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 2

Regular grammatical inference: theory and symbolic methods

This chapter begins with the statement of the basic definitions and theory about formal languages and grammars, finite-state machines (FSMs), finite-state automata (FSA), and regular expressions (REs). The included contents serve as a basis for the study of grammatical inference, and they will be used frequently throughout the work.

Next, the theory of regular grammatical inference (RGI) is reviewed, following closely a recent formalization of the subject in which RGI is viewed as a search problem [DuMV:94]. A brief summary of the current knowledge about the complexity of regular language identification is also included.

The chapter ends with an overview of the symbolic RGI methods that have been reported previously. These methods are classified in three groups, depending on the type of the supplied data:

1) RGI techniques from positive examples,
2) RGI techniques from both positive and negative examples, and
3) RGI methods using queries.

The first group is further divided into heuristic and characterizable techniques. For each method, the main features and technical aspects are described.

2.1 Basic concepts, definitions and notation

This section includes the basic definitions and concepts of formal languages and automata theory that are used in the sequel. Many books have been published that cover this issue (e.g. [Booth:67,Salo:73,Koha:78,Harr:78,HoUl:79]).

2.1.1 Languages and grammars

Let Σ denote a finite alphabet of symbols, and let Σ^* denote the set of all the strings over Σ. Let $|u|$ represent the length of the string u. Let λ denote the *empty string*, i.e. the only string of length zero. A *language* (over Σ) is any subset of Σ^*.

Let u, v, w denote strings over Σ, and let uv denote the concatenation of u and v. We say that u (v) is a *prefix* (*suffix*) of w if there exists v (u) such that $uv = w$. If L is a language over Σ, let $Pr(L) = \{u \in \Sigma^* \mid \exists v \in \Sigma^*,\ uv \in L\}$ denote the set of prefixes of L, and let $L/u = \{v \in \Sigma^* \mid uv \in L\}$ denote the *left-quotient* of L by u. It is clear that $L/u \neq \emptyset \Leftrightarrow u \in \mathrm{Pr}(L)$.

Let L, L_1, L_2 be languages over Σ. The *concatenation* of L_1 and L_2, denoted L_1L_2, is the language $\{xy \mid x \in L_1 \wedge y \in L_2\}$. Let $L^0 = \{\lambda\}$ and $L^i = LL^{i-1}$ for $i \geq 1$. The *closure* of L, denoted L^*, is the language

$$L^* = \bigcup_{i=0}^{\infty} L^i.$$

A *grammar* is a four-tuple $G = (V_N, V_T, P, S)$, where V_N and V_T are disjoint finite sets of nonterminal and terminal symbols, respectively, P is a finite set of production rules, and S is a special nonterminal called the start symbol. Grammars can be classified according to the allowed forms of the production rules. The Chomsky hierarchy [Chom:59] establishes four types of grammars: *unrestricted*, *context-sensitive*, *context-free*, and *regular grammars*. However, other types can be defined that are subtypes of the above.

Let $V = V_N \cup V_T$ and let V^* denote the set of all the strings composed by terminal and/or nonterminal symbols. The most general family of grammars is the (*type 0*) *unrestricted grammars*, which permits productions of the form $\alpha \longrightarrow \beta$ $(\alpha, \beta \in V^*,$ $\alpha \neq \lambda)$.

Let $\alpha \rightarrow \beta$ be a production rule of P for a grammar $G = (V_N, V_T, P, S)$. By applying $\alpha \rightarrow \beta$ to a string $\gamma\alpha\delta$ (where $\gamma, \delta \in V^*$) the string $\gamma\beta\delta$ is obtained, and we say that *$\gamma\beta\delta$ is directly derived from $\gamma\alpha\delta$* in grammar G, denoted $\gamma\alpha\delta \Rightarrow_G \gamma\beta\delta$. For any $\alpha, \beta \in V^*$, we say *β is derived from α* in grammar G or $\alpha \overset{*}{\Rightarrow}_G \beta$ if β follows from α by application of zero or more production rules of P. That is, $\overset{*}{\Rightarrow}_G$ is the reflexive and transitive closure of $\Rightarrow_G$.

Every grammar G *generates* a language $L(G)$, which is defined as $\{w \mid w \in V_T^* \wedge S \overset{*}{\Rightarrow}_G w\}$. That is, a string is in $L(G)$ if it consists solely of terminals and it can be derived from the start symbol. Two grammars G_1 and G_2 are *equivalent* if $L(G_1) = L(G_2)$. It can be proved that the languages generated by the unrestricted grammars are exactly the *recursively enumerable languages* (or *r.e. sets*), which are the languages accepted by Turing machines [HoUl:79].

A grammar G is (of *type 1* or) *context-sensitive* if every production rule $\alpha \rightarrow \beta$ satisfies $|\alpha| \leq |\beta|$ (with maybe the only exception of $S \rightarrow \lambda$ [Salo:73]). The languages generated by *context-sensitive grammars* (CSGs) are called *context-sensitive languages* (CSLs). A normal form for CSGs is given by restricting the production rules to the form $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ where $A \in V_N$, $\alpha_1, \alpha_2, \beta \in V^*$, $\beta \neq \lambda$.

G is a (*type 2* or) *context-free grammar* (CFG) if every production rule $p \in P$ is of the form $A \rightarrow \alpha$, where $A \in V_N$, $\alpha \in V^*$. The languages generated by CFGs are called *context-free languages* (CFLs). If every $p \in P$ is of one of the forms $A \rightarrow BC$ or $A \rightarrow a$, where $A, B, C \in V_N$, $a \in V_T$, then G is in Chomsky normal form. If every $p \in P$ is of the form $A \rightarrow a\alpha$, where $\alpha \in V_N^*$, then G is in Greibach normal form. In both cases, the rule $S \rightarrow \lambda$ must be added whenever $\lambda \in L(G)$.

Let $A, B \in V_N$, $w \in V_T^*$. We say that a CFG G is *right-linear* (*left-linear*) if all the production rules of G are of the form $A \rightarrow wB$ or $A \rightarrow w$ ($A \rightarrow Bw$ or $A \rightarrow w$, respectively). G is a (*type 3* or) *regular grammar* (RG) if it is a right- or left-linear grammar. A normal form for *right-linear* grammars restricts the production rules to the form $A \rightarrow aB$ or $A \rightarrow a$, where $a \in V_T$, except for the possible rule $S \rightarrow \lambda$. A similar normal form applies to the *left-linear* grammars. It can be proved [HoUl:79] that the languages generated by RGs are exactly the *regular languages* (or *regular sets*), which are the languages accepted by finite-state automata and denoted by regular expressions (see next subsections).

The Chomsky hierarchy is based on the following theorem, the proof of which is available elsewhere (e.g.[HoUl:79]).

Theorem 2.1. *Regular languages* $\subset$ CFLs $\subset$ CSLs $\subset$ *r.e. sets.*

2.1.2 Finite-state machines and finite-state automata

Informally, a finite-state machine (FSM) is an abstract device that reads a sequence of input symbols (from a tape) and writes an associated sequence of output symbols (to another tape) under the control of a finite set of states. There are two models of FSMs: the *Mealy machines* and the *Moore machines* [Booth:67].

A Mealy machine is a six-tuple $M = (Q, \Sigma, \Gamma, \delta, \eta, q_0)$, where Q is a finite set of states, Σ is a finite alphabet of input symbols, Γ is a finite alphabet of output symbols, $\delta : Q \times \Sigma \rightarrow Q$ is the *state transition function*, $\eta : Q \times \Sigma \rightarrow \Gamma$ is the *output function*, and q_0 is the *initial state*. A Moore machine is also a six-tuple $M = (Q, \Sigma, \Gamma, \delta, \eta, q_0)$, where all is as in the Mealy machine, except that the output function depends only on the states, i.e. $\eta : Q \rightarrow \Gamma$. It can be proved that these two FSM types produce the same class of input-output mappings (called *sequential functions*) by designing two algorithms that, given a machine of one type, they construct an equivalent machine of the other type [Booth:67].

A *deterministic finite-state automaton* (DFA) is a five-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q, Σ, δ and q_0 are as in a FSM, and $F \subseteq Q$ is the set of *accepting* or *final states*. A DFA may be viewed as a special case of a Moore machine where the output alphabet is $\Gamma = \{0, 1\}$ and a state $q \in Q$ is an accepting state if and only if $\eta(q) = 1$.

Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$, an associated function $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ can be defined, such that $\hat{\delta}(q, w)$ indicates the state reached after reading the string w starting in state q, in the following manner:

i) $\hat{\delta}(q, \lambda) = q$
ii) $\forall w \in \Sigma^* \; \forall a \in \Sigma : \; \hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$

A string w is said to be *accepted* by A if $\hat{\delta}(q_0, w) \in F$. Every DFA A *accepts* a language $L(A)$, which is defined as $\{w \mid \hat{\delta}(q_0, w) \in F\}$. It is clear that $L(A) \subseteq \Sigma^*$.

A *nondeterministic finite-state automaton* (NFA) is a five-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0 and F have the same meaning as for a DFA, but now $\delta : Q \times \Sigma \rightarrow 2^Q$ is a nondeterministic transition function, such that $\delta(q, a)$ indicates the set of all states to which there is a transition labeled a from state q. The function δ can be extended to a function $\hat{\delta} : Q \times \Sigma^* \rightarrow 2^Q$ as follows:

i) $\hat{\delta}(q, \lambda) = \{q\}$
ii) $\forall w \in \Sigma^* \; \forall a \in \Sigma : \; \hat{\delta}(q, wa) = \{p \mid \exists r \in \hat{\delta}(q, w) : \; p \in \delta(r, a)\}$

Every NFA $A = (Q, \Sigma, \delta, q_0, F)$ *accepts* a language $L(A)$, which is defined as $\{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$.

We will refer to the union of the set of all DFAs and the set of all NFAs as the class of *finite-state automata* (FSA). A directed graph, called a *transition diagram*, is associated with each FSA, where the vertices of the graph correspond to the states of the FSA, and there is an arc labeled a from state q to state p whenever $p \in \delta(q, a)$; graphically, final states are marked as encircled vertices and the start state is distinguished in some way[1] (see Fig. 2.1). Two FSA A_1 and A_2 are *equivalent* if $L(A_1) = L(A_2)$.

