Learning Subsequential Transducers for Pattern Recognition Interpretation Tasks*

José Oncina, Pedro García, and Enrique Vidal

Departamento de Sistemas Informáticos y Computación

Universidad Politécnica de Valencia

Abstract

The "*interpretation*" framework in Pattern Recognition (PR) arises in the many cases in which the more classical paradigm of "*classification*" is not properly applicable, generally because the number of classes is rather large, or simply because the concept of "*class*" does not hold. A very general way of representing the results of Interpretations of given objects or data is in terms of sentences of a "*Semantic Language*" in which the actions to be performed for each different object or datum are described. Interpretation can therefore be conveniently formalized through the concept of *Formal Transduction*, giving rise to the central PR problem of how to automatically learn a transducer from a training set of examples of the desired input-output behavior. This paper presents a formalization of the stated transducer learning problem, as well as an effective and efficient method for the inductive learning of an important class of transducers, namely, the class of *Subsequential Tranducers*. The capabilities of subsequential transductions are illustrated through a series of experiments which also show the high effectiveness of the proposed learning method to obtain very accurate and compact transducers for the corresponding tasks.

* Work partially supported by the Spanish CICYT under grant TIC-0448/89

I. INTRODUCTION

Syntactic Pattern Recognition (SPR) is currently seen as a very appealing approach to many Pattern Recognition (PR) problems for which the most traditional Decision-Theoretic or Statistical approaches fail to be effective [1-3]. One of the most interesting features of SPR consists of the ability to deal with highly structured (representations of) objects and to uncover the underlying structure by means of parsing. In fact, it is often claimed that such a distinctive feature is indeed what would enable SPR to go beyond the capabilities of other traditional approaches to PR. However, though parsing is perhaps the most fundamental and widely used tool of SPR, it is often used just to determine the (likelihood of) membership of the test objects to a set of classes, each of which being modeled by a different syntactic model. Any possible structural byproduct of the parsing process is therefore discarded, thus considerably wasting the high potential of SPR.

Nevertheless, membership determination is in fact all that is needed if the problem which is considered is just one of *classification*. But SPR can be seen as particularly useful for many other interesting PR problems for which the classification paradigm is not properly applicable – simply because the concept of "class" does not hold and/or because the number of underlying classes is very large or infinite. In order to properly deal with these problems, one should move from the traditional, rather comfortable *Classification* point of view to the most general paradigm of *Interpretation* (see e.g.,[4-5]).

Within this framework, a PR system is assumed to accept (convenient representations of) a certain class of objects as input and must produce, as output, adequate interpretations of these objects which are consistent with the a-priori knowledge (model) with which the system's Universe is embodied. Without loss of generality, interpretation results can be seen as *"Semantic Messages"* (SM) out of a (possibly infinite) set, which constitutes the "*Semantic Universe"* (SU) of the problem considered [4]. The complexity of the interpretation task is therefore determined by the complexity (size) of this universe. If the SU is "small" (i.e., a small number of different SM's is expected), interpretation can be viewed as the traditional process of classification, by letting the output consist of just a label specifying the obtained SM (class). On the other hand, if the SU is "large", a simple label can be inadequate to completely specify an SM and the output should now consist of some type of *structured information*. In this case, rather than speaking of (Pattern) "Recognition", the term "*Understanding*" is usually adopted instead. [4].

One convenient and general way of representing SM's is by means of sentences from a "Semantic Language" which specifies the actions to be carried out as a response to the interpretation of a given object. For instance, the sequence of operations to be performed by a robotized machine tool in order to correctly position and mechanize a hypothesized (interpreted) input piece, can be properly specified by a sentence of the command language of the machine.

Clearly, this calls for applying Formal Language techniques and then quite naturally puts Interpretation within the SPR framework. However, one should realize that the Interpretation paradigm no longer supports the traditional assumption of one (syntactical) model per class; rather, all the system's a-priori knowledge should now be represented, as a whole, into a single global model. In the theory of Formal Languages, such a representation can be properly accomplished through the concept of *Transduction*. A transducer is a formal device which inputs structural representations of objects (strings) from a given input set (language) and outputs strings from another (usually different) output language. Many interesting results on properties of rational or finite-state transducers are well known from the classical theory of Formal Languages (see e.g., [6]), and some examples of their application to SPR are often given in the textbooks of SPR [1-2]. However, for the transduction framework to be really fruitful, it is absolutely necessary to first solve the important problem of *how to learn or "train" these models from known instances of input-output strings*.

This problem, which is *central from a PR point of view*, has very rarely been dealt with in the literature. Apart from the particular treatment involved in Gold's study of the complexity of automata identification [7], only very limited and/or heuristic work seems to have been carried out so far. Perhaps one of the earliest works is that of Velenturf (1978) [8], in which a (non incremental) algorithm is proposed to obtain a (reduced) Mealy machine which is compatible with the given training data. If the source transducer is a (canonical) n-state Mealy machine and the training data contains all the input-output pairs of lengths smaller than $2n-1$, then the algorithm is shown to identify a machine that is exactly equivalent to the source transducer. Another related work is that of Luneau et al. (1984) [9], in which a heuristic incremental procedure is proposed to infer Moore machines. Apart from the "heuristic" nature of this method, an important drawback is that the obtained transducers are not guaranteed to be compatible with the training data. One still more recent reference to the transducer inference problem is given in [10] in relation with a context-free grammar inference technique. The main drawback of all these works, however, is that they are strictly restricted to the inference of *sequential* transductions which, as will be discused below, introduce strong limitations that are inadmissible in many cases of interest. Finally, Vidal et al. (1990) [11] proposed a (heuristic) *methodology* for the inference of transducers, which was reminiscent from the "Morphic Generator Grammatical Inference Methodology" (MGGI) [20] for the inference of general regular languages. Although this work was based on rather sound theoretical arguments and interesting empirical results were reported, no properties of identificability were given and the (heuristic) application of the methodology to each particular task required considerable skills on the part of the designer.

In this paper, we present a first step towards a formalization of the Transducer Learning (TL) problem from the point of view of its applicability within the Interpretation paradigm. We have adopted a framework somewhat similar to that of Grammatical Inference (GI) [12-15], though one should realize that many important differences between GI and TL exist. In particular, we have tried to develop our work in the same direction as that of the so-called "characterizable" methods of GI [14], [16-17]; that is, we have identified a (characterized) *class of transductions* and have developed a TL method that, under suitable conditions, and using only positive presentations of input-output pairs, can be proved to correctly identify such a class.

This class is the class of *Subsequential Transductions* which is a subclass of the most general *Rational or Finite-State Transductions* and properly contains the class of *Sequential Transductions* [6]. A Sequential Transduction is one that preserves the increasing length prefixes of input-output strings. While this can be considered as a rather "natural property" of transductions, there are many real-world situations in which such a strict sequentiality is clearly inadmissible. The class of *Subsequential Transductions* comes to make this restriction milder, therefore allowing applicability in quite a few interesting practical situations.

While a detailed theoretical study of the Subsequential Transduction Learning problem has been carried out [18], only those key formal issues needed for a convenient description of the proposed approach will be dealt with here. Greater attention, instead, will be devoted to present the results in such a way that their possible application to Pattern Recognition problems can be easily understood. For this purpose examples are used throughout the paper to clarify concepts and results. Also a number of experiments will be presented that illustrate both the usefulness of Subsequential Transduction and the ability of our learning methods to actually obtaining the corresponding transducers from positive training data.

It should be emphasized that both the methods and the experiments which are presented in the forthcoming sections are mainly aimed at *illustrating* the great potential of the proposed approach to the Interpretation problem in Pattern Recognition. Obviously, rather direct developments, such as stochastic and/or error correcting extensions, are required before these methods can be properly applied to the kind of noisy and distorted data usually encountered in real Pattern Recognition tasks. Nevertheless, we consider that the contents of this paper constitute in fact a required *first step* toward a new framework that would eventually launch Syntactic Pattern Recognition beyond the boundaries of the Classification paradigm.

