# Learning Languages with Help

Christopher Kermorvant        Colin de la Higuera*

March 14, 2002

**Abstract**

Grammatical inference consists in learning formal grammars for unknown languages when given learning data. Classically this data is raw: strings that belong to the language or that do not. We present in this paper the possibility of learning when presented with additional information such as the knowledge that the hidden language belongs to some known language, or that the strings are typed, or that specific patterns have to/can appear in the strings. We propose a general setting to deal with these cases and provide algorithms that can learn deterministic finite automata in these conditions. Furthermore the number of examples needed to correctly identify diminishes drastically with the quality of the added information. We show that this general setting can cope with several well known learning tasks.

## 1 Introduction

Grammatical inference consists in learning formal grammars for unknown languages when given learning data. The task has received a lot of attention, with different settings and variations.

In a classification type of approach one wants to learn a language from a presentation of this language. The presentation may include only instances of the language (we will be learning from *text*) or both examples and counter-examples with their respective labels (learning from a *complete presentation*). In this setting it is unusual to have any more information than these strings and the added bias of the class of grammars from which one should select the candidate from. If no extra hypothesis is taken asuccesfull algorithm has been RPNI[1][OG92]. Heuristics in which the order in which hypothesis are made depend on the quantity of data that sustains it [dlHOV96, LPP98] have been proposed under the names of *data driven* or *evidence driven* algorithms.

If one is allowed to ask questions about the unknown language the setting is that of *active learning* for which the $L^*$ algorithm [Ang87] has been proposed.

In the absence of a source of reliable counter-examples the possibilities of learning become sparse [Gol67]. Additional information that may help in that case is the actual structure of the strings [Sak97]; this is equivalent to been given

*This work was done when the second author visited the Departamento de Lenguajes y Sistemas Informáticos of the University of Alicante, Spain. The visit was sponsored by the Spanish Ministry of Education.

[1]Regular Positive and Negative Inference

the skeleton of the parse tree for each example. Partial structure may also help, as shown in [SM00].

An interesting alternative is to consider the hypothesis that not only is the language regular, but that the distribution also is. In such a case one needs to learn a *Stochastic Finite Automaton.* A better known algorithm for this task is ALERGIA[2][CO94]. Data-driven versions have been provided in [GBE96, YLT00].

In other areas of machine mearning, on the other hand, succesfull techniques have been invented to be able to learn with much more additional information:

*Inductive logic programming* (ILP)[Mug99] 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. The field has used this possibility with great success. The generalization methods may then include different techniques, between which one may even encounter grammatical inference [Bos96].

The possibility of being able to use such background knowledge in a systematical way in grammatical inference would lead to the following advantages, that we give for the case of dfa learning only:

- The search space of the dfa identification task is well defined [DMV94]. The space depends on the learning data one is presented, and can be extremely large. Typing, or excluding specific automata as solutions allows for the reduction of the search space.

- Convergence of dfa learning algorithms depends on the presence of characteristic elements in the learning sets. Some of these elements can be substituted by specific knowledge.

- More importantly, in practical cases one does not have an alphabet of 0s and 1s: one has an alphabet where certain symbols appear in certain positions only, independently of the particular language one is faced with. For instance, in a task involving well formed boolean expressions the sequence "$\neg$)" 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 "$\neg$)" in the characteristic set, for identification to be possible (for some specific merge not to take place), then we would really have no hope to identify. If on the otherhand we have background knowledge that no string can contain "$\neg$)" as a substring, then this merge would not take place and identification could take place even when some counter example cannot *par nature* be given.

The idea of using this sort of extra information is clearly not new in grammatical inference. Earlier work work along these lines include:

- [RST95] propose an algorithm to learn stochastic finite automata. For theoretical reasons they restrict themselves to the class of acyclic automata. They prove that these admit a normalized level form: each state is assigned

[2]Algorithm Regular Grammatical Inference Algorithm

a unique integer, which corresponds to the length of all the strings that reach that state. The restriction can be translated as a specific automaton which has to be used as the backbone of the dfa to be inferred.

- [GBE96] use algorithm ALERGIA [CO94] to learn automata for web-oriented tasks. The authors note that a data driven heuristics [dlHOV96] and a typed approach, give better results. The typed approach consists in noting that locally certain symbols may not follow each other and thus forcing the automaton to respect such rules. This is tested on several benchmarks with clear success.

- [dlHB01] propose an ILP system called GIFT[3] that learns a tree automaton from a set of terms. This automaton is later translated into a logic program. The system is applied to real world data, and a typing of the data is also infered (basically rules that state -for instance- that a person has 2 parents one of which is male and the other female). This type is used to avoid testing for impossible merges (a *man* with a *woman*). A stochastic version (learning stochastic logic programs by means of stochastic tree automata) is proposed in [BH01].

- [Fer01] suggests the task of learning XML grammars from positive data. A very general XML grammar is known, and the idea is to find a better fitting grammar (but that takes into account the general one).

In the next section definitions will be given for the paper to be self-contained. The inference of typed automata will be proved feasible in section 3. In section 4 we will deal with the more general problem of learning $L$, knowing $L'$ and that$L \subseteq L'$. In both cases the new information is used to infer a solution, but incorrectly typed data whether positive or negative is unavailable. Applications of these methods are presented in section 5.