The classes of languages accepted respectively by DFAs and NFAs are identical (namely, the *regular languages*), and this is proved by designing an algorithm that for every NFA constructs an equivalent DFA [HoUl:79] (since, in addition, every DFA is an NFA). The way a DFA simulates an NFA is to allow the states of the DFA to correspond to sets of states of the NFA, and to keep track of all the states that the NFA could be in after reading an input. Therefore, in the worst case, the number of states of the equivalent DFA is exponential with the number of states of the given NFA.

The model of the NFA can be extended to include transitions on the empty input λ. Hence, a *nondeterministic finite-state automaton with λ-moves* is a five-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where now the state transition function is a mapping $\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow 2^Q$. Again, a transition function extended to strings $\hat{\delta}$ can be defined for an NFA with λ-moves [HoUl:79], and this allows to define $L(A)$, the language accepted by A, similarly. Any NFA with λ-moves can be simulated by an associated NFA without λ-moves [HoUl:79], and thus, the class of accepted languages is not increased by the addition of the λ-moves. However, NFAs with λ-moves can be useful as an intermediate step when constructing an FSA from a regular expression or from a regular grammar [HoUl:79].

A direct mapping from regular grammars to NFAs (without λ-moves) is provided by the following simple algorithm. Let $G = (V_N, V_T, P, S)$ be a regular grammar in right-linear normal form. Let $Q = V_N \cup \{q_f\}$, $\Sigma = V_T$ and $q_0 = S$. Let $\delta : (Q \times \Sigma) \rightarrow 2^Q$ be defined as follows: for each production rule of the form $A \rightarrow aB$ let $B \in \delta(A, a)$, and for each production rule of the form $A \rightarrow a$ let $q_f \in \delta(A, a)$. Finally, if $S \rightarrow \lambda \in P$, then $F = \{q_0, q_f\}$, and otherwise $F = \{q_f\}$. It can be proved that the NFA $A = (Q, \Sigma, \delta, q_0, F)$ so formed satisfies $L(A) = L(G)$.

It is also easy to construct a regular grammar equivalent to a given NFA $A = (Q, \Sigma, \delta, q_0, F)$. Let $G = (Q, \Sigma, P, q_0)$, where P consists of production rules $p \rightarrow aq$ whenever $q \in \delta(p, a)$ and also $p \rightarrow a$ whenever $\delta(p, a) \cap F \neq \emptyset$; if $q_0 \in F$ then the rule $q_0 \rightarrow \lambda$ is added to P. Again[2], it can be demonstrated that $L(G) = L(A)$.

[1]For instance, by means of an unlabeled incoming arrow, or by the special state labels q_0 or λ (the last when shortest prefixes are used for state labeling).

[2]The proofs of this and the previous statement are omitted for concision and clarity purposes.

2.1.3 Regular expressions

Let $\Sigma = \{a_1, ..., a_m\}$ be an alphabet. The *regular expressions* (REs) over Σ and the languages that they describe are defined recursively as follows.

i) $\emptyset$ is a regular expression and describes the empty set.
ii) λ is a regular expression and describes the set $\{\lambda\}$.
iii) For each $a_i \in \Sigma\ (1 \leq i \leq m)$, a_i is a regular expression and describes the set $\{a_i\}$.
iv) If P and Q are regular expressions describing the languages L_P and L_Q, respectively, then $(P+Q)$, (PQ), and (P^*) are regular expressions that describe the languages $L_P \cup L_Q$ (their *union*), $L_P L_Q$ (their *concatenation*) and L_P^* (the *closure* of L_P), respectively.
v) No other expressions are regular unless they can be generated in a *finite* number of applications of the above rules.

By convention, the precedence of the operations in decreasing order is $*$ (star), (concatenation), $+$ (union). This precedence together with the associativity of the concatenation and union operations allows to omit many parentheses in writing a regular expression.

We write $L(R)$ for the language described by RE R. Two regular expressions P and Q are said to be *equivalent*, denoted by $P = Q$, if they describe the same language. The following are some basic equivalence rules:

$$\emptyset + R = R \tag{2.1}$$
$$\emptyset R = R\emptyset = \emptyset \tag{2.2}$$
$$\lambda R = R\lambda = R \tag{2.3}$$
$$\lambda^* = \lambda \tag{2.4}$$
$$\emptyset^* = \lambda \tag{2.5}$$
$$R + R = R \tag{2.6}$$
$$PQ + PR = P(Q+R) \tag{2.7}$$
$$PQ + RQ = (P+R)Q \tag{2.8}$$
$$R^*R^* = R^* \tag{2.9}$$
$$RR^* = R^*R \tag{2.10}$$
$$(R^*)^* = R^* \tag{2.11}$$
$$\lambda + RR^* = R^* \tag{2.12}$$
$$(PQ)^*P = P(QP)^* \tag{2.13}$$
$$(P+Q)^* = (P^*Q^*)^* = (P^* + Q^*)^* \tag{2.14}$$
$$(P+Q)^* = P^*(QP^*)^* = (P^*Q)^*P^* \tag{2.15}$$

Although the algebraic manipulation of regular expressions, using the preceding (and other derived) identities, can be used to simplify them or to demonstrate their equivalence, it does not provide an automatic tool for determining the equivalence of two regular expressions. It is a better approach to convert the two given REs into their equivalent FSA and to test these for equivalence [HoUl:79].

A language is said to be *rational* if and only if it can be described by a regular expression. Every language accepted by an FSA can be represented by a regular expression and every language denoted by a regular expression can be recognized by an FSA (Kleene's theorem [Koha:78]), and thus, *rational* and *regular* languages are the same family of languages. An NFA with λ-moves A_λ is easily constructed that accepts $L(R)$ for a given RE R [HoUl:79]; then A_λ can be further transformed into an NFA without λ-moves, or a DFA if desired.

Given an FSA A, there can be many equivalent REs R such that $L(A) = L(R)$. Several algorithms have been proposed to find a regular expression that describes the language accepted by a given FSA [Koha:78, HoUl:79]. By selecting a specific algorithm, a deterministic mapping ψ can be established from FSA to REs, this is, a canonical RE R can be chosen for each FSA A, $R = \psi(A)$. In Chapter 8, a modification of Arden's algorithm [Arden:60] will be proposed as the mapping ψ in the context of learning *augmented* REs[3]. Arden's algorithm is based on the following theorem, known as Arden's lemma, the proof of which can be found in [Koha:78].

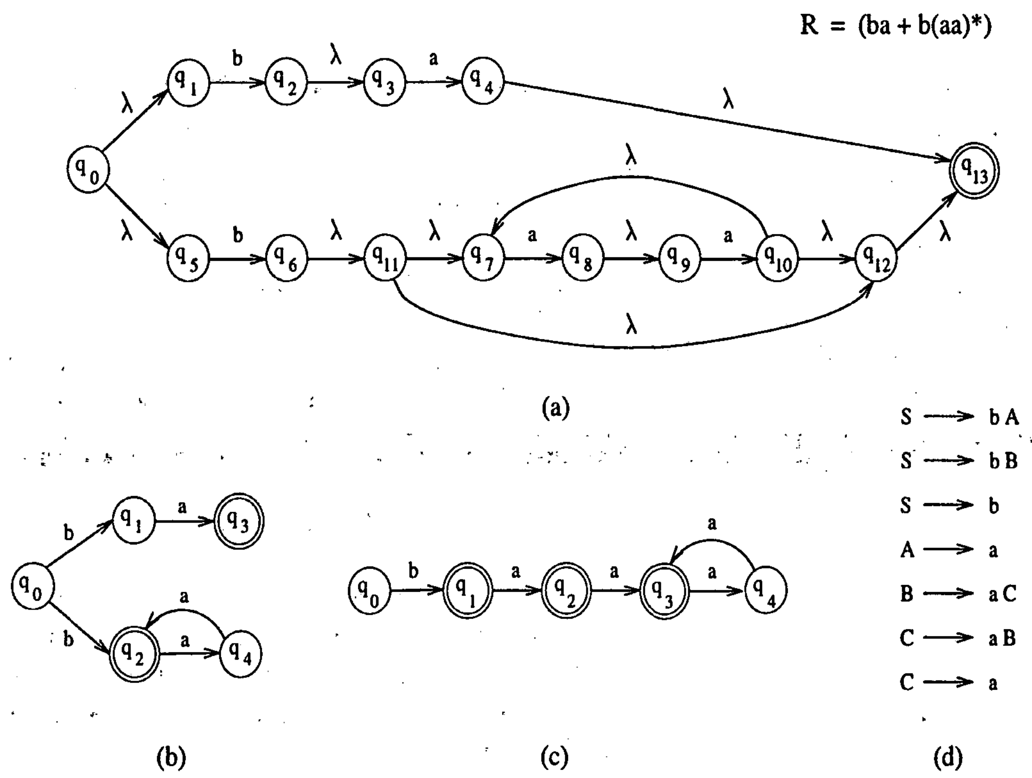
Theorem 2.2. *Let Q, P and R be regular expressions over a finite alphabet. Then, if $\lambda \notin L(P)$, the equation $R = Q + RP$ has a unique solution given by $R = QP^*$.*

The following theorem summarizes the equivalence relationships among the different representations of regular languages. The proof can be done by parts showing pairwise equivalence, and each part is indeed a known theorem, which proof is based on giving the algorithms for going from one description to the other and viceversa [HoUl:79].

Theorem 2.3. *Regular grammars, FSA, and regular expressions are distinct classes of descriptions of the same class of languages, the so-called regular languages.*

In Fig. 2.1, given the regular language described by the RE over $\{a, b\}$ $R = (ba + b(aa)^*)$, we show an NFA with λ-moves A_λ, an NFA without λ-moves A_N, a DFA A_D, and a regular grammar G, such that $L(R) = L(A_\lambda) = L(A_N) = L(A_D) = L(G)$. In fact, the sequence of automata A_λ, A_N, A_D, is obtained from R by applying the aforementioned algorithms [HoUl:79], and G is obtained from A_N by applying the NFA-to-RG algorithm described previously.

