Introducing Domain and Typing Bias in Automata Inference

François Coste[1], Daniel Fredouille[2],
Christopher Kermorvant[3], and Colin de la Higuera[4]

[1] IRISA, Campus de Beaulieu, Rennes, France
Francois.Coste@irisa.fr
[2] The Robert Gordon University, Aberdeen, UK
df@comp.rgu.ac.uk
[3] Dept. IRO, Université de Montréal, Canada
kermorvc@iro.umontreal.ca
[4] EURISE, Université Jean Monnet, St Etienne, France
cdlh@univ-st-etienne.fr

Abstract. Grammatical inference consists in learning formal grammars for unknown languages when given sequential learning data. Classically this data is raw: Strings that belong to the language and eventually strings that do not. In this paper, we present a generic setting allowing to express domain and typing background knowledge. Algorithmic solutions are provided to introduce this additional information efficiently in the classical state-merging automata learning framework. Improvement induced by the use of this background knowledge is shown on both artificial and real data.

Keywords: Automata Inference, Background Knowledge.

Toward Grammatical Inference with Background Knowledge. Grammatical inference consists in learning formal grammars for unknown languages when given sequential learning data. Classically this data is raw: Strings that belong to the language and eventually strings that do not. If no extra hypothesis is taken, state merging algorithms have shown to be successful for the case of *deterministic finite automata* (Dfa) learning. One of the most representative and simplest algorithm in this family is Rpni [14]. Convergence of Dfa learning algorithms depends on the presence of characteristic elements in the learning sets. In real applications some of these elements may be missing, but could be compensated by other sources of information. For instance if the learner is allowed to ask questions about the unknown language the setting is that of *active learning*: In the case of learning Dfa, algorithm L^* [1] has been proposed. In the absence of a source of reliable counter-examples, the class of learnable languages is more restricted. Either subclasses of the regular languages are used, or statistical regularities can be used to help the inference. Then, with the hypothesis that not only the language but also the distribution is regular, one can hope to learn a *stochastic deterministic finite state automaton*, as with algorithm Alergia [4].

In other areas of machine learning, successful techniques have been invented to be able to learn with more additional information and especially with the

G. Paliouras and Y. Sakakibara (Eds.): ICGI 2004, LNAI 3264, pp. 115–126, 2004.

capacity of including expert knowledge: A typical example is that of *inductive logic programming* (ILP) [13]. ILP is concerned with learning logic programs from data that is also presented as facts. Furthermore some *background knowledge* can be presented to the system using the same representation language (first order logics) as the data or the program to be learned.

In grammatical inference, specific knowledge relative to the application is often included by *ad hoc* modification of the algorithm (see for instance [10]). Very few automata inference algorithms allow the user to express a bias to guide the search. In this article, we propose a method enabling to integrate two kinds of background knowledge into inference: First, *domain bias* which is a knowledge on the language recognised by the target automata, and second *typing bias* which considers semantic knowledge embedded in the structure of the automata.

Domain and Typing Background Knowledge.

Domain bias is often available and may sometimes supply the lack of counter-examples. In [15], domain information may be provided for learning subsequential transducers. The domain is the set of sequences such that translating them make sense, *i.e.*: Those sequences that belong to the original language model. If the exact domain is given, (partial) subsequential transducers may be identified in the limit. In classification tasks performed by automata, the domain may be the set of sequences of interest to be classified inside or outside the concept to learn. Take for instance a task involving well formed boolean expressions: The sequence "¬)" should not appear. But it will neither appear in the correctly labelled examples (those that will evaluate to true), nor in those that will evaluate to false. If – as would be the case with a typical state merging algorithm – one depends on the presence of a counter-example containing "¬)" in the characteristic set for identification to be possible, then we would really have no hope to identify. If on the other hand we can express as domain background knowledge that no string can contain "¬)" as a substring, then identification could take place even when some counter-example cannot be given due to the intrinsic properties of the problem. We propose in section 2 an algorithm which allows to introduce this kind of background knowledge into automata inference.

Typing Bias: In [10], the authors add types to the states of the inferred automaton such as to introduce semantics on the symbols of the alphabet. These types are then used to constraint the structure of the inferred automaton, therefore embedding the type semantic in its structure. In [11], two of the authors proposed a formalism to represent the typing information. However, in this framework, the conditions to be met by the typing information are restrictive. Section 3 proposes an algorithm enabling to get rid of the limitations of this previous framework.