II. MATHEMATICAL BACKGROUND AND NOTATION

Let X be a finite set or *alphabet*, and X^* the *free-monoid* over X. For any string $x \in X^*$, $|x|$ denotes the *length* of x and λ is the symbol for the string of length zero. For every $x,y \in X^*$, xy is the *concatenation* of x and y, with $|xy| = |x| + |y|$. If v is a string in X^*, then X^*v denotes the set of all strings of X^* that end with v; i.e., $X^*v=\{x \in X^*: uv= x, u \in X^*\}$. Similarly, $uX^*= \{x \in X^*: uv=x, v \in X^*\}$. On the other hand, $\forall x \in X^*$, $Pr(x)$ denotes the set of *prefixes* or *left factors* of x; that is $Pr(x) = \{u \in X^*: uv = x, v \in X^*\}$. Given $u,v \in X^*$, with $u \in Pr(v)$, the *right quotient* $u^{-1}v$ is the suffix of v that results after eliminating its prefix u; that is, $u^{-1}v=w \Leftrightarrow v=uw$. Given a set $L \subseteq X^*$, the *longest common prefix* of L is denoted as $lcp(L)$, where

$$lcp(L) = w \;\; iff \;\; w \in \bigcap_{x \in L} Pr(x) \wedge \forall w' (w' \in \bigcap_{x \in L} Pr(x) \Rightarrow |w'| \le |w| \;)$$

In general, a *transduction* from X^* to Y^* is a relation $t \subseteq (X^* \times Y^*)$. In what follows, only those transductions which are (partial) *functions* from X^* to Y^* will be considered. Also, in order to simplify the forthcoming presentation and without loss of generality, all the functions will be assumed with $t(\lambda)=\lambda$. For a partial function $t: X^* \rightarrow Y^*$, $Dom(t)$ denotes the subset of X^* where f is defined. A function t is *finite*, if so it is $Dom(t)$.

A *finite state* or *rational transducer* τ is defined as a 6-tuple $\tau = (Q,X,Y,q_0,Q_F,E)$, where Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, $Q_F \subseteq Q$ is a set of *Final States* and E is a finite subset of $(Q \times X^* \times Y^* \times Q)$ whose elements are called *edges*. Associated to an *edge* (q',x,y,q) is a *transition* $(q',x,q) \in (Q \times X^* \times Q)$ which corresponds to the conventional concept in automata.

A *Sequential Transducer* is a rational transducer in which $E \subset (Q \times X \times Y^* \times Q)$, $Q_F=Q$ and $(q,a,u,r),(q,a,v,s) \in E \Rightarrow (u=v \wedge r=s)$ (*determinism condition*) [6]. Correspondingly, Q_F can be omitted and a Sequential Transducer is completely specified as a 5-tuple $\tau = (Q,X,Y,q_0,E)$. Sequential Transducers are also called "*Generalized Sequential Machines*" (*GSM*), from which *Mealy* and *Moore Machines* are restricted instances.

A *path* in a sequential transducer, τ, is a sequence of edges $\pi = (q_0,x_1,y_1,q_1)(q_1,x_2,y_2,q_2) \ldots (q_{n-1},x_n,y_n,q_n)$, $q_i \in Q$, $x_i \in X, y_i \in Y^*, 1 \le i \le n$. Whenever the states which are involved in a path are not of particular concern, a path will be written as $\pi = (q_0,\, x_1x_2 \ldots x_n,\, y_1y_2 \ldots y_n,\, q_n)$. Let Π_τ be the set of all possible paths over τ. The transduction realized by a sequential transducer τ is the partial function $t : X^* \rightarrow Y^*$ defined as $t(x) = y$ *iff* $\exists (q_0,x,y,q) \in \Pi_\tau$.

Sequential Transduction has the property of preserving left factors; that is, $t(\lambda) = \lambda$ and, if $t(uv)$ exists, then $t(uv) \in t(u)Y^*$. Such a seemingly "natural" property (a restriction, in fact), however, can be quite inadmissible in many cases of interest. For instance, there are finite functions that are not sequential. In order to make this restriction milder the following extension is introduced [6].

A *subsequential transducer* is defined as a 6-tuple $\tau=(Q,X,Y,q_0,E,\sigma)$, where $\tau'=(Q,X,Y,q_0,E)$ is a sequential transducer and σ: $Q \rightarrow Y^*$ is a (partial) function that assigns output strings to the states of τ '. The partial function t: $X^* \rightarrow Y^*$ that is realized by τ is defined as $t(x) = y\sigma(q)$ *iff* $\sigma(q)$ is defined and $(q_0,x,y,q) \in \Pi_\tau$. The class of subsequential functions properly contains the class of sequential functions. In the same way as for conventional finite transducers, a Subsequential Transducer can be represented as a graph with the only difference that nodes are labelled as pairs $(q,\sigma(q))$ where $q \in Q$.

By using an additional input symbol "#", not in X, to mark the end of the input strings, any subsequential transduction t: $X^* \rightarrow Y^*$ can be obtained as a restriction to $X^*\#$ of a sequential transduction t': $(X \cup \{\#\})^* \rightarrow Y^*$, so that, $\forall x \in X^*$ $t(x) = t'(x\#)$ [6]: If $\tau=(Q,X,Y,q_0,E,\sigma)$ is a subsequential transducer that realizes t, then t' can be realized by the sequential transducer $\tau' = (Q \cup \{q_F\}, X \cup \{\#\}, Y, q_0, E')$, where $q_F \notin Q$ and $E' = E \cup \{(q,\#,\sigma(q),q_F)$: $q \in Q\}$. Note that, any given Subsequential Transducer admits several different Sequential Representations. For instance, an alternative to the above sequential representation of τ can be obtained by eliminating q_F and replacing every $(q,\#,\sigma(q),q_F)$ for $(q,\#,\sigma(q),q_0)$. In what follows the term "Subsequential Transducer" will be used, at convenience, either to denote a Subsequential Transducer as defined by (Q,X,Y,q_0,E,σ), or a sequential representation thereof. Fig. 1 shows an example of subsequential transducer and a sequential representation.

Example 1: The function t: $\{a\}^* \rightarrow \{b,c\}^*$ defined by

$$t(a^n) = \begin{cases} \lambda & n = 0 \\ b^{n+1} & n \text{ odd} \\ b^n c & n \text{ even} \end{cases}$$

is subsequential, but not sequential, since it does not preserve left factors. The function t is realized by the subsequential transducer $\tau = (Q,X,Y,q_0,E, \sigma)$ in which $Q = \{q_0,q_1,q_2\}$, $X = \{a\}$, $Y = \{b,c\}$, $E = \{(q_0,a,b^2,q_1), (q_1,a,\lambda,q_0), (q_2,a,b^2,q_1)\}$, and $\sigma(q_0) = \lambda$, $\sigma(q_1) = \lambda$, $\sigma(q_2) = c$.

The graph of τ and that of a sequential representation of it are shown in Fig.1.

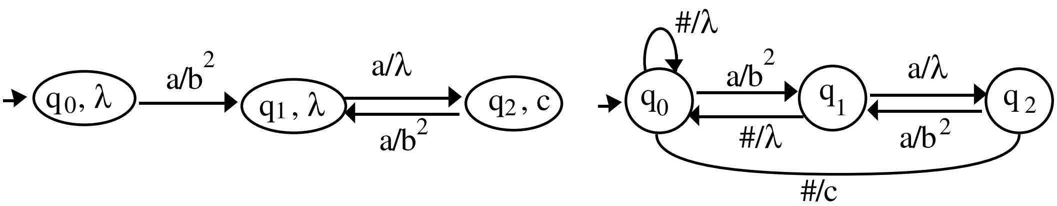


Fig. 1. Subsequential transducer of Example 1 and a Sequential representation.

Following, e.g. [12] and [14], an appropriate framework for the Transducer Learning Problem can be established as follows: Let $f : X^* \to Y^*$ be a partial recursive function, a Transducer Learning Algorithm $\mathcal{A}$ is said to *identify f in the limit* if, for any (positive) presentation of the input-output pairs (graph) of f, $\mathcal{A}$ converges to a transducer τ that realizes a function $g: X^* \to Y^*$ such that $\forall x \in Dom(f),\ g(x) = f(x)$.

III. ONWARD SUBSEQUENTIAL TRANSDUCERS

The transducer learning algorithm to be presented in the next section requires some additional new concepts to be introduced. An *Onward Subsequential Transducer* (*OST*) is one in which the output strings are assigned to the edges in such a way that they are as "close" to the initial state as they can be. Formally, a subsequential transducer $\tau' = (Q,X,Y,q_0,E,\sigma)$ is an *OST* if

$$\forall p \in Q\text{-}\{q_0\} \quad lcp\ (\{y \in Y^* \mid (p,a,y,q) \in E\} \cup \{\sigma(p)\}) = \lambda.$$

Much in the same way as it happens in Regular Languages, any Subsequential Transduction admits a characterization in terms of a *Canonical Transducer in Onward form*; in other words, for any Subsequential Transduction there exist an *OST* which has a minimum number of states and which is unique up to isomorphism. This transducer is obtained through the following construction:

Let $t: X^* \to Y^*$ be a partial function and let $x \in Pr(Dom(t))$. The *set of tails* of x in t, $T_t(x)$, is defined as:

$$T_t(x) = \{(y,v) \mid t(xy) = uv,\ u = lcp\ (t(xX^*))\}$$

with $T_t(x) = \emptyset$ if $x \notin Pr(Dom(t))$

It can be seen [18] that, if t is subsequential, the number of different sets of tails in t is finite. Therefore, these sets can be used to build a subsequential transducer $\tau(t) = (Q,X,Y,q_0,E,\sigma)$ as follows:

$Q = \{T_t(x) \mid x \in Pr(Dom(t))\}$; $q_0 = T_t(\lambda)$;

$E = \{(T_t(x), a, lcp(t(xX^*))^{-1}lcp(t(xaX^*)), T_t(xa)) \mid T_t(x), T_t(xa) \in Q\}$

$$\sigma(T_t(x)) = \begin{cases} lcp(t(xX^*))^{-1}t(x) & \text{if } x \in Dom(t) \\ \varnothing & \text{Otherwise} \end{cases}$$

It can be easily seen [18] that such a transducer is Onward and that any other *OST* for *t* either has a greater number of states or is isomorphic with $\tau(t)$.