[3]*Augmented regular expressions* will be defined in Chapter 8.



$S \longrightarrow bA$

$S \longrightarrow bB$

$S \longrightarrow b$

$A \longrightarrow a$

$B \longrightarrow aC$

$C \longrightarrow aB$

$C \longrightarrow a$

Fig. 2.1 *Different descriptions for the regular language denoted by* $(ba + b(aa)^*)$. (a) *NFA with λ-moves A_λ* (b) *NFA A_N* (c) *DFA A_D* (d) *Regular grammar G*

2.2 Basic theory for regular grammatical inference

This section comprises the material of automata theory and the theoretical developments of previous researchers that provide the basis for regular grammatical inference (RGI). Most of the contents presented here are picked from a recent paper by Dupont *et al.* [DuMV:94], which gives a comprehensive statement of RGI as a search problem. Other references that may be consulted about this topic are [FuBo:75, Gold:78, Fu:82, AnSm:83, KuSh:88, Micl:90, OnGa:92a-b, Greg:94].

2.2.1 Statement of the RGI problem

A *sample* of a language L over an alphabet Σ is a tuple $S = (S^+, S^-)$, where S^+ is a finite subset of L (called *positive sample*), and S^- is a finite subset of the complementary language $\Sigma^* - L$ (called *negative sample*). Consequently, S^+ and S^- are two disjoint finite subsets of Σ^*.

The problem of *regular grammatical inference* (RGI), or simply *regular inference*, can be stated as follows: given a sample $S = (S^+, S^-)$, such that $S^+ \neq \emptyset$, discover the unknown FSA A_T, called the *target automaton*, from which S^+ is supposed to have been generated (and which does not accept any of the strings in S^-, whenever $S^- \neq \emptyset$). Hence, A_T must satisfy $L(A_T) \supseteq S^+ \wedge (\Sigma^* - L(A_T)) \supseteq S^-$, i.e. A_T must be *consistent* with the supplied data.

Recall that an equivalent regular grammar is easily obtainable from any FSA A_T, so the term "grammatical inference" is perfectly valid. Also note that the unknown language L, from which the sample S is taken, is assumed to be *regular*. If this were not true, any FSA yielded by a RGI method applied to S would only represent a regular approximation of L, that may be quite imperfect.

Even when L is actually regular, there are theoretical and practical difficulties in solving the problem. Since S is a finite set, it is clear that a huge (infinite) number of FSA consistent with S exist. By establishing some reasonable assumptions (namely, the *structural completeness* of the sample, to be defined later), the A_T candidates can be reduced to a finite but still excessively large number. In practice, one or more criteria (inductive biases) must be imposed to select just one or a few plausible solutions. Simplicity and maximal generalization are the usually preferred inductive criteria; however, other heuristic criteria can be applied, specially when there is some prior (partial) knowledge of the solution.

2.2.2 Further definitions

Next, some additional definitions are introduced that involve FSA (*derived FSA*, *canonical DFA*, *acceptances*), FSA associated with language samples (the MCA, PTA, CDA, and *universal* automata), and lattices of FSA, which are required to characterize the RGI problem and its possible solutions.

2.2.2.1 Derived automata

For any set Z, a partition π is a set of pairwise disjoint nonempty subsets of Z whose union is Z. Let $z \in Z$ and let $B(z,\pi)$ denote the unique element, or *block*, of π containing z. A partition π_1 *refines*, or *is finer than*, other partition π_2 if and only if every block of π_2 is either a block of π_1 or a union of two or more blocks of π_1.

Let $A = (Q, \Sigma, \delta, q_0, F)$ be an FSA. The *FSA derived from A with respect to the partition* π of Q, also called the *quotient automaton* A/π, is defined as $A/\pi = (Q', \Sigma, \delta', B(q_0,\pi), F')$, where

$$\begin{aligned} Q' &= Q/\pi = \{B(q,\pi) \mid q \in Q\}, \\ F' &= \{B \in Q' \mid B \cap F \neq \emptyset\}, \text{ and} \\ \delta' &: Q' \times \Sigma \to 2^{Q'} \text{ is given by } \forall B, B' \in Q', \forall a \in \Sigma : \\ &\qquad B' \in \delta'(B,a) \Leftrightarrow \exists q, q' \in Q, q \in B, q' \in B' \wedge q' \in \delta(q,a). \end{aligned}$$

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

2.2.2.2 Canonical automaton of a regular language, k-tails and DFA minimization

Let L be a regular language over Σ. Let $A(L) = (Q, \Sigma, \delta, q_0, F)$ be a DFA, such that $Q = \{L/u \mid u \in Pr(L)\}$, $q_0 = L/\lambda$, $F = \{L/u \mid u \in L\}$ and $\delta(L/u, a) = L/ua$ where $u, ua \in Pr(L)$. $A(L)$ is called the *canonical automaton* of L, and it can be proved that every DFA which accepts exactly L and contains the *minimal number of states* for this language is isomorphic to A(L) [HoUl:79].

Let $A = (Q, \Sigma, \delta, q_0, F)$ be an FSA. A string u is a *tail* of a state $q \in Q$ if $\hat{\delta}(q,u) \in F$; u is a *k-tail* of q if it is a tail of q and $|u| \leq k$. The set of k-tails of q is written Ψ_q^k. We say that two distinct states q, q' of A are distinguishable by $u \in \Sigma^*$ if either $\hat{\delta}(q,u) \in F \wedge \hat{\delta}(q',u) \notin F$ or $\hat{\delta}(q,u) \notin F \wedge \hat{\delta}(q',u) \in F$; and q, q' are *k-equivalent*, denoted $q \overset{k}{\equiv} q'$, if they are not distinguishable by any string u such that $|u| \leq k$. This is, q and q' are *k-equivalent* if and only if they have the same set of k-tails, i.e.

$$\forall q, q' \in Q : \; q \overset{k}{\equiv} q' \iff \Psi_q^k = \Psi_{q'}^k.$$

Finally, q and q' are *equivalent*, denoted $q \overset{\infty}{\equiv} q'$ or simply $q \equiv q'$, if they have the same set of tails. It is easily shown that both $\overset{k}{\equiv}$ and $\overset{\infty}{\equiv}$ are true equivalence relations on the set of states Q.

The following property is at the basis of Moore's algorithm for DFA minimization:

$$\forall q, q' \in Q : \; q \overset{k+1}{\equiv} q' \iff q \overset{k}{\equiv} q' \wedge \forall a \in \Sigma : \delta(q,a) \overset{k}{\equiv} \delta(q',a) \tag{2.16}$$

Let π_k and π_∞ be the partitions of the states of $A_0 = (Q, \Sigma, \delta, q_0, F)$ resulting from the relations $\overset{k}{\equiv}$ and $\overset{\infty}{\equiv}$, respectively. Then, it follows that, if A_0 is a DFA, the derived automaton A_0/π_∞ is the minimal DFA accepting $L(A_0)$, i.e. A_0/π_∞ is the canonical automaton of $L(A_0)$; and the DFA minimization algorithm can be described as follows:

```
**ALGORITHM 2.1:** *Moore's algorithm* ($A_0$)
begin
$k := 0$
compute the partition $\pi_k$ for the relation $\overset{0}{\equiv}$ on $Q$
*convergence* :=FALSE
while $k \leq |Q| - 2$ and not *convergence* do
    $k := k + 1$
    compute $\pi_k$ as a function of $\pi_{k-1}$ using eq.(2.16);
    *convergence* := $(\pi_k = \pi_{k-1})$
end_while
compute the derived automaton $A_0/\pi_k$ { i.e. merge the $k$-equivalent states }
returns $A_0/\pi_\infty$ { since $A_0/\pi_k = A_0/\pi_\infty$ after convergence }
end_algorithm
```

It should be noted that if Moore's algorithm is applied to a non-deterministic FSA A_0, there is no guarantee that the result be the canonical automaton $A(L(A_0))$ nor the minimal NFA recognizing $L(A_0)$.

2.2.2.3 Structural completeness of a sample

An *acceptance* of a string $u = a_1 ... a_l$ by an FSA $A = (Q, \Sigma, \delta, q_0, F)$, denoted $\mathcal{AC}(u, A)$, is a sequence of $l + 1$ states $(q^0, ..., q^l)$ such that $q^0 = q_0$, $q^l \in F$, and $q^{i+1} \in \delta(q^i, a_{i+1})$, for $0 \leq i \leq (l - 1)$, i.e. a path of the associated transition diagram starting at q_0 and ending in a final state that is labeled by u. The $l + 1$ states in the path are said to be *reached* for $\mathcal{AC}(u, A)$ and the state q^l is said to be *used as accepting state.* The l transitions (elements of δ) traversed in the path are said to be *exercized* by $\mathcal{AC}(u, A)$.

A state q of an FSA A is *useful* if there exists a string $u \in L(A)$ for which q may be reached for some acceptance of u. Otherwise, the state is said to be *useless.* An FSA that contains no useless state is said to be *stripped.* An FSA A is *ambiguous* if there is at least one string u for which there are several acceptances; clearly, only NFAs may be ambiguous.

An acceptance of a set of strings S_s by an FSA A, denoted $\mathcal{AC}(S_s, A)$, is a set of acceptances $\mathcal{AC}(u, A)$ of the strings $u \in S_s$ by A, such that there is only one acceptance for each string. The set of transitions exercized by (respectively states reached for) $\mathcal{AC}(S_s, A)$ is the union of the sets of transitions exercized by (respectively states reached for) each string acceptance in $\mathcal{AC}(S_s, A)$. A similar extension serves to define the set of states used as accepting states for $\mathcal{AC}(S_s, A)$.

According to the new correct definition given in [DuMV:94], a positive sample S^+ is said to be *structurally complete* with respect to an FSA $A = (Q, \Sigma, \delta, q_0, F)$, if there exists an acceptance $\mathcal{AC}(S^+, A)$ of S^+ such that

i) every transition in δ of A is exercized by $\mathcal{AC}(S^+, A)$, and
ii) every state in F of A is used as accepting state for $\mathcal{AC}(S^+, A)$.