The remainder of the paper is structured as follows: First we review the basics of automata inference by state merging methods. Then, we apply the ideas presented in [5] to take specifically into account some *domain* bias in the inference process. The same algorithmic ideas are then applied for tackling more expressive *typing* functions than in [11]. We end this study by two particular cases allowing a more efficient treatment: We revisit the results of [11] under the formulation of the specialisation of a (type) automaton and we study the practical case when the sole information available is a typing of the sequences.

1 State Merging Inference of Automata

Languages and Automata. An alphabet Σ is a finite nonempty set of symbols. Σ^* denotes the set of all finite strings over Σ. A language L over Σ is a subset of Σ^*. In the following, unless stated otherwise, symbols are indicated by $a, b, c \ldots$, strings by $u, v, \ldots$, and the empty string by λ. A *finite state automaton (*Fsa*)* A is a tuple $\langle \Sigma, Q, Q_0, F, \delta \rangle$ where Σ is an alphabet, Q is a finite set of *states*, $Q_0 \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function. The transition function δ is classically extended to sequences by: $\forall q \in Q,\ \delta(q, \lambda) = \{q\}$, $\forall q \in Q,\ \forall w \in \Sigma^*,\ \forall a \in \Sigma,\ \delta(q, aw) = \bigcup_{q' \in \delta(q,a)} \delta(q', w)$. The language $L(A)$ recognized by A is $\{w \in \Sigma^* : \exists q_0 \in Q_0, \delta(q_0, w) \cap F \neq \emptyset\}$. Languages recognized by Fsa are called regular. For a given state q, the prefix and the suffix sets of q are respectively $P(q) = \{w \in \Sigma^* : \exists q_0 \in Q_0, q \in \delta(q_0, w)\}$ and $S(q) = \{w \in \Sigma^* : \exists q_f \in F, q_f \in \delta(q, w)\}$. The above definitions allow Fsa to have inaccessible states or useless states (from which no string can be parsed) but we will not consider such automata in this paper, *i.e.*: In the sequel, we will assume that $\forall q \in Q, P(q) \neq \emptyset$ and $S(q) \neq \emptyset$. A *deterministic finite automaton (*Dfa*)* is an Fsa verifying $|Q_0| = 1$ and $\forall q \in Q, \forall a \in \Sigma, |\delta(q, a)| \leq 1$. An automaton is *unambiguous* if there exists at most one accepting path for each sequence. Dfa are trivially unambiguous. For each regular language L, the canonical automaton of L is the smallest Dfa accepting L; It is denoted $A(L)$.

State Merging Algorithm. Regular languages form the sole class in the Chomsky Hierarchy to be efficiently identifiable from given data [7]. The most popular approach is the state merging scheme used to learn Fsa but also applied to the inference of sub-sequential transducers, probabilistic automata and more expressive grammars. This approach relies on the state merging operation which consists in unifying the states (or for grammars, the non-terminals). This operation increases the number of accepted sequences and is thus a generalisation operation. The general framework of state merging inference is given by algorithm 1. It consists in first building from the set I_+ of example strings a *maximal canonical automaton*, in short Mca, that recognizes only these strings, and then in applying the state merging operation iteratively while preserving a "compatibility" property used to avoid over-generalisation. When a set I_- of counter-example strings is available, the compatibility condition prevents over-generalisation by rejecting automata accepting at least one string of I_-. The search may be restricted to deterministic (respectively unambiguous) automata by using a *merging for determinisation* (respectively for disambiguisation) operation after each `merge`() [9, 6].

Two functions are likely to be modified to introduce an application based bias. The function `choose_states`() allows to introduce search heuristics while the function `compatible`() controls the generalization and restricts the search space. In this article, we will focus on the latter to integrate background knowledge.

Algorithm 1 Generic State Merging Algorithm

Require: training set I_+ (set of example strings)
Ensure: $A = \langle \Sigma, Q, F, \delta, Q_0 \rangle$ is compatible
  $A \leftarrow \text{MCA}(I_+)$
  while $\langle q_1, q_2 \rangle \leftarrow$ `choose_states`(Q) **do**
    $A' \leftarrow$ `merge`$(A, \langle q_1, q_2 \rangle)$
    if `compatible`(A') **then** $A \leftarrow A'$

2 Using Domain Bias

A first natural way to give information helping the inference is to force the hypothesis language, denoted hereafter by L, to be included in a more general one, denoted by L_G. In practice, this more general language may be defined according to the knowledge of the *maximal domain* of the sequences such that classifying them (as being positive or negative) makes sense. One can also consider that this general language is simply an *over general hypothesis*, obtained from a machine learning system or from an expert, which has to be specialized. Since $L \subseteq L_G \Leftrightarrow L \cap (\Sigma^* - L_G) = \emptyset$, learning a language included in L_G may also be seen as the inference of a language given an (eventually) *infinite set of counter-examples* defined by a language $L_- = \Sigma^* - L_G$ instead of (or in addition to) the traditional finite set I_-. Obviously these three interpretations (L_G, L_- and I_-) are not exclusive and may be combined to obtain a more general language L_G from the expert knowledge on each of these aspects.