Example 2: Let t: $\{a\}^* \rightarrow \{b,c\}^*$ be a function defined as in *example 1*. The different sets of tails are:

$T_t(\lambda) = \{(\lambda,\lambda)\} \cup \{(a^{2n+1},b^{2n+2}) \mid n\geq 0\} \cup \{(a^{2n},b^{2n}c) \mid n\geq 1\}$
$T_t(a) = \{(\lambda,\lambda)\} \cup \{(a^{2n+1},b^{2n}c) \mid n\geq 0\} \cup \{(a^{2n},b^{2n}) \mid n\geq 1\}$
$T_t(a^2) = \{(\lambda,c)\} \cup \{(a^{2n+1},b^{2n+2}) \mid n\geq 0\} \cup \{(a^{2n},b^{2n}c) \mid n\geq 1\}$
$T_t(a^3) = T_t(a)$

and the Canonical Onward Subsequential Transducer that realitzes *t,* shown in Fig.2, is isomorphic with the corresponding transducer shown in Fig.1.

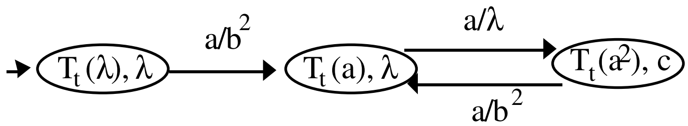


Fig. 2. Canonical Transducer for the subsequential function of Example 2.

The transduction learning algorithm to be presented in the next section requires a finite sample of input-output pairs $T \subset (X^* \times Y^*)$, which is assumed to be non-ambiguous or *single-valued*; i.e., $(x,y), (x,y') \in T \Rightarrow y = y'$. Such a sample can be properly represented by a *"Tree Subsequential Transducer"* (*TST*) $\tau = (Q,X,Y,q_0,E,\sigma)$, with

$Q = \cup_{(u,v)\in T} Pr(u)$, $q_0 = \lambda$,
$E = \{(w,a,\lambda,wa) \mid w,wa \in Q\}$,
$\sigma(u) = v$ if $(u,v)\in T$ with $\sigma(u)$ being undefined otherwise.

A possible sequential representation, τ', of this *TST* can be obtained by eliminating σ of τ and extending its set of states and edges as follows:

$Q' = Q \cup \{u\# \mid (u,v)\in T\}$; $E' = E \cup \{(u,\#,v,u\#) \mid (u,v)\in T\}$

Whenever such a sequential representation is used, the training pairs (u,v) will be written as $(u\#,v)$.

Given T, an *Onward Tree Subsequential Transducer* (*OTST*) representing T, *OTST(T)*, can be obtained by building an *OST* equivalent to the *TST* of T. This can be accomplished through the following construction (directly given for a sequentially represented *TST* $\tau' = (Q',X\cup\{\#\},Y, q_0,E')$).

$\forall q\in Q'$, **if** $w = lcp\{v\in Y^* \mid (q,a,v,r)\in E'\} \wedge w \neq \lambda$ **then**

1: substitute every outgoing edge of q : (q,a,wz,r) for (q,a,z,r)

2: substitute every ingoing edge of q : (p,b,y,q) for (p,b,yw,q)

Example 3: Let $T= \{(\#,\lambda), (a\#,bb), (aa\#,bbc), (aaa\#,bbbb), (aaaa\#,bbbbc)\}$. Sequential representations of the *TST* of T and *OTST*(T) are as shown in Fig.3.

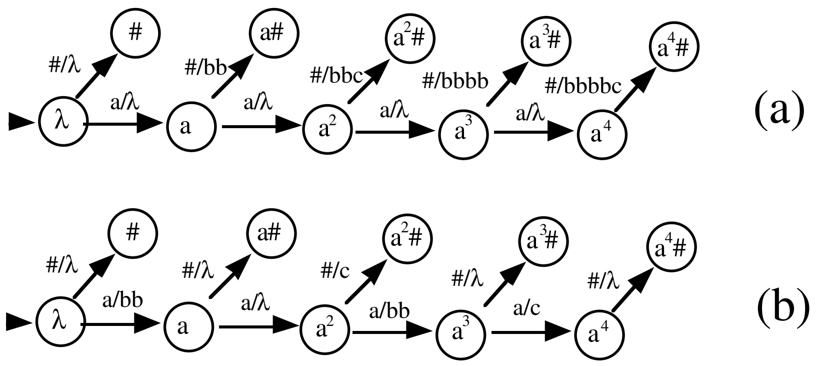


Fig. 3. Tree Subsequential Transducer (*TST*) (a), and Onward Tree Subsequential Transducer *OTST*(T) (b) representing the sample $T= \{(\#,\lambda), (a\#,bb), (aa\#,bbc), (aaa\#,bbbb), (aaaa\#,bbbbc)\}$

IV. THE TRANSDUCER LEARNING ALGORITHM

Let $T \subset (X^*\# \times Y^*)$ be a finite single-valued training set. The proposed algorithm consists of a non incremental procedure that starts building the *OTST(T)*, $\tau = (Q,X\cup\{\#\},Y,q_0,E)$, and then proceeds by orderly trying the merge of states of τ. This state merging may result in transducers which are not subsequential as defined in Section II. In particular they may often violate the condition of *determinism*: $(q,a,u,r), (q,a,v,s)\in E \Rightarrow (u=v \wedge r=s)$. In these cases, we may still insist to preserve the sequential nature of the resulting transducers. For this to be possible, some output (sub)strings associated to the edges of τ often need to be *"pushed back"* towards the leaves of τ in a process which, to a limited extent, reverses the forward (sub)string displacements carried out to transform the initial *TST* associated to *T* into τ. The test as to whether a transducer τ is subsequential, as defined in Sect. II, is assumed to be supplied by a (hypothetical) procedure "*subseq*" that takes a transducer and outputs *true* if such a property holds, *or false* otherwise. One should realize, however, that when embedded in the algorithm to be described below, such a test can be carried out at no cost as a byproduct of the remaining operations of the core of the algorithm.

The merging process mentioned above requires the states of τ to be successively taken into account in a *lexicographic order* of the names given to these states through the *TST* construction (Sect.III) [18]. Let "<" be such an order on Q, with *first* (τ) and *last* (τ) being the first and last states with respect to "<", and $\forall q \in Q - \{last(\tau)\}$ let *next* (τ,q) denote the state which immediately follows q in the order "<". The merging of any two states $q',q \in Q$, with $q'<q$, results in a new transducer $\tau' = (Q',X \cup \{\#\},Y,q_0,E')$ in which the state q no longer exists in Q' and all the outgoing edges of q in E are assigned to q' in E'. Let τ' = *merge* (τ,q',q) represent this merging operation which is quite similar to the merging of states in finite-state automata [16].

The process of "pushing back" output substrings in a transducer $\tau = (Q,X \cup \{\#\},Y,q_0,E)$ requires some more explanation. Let $q \in Q$ be a state of τ and $(q',a,w,q) \in E$ (one of) its ingoing edge(s). If $u \in Y^*$ is a prefix of w, then the suffix $v=u^{-1}w$ can be *"pushed back"* to behind q and distributed throughout all the outgoing edges of q to produce another transducer $\tau' = (Q,X \cup \{\#\},Y,q_0,E')$ as follows (see an illustration in Fig.4):

$$E' = (E - \{(q',a,uv,q)\}) \cup \{(q',a,u,q)\} \cup \{(q,b,vz,r) : (q,b,z,r) \in E\}$$

Let τ' = *push_back* $(\tau,v,(q',a,uv,q))$ denote this operation, for which the following *equivalence property* can be easily established [18]:

If $v = \lambda$ or if (q',a,w,q) is the *only* edge that enters q, then transductions are preserved and the transducer τ' is *equivalent* to τ.