Note that the existence of $\mathcal{AC}(S^+, A)$ implies $S^+ \subseteq L(A)$. By extension, we say that a sample $S = (S^+, S^-)$ is *structurally complete* with respect to an FSA A if S^+ is structurally complete with respect to A and $S^- \cap L(A) = \emptyset$.

2.2.2.4 Automata related to a positive sample

Let $S^+ = \{u_1, ..., u_M\}$ be a positive sample, where $u_i = a_{i,1}...a_{i,|u_i|}$, $1 \le i \le M$. A *canonical FSA with respect to* S^+ is any FSA A_C such that $L(A_C) = S^+$. Three significant FSA related to S^+ are defined as follows.

The *maximal canonical automaton with respect to* S^+ is the FSA $MCA(S^+) = (Q, \Sigma, \delta, q_0, F)$, where

Σ is composed of all the symbols that appear in S^+,
$Q = \{v_{i,j} \mid 1 \le i \le M,\ 1 \le j \le |u_i|,\ v_{i,j} = a_{i,1}...a_{i,j}\} \cup \{\lambda\}$,
$q_0 = \lambda, \quad F = S^+$,
$\forall a \in \Sigma : \ \delta(\lambda, a) = \{v_{i,1} \mid v_{i,1} = a,\ 1 \le i \le M\}$, and
$\delta(v_{i,j}, a) = \{v_{i,j+1} \mid v_{i,j+1} = v_{i,j}a_{i,j+1} \wedge a_{i,j+1} = a\}\ 1 \le i \le M,\ 1 \le j \le |u_i|-1.$

Hence, $L(MCA(S^+)) = S^+$ and $MCA(S^+)$ is the FSA having the largest number of useful states, with respect to which S^+ is structurally complete. $MCA(S^+)$ is generally an NFA, where the only possible nondeterministic transitions are from the start state.

The *prefix tree acceptor* of S^+, denoted $PTA(S^+)$, is the quotient automaton $MCA(S^+)/\pi_{pr}$, where the partition π_{pr} is defined by

$$B(q, \pi_{pr}) = B(q', \pi_{pr}) \quad \Leftrightarrow \quad Pr(q) = Pr(q')$$

That is, the prefix tree acceptor $PTA(S^+)$ is obtained from the $MCA(S^+)$ by merging the states that share the same prefixes. Note that $PTA(S^+)$ is always a DFA.

The *canonical derivative acceptor*, denoted $CDA(S^+)$, is the canonical automaton of the language S^+, i.e. $CDA(S^+) = A(S^+)$, and thus $CDA(S^+)$ is the DFA that is the *minimal canonical automaton with respect to* S^+. The canonical derivative acceptor $CDA(S^+)$ can be obtained from the $PTA(S^+)$ by merging the states that share the same set of tails, i.e. $CDA(S^+) = PTA(S^+)/\pi_{tl}$, where the partition π_{tl} is defined by

$$B(q, \pi_{tl}) = B(q', \pi_{tl}) \quad \Leftrightarrow \quad S^+/Pr(q) = S^+/Pr(q').$$

The *universal automaton* over an alphabet Σ is the DFA $UA = (Q, \Sigma, \delta, q_0, F)$, where $Q = F = \{q_0\}$ and $\forall a \in \Sigma : \delta(q_0, a) = q_0$. It is clear that $L(UA) = \Sigma^*$ and UA is the smallest FSA with respect to which every positive sample of Σ^* including all the symbols in Σ is structurally complete.

2.2.2.5 Lattices of automata

Let $P(A)$ denote the set of partitions π of the set of states Q of an FSA A. Let $|\pi_i|$ denote the number of blocks of the partition π_i. Let $\pi_1 = \{B_{1,1} \ldots B_{1,|\pi_1|}\}$ and π_2 be two partitions of $P(A)$. We say that π_2 *is directly derived from* π_1, denoted $\pi_1 \preceq \pi_2$, if π_2 is constructed from π_1 as $\pi_2 = \{B_{1,j} \cup B_{1,k}\} \cup (\pi_1 - \{B_{1,j}, B_{1,k}\})$ for some $j, k \in [1, |\pi_1|], j \neq k$. Consequently, $|\pi_2| = |\pi_1| - 1$. By extension to quotient automata, we also say that A/π_2 *is directly derived from* A/π_1, or $A/\pi_1 \preceq A/\pi_2$.

The preceding operation $\preceq$ defines a partial orden relation on $P(A)$. Let $\ll$ denote the reflexive and transitive closure of $\preceq$. It follows that π_j *is derived from* π_i or $\pi_i \ll \pi_j \Leftrightarrow \pi_i$ is finer than π_j. In that case, we also say that A/π_j *is derived from* A/π_i (denoted $A/\pi_i \ll A/\pi_j$). By the way a quotient automaton is built, it is readily proved [FuBo:75] that

$$A/\pi_i \ll A/\pi_j \quad \Rightarrow \quad L(A/\pi_i) \subseteq L(A/\pi_j) \tag{2.17}$$

The set of FSAs derived from a given FSA A is partially ordered by the relation $\preceq$, and furthermore, it is a lattice, denoted $Lat(A)$, of which A and the universal automaton UA are respectively the null and universal elements[4]. The *depth* of an FSA A/π in $Lat(A)$ is given by $n - |\pi|$, where n is the number of states of A. Therefore, the depths of A and UA in $Lat(A)$ are 0 and $n - 1$, respectively.

[4]Strictly speaking, UA is the universal element of $Lat(A)$ whenever the set of final states F of A is not empty; otherwise (a degenerated case), the universal element is similar to UA except that its unique state is not accepting, and all the FSA in $Lat(A)$ accept the empty language.

2.2.3 The search space of the RGI problem

As commented before, the statement of the RGI problem allows for an infinite number of candidates to be the target automaton A_T. Given the additional hypothesis of structural completeness of the sample S, the RGI problem may be considered as a search through a (finite) lattice built from the positive sample S^+. This property follows from the following three theorems, the proof of which is available in [DuMV:94]. Three corollaries are appended here, one for each theorem, that cover the general case of having both a positive and a negative sample.

Theorem 2.4. *Let S^+ be a positive sample of any regular language L and let A_T be any FSA accepting exactly L. If S^+ is structurally complete with respect to A_T then A_T belongs to $Lat(MCA(S^+))$.*

Corollary 1. *Let $S = (S^+, S^-)$ be a sample of any regular language L and let A_T be any FSA accepting exactly L. If S is structurally complete with respect to A_T then A_T belongs to $Lat(MCA(S^+))$.*

Proof. If S is structurally complete with respect to A_T, then S^+ is structurally complete with respect to A_T (implying $S^+ \subseteq L(A_T)$) and $S^- \cap L(A_T) = \emptyset$. Hence, A_T is consistent with S and by Theorem 2.4 it belongs to $Lat(MCA(S^+))$. □

Theorem 2.5. *Let S^+ be a positive sample of any regular language L and let $A(L)$ be (the DFA that is) the canonical automaton of L. If S^+ is structurally complete with respect to $A(L)$ then $A(L)$ belongs to $Lat(PTA(S^+))$.*

Corollary 1. *Let $S = (S^+, S^-)$ be a sample of any regular language L and let $A(L)$ be the canonical automaton of L. If S is structurally complete with respect to $A(L)$ then $A(L)$ belongs to $Lat(PTA(S^+))$.*

Proof. The same argument than in the previous corollary holds substituting $A(L)$ for A_T, $PTA(S^+)$ for $MCA(S^+)$, and Theorem 2.5 for Theorem 2.4. □

Theorem 2.6. *Let S^+ be a positive sample. Let $\mathcal{A}_{S^+}$ be the set of FSAs such that S^+ is structurally complete with respect to any FSA belonging to $\mathcal{A}_{S^+}$. The set $\mathcal{A}_{S^+}$ is equal to $Lat(MCA(S^+))$.*

Corollary 1. *Let $S = (S^+, S^-)$ be a sample. Let $\mathcal{A}_S$ be the set of FSAs such that S is structurally complete with respect to any FSA belonging to $\mathcal{A}_S$. The set $\mathcal{A}_S$ is a subset of $Lat(MCA(S^+))$.*

Proof. The set $\mathcal{A}_S$ is the intersection of the set $\mathcal{A}_{S^+}$ and the set $\mathcal{A}'_S$ of FSA consistent with S. Since $\mathcal{A}_{S^+} = Lat(MCA(S^+))$ by Theorem 2.6, it follows that $\mathcal{A}_S \subseteq Lat(MCA(S^+))$. When $S^- = \emptyset$, $\mathcal{A}_S = Lat(MCA(S^+))$, but otherwise $\mathcal{A}_S \subset Lat(MCA(S^+))$, since at least the universal automaton $UA \in Lat(MCA(S^+))$ is inconsistent with any S with nonempty S^-. Also, $\mathcal{A}_S \neq \emptyset$, since at least $MCA(S^+) \in \mathcal{A}_S$ always. □

The so-called *minimal DFA consistency problem* consists of finding the DFA that is consistent with a sample $S = (S^+, S^-)$ and has the minimal number of states. This problem has been proved to be NP-hard [Gold:78,Angl:78]. The minimal DFA consistency problem can be viewed as the search of the DFA in $\mathcal{A}_S$ with minimal number of states, where the simplicity of the inferred FSA is used as criterion to select the target A_T. However, the search space for this problem can be further delimited using the concept of *border set* [DuMV:94], which is introduced next.

An *antistring* $\overline{as}$ in a lattice of FSAs is a set of FSAs such that any element of $\overline{as}$ is not related by $\ll$ with any other element of $\overline{as}$. An FSA A is *at a maximal depth* in a lattice of FSAs with respect to a negative sample S^-, if there is no FSA A' which may be derived from A such that $L(A') \cap S^- = \emptyset$. Given a sample $S = (S^+, S^-)$, the *border set* $BS_{MCA}(S)$, respectively $BS_{PTA}(S)$, is the antistring in $Lat(MCA(S^+))$, respectively $Lat(PTA(S^+))$, of which each element is at a maximal depth with respect to S^-.

The search space of the RGI problem presents some interesting properties that are listed next.