The boolean formulae example given in introduction, or a known constraint on the lenghts of the strings of the target language are both instances of domain bias. We give in this section a practical algorithm to ensure during the inference that the hypothesis language L remains included in L_G. More precisely, we take the equivalent "counter-example language" point of view, *i.e.*: We assume that we are given an automaton A_- such that $L(A_-) = L_- = (\Sigma^* - L_G)$, and we propose to ensure that $L \cap L_- = \emptyset$. If we are given the (deterministic) canonical automaton $A(L_G)$, A_- can be easily computed by completing $A(L_G)$ before inverting final and non final states. One can remark that no assumption of determinism is required on A_- allowing a compact representation, if available, of L_-. In particular, A_- can easily be defined as the union of different (non deterministic) automata representing different sources of information without needing a potentially costly determinization. In return, we have to note that this is not true for L_G since a non deterministic automaton of this language would have to be determinized before complementation.

If we denote by A the current automaton representing the current language hypothesis L, a direct way to handle this problem would be to build the intersection product of A and A_- to check whether the intersection language is empty at each step of the inference. The algorithm presented hereafter can be seen as a simulation of this intersection product to detect incrementally non emptiness by introducing and maintaining a *common prefix* relation between the states of each automata defined as follow: Let $A_1 = \langle \Sigma, Q_1, Q_{01}, F_1, \delta_1 \rangle$ and

Algorithm 2 Computation and incremental update of common prefix relation (using the notations $A_1 = \langle \Sigma, Q_1, Q_{01}, F_1, \delta_1 \rangle$ and $A_2 = \langle \Sigma, Q_2, Q_{02}, F_2, \delta_2 \rangle$)

Function `common_prefix`(A_1, A_2)
$\mathcal{E}_p \leftarrow \emptyset$ {common prefix relation storage}
for all $q_1 \in Q_{01}, q_2 \in Q_{02}$ **do**
  `add_to_common_prefix`$(A_1, A_2, q_1, q_2, \mathcal{E}_p)$
return $\mathcal{E}_p$
Procedure `add_to_common_prefix`$(A_1, A_2, q_1, q_2, \mathcal{E}_p)$
if $\langle q_1, q_2 \rangle \notin \mathcal{E}_p$ **then**
  $\mathcal{E}_p \leftarrow \{\langle q_1, q_2 \rangle\} \cup \mathcal{E}_p$
  `propagate_forward`$(A_1, A_2, q_1, q_2, \mathcal{E}_p)$
Procedure `propagate_forward`$(A_1, A_2, q_1, q_2, \mathcal{E}_p)$
for all $a \in \Sigma, q_1' \in \delta_1(q_1, a),\ q_2' \in \delta_2(q_2, a)$ **do**
  `add_to_common_prefix`$(A_1, A_2, q_1', q_2', \mathcal{E}_p)$
Procedure `update_after_state_merging`$(A_1, A_2, q, q', \mathcal{E}_p)$
Require: q is the resulting state of merging q and q' in A_1
for all $q_2 : \langle q, q_2 \rangle \in \mathcal{E}_p$ **do**
  `propagate_forward`$(A_1, A_2, q, q_2, \mathcal{E}_p)$ {handling new transitions added to q}
for all $q_2 : \langle q', q_2 \rangle \in \mathcal{E}_p$ **do**
  `add_to_common_prefix`$(A_1, A_2, q, q_2, \mathcal{E}_p)$ {reporting relations of q'}

$A_2 = \langle \Sigma, Q_2, Q_{02}, F_2, \delta_2 \rangle$. Two states q_1, q_2 in $Q_1 \times Q_2$ *share a common prefix* iff $P(q_1) \cap P(q_2) \neq \emptyset$. It is easy to see that the intersection of $L(A_1)$ and $L(A_2)$ is not empty *iff* there exists a couple of final states $q_1, q_2 \in F_1 \times F_2$ in common prefix relationship. Thus to apply this detection to A and A_-, it is sufficient to compute the set of states of A and A_- in common prefix relationship and to test whether two final states are in relation.