$\tau' = push\_back(\tau, 01, (q',1,001,q))$

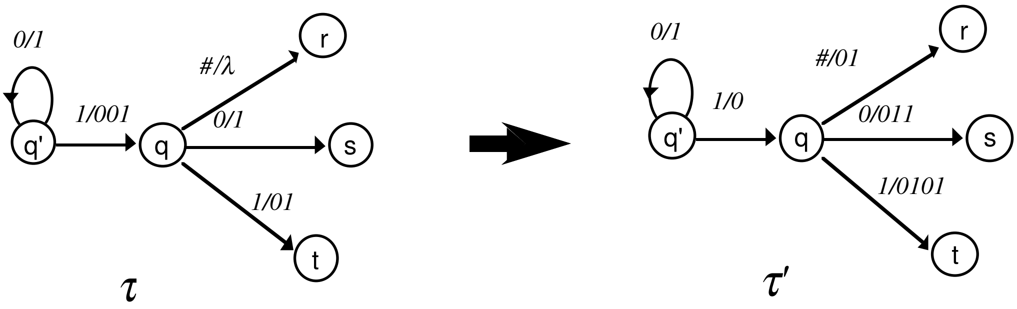


Fig.4. Example of a push-back operation

The algorithm that performs the above outlined procedures is called *"Onward Subsequential Transducer Inference Algorithm* (*OSTIA*)*"* and is formally presented in Fig. 5 below.

Algorithm *OSTIA* //Onward Subsequential Transducer Inference Algorithm//

```
INPUT: Single-valued finite set of input-output pairs T⊂ (X*# x Y*)
OUTPUT: Onward Subsequential Transducer τ consistent with T
τ := OTST(T)
q:= first(τ )
while q < last(τ ) do
    q:= next(τ,q);
    p:= first(τ );
    while p < q do
            τ':= τ
            τ := merge(τ,p,q)
            while ¬subseq(τ ) do
            let (r,a,v,s), (r,a,w,t) be two edges of τ
                that violate the subseq condition, with s < t
                if ((v≠w) and (a=#)) or (s<q and v∉ Pr(w)) then exit while
                u:= lcp(v,w)
                τ := push_back(τ, u^-1v,(r,a,v,s))
                τ := push_back(τ, u^-1w,(r,a,w,t))
                τ := merge(τ,s,t)
            end while //¬subseq(τ )//
            if ¬subseq(τ ) then τ := τ' else exit while
            p:= next(τ ,p)
    end while // p < q //
    if ¬subseq(τ ) then τ := τ'
end while // q < last(τ)//
end //OSTIA//
```

Fig. 5. The Transducer Inference Algorithm

The computational cost of this algorithm can easily shown to be polynomial. Let $n = \sum_{(x,y)\in T}|x|$, $m = max_{(x,y)\in T}|y|$, and $k = |X|$ be parameters measuring the size of the input training set *T*. The initial *OTST(T)* has a number of states that is linear with *n* and, therefore, its construction requires computing time that grows linearly with $n \cdot m$.

The two outer while loops of *OSTIA* entail, in principle, a total number of iterations that is quadratic with the number of states of the initial transducer $\tau = OTST(T)$ – hence with *n*. This could be considered exceedingly pessimistic, since many states could rapidly be eliminated from τ by successive *merge* operations. Nevertheless, a *worst-case* can be identified in which, after having carried out all possible *merge* operations of the inner **while** loop and having exhausted all the remaining states, this loop is **exited** without success (**if** condition). In this case, the inner loop would perform at most $O(n)$ iterations, resulting in an $O(n^3)$ total number of executions of the core of the algorithm. Except for *lcp* and *push_back*, all the other operations involved in the *OSTIA* can be easily implemented to run with *unit cost*; that is, computing time independent of *n, m* and *k*. The *lcp* operation can also be easily implemented to work in $O(m)$ time, while *push_back* may require $O(k)$ steps if string indexes rather than actual (sub)strings are used to represent the input-output substrings of the edges of τ. This amounts to a total cost in $O(m+k)$ for the core operations and a total computing time which can be bounded as $O(n^3(m+k))$. Obviously, in many cases of interest this bound is in fact pessimistic. Real empirical timing results will be shown in Section VI.

The correctness of this algorithm is also clear. The proof starts by realizing that the property $q<t$ results in being a loop-invariant of the algorithm. Given the tree structure of the initial *OTST*, along with the order in which states are considered for merging, this invariant implies that only *one* edge can enter the state *t*. Correspondingly, from the above *equivalence property*, the push-back operation involving *t* is always transduction-preserving. On the other hand, the other push-back operation, which involves the state s, can only be executed in two very precise situations: 1) $s<q$, in which case $v \in Pr(w)$ and thus $u^{-1}v = \lambda$; or 2) $q<s$, which implies that only one edge can enter *s*. Therefore, the *equivalence property* guarantees that, in both cases, the operation also results in being transduction-preserving. Since the initial *OTST* is obviously consistent with *T*, proceeding by induction we see that every iteration yields a transducer τ that is *subsequential* and *consistent* with *T*.

Similar arguments support the assertion that the resulting transducer is *Onward*. The initial *OTST* is obviously *Onward*, a property which is clearly preserved by the successive executions of the operation "$merge(\tau,p,q)$". However, the two *push_back* operations (which aim at trying to recover the *subseq* condition after its possible violation by the merge of two states) may result in loosing the *Onward* property. But, in this case the subsequent "$merge(\tau,s,t)$" operation directly restores the property yielding again an *Onward* transducer.

Following this discussion, it can be seen that the *OSTIA* produces a subsequential transducer which is a state-merging-based generalization of the initial Subsequential Tree Transducer representing the given sample *T*. This generalization, however, tends to progressively shrink as *T* gets larger and includes more training pairs that are "representative" of the unknown (subsequential) transduction from which *T* is assumed to have been drawn. As it will be discussed in the next section, this leads to the the important result that, *using the OSTIA, the class of subsequential functions can be identified in the limit*.

Using the training pairs of Example 3 and the corresponding *OTST* in Fig.3, Fig.6 illustrates some key steps of the *OSTIA*. Starting from the *OTST*(T) of Fig. 3(b), the algorithm attempts the merging of states λ and #. Fig. 6(a) shows the obtained transducer. Then, *OSTIA* goes on trying the merging of states a with λ (Fig.6b). This transducer has the two edges (λ,a,bb,λ) and (λ,a,λ,aa) which violate the *subseq* condition. Given that $\lambda < a$ and, $bb \notin Pr(\lambda)$ the innermost loop of *OSTIA* is exited and states λ and $a\#$ are merged in the previous transducer (Fig.6a), resulting in the transducer of Fig.6c. The next step tries the merging of the states λ and aa. The existence of edges $(\lambda,\#,\lambda,\lambda)$ and $(\lambda,\#,c,aa\#)$ causes the exit of the innermost loop of *OSTIA*. After merging the states λ with $aa\#$ and given the imposibility of merging λ with aaa, Fig.6e shows the result of the merging of the states aaa with a. The edges (a,a,λ,aa) and $(a,a,c,aaaa)$ of the resulting transducer violate the *subseq* condition; nevertheless $lcp(\{\lambda,c\}) = \lambda$, and the operations $push\_back(\tau,c,(a,a,\lambda,aa))$ and $push\_back(\tau,c,(a,a,c,aaaa))$, produce the transducer shown in Fig.6.f. The rest of the work carried out by the innermost loop of *OSTIA* finally leads to the subsequential transducer of Fig. 6(g) that is the inferred result from the training sample of Example 3.