1. $Lat(PTA(S^+)) \subseteq Lat(MCA(S^+))$.
2. $\forall A \in Lat(MCA(S^+)): \quad A \text{ is a DFA} \;\Rightarrow\; A \in Lat(PTA(S^+))$.
3. There exist positive samples S^+ for which the languages which may be identified from $Lat(PTA(S^+))$ is properly included in the set of languages which may be identified from $Lat(MCA(S^+))$; in particular, some languages[5] are only represented by NFAs in $Lat(MCA(S^+))$.
4. $BS_{PTA}(S) \subseteq BS_{MCA}(S)$.
5. There may be several distinct languages represented by the FSAs belonging to $BS_{MCA}(S)$.
6. There may exist NFAs belonging to $BS_{MCA}(S)$ which contain less states than the minimal consistent DFA.

[5]Necessarily, S^+ is not structurally complete with respect to the canonical automata of these languages.

7. All the DFAs belonging to $BS_{PTA}(S)$ are necessarily minimal for the language they accept.
8. There may exist canonical automata (for some languages) consistent with S which do not belong to $BS_{PTA}(S)$, since other FSAs in $BS_{PTA}(S)$ may be derived from them.

The property #2 can be proved by showing that all the FSAs that belong to $Lat(MCA(S^+))$ but not to $Lat(PTA(S^+))$ are quotient automata $MCA(S^+)/\pi$ for which the partition π include two distinct blocks B_i and B_j such that $\exists q_i \in B_i, \exists q_j \in B_j : B(q_i, \pi_{pr}) = B(q_j, \pi_{pr})$, i.e. there exists a common prefix of q_i and q_j and also of B_i and B_j, and thus, these FSAs are necessarily NFAs. The rest of the previous properties are shown in [DuMV:94].

The minimal DFA consistency problem might be considered as the discovery of the smallest DFA in $BS_{PTA}(S)$. However, the minimal DFA consistent with S could belong or not to $BS_{PTA}(S)$ (by property 8). Therefore, to delimit the search space for the minimal consistent DFA precisely, the concept of *deterministic border set* has to be introduced as follows.

Given a sample $S = (S^+, S^-)$, the $DBS_{PTA}(S)$, is the antistring in $Lat(PTA(S^+))$, of which each element is a DFA A such that there is no DFA A' which may be derived from A satisfying $L(A') \cap S^- = \emptyset$. It is obvious that the minimal consistent DFA must belong to the deterministic border set $DBS_{PTA}(S)$.

2.2.4 The complexity of regular language identification

In a classical paper [Gold:67], Gold showed that all the recursively enumerable classes of languages (context-sensitive and below) can be identified in the limit using both positive and negative examples. On the other hand, no *superfinite* class of languages, i.e. one that contains all finite languages and at least a language that is infinite, can be identified in the limit using only positive examples. Both statements apply to the case of regular languages. However, there are certain (not superfinite) sub-classes of regular languages that can be identified in the limit using only positive examples, e.g. the classes of *k-reversible* languages [Angl:82], *k-testable* languages [GaVi:90], and *terminal distinguishable regular languages* [RaNa:87]. The conditions that are required to identify in the limit a class of languages from just positive data were stated by Angluin [Angl:80a].

On the other hand, there has been a long controversy about the *practical* tractability of learning DFAs from both positive and negative examples. The reader is referred to

[Pitt:89] for an overview on the computational complexity of DFA learning. Dupont *et al.* [DuMV:94] summarized the key negative theoretical evidence in the following three points:

- The problem of finding the minimal DFA consistent with $S = (S^+, S^-)$ is NP-hard [Gold:78,Angl:78].
- The minimal DFA consistency problem cannot be approximated within any polynomial of the size of the optimal solution [Pitt:89].
- Approximate inference of FSAs from sparse labeled examples is NP-hard if an adversary chooses both the target automaton and the sample [KeVa:89].

The relationship between the minimal DFA consistency problem and the regular language identification problem is given by the fact that an algorithm which solved the former for each new (positive or negative) example would identify in the limit any regular language. However, this does not mean that there do not exist polynomial-time algorithms which return a possible solution of the RGI problem and identify in the limit any regular language from a positive and negative sample; indeed, some methods with these properties have been reported [Gold:78,OnGa:92a-b,Lang:92] (see next section).

Even earlier, Trakhtenbrot and Barzdin proposed a state-merging algorithm for building the smallest DFA consistent with a *complete* sample [TrBa:73]. This complete sample is made of all the strings up to the length l, each of them being labeled either as positive or negative. The length l depends on some features of the target DFA and has been shown to be equal to $2n - 1$, in the *worst case*, where n is the size of the target DFA [TrBa:73]. The time complexity of Trakhtenbrot-Barzdin method is $O(mn^2)$, where m is the size of the $PTA(S^+)$. However, whenever n is large, the size of the complete sample becomes prohibitive. Furthermore, in the worst case, the exact FSA identification is not feasible if a vanishingly small fraction of the complete sample is lacking [Angl:78].

Nevertheless, the computational complexity is better in the *average case*. For DFAs randomly drawn from a well defined probability distribution, the expected value of the complete sample size $|S|$ is $(|\Sigma|^2 n^C log_2 n - 1)/(|\Sigma| - 1)$, where C only depends on Σ [TrBa:73]. Although, for small alphabets, the expected average size of a complete sample might suggest that exact identification of random DFAs is feasible, there is no practical guarantee to have a complete sample available, whatever its size, unless an oracle which answers membership queries is accessible (see next section). A more realistic assumption is to consider the availability of only *sparse* data, i.e. two disjoint sets S^+ and S^- without any further restriction.

In this framework, Lang showed empirically that highly accurate approximate identification may be achieved by using a (randomly drawn) vanishingly small fraction of the complete sample (needed by the Trakhtenbrot and Barzdin's method); that is, a fraction which decreases as the size of the target DFA increases [Lang:92]. Furthermore, Oncina and García have shown that if the sample contains a representative set of the target regular language, their RGI algorithm is proved to produce the canonical DFA for this language (which is also the minimal consistent DFA for the sample); the size of the representative sample is only of order $O(n^2)$, where n is the number of states of the canonical DFA [OnGa:92a-b] .

Finally, Angluin has demonstrated that, whenever a teacher is available, the combined use of a structurally complete sample and membership queries to identify the target language can lead to an improvement in efficiency [Angl:81]. This is shown by defining a procedure for this setting that identifies any regular language using a number of queries that is polynomial in n, $|\Sigma|$, and the size of the sample, and which can be run in time polynomial in these variables.

2.3 Symbolic methods for regular grammatical inference

In this section, an overview of the most outstanding symbolic methods proposed for regular grammatical inference is given. The reader is referred to a recent survey by Gregor [Greg:94] and to the original papers describing the methods for more details. See also the previous surveys by Fu [Fu:82] and Miclet [Micl:90].

2.3.1 Methods for RGI from a positive sample

Since the class of regular languages cannot be identified in the limit from only positive data [Gold:67], most of the methods for RGI from a positive sample are *heuristic techniques* that are based on selecting one or more inductive biases (*heuristic criteria*) to determine a plausible hypothesis. Nevertheless, some other methods have been proposed that are able to identify from positive data a subclass of the regular languages; these are called *characterizable techniques* [AnSm:83] because their results, i.e. the hypotheses selected, are characterizable according to some restriction.

Besides the discrimination between heuristic and characterizable techniques [AnSm:83], the methods for RGI from positive data can be classified into three groups: *equivalence relation methods*, *miscellaneous methods* and *error correcting procedures* [Greg:94]. In the two former groups, the data is assumed to be perfect positive examples, and the methods can often cope with the availability of a negative sample by adjusting some parameter in order to choose a consistent hypothesis. On the other hand, the error correcting procedures handle (potentially) imperfect examples and therefore they do not allow for a negative sample.

2.3.1.1 Heuristic techniques

The methods in the following list use a variety of equivalence relations to form either a few sets, or a unique set, of equivalence classes among the states of a canonical automaton with respect to S^+. For each one of these partitions, a derived FSA A_D is obtained that can be selected as the inferred FSA. However, there is no guarantee that any of these A_D correspond to the target FSA A_T. Moreover, if a negative sample S^- is available, the inferred FSA A_D could be inconsistent with S^-. In the methods which offer a limited set of hypotheses, the consistency requirement may be used to filter this set.

1) The *k-tails* method [BiFe:72].
2) The *tail-clustering* method [Micl:80].
3) The *successor* method [RiVe:84].
4) The *predecessor-successor* method [VeRi:84].
5) The *partial similarities* method [KuSh:88].

The *k-tails* method [BiFe:72] offers a family of FSAs A_k derived from a canonical FSA with respect to S^+ (e.g. $MCA(S^+)$ or $PTA(S^+)$) using the *k-equivalence* relation $\overset{k}{\equiv}$ (for $0 \leq k \leq |Q| - 2$), where $|Q|$ is the number of states of the canonical FSA. Adjusting the parameter k affects the characteristics of the inferred FSA considerably. If $k \geq \max_{u \in S^+} |u|$ then $L(A_k) = S^+$, and if, in addition, the starting FSA is the $PTA(S^+)$, then $A_k = CDA(S^+)$, i.e the canonical derivative acceptor. If S^- is non-empty, the value of k may initially be set to 0 and then increased until an FSA is found that is consistent with $S = (S^+, S^-)$. Because the *k-tails* method gives great weight to the string terminations in selecting the hypothesis, it is ill adapted to pattern recognition problems, where the result should take into account the overall properties of the sample.

The *tail-clustering* method [Micl:80] is a generalization of the k-tails approach. There is no parameter k to be adjusted and a unique FSA is returned as selected hypothesis. This method iteratively groups the states of the $MCA(S^+)$ using distance measures between sets of tails and a hierarchical clustering algorithm. It provides better results than the k-tails method [GrJu:92], at the expense of a higher computational cost.