Moreover, since common prefixes are preserved by state merging, this relation may be maintained incrementally during the inference process. The sketch of the algorithm computing and incrementally updating the common prefix relation (algorithm 2) is then the following. At the initialization step, the common prefix relation between A and A_- can be computed by the `common_prefix`() function. Then, after each merge in A, new common prefixes created by the state merging are propagated using the `update_after_state_merging`() procedure. Detection of the constraint violation may be done simply by checking when a new common prefix relation is detected between a final state of the current automata A and a final state of A_-. At the initialization step, this violation means that the problem cannot be solved with the given data (*e.g.*: L_- includes a sequence of I_+). During the inference, it allows to detect that the last state merging is invalid and should not be done (involving a backtrack).

The theoretical complexity of function `common_prefix`() is in $O(|A| \times |A_-| \times t_A \times t_{A_-})$, where t_A and t_{A_-} are the maximum number of outgoing transitions by the same symbol from states of A and A_-. In practice, a maximum sequence of $|A|$ successful state-mergings can be achieved during inference. During such a sequence of state-mergings the complexity of the naïve approach (*i.e.*: Recom-

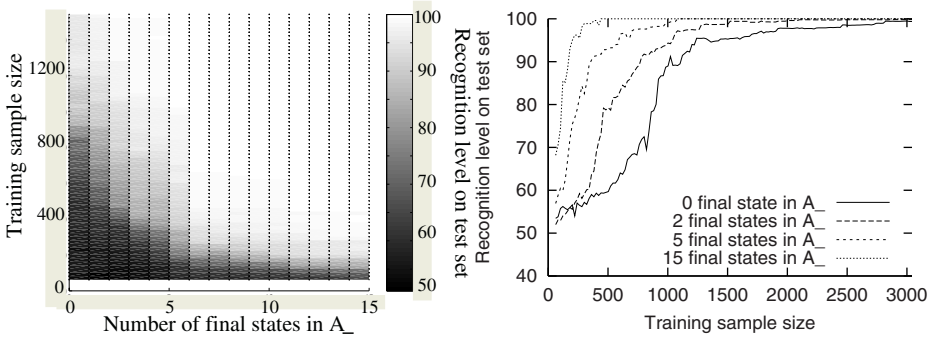

Fig. 1. Recognition level on the test set (*i.e.*: The number of correctly classified words of the test set divided by the test set size) of the solution given by RPNI [14] (extended with common prefix relation). The left plot shows how the recognition level improves (grey level) when considering an increasing "quantity" of domain bias (abscissa), and an increasing training sample size (ordinate). The right plot represents vertical sections of the left plot. The experimental setting is the following: A 31 states target automata, a training and a testing set have been generated by the GOWACHIN server (http://www.irisa.fr/Gowachin/). The complementary A_- of the target automata has been computed to provide progressively domain information by considering an increasing number of its 15 final states. Each point in a plot is a mean of nine executions with different training sets and different choices for the final states of A_-. Even if only illustrative, on can remark that the size of the training sample needed for RPNI to converge is strongly reduced when introducing domain bias (even for "small" bias) but that using only this information without providing a sample of a reasonable size does not enable the algorithm to converge.

puting relations after each merge with function `common_prefix`()) is therefore $O(|A|^2 \times |A_-| \times t_A \times t_{A_-})$. Using the incremental `update_after_state_merging`() procedure improves this complexity by a $|A|$ factor. Indeed, the worst case complexity of this procedure is achieved when $|A| \times |A_-|$ relations are stored from the beginning in $\mathcal{E}_p$. Next $|A|$ calls to `update_after_state_merging`() during the inference process have a complexity of $O(|A_-| \times t_A \times t_{A_-})$. Indeed, the `update_after_state_merging`() procedure try to propagate the $|A_-|$ relations existing on the merged state, but all propagations stop since the set of relations is already complete. This leads to a global complexity of $O(|A| \times |A_-| \times t_A \times t_{A_-})$.

We have presented here the introduction of domain bias in a state merging inference algorithm. The proposed approach is generic enough to be adapted to other inference algorithms whenever they proceed by automaton generalization. In particular, since DeLeTe2 generalization operations [8] preserve common prefix relations, this approach could be easily applied to this algorithm. An experiment on artificial data showing the interest of using domain bias during automata inference is provided figure 1.