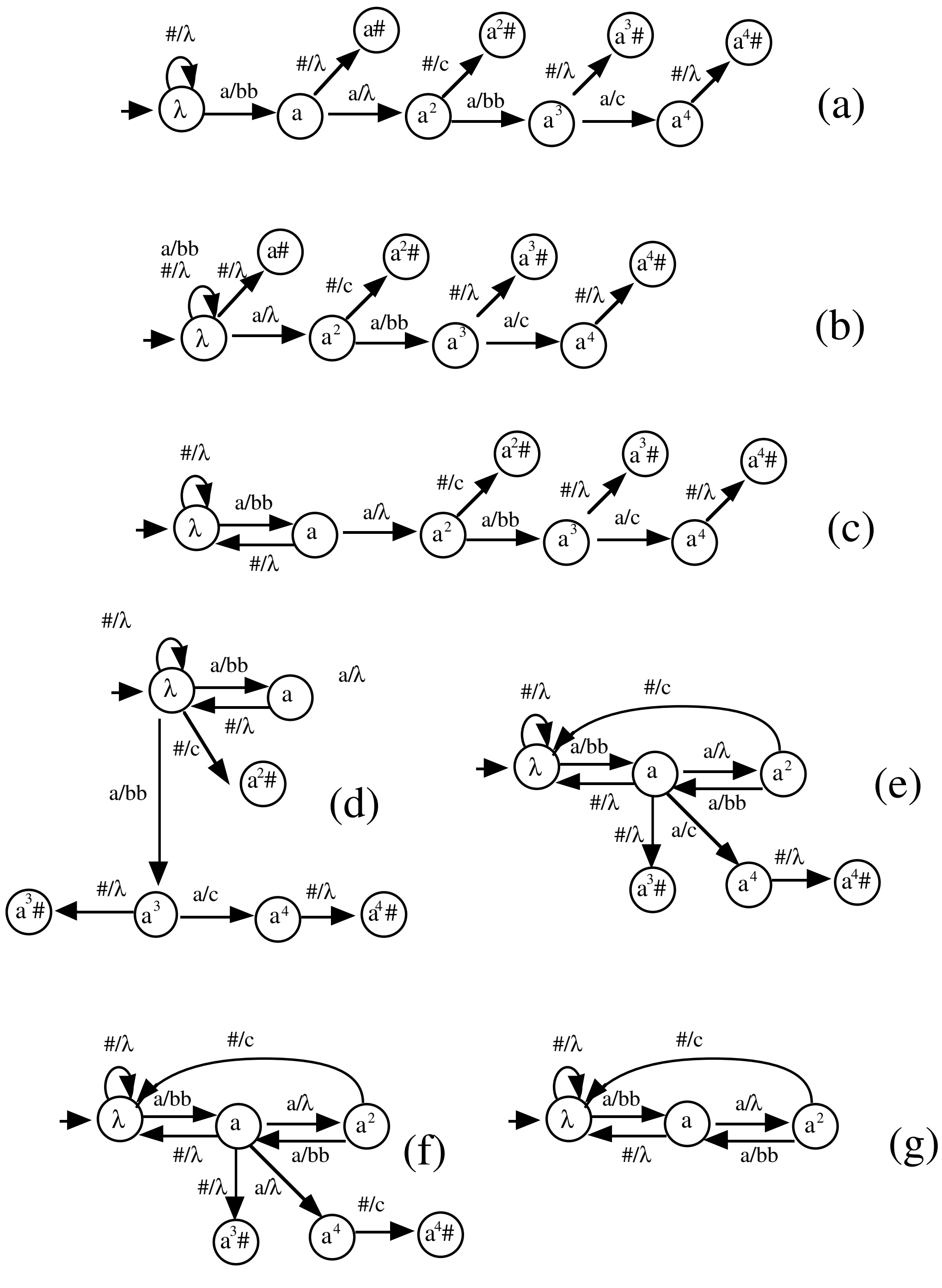


Fig. 6. Some key steps of the *OSTIA* as applied to the the *OTST* in Fig. 3.b

V. IDENTIFICATION OF SUBSEQUENTIAL TRANSDUCTIONS IN THE LIMIT.

In this section we will shown how the *OSTIA* identifies the class of the Subsequential functions from positive data (input-output pairs) in the limit. To this end, we start defining a *finite representative sample T* of a total subsequential function *t*. We will show that the transducer *OTST(T)* contains a subset of states, referred to as *kernel,* which induces a subtree in *OTST(T)* that contains representations of all the edges of the canonical *OST* of *t*, $\tau(t)$, and only these edges. This result will allow us to establish that, if a representative sample is contained in the input to *OSTIA,* then the output will be the canonical *OST* of the function *t*. Given that any positive presentation of *t* will eventually include a representative sample, the property of the *identification in the limit* will result. In what follows only the main arguments of the proofs will be sketched. A full account of the complete results can be found in [18].

Definition 1. Let $t\colon X^* \rightarrow Y^*$ be a subsequential function. The string *w* is a *Short Prefix* of *t* iff $w \in Pr(Dom(t))$ and $\forall v \in X^*\ (T_t(w) = T_t(v) \Rightarrow v \geq w)$. The *set of Short Prefixes* of *t*, $SP(t)$, includes a short prefix for every different tail in $\{T_t(w) \mid w \in Pr(Dom(t))\}$.

Definition 2. Let $t\colon X^* \rightarrow Y^*$ be a subsequential function. The *kernel* of *t* is the set:
$K(t) = (SP(t)X \cap Pr(Dom(t))) \cup \{\lambda\}$

In other words, $K(t)$ contains the empty string along, whith the extensions of all of the strings in $SP(t)$ whith all the symbols in *X*, as long as these extensions are prefixes of the strings in the domain of *t*.

Definition 3. Given a *total* subsequential function $t\colon X^* \rightarrow Y^*$, a *sample T* of *t* is said to be *representative* of *t* iff:

1) $\forall u \in K(t)\ \exists (uv,w') \in T \mid v \in X^*,\ w \in Y^*$
2) $\forall u \in SP(t)\ \forall v \in K(t)\quad (T_t(u) \neq T_t(v) \Rightarrow \exists\, (uw,u'w'),(vw,v'w'') \in T \mid$
   $(w,w') \in T_t(u),\ (w,w'') \in T_t(v),\ w' \neq w'')$
3) $\forall u \in K(t)$
   $\exists (uv,u'v'),(uw,u'w') \in T \mid\ (v,v'),(w,w') \in T_t(u),\ lcp(\{v',w'\}) = \lambda$

In the forthcoming theorems, condition 1) will guarantee that all the states and *transitions* of the canonical *OST* $\tau(t)$ are represented in *OTST(T)*. Condition 2) distinguishes those states of *OTST(T)* for which the corresponding states of $\tau(t)$ are different. Finally, condition 3) will make it possible that, for every state of *OTST(T)* which is reachable by a string in the *kernel* of *t*, the edges will be identical to the corresponding edges in $\tau(t)$.

In the forthcoming theorems the following notation will be used:

$\tau_T = (Q_T, X, Y, \lambda, E_T, \sigma_T)$ and $\tau(t) = (Q_t, X, Y, q_0, E_t, \sigma_t)$ will denote the $OTST(T)$ and the canonical *OST* of *t*, respectively; and $\tau_i = (Q_i, X \cup \{\#\}, Y, \lambda, E_i)$ will denote *OST* obtained after the *i*-th iteration of the outermost *while* loop of the *OSTIA* with input $T \subset (X^*\# \times Y^*)$.

Theorem 1. Let *T* be a representative sample of $t: X^* \to Y^*$. A function $\varphi: Q_T \to Q_t$ exists such that $\forall u, ua \in K(t)\ ((u,a,v,ua) \in E_T \Leftrightarrow (\varphi(u),a,v,\varphi(ua)) \in E_t)$ and $\forall u \in K(t)\ \sigma_T(u) = \sigma_t(\varphi(u))$.

Proof: If $u \in K(t)$ and $t': X^* \to Y^*$ is the finite function realized by τ_T, the third condition for a representative sample (def.3 above) allows us to establish that $lcp(t'(uX^*)) = lcp(t(uX^*))$. Then, by defining $\varphi: Q_T \to Q_t$ as $\varphi(u) = T_t(u),\ u \in Q_T$ and using the first condition of representative sample (def.3) the theorem follows.

□

Theorem 2. Let $q \in Q_i$ be the state of τ_i considered in the $i+1$ iteration, of the oputermost *while* of *OSTIA* let $Q'_i = \{p \in Q_i \mid p < q\}$ and let $\tau'_i = (Q_i, X \cup \{\#\}, Y, \lambda, E'_i)$be the subtransducer which is induced by Q'_i. If *T* is a representative sample of the total subsequential function $t: X^* \to Y^*$, then τ'_i is isomorphic with a (sequential representation of a) subtransducer of $\tau(t)$.

Proof. Let $\varphi: Q'_i \to Q_t$ be defined as $\varphi(u) = T_t(u),\ u \in Q'_i$. By induction in the number of iterations, and given that the second condition of the definition of representative sample guarantees us that *OSTIA* will perform the merge of two states only if both have the same image with φ, we can establish:

a) $(u,a,v,ua) \in E'_i \Rightarrow (\varphi(u),a,v,\varphi(ua)) \in E_t$
b) $(u,\#,v,u\#) \in E'_i \Rightarrow \sigma_t(\varphi(u)) = v$

□

Corolary 1. If the input to *OSTIA* includes a representative sample of the total function *t*, then the output is an Onward Subsequential Transducer which is isomorphic with $\tau(t)$.

Theorem 3. Using the Onward Subsequential Transducer Inference Algorithm (*OSTIA*) the class of subsequential functions can be identified in the limit with positive presentation.