The *successor* method [RiVe:84] infers a minimal DFA by analysing pairwise successions of symbols in the strings of S^+. For each symbol a in $\Sigma^\lambda = \Sigma \cup \{\lambda\}$, the set $Succ(a)$ is defined as the symbols succeeding a in S^+, where λ succeeds any trailing symbol and it is succeeded by any starting symbol. Firstly, a DFA is created containing just one state for each symbol in Σ^λ; then, the symbols that have identical sets of successors are said to be equivalent and a derived DFA is built according to the resulting equivalence classes. Usually, the result is a strong generalization with respect to S^+; in many cases (e.g. for samples including long strings in comparison with $|\Sigma^\lambda|$) the sets $Succ(a)$, for $a \in \Sigma^\lambda$, tend to include most symbols and, hence, the universal automaton UA is likely inferred.

The *predecessor-successor* method [VeRi:84] also infers a DFA (though not necessarily minimal) and it compensates, to some extent, for the drawbacks of the successor method. It consists of establishing a relabeling scheme for the sample S^+ (giving rise to a new alphabet and sample set), applying afterwards the successor method, and translating the resulting DFA to the original alphabet. The relabeling is aimed at associating each symbol in Σ with a non-unique number of states. To this end, an equivalence relation is defined for the different occurrences in S^+ of each symbol that is based on the predecessor symbols.

The *partial similarities* method [KuSh:88] is indeed a general framework which embodies a large number of "reasonable" ways by which the states of the $MCA(S^+)$ can be partitioned. Its flexibility is achieved through the definition of several equivalence relations based on similarities of either predecessor or succesor substrings, or both. The *partial similarities* framework covers the k-tails and successor methods and allows for other strategies of which the authors proposed three parameterized methods: $\wedge(m,n)$, $\vee(m,n)$ and $+(k)$ [KuSh:88].

Let $Pred_m(q)$ and $Succ_n(q)$ denote the predecessor and successor substrings of length m and n, respectively, for a state $q \in MCA(S^+)$, such that an entire prefix or suffix shorter than the specified length can be adopted. The equivalence relations $\mathcal{R}_{p_m}$ and $\mathcal{R}_{s_n}$ are given by $q\mathcal{R}_{p_m}q' \Leftrightarrow Pred_m(q) = Pred_m(q')$ and $q\mathcal{R}_{s_n}q' \Leftrightarrow Succ_n(q) = Succ_n(q')$, respectively. Then, for $\wedge(m,n)$, two states are considered equivalent when $q\mathcal{R}_{p_m}q' \wedge q\mathcal{R}_{s_n}q'$ (i.e. both their predecessors and successors are identical). For $\vee(m,n)$, the equivalence relation is given by the transitive closure of $q\mathcal{R}_{p_m}q' \vee q\mathcal{R}_{s_n}q'$ (i.e. either predecessors or successors identical). Finally, for $+(k)$, two states are equivalent if they share a common substring of length k in the concatenation of predecessor and successor substrings ($m + n = k$ with variable m), and again the transitive closure is required. By adjusting the length parameters, these three methods each offer a family of derived FSA, from which a hypothesis can be chosen on basis of generalization degree or compatibility with a negative sample.

Other *miscellaneous* heuristic methods [Greg:94], which are not based on state equivalence relations, are listed and commented below.

6) The uv^kw method [Micl:76].

7) The *Itoga's recursive merging* method [Itoga:81].

8) The *morphic generator* methods [GaVC:87].

The uv^kw method [Micl:76] infers a regular expression by detecting repeated substrings in S^+ and generalizing the sample using the pumping lemma for regular languages:

If L is a regular language over Σ, then $\exists p \in \mathcal{N}$ such that $\forall s \in L$ with $|s| > p$, s can be written $s = uvw$, where $|v| \geq 2$, so that $\forall k \geq 0$, $uv^kw \in L$.

The assumption behind this method is that the sample contains strings that are long in relation to the number of states of the generating FSA. Different candidate substrings can be detected that fit in the searched pattern, and one is selected that provides the "best matching" to S^+. When a possible recursion has been identified, an inference step is completed by rewriting the sample with a new symbol replacing each occurrence of the selected substring. Several inference steps can be made until no substring is found that is "sufficiently consistent" with S^+. The sample is thus rewritten several times, each time giving a regular expression that includes its predecessor. The size of the inferred language (denoted by the inferred regular expression) can be reduced by augmenting the required number of repetitions k of the searched substring, since it can be shown that $L(k+1) \subseteq L(k)$. The uv^kw method is computationally expensive, and the notions of "best matching" and "sufficiently consistent" are not well-defined.

The *Itoga's* method [Itoga:81] infers a DFA by recursively merging the states of the $CDA(S^+)$. The rule for state merging is not based on an equivalence relation but on the following heuristic criterion: two states q_i, q_j are merged (into q_i) if $R_j \subseteq R_i$, where $R_i = \{(a, q_k) \mid \delta(q_i, a) = q_k,\ a \in \Sigma\}$. The process is repeated until no more states can be merged. During the recursive merging of states, q_j could be merged with several candidate states q_i, and the choice may influence the structure of the inferred DFA. This method tends to produce low-complexity DFA and is claimed to perform well for small samples while it might be unsuitable for larger sets.

The *morphic generator* methodology [GaVC:87] permits to use a GI algorithm together with symbol renaming functions which incorporate application-dependent knowledge. This methodology is based on the following property: *for any regular* *language L there exists a local language*[6] *l and a letter-to-letter morphism*[7] *h such that* $L = h(l)$. Firstly, a symbol renaming function $g : \Sigma^* \rightarrow \Sigma'^*$ is applied to S^+ yielding a transformed sample $S'^+ = g(S^+)$, under the assumption that a local language $l(S'^+)$ captures more of the underlying structure of the sample than does $l(S^+)$. Then, an FSA accepting a local language $l(S'^+)$ is built; for example, using the successor method, the minimal DFA associated with a local language $l(S'^+)$ is produced. Finally, a letter-to-letter morphism $h : \Sigma'^* \rightarrow \Sigma^*$ is applied to obtain a translation back into the original alphabet, thus giving an FSA that accepts the regular language $L = h(l(g(S^+)))$. The morphism h is typically (but not necessarily) the inverse of g; in such a case, it follows that $S^+ \subseteq L \subseteq l(S^+)$.

The *morphic generator* methodology covers the predecessor-successor method, but a great number of other renaming functions can be devised. However, it is unclear how to define suitable symbol renaming functions that ensure a good generalization. An application of the methodology to speech recognition has been reported [GaSV:90].

A third class of RGI heuristic techniques is given by the error-correcting procedures. Usually, in a real application, the examples in S^+ are imperfect representations due to an original noisy input and/or segmentation errors. These imperfections may be modelled as insertion, deletion and substitution edit operations on an ideal string. During recognition, error correction permits to compensate for this problem by temporarily adding states and transitions to an FSA that account for the allowed edits; a local cost is assigned to each of the edit operations, and dynamic programming is used to optimize the global cost for a full edit sequence, so that each string is recognized with a certain cost. By making the edit-derived states and transitions permanent, the error correction concept can also be applied for FSA inference.

Two such error-correcting RGI methods are commented next, which have some common properties. An ambiguous NFA A is generally inferred, that contains no loops or cycles in its transition diagram, so that $L(A)$ is finite; however, $S^+ \subseteq L(A)$, i.e. some generalization is made. Moreover, each state of A has all of its incoming transitions labeled by the same symbol, and there is only one final state. The first given example constitutes the initial structure of the FSA, and subsequent examples are optimally aligned (by computing the lowest-cost edit sequence) and incorporated (by editing the FSA) one-by-one using an error correction scheme. Both methods are well-suited for applications where substrings occur at approximately the same locations in many examples in S^+, and they differ mainly on the choice of the cost function.

[6]Local languages are a subclass of the regular languages, such that each local language can be characterized by a finite set of substrings of length two that are allowed to occur.

[7]A letter-to-letter morphism is a function $h : \Sigma'^* \rightarrow \Sigma^*$ which preserves concatenation and ensures that $\forall a \in \Sigma' : h(a) \in \Sigma$.

9) The *Markov network inference* method [ThGr:86].
10) The *ECGI* algorithm [RuVi:87].

The method by Thomason and Granum [ThGr:86] infers stochastic FSA called *Markov networks.* The cost function used for dynamic programming error correction is designed such that an example is optimally aligned when the edited automaton will generate it with maximum probability. The cost of a specific edit operation is thus the probability with which it is part of an alignment, and it is computed as a relative-frequency estimate of the corresponding transition probability. These estimates are obtained from frequency counts asigned to each transition and incrementally updated when the transition is used in an alignment. The same cost function is also applied for error correction by use of stochastic FSA. In addition, several global characteristics of S_+^+ examples are preserved, e.g. average number of appearances of specific symbols, average string lengths, and average distance between two identical symbols.

The *ECGI* algorithm by Rulot and Vidal [RuVi:87] infers FSAs that are not necessarily stochastic. The cost function used for dynamic programming error correction computes a normalized weighted Levenshtein distance between an example string and the nearest string belonging to the language accepted by the unedited FSA. The *ECGI* algorithm can be extended to infer stochastic FSA by computing relative-frequency estimates of the transition probabilities based on frequency counts of use (as before). Each type of error correction (depending only on the symbols involved) has its probability approximated in a similar way. A different cost function is defined for error correction by use of stochastic FSA [Greg:94].

2.3.1.2 Characterizable techniques

The following GI methods permit to identify in the limit a certain subclass of regular languages using only positive examples. If there is some a-priori knowledge which indicates that the target language belongs to one of these subclasses, then the corresponding method should prevail.

1) The *k-reversible languages* method [Angl:82].
2) The *skeleton* method [RaNa:87].
3) The *k-testable languages* method [GaVi:90].

A regular language L is *k-reversible* [Angl:82], if whenever two prefix strings, whose last k symbols are identical, have one tail in common then they have all tails in common, i.e. $u_1vw, u_2vw \in L \;\wedge\; v \in \Sigma^k \;\Rightarrow\; (\forall z \in \Sigma^* : \; u_1vz \in L \Leftrightarrow u_2vz \in L)$. Let R_k denote the class of k-reversibles languages over Σ for a specific value of k. The union of all R_k, $k = 0, 1, 2, ...$, is a hierarchy of classes that contains all finite languages over

Σ where $R_k \subset R_{k+1}$. This hierarchy defines the class of reversible languages, which is a proper subclass of the regular languages. k-reversible languages are accepted by k-reversible FSAs. An FSA $A = (Q, \Sigma, \delta, q_0, F)$ is k-reversible iff A is a DFA and the FSA $A^r = (Q, \Sigma, \delta^r, F, \{q_0\})$, obtained by reversing each of the transitions of A, is *deterministic with lookahead k*. An FSA is deterministic with lookahead k if $\forall q, q' \in Q : (\exists q'' \in Q, a \in \Sigma : q, q' \in \delta(q'', a)) \Rightarrow \neg(\exists v \in \Sigma^k : \hat{\delta}(q, v) \in Q \wedge \hat{\delta}(q', v) \in Q)$.