In the next section, we propose to use the common prefix relation to take into account typing prior knowledge in the inference process.

3 Using Typing Information

In applications, information on the symbols of the examples is often available. For instance, in genomics, the secondary structure to which the amino acid belongs in the protein is often known. In linguistics, one can have the result of the part of speech tagging or chunking of the sequences. This information may be considered as a typing of the symbols in the sequences and may have been given either by an unknown machine or an expert (in this case, only the result of the typing of example sequences is available) or by a typing machine provided by the expert as background knowledge (in this case, the typing is known for the examples but also for all the sequences in the domain of the machine). In sections 3.1 and 3.2, we will focus on the second case and will assume that we are given a typing machine by the expert. The first case will be studied specifically in section 3.3.

3.1 Inference of Compatible Type Automata

Formalisation of the Expert Knowledge: If we designate by Σ the symbol alphabet and by S the sort alphabet, a *symbol typing function* τ is defined from $\Sigma^* \times \Sigma \times \Sigma^*$ to S. We then designate by $\tau(u, a, v)$ the sort associated to symbol a in string uav. From this definition, the sort can depend both on u, called the *left context*, on v, called the *right context*, and on the typed symbol itself a. The expert knowledge on sorts can be represented as a typing function τ provided to the inference process. Therefore, the sorts returned by the τ function have a semantic meaning on the basis of the expert knowledge.

Since we are considering the inference of Fsa, we will assume here that the typing function provided is somehow "rational", *i.e.*: That the typing could have been generated by a finite state machine. Thus, following [11], we consider that the typing function τ provided by the expert can be encoded in a *type automaton*: A finite state type automaton T is a tuple $\langle \Sigma, Q, Q_0, F, \delta, S, \sigma \rangle$ where $\Sigma, Q, Q_0, F, \delta$ are the same as in classical Fsa, S is a finite set of sorts and $\sigma : \ Q \rightarrow S$ is the state typing function. A type automaton T represents a typing application τ from $\Sigma^* \times \Sigma \times \Sigma^*$ to 2^S defined by $\tau(u, a, v) = \{\sigma(q) / \exists q_0 \in Q_0, \exists q_f \in F, q \in \delta(q_0, ua)$ and $q_f \in \delta(q, v)\}$. Such as to consider typing functions (*i.e.*: From $\Sigma^* \times \Sigma \times \Sigma^*$ to S) instead of typing applications, we will restrict ourself to type automata such that $\forall u, v \in \Sigma^*, \ \forall a \in \Sigma : \ |\tau(u, a, v)| \leq 1$. As improvments on [11], we authorize type automata to be non-deterministic to handle typing functions depending on the left context but also on the right context of the word. We also authorize incompletely specified typing functions as it is often the case in practice. Figure 2 gives an example of a type automaton.

Integration of the Expert Knowledge in Inference: To integrate the typing information in the learned automaton, this one has to embed this information in its structure. To achieve this goal, one associate a sort to each state of the learned automaton, which can therefore be considered as a type automaton. The idea is that the typing function embedded in the structure of the inferred automaton has to be *compatible* with the typing function provided as expert knowledge.

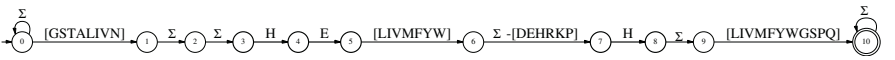

Fig. 2. Expert knowledge on proteins family is available in the PROSITE database (http://au.expasy.org/prosite/). This type automaton has been constructed from the PROSITE pattern PS00142 (ZINC_PROTEASE). Σ is the amino acids alphabet, the state typing function may be defined to return the state number on significative states. Knowledge embedded in this pattern is for instance the presence of the two amino acids H (Histidine) in the pattern because of their good properties to fix Zinc molecules.

We denote by $\mathcal{D}_\tau$ the domain of a typing function τ which is the set of tuple $\langle u, a, v\rangle$ such that $\tau(u, a, v)$ is defined. Thanks to our distinction between the domain and the typing knowledge, we use a weaker definition of the compatibility of two typing functions than the one proposed in [11]: Two typing functions τ and τ' are *compatible* iff $\forall \langle u, a, v\rangle \in \mathcal{D}_\tau \cap \mathcal{D}_{\tau'},\ \tau(u, a, v) = \tau'(u, a, v)$.