Proof. If $t: X^* \to Y^*$ is a total function, then any positive presentation of *t* will include a representative sample. Hence, *OSTIA* will converge to a subsequential transducer that is isomorphic with $\tau(t)$. If *t* is not total, the second condition of the definition of representative sample in not guaranteed. Correspondingly, *OSTIA* may merge states which are associated to states of $\tau(t)$ that are different to each-otter. But, in this case, no finite information sequence can in fact distinguish such states. Thus, *OSTIA* would converge to a subsequential transducer τ that realizes a function t' such that $\forall x \in Dom(t)\ t'(x) = t(x)$, which is precisely the condition assesing the *identification of t in the limit* (Sect. II).

□

VI. EXPERIMENTS

Theoretical properties assessing the adequate behavior in the limit (convergence) of subsequential transduction identification using the *OSTIA* have been discused in the previous section. However, from a Pattern Recognition viewpoint, results concerning finite data behavior seem to be of greater interest. In order to obtain such results, and at the same time illustrate the capabilities of Subsequential Transduction and their *OSTIA* learning, a number of experiments have been carried out. Some of these experiments and their results will be presented in this section.

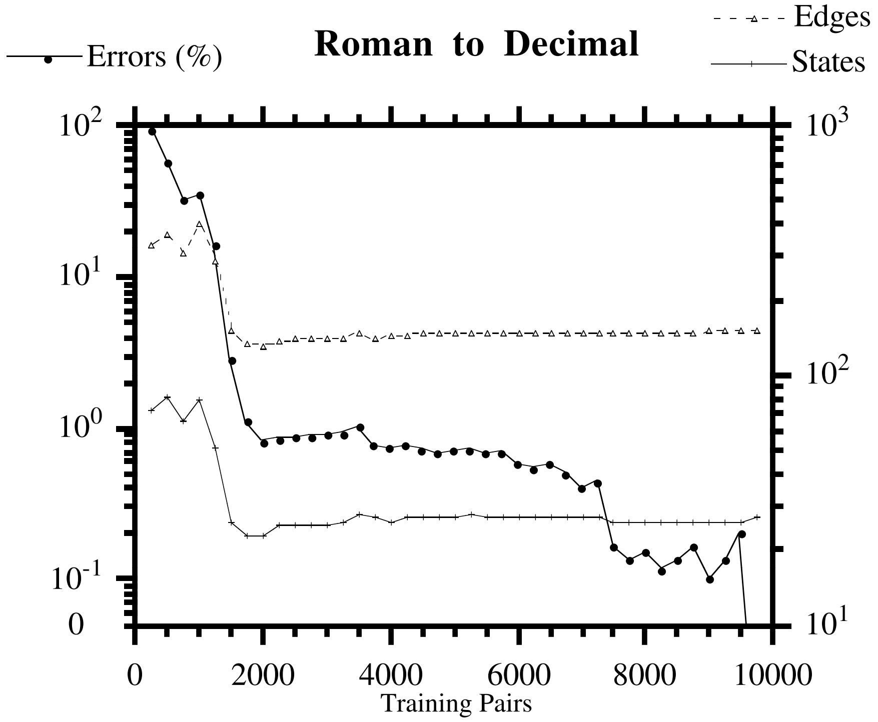
For the first group of experiments the task of translating *Roman Numbers* into their *decimal representation* has been chosen. It should be taken into account that such a task can by no means be supported by pure sequential transduction, since many input-output pairs exist in which identical input prefixes must lead to different output substrings. As will be seen below, subsequential transduction easily overcomes this difficulty, allowing the task to be properly accomplished.

For the first experiment, a series of *50* increasing-size random *training-sets*, each including the previous ones, were drawn from a uniform distribution in the range of (Roman and Decimal) numbers from *1* to 10^4-1. The random procedure was prevented from generating repeated samples and the *test-set* consisted of all the Roman-Numbers from *1* to 10^4-1 that where not included in the training set. Some selected training pairs are shown in fig 7.

(III#, 3)	(LXXIV#,74)
(IV#, 4)	(XCV#, 95)
(VI#, 6)	(CXV#, 115)
(IX#, 9)	(CDII#, 402)
(XI#, 11)	(CMLXXXIX#,989)
(XIX#, 19)	(MCXI#, 1111)
(XLII#, 42)	(MMMMMMMMDCCCLXXXVIII#, 8888)

Fig.7. A sample of selected training pairs for the Roman to Decimal translation task

Every training set was submitted, in turn, to the *OSTIA* and each resulting Subsequential Transducer was used to translate all the Roman-Number strings of the test set. The results of this experiment appear in fig. 8. The items shown are: (a) the test-set error rates, (b) the sizes of the learnt transducers and (c) the computing time required for each execution of the *OSTIA*, on a ~*25* mips. conventional RISC Computer (HP 9000/35).



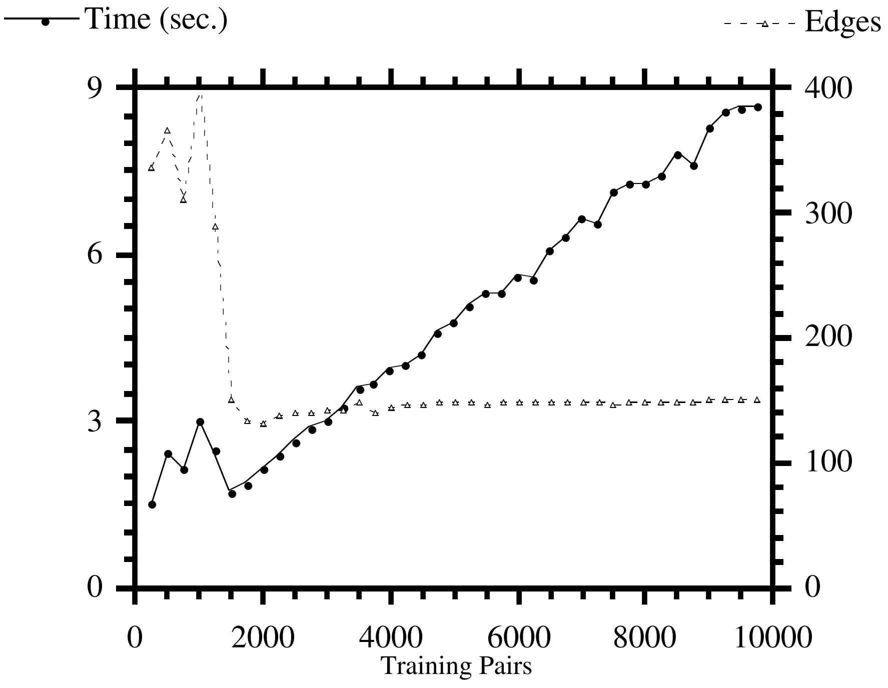


Fig. 8. Behavior of the OSTIA for the Roman to Decimal translation task

It is worth noting that, with small training-sets, the inferred transducers tend to be rather large and error-prone, while both sizes and errors reduce dramatically as enough source structure is made available through the training data.

It should be taken into account that these results were obtained without concern as to how “relevant” the (random) training data were for the considered learning task. In order to investigate how small the training set could become if appropriately selected, a further experiment was carried out involving the following *greedy* procedure. First, starting with a transducer learnt from the first training pair, “(I,1)”, the test data was submitted in numerical order to transduction until one Roman-Number string appeared which was incorrectly translated. Then this number, along with its correct decimal transduction, was used as a training pair. The first phase of this procedure stopped when all the strings were correctly translated. In the second phase, the Roman-number (training) pairs that were selected in the first phase were considered, in turn, to see whether they could be discarded without change in the inferred transducer. After this second phase, a training set of *210* relevant Roman-Decimal pairs was identified which led to the correct identification of an exact Subsequential Transducer that was identical to the one obtained in the previous experiment with a random training set of *8800* pairs. A part of this transducer is shown in Fig. 9.

A second group of experiments was concerned with the task of translating *English numbers* in the $0...10^6\text{-}1$ range into their conventional *decimal representation*. Although this task is rather better suited to sequential transduction than that of the Roman-Decimal task, there are also in this case situations which prevent pure sequentiality; e.g.: "ninety" $\rightarrow$*"90"*, "nineteen" $\rightarrow$*"19"* etc. The experimental framework was similar to that of the previous experiments, except that in this case, the numbers were randomly drawn from a non-uniform distribution in which the lengths of the (decimal) strings were (approximately) equiprobable. In this case, *60* training-sets of sizes increasing from *200* up to *12000* were used and the test set consisted of the whole range of 10^6 different English-number strings. In order to approach the conditions of phonemic recognition, these strings were written with no separating blanks between the different words, what considerably increased the difficulty of the task.

Fig.10 shows some training-pairs that have been selected from the training material. The results appear in Fig. 11 which shows the same items as Fig. 8 above, with a rather similar behavior. Also, a greedy procedure similar to the one described above was carried out, ending with a representative set of 260 input-output pairs.

