithm in three different presentation orders: lexicographic, alphabetical and random. For all the test languages and presentation orders, the convergence length of the method was identical to that of the non-incremental procedure with the same bias (second row of Table 5.6). Therefore, this result is an empirical validation of the convergence property of the *SM_1*-based method.

Finally, the learning performance of the pseudo-incremental *SM_2*-based method (also with "maximize positive and negative generalization" bias) from complete samples was tested. Here, the experimental protocol was a little bit different. For each language, the parameter l of the complete sample was set to the convergence value of the *SM_1*-

based method (except that a minimal length $l = 3$ was set for the languages L_1 and L_{13}). Again, each complete sample was presented in lexicographic, alphabetical and random order.

Table 5.7 displays the classification rates (with respect to longer strings) of the UFSAs inferred by the *SM_2*-based method, together with their number of states and transitions, for each test language and presentation order. The minimal complete DUFA $U(L)$ was obtained in the cases where a 100% total classification rate is displayed. The behaviour of the method was irregular; in average, a total classification rate around 85% was computed for the lexicographic and alphabetical complete presentations, and this grew up to a 90% for the random order. In general, and specially for the test languages associated with the largest $U(L)$ in the set, the *SM_2*-based method showed a tendency towards the induction of excessively complex UFSAs.

	Pos.class			Neg.class			Tot.class			(states,transitions)		
	Lex.	Alph.	Rand.	Lex.	Alph.	Rand.	Lex.	Alph.	Rand.	Lex.	Alph.	Rand.
L_1	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	(2,4)	(2,4)	(2,4)
L_2	100.0	100.0	0.0	100.0	100.0	100.0	100.0	100.0	99.2	(3,6)	(3,6)	(4,8)
L_3	97.8	100.0	81.9	55.4	61.2	70.8	71.5	75.9	75.0	(17,34)	(19,38)	(21,42)
L_4	100.0	100.0	100.0	49.8	100.0	44.3	78.7	100.0	76.4	(6,12)	(4,8)	(6,12)
L_5	100.0	78.8	80.0	100.0	80.8	85.1	100.0	80.4	84.3	(4,8)	(8,16)	(8,16)
L_6	100.0	33.9	100.0	100.0	67.0	100.0	100.0	56.0	100.0	(3,6)	(4,8)	(3,6)
L_7	100.0	100.0	83.9	33.1	33.9	54.0	56.0	56.6	64.3	(10,20)	(11,22)	(20,40)
L_8	100.0	100.0	0.0	68.2	72.0	100.0	68.7	72.5	98.5	(9,18)	(9,18)	(6,12)
L_9	100.0	100.0	100.0	84.1	88.2	99.0	84.2	88.2	99.0	(12,36)	(14,42)	(11,33)
L_{10}	53.3	40.0	73.3	82.0	91.7	100.0	81.6	90.8	99.6	(20,40)	(18,36)	(9,18)
L_{11}	48.8	10.1	58.3	82.9	91.8	78.7	71.4	64.1	71.8	(5,10)	(8,16)	(6,12)
L_{12}	100.0	0.0	100.0	88.9	98.5	99.4	89.1	97.3	99.4	(14,28)	(8,16)	(5,10)
L_{13}	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	(2,4)	(2,4)	(2,4)
L_{14}	100.0	100.0	100.0	82.0	87.8	93.6	82.4	88.1	93.8	(12,24)	(11,22)	(11,22)
L_{15}	50.0	100.0	50.0	88.5	94.0	99.9	88.3	94.0	99.7	(30,90)	(15,45)	(13,39)
Mean	90.0	77.5	75.2	81.0	84.5	88.3	84.8	84.3	90.7	-	-	-

Table 5.7. *Inference results of the pseudo-incremental method SM_2 for differently ordered complete samples of the test languages.*