In the following, two type automata will be considered compatible if their typing functions are compatible. We propose to ensure the compatibility of the learned automaton A with the provided type automaton A_T during the inference process by a similar scheme to the one presented in section 2. But, since left and right contexts are considered for typing, common prefix relation is not sufficient. To propagate the acceptance information backward to the states, the *common suffix* relation is also used. It is defined as follow: Let $A_1 = \langle \Sigma, Q_1, Q_{01}, F_1, \delta_1\rangle$ and $A_2 = \langle \Sigma, Q_2, Q_{02}, F_2, \delta_2\rangle$. Two states q_1, q_2 in $Q_1 \times Q_2$ are said to share a *common suffix* iff $S(q_1) \cap S(q_2) \neq \emptyset$. This definition is completely symmetric to the the definition of the common prefix relation and share the same properties. Thus computation and incremental maintenance of this relation can be done by similar functions to those given in algorithm 2 simply by replacing the set of initial states by the set of final states and by propagating the relations backward instead of forward by using the inverse function of transition.

The common prefix and common suffix relations are used during the inference process to ensure the compatibility between the inferred automaton A and the automaton embedding the expert typing function A_T. This is done by assigning to each state q of A sharing both a common prefix and a common suffix with a state q_T of A_T the sort of q_T and by prohibiting assigning different sorts to the same state. At the initialization step, this projection of the sorts allows to initialize the state typing function of A. A failure at this point (*i.e.*: Trying to assign two different types to one state) shows that the typing function of A_T is not functional or, in the deterministic case when the prefix tree acceptor is used as MCA [9], that the typing can not be realized by a DFA. During the inference, a failure detection allows to detect that the last state merging is invalid and should not be done (involving a backtrack). As for domain background knowledge, an experiment on artificial data showing the interest of typing bias during automata inference is provided (figure 3).

It may be easily shown that states of A with different sorts will always fail to be merged and that trying to merge them is useless. In figure 4, we show that the initial typing of the states is not sufficient, even for deterministic type automata, to ensure compatibility and that propagation of relations has to be

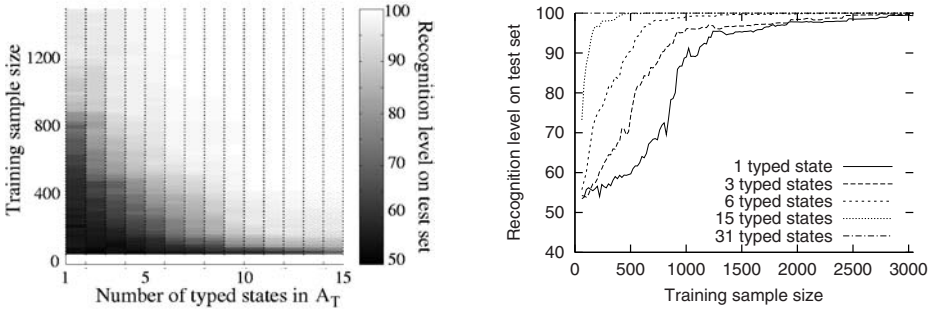

Fig. 3. Recognition level when given a type automaton A_T. The same experimental setting and the same automaton than in figure 1 is used except that instead of giving final states information, typing information is given progressively (here, between 1 and 31 typed states of A_T chosen randomly, all the states are final, the left plot is truncated at 15 typed states). Convergence is faster since the given typing information is richer than the domain one.

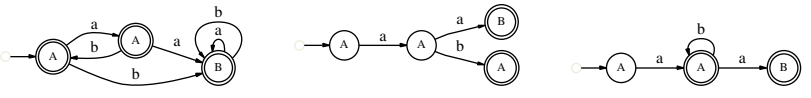

Fig. 4. A_T, deterministic MCA for $I_+ = \{aa, ab\}$ and A obtained by merging source and target state of the transition by b. Although the merged states have the same sort A, in A_T $\tau(ab, a, \lambda) = \text{A}$ whereas in the merged automaton $\tau(ab, a, \lambda) = \text{B}$ (and the problem remains even if we consider a complete sample wrt to A_T, *e.g.*: $\{aa, ab, ba, bb\}$).

done after state merging to ensure that new accepted sequences are correctly typed. We will study in the next sections two special cases such that avoiding to merge states with different sorts is sufficient to ensure the compatibility.