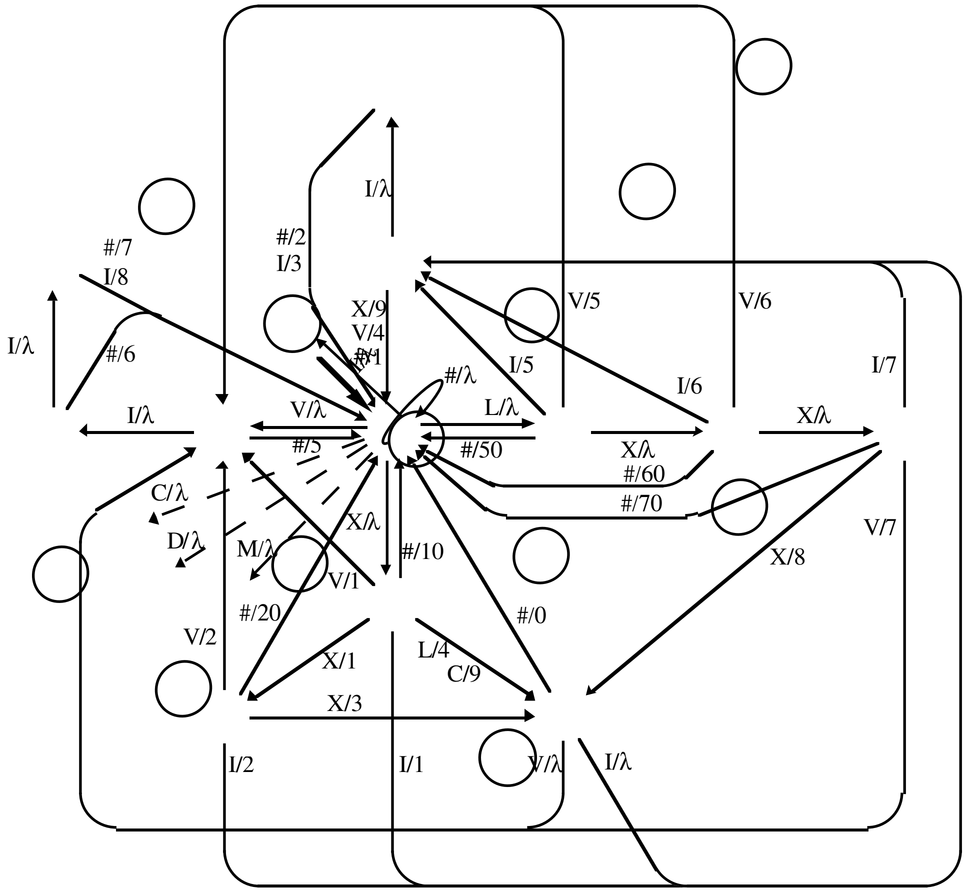


Fig.9. A part of the Subsequential Transducer that was learnt for the Roman to Decimal translation task. The part shown correctly translates the Roman numbers from *1* to *99*. The dashed lines correspond to edges to the remaining parts (not shown) of the whole transducer.

(nine#, 9)

(ninety#,90)

(nineteen#, 19)

(ninehundred#, 900)

(nineteenthousandandeight#, 19008)

(ninehundredthousandandeighteen#, 900018)

(twohundredandthirteenthousandandtwelve#, 213012)

(fourhundredandonethousandeighthundred#, 401800)

(eighthundredandeightthousandeighthundredandseven#, 808807)

(sixhundredandseventeenthousandsixhundredandsixteen#, 617616)

(threehundredandninetyfivethousandonehundredandsixtythree#, 395163)

(sevenhundredandeighteenthousandsevenhundredandseventeen#, 718717)

Fig.10. A sample of selected training pairs for the English to Decimal translation task

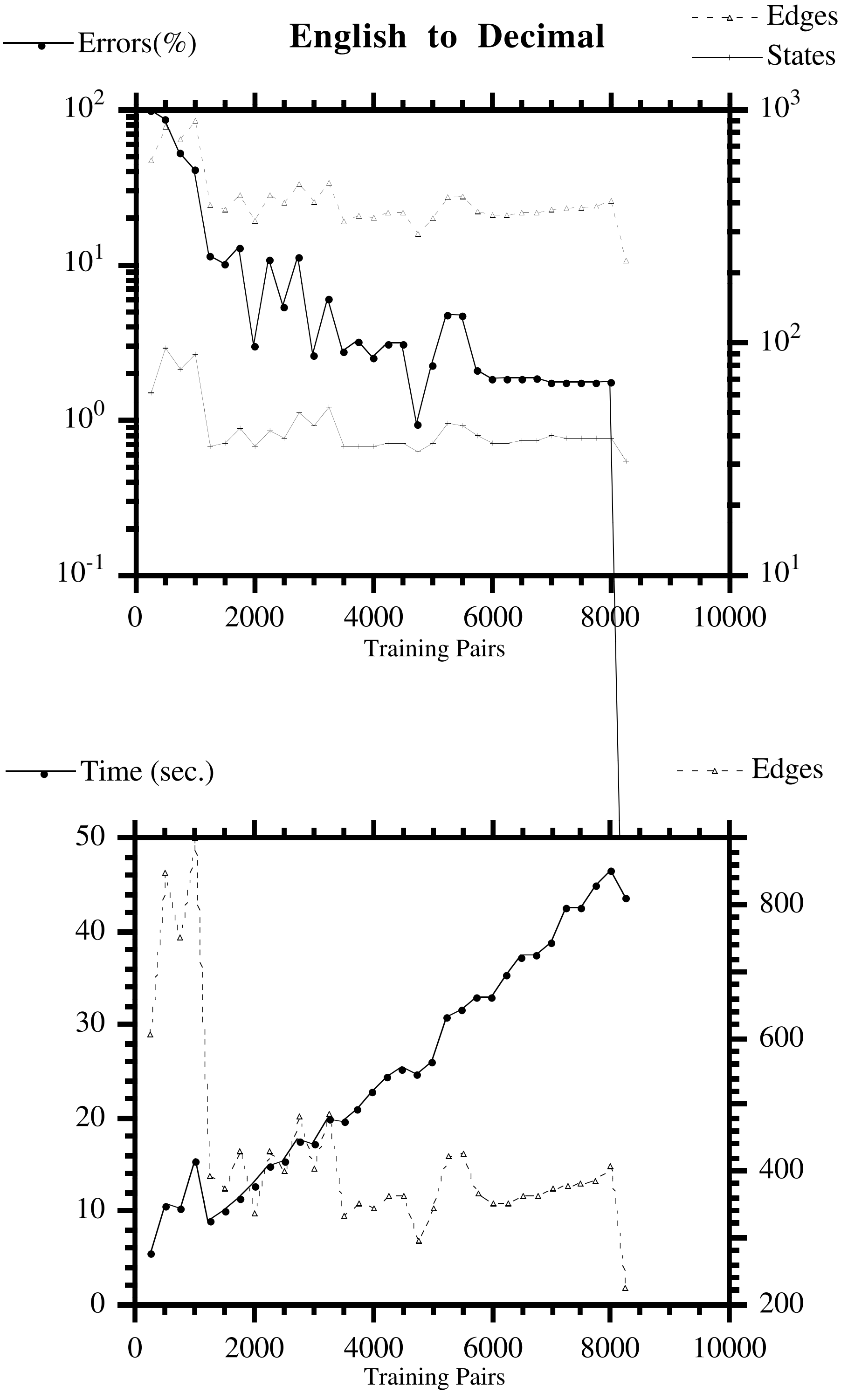


Fig. 11. Behavior of the *OSTIA* for the English to Decimal translation task

Apart from these experiments, many other similar experiments were carried out, involving these and many other tasks. For instance, *number translation from English to*

Spanish and from *Spanish to Decimal* have been successfully learnt from English-Spanish pairs and Spanish-Decimal pairs, respectively. Also, transducers that implement *integer division* of a decimal number by an arbitrary divisor *and* (reversed) *multiplication* by an arbitrary factor were easily learnt with the *OSTIA* from data-result examples [19]. The behavior of the *OSTIA*, observed in all these experiments, was rather similar to that shown in the experiments described above, presentation being omitted here for the sake of brevity.

The last group of experiments was aimed at establishing some *worst-case* results on the *computational requirements* of the *OSTIA*. While a worst-case polynomial behavior can be easily deduced from the properties of the *OSTIA* (c.f. Section IV), the exact degree seems rather difficult to establish theoretically. Nevertheless, it seems clear that worst-case behavior would be exhibited for training data corresponding to non-subsequential transduction. In other cases, when enough source transducer structure is represented in the training data, the state merging process quickly reduces the (large) number of states of the initial *OTST*, leading to the quasi-linear, low computing-time growing rates shown in figures *8* and *11* for large-enough training-sets. Therefore, in order to investigate how fast the *OSTIA* computing-time can grow, an experiment involving non-subsequential transduction was carried out.