The *k-reversible languages* method [Angl:82] is an equivalence relation based method [Greg:94], that builds from the $MCA(S^+)$ a derived (k-reversible) DFA which generates the smallest k-reversible language containing S^+. For a fixed value of k, the method provides a well-defined FSA from S^+ in a non-heuristic manner. When the parameter k is an estimate, a setting of k lower than that of the target language results in an over-generalization, while a higher setting yields a too conservative inference. If a negative sample S^- is available, the value of k may initially be set to 0 and then increased until a consistent DFA is found.

The class of *terminal distinguishable regular languages* (TDRLs) [RaNa:87] is defined as follows. Let $\Sigma(w) \subseteq \Sigma$ denote the set of distinct symbols in a string $w \in \Sigma^*$. A regular language L is terminal distinguishable iff $\forall u, v, w, z \in \Sigma^* : uw, vw \in L \wedge \Sigma(w) = \Sigma(z) \Rightarrow (uz \in L \Leftrightarrow vz \in L)$. The class of TDRLs properly contains the class of zero-reversible languages, some languages from the class of k-reversible languages for all values of k, and also some nonreversible languages.

The *skeleton* method [RaNa:87] is also an equivalence relation based method [Greg:94], that builds from the $MCA(S^+)$ a derived unambiguous FSA A which generates the smallest language in the class of TDRLs containing S^+. If S^+ is a positive sample of a target regular language L, then it can be shown that $S^+ \subseteq L(A) \subseteq L$.

The class of *k-testable languages in the strict sense* (k-TLSS) [GaVi:90] is a subclass of the regular languages, such that each k-testable language can be characterized by a finite set of substrings of length k that are allowed to occur. This is a generalization of the local languages, for which $k = 2$. The class of k-TLSS is included in the class of k-reversible languages.

The *k-testable languages* method [GaVi:90] infers a DFA A_k which accepts the smallest language in k-TLSS containing S^+. Adjusting the parameter k has a considerable influence on the resulting DFA. For example, $L(A_1) = \Sigma^*$ while $L(A_k) = S^+$ if $k \geq \max_{u \in S^+} |u|$. In general, $S^+ \subseteq L(A_{k+1}) \subseteq L(A_k)$. The choice of k is tentative, but if $S^- \neq \emptyset$, the value of k may initially be set to 2 and then increased until a consistent DFA is found.

The main advantages of the three methods here discussed are the same: identification in the limit from positive presentation, and a well-characterized language accepted by the inferred FSA.

2.3.2 Methods for RGI from a positive and a negative sample

The following methods infer FSAs that are solutions of the RGI problem, given a sample containing both positive and negative examples (which, in general, represents sparse data). Some of them can be proved to identify in the limit the class of regular languages.

1) The *Gold's identification* method [Gold:78].
2) The *Tomita's hill climbing* method [Tomi:82].
3) The *Oncina's recursive merging* methods [OnGa:92a,OnGa:92b].
4) The *border set finding (BIG and RIG)* methods [MiGe:94].
5) The *border set heuristic pruning (BRIG)* method [MiGe:94].
6) The *genetic search (GIG)* method [Dupo:94].
7) The *RPNI2* method [Dupo:96].

In the same paper where Gold demonstrated that the problem of finding the minimal DFA consistent with a given sample $S = (S^+, S^-)$ is NP-hard [Gold:78], he proposed a non-incremental algorithm to identify in the limit any regular language. Gold's algorithm is polynomial in time (obviously, it is not based on finding the minimal consistent DFA), but it has an important drawback, it cannot generalize S^+ unless this sample contains a characteristic set. When there are not enough examples, the algorithm does not ensure the consistency with the data, and therefore, it is not appropriate for pattern recognition tasks, where the lack of data is usual.

Tomita presented an algorithm for finding an FSA consistent with S based on heuristic search by hill climbing [Tomi:82]. A function of merit to be optimized was defined for the FSAs in a set which was not explicitly related to the lattice of FSA derived from a canonical automaton.

More recently, Oncina and García [OnGa:92a-b] have proposed two polynomial-time algorithms that identify in the limit the regular languages from a sample $S = (S^+, S^-)$, and which overcome the practical limitations of Gold's method. Both algorithms are very similar, they perform a greedy search in $Lat(PTA(S^+))$ that yields a "locally optimal" solution to the consistency problem, with the basic difference that the first one [OnGa:92a] may return an NFA A_N as solution (such that $A_N \in BS_{PTA}(S)$), while the second one [OnGa:92b], sometimes called the *RPNI method*, constrains the set of hypotheses to DFAs and always returns a DFA $A_D \in DBS_{PTA}(S)$.

In both versions, the selected FSA is usually small with respect to the $PTA(S^+)$, since the algorithms try to maximize the generalization of positive information, but they do not necessarily obtain the minimal DFA consistent with S, except if the data contain a representative sample. Oncina and García have characterized the representative sample $S_r = (S_r^+, S_r^-)$ proper to their algorithms to identify a regular language L, and they have shown that its size is of order $O(n^2)$, where n is the number of states of the canonical DFA $A(L)$. When $S_r \subseteq S$, the returned solution is always isomorphic to $A(L)$ [OnGa:92a]. It should be pointed that the RPNI algorithm was also developed independently by Lang [Lang:92].

Since this algorithm will be extended in a later chapter, it is worthwhile to describe it more precisely. The key idea consists of establishing a lexicographic order among the states of the prefix tree acceptor of the positive sample $PTA(S^+)$ and trying to merge pairwise the states following this order. The lexicographic order in Σ^* (let refer to it as $<$) gives a breadth-first trajectory on the tree, since each state is labeled by its single prefix. Once a partition π of $PTA(S^+)$ is defined, the lexicographic order is extended to the blocks $B \in \pi$ in the following way: $B_i < B_j$ iff $\exists u \in B_i \; : \; \forall v \in B_j, u < v$. A deterministic merge operation $DMerge(\pi, B_i, B_j)$ is employed at each step, which yields a cascade of simple merge operations (beginning with $merge(\pi, B_i, B_j)$) that stops when a deterministic FSA is reached. The whole algorithm has a worst time complexity of $O((||S^+|| + ||S^-||)\; ||S^+||^2)$ and it can be described as follows:

ALGORITHM 2.2: *Oncina-García non-incremental DFA-based RGI*

```
begin
$A_1 := PTA(S^+)$; let $m$ be the number of states of $PTA(S^+)$; and
 let $\pi_1 := \{\{u\} \mid u \in Pr(S^+)\} = \{\{u_1\}, ..., \{u_m\}\}$ be the initial partition, where
 the prefixes are indexed in lexicographic order, so $u_1 = \lambda$;
for j:= 2 to m do
    if $\exists B \in \pi_{j-1} \; : \; u_j \in B \wedge B < u_j$ then $\pi_j := \pi_{j-1}$
    else find within $B \in \pi_{j-1}, B < u_j$, the lowest block $B_i$ such that
            $S^- \cap L(PTA(S^+)/DMerge(\pi_{j-1}, B_i, u_j)) = \emptyset$
        if found such $B_i$ then $\pi_j := DMerge(\pi_{j-1}, B_i, u_j)$
        else $\pi_j := \pi_{j-1}$
        end_if
    end_if
    $A_j := PTA(S^+)/\pi_j$ (which is always a DFA)
end_for
return $A_m = PTA(S^+)/\pi_m$ as the selected solution
end_algorithm
```

Two exponential-time algorithms, BIG and RIG, have been recently presented by Miclet and Gentile [MiGe:94] that find the complete *border set* $BS_{MCA}(S)$, containing all the consistent maximal generalizations of the positive sample (in close agreement with Mitchell's version space scheme for concept learning [Mitc:82]). Both algorithms are breadth-first search methods, which use to prune the search space the property that once an inconsistent FSA is found, all of its derived FSAs are also inconsistent. Whereas the BIG method requires the storage, at each step i, of all the partitions corresponding to the FSAs of depths i and $i+1$ in $Lat(MCA(S^+))$, the RIG method only requires the storage of the subsets of partitions corresponding to the *consistent* FSA of depths i and $i+1$. The main purpose of the RIG method is to obtain the subset B^* of the border set $BS_{MCA}(S)$ that contains the FSAs of maximal depth (i.e. with minimal number of states).

Besides their theoretical interest, the BIG and RIG algorithms solve the minimal consistent DFA problem, and therefore, they identify in the limit any regular language, but their exponential complexity makes them completely unpractical. Consequently, the same authors have proposed a polynomial-time heuristic version, the BRIG algorithm, that is based on a random pruning of the consistent FSAs and returns just a subset of the FSAs in $BS_{MCA}(S)$ with minimal number of states, i.e. a subset $B^*_{HL} \subseteq B^*$ [MiGe:94]. The BRIG algorithm is still a breadth-first approach but which uses a beam-search strategy to reduce the complexity; the BRIG method is provided with two parameters H and L, which determine the number of consistent FSAs randomly selected at each depth, and the percentage of FSAs directly derived from them that are tested, respectively. This method tries to exploit the great redundancy in the lattice of derived automata, since any FSA of depth i can be directly derived from many FSAs of depth $i-1$. Some experiments have been reported which show that the underlying assumption is reasonable, since the BRIG method was usually able to obtain the target FSA in the set of returned solutions, for a few simple regular languages [MiGe:94].