3.2 Specialization of a Regular Grammar

In the setting of [11], typing is feasible only if: 1. The expert type automaton is deterministic, 2. It has a completely defined state typing function, 3. It accepts at least all words of I_+, 4. It has different sorts for all its states (*i.e.* $\forall q, q' \in Q_T, q \neq q' : \sigma_T(q) \neq \sigma(q')$), and 5. The inferred automaton is deterministic. The setting presented in section 3.1 removes all these constraints to the cost of a more important complexity. If we consider these restrictions, [11] show that the initial typing of the MCA can be realised in $O(|\,\text{MCA}\,|)$, and the maintenance of types can be realized in $O(1)$ after each state merging.

In fact, the determinism restrictions 1. and 5. can be removed while keeping the $O(1)$ type managment after each state merging. This is done by realizing the initial typing as proposed in subsection 3.1. The maintenance of types along inference is realized in $O(1)$ by preventing to merge states with different state typing functions. The formal proof follows the same lines as the one provided in [11]. This extension is interesting because it allows to handle efficiently some

typing functions that cannot be represented by deterministic type automata with different sorts for all states (typing functions τ such that $\tau(u, a, bv) = \tau(u', a', bv')$ and $\tau(ua, b, v) \neq \tau(u'a', b, v')$) but that can be represented by non deterministic ones.

In [11], the semantic of the restrictions stays unclear. We discuss here an interpretation and the links between this typing framework and domain background knowledge. Under constraints 2. to 4., all compatible automata of the search space in the state-merging framework can be seen as *specializing* the structure of A_T. Indeed, the constraint 4. implies a one to one correspondence between the sorts and the states of A_T and thus the information given by typing is the structure of A_T. Let A'_T be the sub-automaton of A_T obtained by discarding useless states for the acceptance of I_+. We can show that A'_T can be obtained by merging states of any compatible automaton of the search space. Therefore these automata are specializations of A_T.

By inferring automata A specializing the structure of A_T, we also specialize the recognized language, *i.e.* $L(A) \subseteq L(A_T)$. Then, by constraining the structure, the domain is also constrained. We obtain a setting such that both the structure and the language of the automaton A_T are specialized and we should rather consider that this setting corresponds to the *specialization of an automaton* (not necessarily typed). In particular, it should be noticed that in this setting some automata verifying the language constraint, but not the structural one, can be excluded from the search space.

3.3 Inference Given Typed Examples

A practical case such that updating common prefix and common suffix relations is not necessary is when we are given only typed examples. A type automaton A_T could easily be constructed from the typed examples and would verify $L(A_T) = I_+$. In that case, no typing constraint is given about sequences outside the training set and the sole remaining constraint is to avoid creating two acceptance paths leading to different typing for a sequence of I_+ (to ensure compatibility) what can be avoided by considering the inference of DFA or more generally of unambiguous type automata [6] (allowing also to ensure functionality of the learned type automaton). Then, one can show that the initial projection of the sorts to the MCA and forbidding merging states with different sorts is sufficient to ensure the compatibility. But it should be noticed that some complementary state merging (for determinization or for disambiguisation) may have to be done before detecting the failure. This case should rather be considered factually as a case such that the typing information can be directly encoded in the automata A and does not need a second automata A_T besides it.

Inference given typed examples is interesting when no type automaton is given and only a tagging of the examples is available. This setting is also interesting if a typing function that cannot be converted into a typing automaton is available. In this case, the function can be used at least to type the examples. We illustrate this by an experiment on the ATIS task using Part-Of-Speech Tagger typing information (figure 5).

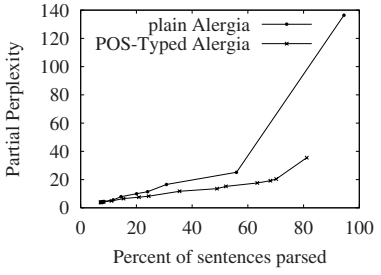

Fig. 5. We have tested the use of type information for automaton inference on the Air Travel Information System (ATIS) task [12]. This corpus has been widely used in the speech recognition community to test various language models. The type information was composed of Part-Of-Speech (POS) tags provided by the Brill tagger [2]. Here we inferred stochastic automata using Alergia inference algorithm [4] with and without the POS-tag information. Each word was tagged with its most probable POS-tag, disregarding the context-rules in the tagger. In this task, a usual quality measure is the *perplexity* of the test set S (ordinate) given by $P = 2^{-\frac{1}{|S|}\sum_{w \in S} \log_2 P(w)}$, where $P(w)$ is the probability given by the stochastic automaton to the word w. The smaller the perplexity the better the automaton can predict the next symbol. The sentences with 0 probability are ignored in this score (the presence of one of these sentence would lead to an infinite perplexity). So to evaluate the results, we also have to represent the percentage of sequences accepted, *i.e*: With non 0 probability (abscissa). In the Alergia algorithm, generalisation is controled by one parameter. Different values for this parameter provided the different points of the figure. The best results are situated in the bottom right corner as they correspond to high coverage and small perplexity. For a given number of sentences parsed, the use of POS-tag based type reduces the partial perplexity and provides better models.