The chosen task was that of reversing strings (representing decimal numbers in the 10^4 range), which by no means can properly be accomplished by subsequential transduction. Using a similar experimental set-up as in the previous experiments, the *OSTIA* was presented with *20* training sets of sizes increasing from *200* up to *4000*. Moreover, in order to as much as possible approach an absolute worst-case data conditioning for the *OSTIA*, the training data were presented in strict number ordering within each training-set. Obviously, in this case no appropriate transducer was ever obtained, leading to a *100%* error-rate for those strings not used for training. The computing times required by the *OSTIA* in this case are presented in Fig. 12 along with the corresponding sizes of the inferred transducers. A second-degree polynomial which was least-squared fitted to the experimental values is also shown. It is worth noting that, while a first degree fit would clearly be inappropriate, the quadratic coefficient of the fitted polynomial is relatively small, and, for the range of sizes considered, a cubic coefficient would have not improved the goodness of the fit.

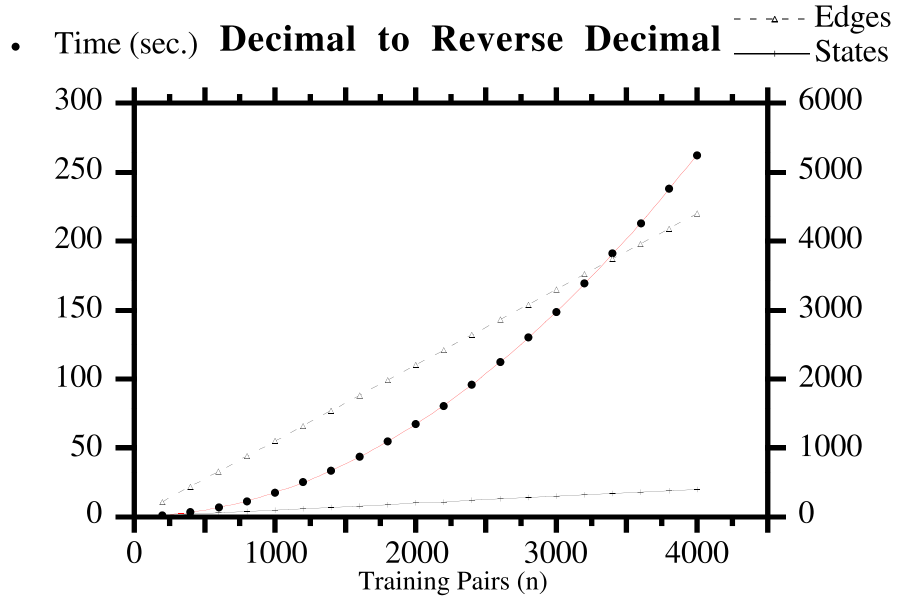


Fig. 12. A worst-case behavior of the OSTIA as applied to a non-subsequential transduction learning task. The computing-times have been least-squared fitted to a second degree polynomial: Time = $1.6{\cdot}10^{-5}n^2 + 0.00013\,n + 0.2595$

VI. DISCUSSION AND CONCLUSION

The results of the experiments described in the last section clearly indicate both the versatility of subsequential transduction and the effectiveness of the *OSTIA* to learn subsequential transducers from training input-output examples. While, for (small) training-sets not conveying enough structure of the unknown source transduction, the transducers produced by *OSTIA* tend to be rather large and inaccurate, very compact and accurate solutions are always obtained once the training data contain a small number of *appropriate* input-output pairs. The existence of such small sets of appropriate training data has been shown through some of the experiments reported in the last section, and some theoretical results regarding what an "appropriate" set of input-output training pairs is, are discussed in Section IV. However, the only practical hint these results seem to suggest is that such training data should contain the "simplest" (usually also the shortest) transduction examples, and how to actually choose adequate and small sets of training data remains an open issue of practical concern. In any case, by relying on chance alone, good results (i.e. *small* error-rates) tend to be obtained with reasonably small sets of training-pairs.

Apart from the issue of training data selection, other interesting (practical) issues remain to be investigated. First of all, since the *OSTIA* is essentially a non-incremental learning algorithm, it should be worth studying the possibility of small *incremental adaptation* to new training data of a transducer that was already fairly well established from previous (non-incremental) *OSTIA* learning. On the other hand, the possibility to incorporate an appropriate (also learnt) *error-model* into the learnt transducers, as well as to make the resulting transducers *stochastic*, needs be investigated if dealing with real and natural data such as speech or images is required. Finally, there are many real (Pattern Recognition) Interpretation problems involving the transduction between pseudo-natural and/or task-oriented languages, that are worth attempting to approach through the methods proposed here.

In any circumstance, the work presented in this paper constitutes in fact a required step that would allow these and many other interesting issues to be examined under the light of the *Interpretation* paradigm of (Syntactic) Pattern Recognition.

VII. REFERENCES.

[1] R. C. González and M. G. Thomason. "Syntactic Pattern Recognition, an introduction". *Addison-Wesley*, Reading Mass., 1978.

[2] K. S. Fu. "Syntactic Pattern Recognition and Applications". *Prentice-Hall,* New York, 1982.

[3] L. Miclet. "Structural Methods in Pattern Recognition". *North-Oxford Academic* 1986.

[4] R. de Mori (chairman) et al. "Working Group B: Waveform and Speech Recognition". *NATO adv. Research Workshop on Syntactic and Structural Pattern Recognition*. Sitges (Barcelona), October 1986. In *Syntactical and Structural Pattern Recognition*. G. Ferrate, T. Pavlidis, A. San Feliu, H. Bunke (eds.) Springer-Verlag, pp 446-451, 1988.

[5] J.C. Simon, E. Backer, and J. Sallantin, "A structural approach to Pattern Recognition". *Signal Processing*, No. 2, pp. 5-22, 1980.

[6] J. Berstel. "Transductions and Context-Free Languages". *Teubner,* Stuttgart 1979.

[7] E. M. Gold. "Complexity of automaton identification from given data". *Information and Control*, 37 pp 302-320, 1978.

[8] L. P. J. Veelenturf. "Inference of sequential Machines from sample Computation". *IEEE Trans. in Computers*. 27, pp 167-170, 1978.

[9] P. Luneau, M. Richetin and C. Cayla. "Sequential Learning from Input-Output Behavior". *Robotica* 1, pp 151-159, 1984.

[10] Y. Takada. "Grammatical inference for even linear languages based on control sets". *Information Processing Letters*, 28(4): 193-199, 1988.

[11] E. Vidal, P. García and E. Segarra. "Inductive Learning of finite-state transducers for the interpretation of unidimensional objects". In *Structural Pattern Analysis*. R. Mohr, T. Pavlidis and A. San Feliu (eds.). World Scientific, pp 17-35, 1990.

[12] E. M. Gold. "Language identification in the limit". *Information and Control,* 10: pp 447-474, 1967.

[13] K. S. Fu and T. L. Booth. "Grammatical Inference: Introduction and Survey". parts 1 and 2. *IEEE Trans. Sys. Man and Cyber.*, SMC-5: 95-111, pp 409-423, 1975.

[14] D. Angluin and C. H. Smith. "Inductive inference: theory and methods". *Computing Surveys*, 15(3), pp 237-269, 1983.

[15] L. Miclet. "Grammatical Inference". In *Syntactic and Structural Pattern Recognition*. H. Bunke and A. San Feliu (eds.) *World Scientific*, pp 237-290, 1990.

[16] D. Angluin. "Inference of Reversible Languages". *J. ACM*, 29(3), pp 741-765, 1982.

[17] P. García and E. Vidal. "Inference of k-Testable Languages in the Strict Sense and application to Syntactic Pattern recognition". *IEEE Trans. Pattern Analysis and Machine Intelligence,* Vol. PAMI-12(9), pp. 920-925, 1990.

[18] J. Oncina, P. García. "Aprendizaje de Funciones Subsecuenciales". Universidad Politécnia de Valencia, Tech. Report DSIC II-5/1991.

[19] A. M. Corbí. "Estudio de un Algoritmo de Inferencia de Transductores Subsecuenciales". Facultad de Informática, Universidad Politécnica de Valencia, Proyecto fin de carrera, 1991.

[20] P. García, E. Vidal, and F. Casacuberta. "Local Language, the Successor method, and a step towards a General Methodology for the Inference of Regular Grammars" *IEEE Trans. Pattern Analysis and Machine Intelligence,* Vol. PAMI-9(6), pp. 841-845, 1987.