Dupont has recently reported a genetic algorithm approach, the GIG method, to the RGI problem from a sample $S = (S^+, S^-)$ [Dupo:94]. The RGI problem is regarded as an optimization problem, that is, the search through the lattice $Lat(MCA(S^+))$ for the FSA A for which an evaluation function is minimal. The evaluation function proposed is related to the number of states of the automaton and strongly penalizes the acceptance of negative examples, thus it is aimed at rewarding the consistent FSA of maximal depth. In order to apply the genetic search scheme, each individual in the population corresponds to a partition π of the set of states of the $MCA(S^+)$, and a left-to-right canonical group-number encoding is selected to represent each partition as a chromosome (or genotype) [Dupo:94]. Moreover, two genetic operators are defined (structural mutation and structural crossover) that perform simple transformations on partitions but which actually work on the associated genotypes.

As the sample size grows the complexity of the genetic search increases, and to cope with this drawback, Dupont proposes a "semi-incremental" version of the GIG method, in which all the examples are available at once, but the positive ones are processed sequentially [Dupo:94]. Both versions are heuristic techniques, which do not guarantee to arrive at the minimal consistent DFA. The GIG algorithms were experimentally compared to Oncina's method for inferring simple regular languages, and a similar performance was reported for the task of inferring the target FSA [Dupo:94].

A pseudo-incremental version of the RPNI algorithm, the so-called *RPNI2 algorithm*, has been presented very recently by Dupont in the ICGI'96 colloquium held in Montpellier (France) in September 1996 [Dupo:96]. The RPNI2 algorithm maintains the prefix tree of the positive sample and the list of negative examples and keeps a record of the partition of prefix tree states associated with the current hypothesis. It is a split-and-merge based method that guarantees the identification in the limit property. It should be pointed here that two split-and-merge pseudo-incremental RGI methods will be described in Chapter 5, which work also from positive and negative examples and are rather similar to the RPNI2 algorithm, but these methods were reported previously in a communication to the SSPR'94 workshop held in Nahariya (Israel) in October 1994 [AlSa:95a]. One of the differences between the RPNI2 algorithm and the methods described in Chapter 5 is that the former uses classical FSAs while the latter are based on UFSAs (a type of representation defined in [AlSa:95a] and Chapter 5). The RPNI and RPNI2 algorithms displayed nearly identical classification rates and converged at the same rate in the experiments reported in [Dupo:96].

2.3.3 Methods for RGI using queries

The following methods are able to identify the regular languages using an oracle to which certain types of queries can be addressed:

1) The *Pao-Carr* method [PaCa:78].
2) The *Angluin's ID* procedure [Angl:81].
3) The *Angluin's* L^* algorithm [Angl:87].
4) The *Trakhtenbrot-Barzdin complete sample* method [TrBa:73].
5) The *Porat-Feldman lexicographically-ordered complete sample* method [PoFe:91].
6) The *Sempere-García lexicographically-ordered complete sample* method [SeGa:93].

The *Pao-Carr* method [PaCa:78] uses a mixture of given data and membership queries to identify regular languages. Initially, a structurally complete positive sample S^+ is given, and then the algorithm makes membership queries to a teacher (who knows the language to be inferred) in order to complete the identification. The algorithm uses the information from the teacher to construct a trajectory in the lattice $Lat(MCA(S^+))$, starting from the $MCA(S^+)$ and merging states according to the answers.

At a node in the lattice, corresponding to an FSA A, the Pao-Carr method has to choose between different possible merges of the states of A. For a candidate pair of states q_1, q_2 to be merged, the resulting derived FSA A' is computed and a string x is selected such that $x \in L(A') \wedge x \notin L(A)$. Then the algorithm asks the informant whether x is in the target language. If it is, A' is taken as the next FSA in the trajectory; and otherwise, all the FSAs derived from A' (including itself) can be removed from the lattice of valid hypotheses, and a different pair of states of A is tried. The process continues until there is a single FSA remaining as valid in the lattice.

The Pao-Carr method is rather unrealistic, since the number of queries must be huge for a non-very-small sample, due to the number of partitions of the $MCA(S^+)$ to be considered, and therefore, the computation time becomes astronomical. However, it is theoretically interesting because of its identification property, in the case of a structurally complete S^+, and the use of a teacher instead of an a-priori heuristic criterion.

The Pao-Carr method was considerably improved by Angluin [Angl:81]. Given the same setting of a structurally complete positive sample S^+ and a teacher answering membership queries, the ID procedure [Angl:81] identifies the target regular language L using no more than kmn queries, where $k = |\Sigma|$ (the alphabet size), n is the number of states of the (completed)[8] canonical DFA for L, and $m = |PTA(S^+)| + 1$. Moreover, the ID procedure may be implemented to run in time polynomial in k, n and m; more precisely, its time complexity can be shown to be $O(km^2n)$. In contrast, it is also proved that if the teacher is only complemented by the information of the number of states n (i.e. a structurally complete sample is not given), then the required number of queries is exponential in n.

Let $P = Pr(S^+)$ and $P' = P \cup \{d_0\}$, where d_0 is a special symbol introduced for labeling a possible *useless* state in the completed canonical DFA $A(L) = (Q, \Sigma, \delta, q_0, F)$. Let f be a function on $P' \times \Sigma$ such that $f(d_0, a) = d_0 \;\; \forall a \in \Sigma$ and $f(u, a) = ua \;\; \forall (u, a) \in P \times \Sigma$. The value $f(u, a)$ represents the state reached on input a from the state represented by $u \in P'$. Let $T' = P' \cup \; range(f)$ and let $T = T' - \{d_0\}$. The procedure ID constructs a partition of T' that places all the elements representing a single state in a single block of the partition.

[8]Angluin [Angl:81] considers as the target FSA the minimal DFA $A(L)$ extended to contain a fully-defined transition function δ; this means that at most one *useless* or *dead* state may be added with respect to $A(L)$.

Firstly, ID builds an initial partition of T' (with two blocks) by discriminating between accepting and non-accepting states (asking the teacher when required). Then, ID successively refines this partition in a manner that is analogous to the DFA minimization procedure. In the process, ID builds up a set of strings $V = \{v_0 = \lambda, v_1, ..., v_m\}$ such that any two distinct states q, q' of the completed $A(L)$ are distinguishable by some $v_i \in V$, i.e. $\exists i \in [0, m] : \delta(q, v_i) \in F \wedge \delta(q', v_i) \notin F$ or viceversa. For every iteration $i \in [0, m]$ and $u \in T$, the sets of tails $E_i(d_0) = \emptyset$ and $E_i(u) = \{v_j \mid 0 \leq j \leq i \wedge uv_j \in L\}$ are defined. The i-th partition is defined by putting all elements u with a common value of $E_i(u)$ into the same block. It is clear that if $u_1, u_2 \in T$ and $\delta(q_0, u_1) = \delta(q_0, u_2)$ then $E_i(u_1) = E_i(u_2)$ for $i \in [0, m]$. At the final partition, the value of $E_m(u)$ uniquely determines which state of the completed $A(L)$ the string $u \in T$ represents, and hence, a DFA that is isomorphic to the completed $A(L)$ can be obtained. Note that the returned DFA is an FSA derived from an "extended" prefix tree acceptor $PTA'(S^+)$, given by the above function f.

The L^* algorithm, also by Angluin [Angl:87], is able to identify the regular languages, for the case where the target language L is presented by a *minimally adequate teacher*, which can answer membership queries about the language and equivalence queries, i.e. to test a conjecture and indicate whether it is equal to the unknown L and provide a counterexample if not. A counterexample is a string in the symmetric difference of L and the conjectured language. The L^* algorithm may be implemented to run in time polynomial in the number of states of the canonical DFA $A(L)$ and the maximum length of any counterexample provided by the teacher. In the adopted scheme [Angl:87], the conjectures are described by minimal DFAs (i.e. the set of hypotheses is the set of minimal DFAs), and therefore, to answer conjectures, the teacher may perform a standard polynomial-time algorithm for testing the equivalence of two DFAs, which yields a counterexample in the case they are not equivalent.

The availabilty of a minimally adequate teacher seems a rather demanding assumption in practice, due to the intrinsic difficulties in answering equivalence queries. Taking this into account, Angluin [Angl:87] proposed a modification of the L^* algorithm, the so-called approximate learner L^*_a, which identifies regular languages in the PAC (probably approximately correct) learning criterion established by Valiant [Vali:84]. The key point is that the equivalence queries can be replaced by a random sampling oracle. In this stochastic setting, it is assumed that there is some probability distribution *Prob* (unknown to the learner) on the set of all strings over Σ, so that the random sampling oracle selects a string $s \in \Sigma^*$ according to the distribution *Prob* and returns the pair (s, d), where d is *yes* if $s \in L$ and *no* otherwise. The algorithm L^*_a is given, at the start of the computation, two positive numbers between 0 and 1, the *accuracy* ϵ and the *confidence* δ, from which it is able to determine the number of queries to the random sampling oracle that are needed to test each conjecture.

In addition, the *Trakhtenbrot-Barzdin* algorithm [TrBa:73], discussed in Section 2.2.4, could be regarded as a query-based method, since a complete sample is required, and this can be built using a teacher and membership queries.

Finally, I will mention two incremental algorithms [PoFe:91,SeGa:93] that identify any regular language L from a lexicographically-ordered complete sample by converging to a DFA that is isomorphic to the canonical DFA $A(L)$. Again, since we cannot expect in general that this kind of sample be directly available in a *given data* framework, it is reasonable to assume that the strings are generated in lexicographic order, and for each one, a membership query is addressed to an oracle (i.e. a *requested data* scheme). Whereas the *Porat-Feldman* method uses NFAs as current hypotheses [PoFe:91], the *Sempere-García* method constrains the set of hypotheses to DFAs [SeGa:93]. The latter method uses a constructive technique for state grouping which is inspired on the RPNI algorithm. The total time complexity of both algorithms is of the same order, $nl\sum_{i=1}^{l} k^i(k+1)$, where $n = |A(L)|$, $k = |\Sigma|$, and l is the string length required to obtain a complete sample for L (recall that $l = 2n - 1$ in the worst case). Although the authors claimed that this complexity could be considered "polynomial in an amortized sense" [PoFe:91,SeGa:93], it should be noted that the number of generated strings and queries needed to identify L is exponential in n, as it was proved previously by Angluin [Angl:81].