Discussion, Conclusion and Perspectives

We have proposed a generic setting to use domain and/or typing knowledge for the inference of automata. This setting includes the non deterministic and incomplete knowledge cases and allows different degrees of efficiency. In particular, two practical cases have been identified such that the expert knowledge can be taken into account with a small over-cost. Experiments on real applications are now needed to validate the approach and to quantify (experimentally, but also theoretically) the amount of the given help.

As pointed by one of the reviewers the presented models have to be compared to existing models to learn in helpful environments. A starting point of that research for the domain bias could be a comparison with inference being allowed *membership queries* [1]. Indeed a language of counter-examples provided as an automaton A_- can answer some of the membership queries (the ones concerning words in the language of A_-). For the typing bias, a comparison with the work of [3] has to be realised. We also have to explore the fact that the comportement of our algorithms is unclear when the provided knowledge is erroneous. In this case, a solution could be to use this knowledge as a heuristic instead a pruning constraint.

Another promising perspective is to study the extension of this work to more powerful grammars. In particular, coupling non terminals with typing could provide an interesting framework to include semantic background knowledge in the inference of context-free grammars and should be compared to the parsing skeleton information used with success by Sakakibara [16].

References

1. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Control*, 39:337–350, 1987.
2. E. Brill. Some advances in rule-based part of speech tagging. In *Proc. National Conference on Artificial Intelligence*, 1994.
3. A. Cano, J. Ruiz, and P. García. Inferring subclasses of regular languages faster using RPNI and forbidden configurations. In *Proc. Int. Coll. on Gram. Inference*, volume 2484 of *LNCS*, pages 28–36. Springer, 2002.
4. R. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *Proc. Int. Coll. on Gram. Inference*, number 862 in LNCS, pages 139–150. Springer Verlag, 1994.
5. F. Coste and D. Fredouille. Efficient ambiguity detection in c-nfa, a step toward inference of non deterministic automata. In *Proc. Int. Coll. on Gram. Inference*, pages 25–38, 2000.
6. F. Coste and D. Fredouille. Unambiguous automata inference by means of state-merging methods. In *European Conference on Machine Learning*, pages 60–71, 2003.
7. C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27:125–138, 1997.
8. F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using *rfsa*. In *Proc. Int. Conf. on Alg. Learning Theory*, LNCS, pages 348–363. Springer-Verlag, 2001.
9. P. Dupont, L. Miclet, and E. Vidal. What is the search space of the regular inference? In *Proc. Int. Coll. on Gram. Inference*, number 862 in LNCS, pages 25–37. Springer Verlag, 1994.
10. T. Goan, N. Benson, and O. Etzioni. A grammar inference algorithm for the world wide web. In *Proc. of AAAI Spring Symposium on Machine Learning in Information Access.* AAAI Press, 1996.
11. C. Kermorvant and C. de la Higuera. Learning languages with help. In *Proc. Int. Coll. on Gram. Inference*, volume 2484 of *LNCS*, pages 161–173. Springer, 2002.
12. C. Kermorvant, C. de la Higuera, and P. Dupont. Improving probabilistic automata learning with additional knowledge. In *Proc. of the IAPR International Workshops SSPR and SPR*, 2004.
13. S. Muggleton. Inductive Logic Programming. In *The MIT Encyclopedia of the Cognitive Sciences (MITECS)*. MIT Press, 1999.
14. J. Oncina and P. García. Identifying regular languages in polynomial time. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition*, volume 5, pages 99–108. World Scientific, 1992.
15. J. Oncina and M. A. Varó. Using domain information during the learning of a subsequential transducer. In *Proc. Int. Coll. on Gram. Inference*, volume 1147 of *LNCS*, pages 301–312. Springer, 1996.
16. Y. Sakakibara and H. Muramatsu. Learning context-free grammars from partially structured examples. In *Proc. Int. Coll. on Gram. Inference*, number 1891 in LNCS, pages 229–240. Springer-Verlag, 2000.