## 2 Basic Definitions

### 2.1 Languages and Automata

An alphabet $\Sigma$ is a finite nonempty set of symbols. $\Sigma^*$ denotes the set of all finite strings over $\Sigma$, $\Sigma^+ = \Sigma^* \backslash \{\lambda\}$. A language $L$ over $\Sigma$ is a subset of $\Sigma^*$. In the following, unless stated otherwise, symbols are indicated by $a, b, c \ldots$, strings by $u, v, \ldots$, and the empty string by $\lambda$. $\mathbb{N}$ is the set of non negative integers.

The length of a string $u$ will be denoted $|u|$, so $|\lambda| = 0$. Let $X$ be a finite set of strings, $|X|$ denotes the number of strings in the set and $\|X\|$ denotes the total sum of the lengths of all strings in $S$. Let $L$ be a language, the *prefix set* is $\mathrm{Pref}(L) = \{x : xy \in L\}$. The symmetrical difference between two languages $L_1$ and $L_2$ will be denoted $L_1 \ominus L_2$. Let $u \in \Sigma^*$ and $L \subseteq \Sigma^*$, $u^{-1}L = \{v : uv \in L\}$. By extension $L_1^{-1}L_2 = \{v : \exists u \in L_1 \wedge uv \in L_2\}$.

**Definition 1 (Deterministic Finite Automata).** *A* deterministic finite automaton*(*dfa*) $\mathcal{A} =< \Sigma, Q, F, \delta, q_0 >$ is a quintuple where*

[3] Grammatical Inference for Terms

$\Sigma$ *is a finite alphabet*

$Q$ *is a finite set of* states

$F \subset Q$ *is the set of final states*

$q_0 \in Q$ *is the initial state*

$\delta : Q \times \Sigma \to Q$ *is the transition function*

The above definition admits dfa to have inaccessible states or states that do not lead to any acceptable string. We will not consider such automata in this paper. In the sequel we will admit that:

$\forall q \in Q, \exists w \in \Sigma^* : \delta(q_0, w) = q$

$\forall q \in Q, \exists w \in \Sigma^* : \delta(q, w) \in F$

The above constraint is not restrictive,as any dfa can be simply transformed into a dfa with no useless states [4] .

**Definition 2.** *We extend $\delta$ to a function $Q \times \Sigma^* \to Q$ by $\forall q \in Q : \delta(q, \lambda) = q$ and $\forall q \in Q, \forall a \in \Sigma, \forall u \in \Sigma^* : \delta(q, au) = \delta(a, \delta(q, u))$.*

**Definition 3.** *Let $\mathcal{A} =< \Sigma, Q, F, \delta, q_0 >$ be a* dfa. *The language recognized by $\mathcal{A}$, $L(\mathcal{A})$ is $\{w \in \Sigma^* : \delta(q_0, w) \in F\}$.*

*The language $L_{\mathcal{A}}(q_i) = \{w \in \Sigma^* : \delta(q_0, w) = q_i\}$.*

*The* size *of $\mathcal{A}$ denoted $\|\mathcal{A}\|$ is $|Q||\Sigma|$.*

*$q$ is a* predecececessor *of $q'$ iff $\exists a \in \Sigma : d(q, a) = q'$.*

## 2.2 Polynomial Identification in the Limit from Given Data

The question of learning grammars or automata with help can be tackled in a variety of learning models. Due to the difficulty of the PAC-learning[5] model [Val84] in the case of languages, and the fact that the active learning setting [Ang87] already makes use of external help through the use of queries, we will turn to the polynomial variant of identification in the limit [Gol78, dlH97].

In this setting the learner is asked to learn from a learning *sample*, *i.e.* a finite set of strings, each string labeled by '+' if the string is a positive instance of the language (an element of $L$), or by '$-$' if it is a negative instance of the language (an element of $\Sigma^* \backslash L$). Alternatively we will denote $S = (S_+, S_-)$ where $S_+$ is the sub-sample of positive instances and $S_-$ the sub-sample of negative ones.

**Definition 4 (Polynomial identification in the limit from given data).** *A class $\mathbb{A}$ of automata is polynomially identifiable in the limit from given data* iff *there exist two polynomials $p()$ and $q()$ and a learning algorithm $\mathcal{L}$ such that:*

1. *Given any sample $(S_+, S_-)$, $\mathcal{L}$ returns an automaton $\mathcal{A}$ in $\mathbb{A}$ compatible with $(S_+, S_-)$ in $\mathcal{O}(p(\|S_+\| + \|S_-\|))$ time;*

[4]The dfa for the empty language will nevertheless, for technical reasons, have one single (initial) state

[5]Probably Approximatively Correct

2. *for each automaton $\mathcal{A}$ in $\mathbb{A}$ of size $n$, there exists a characteristic sample $(CS_+, CS_-)$ of size less than $q(n)$ for which, if $CS_+ \subseteq S_+$, $CS_- \subseteq S_-$, $\mathcal{L}$ returns a grammar $\mathcal{A}'$ equivalent to $\mathcal{A}$.*

In this setting it is known that deterministic finite automata are polynomially identifiable in the limit from given data, whereas context-free grammars and non deterministic finite automata are not [dlH97].

## 2.3 Learning Regular Languages: a Generic Algorithm

A variety of algorithms identifying dfa can be considered. We give here a generic algorithm, dependent of function Choose, which we will suppose deterministic. The proof that this algorithm complies with the conditions of definition 4 can be found in [dlHOV96]:

**Algorithm 1** Infer $\mathcal{A}$

```
Require: $S = (S_+, S_-)$, functions Compatible, Choose
Ensure: $L(\mathcal{A}) = L$ if $S$ is characteristic, some compatible language if not where
  $\mathcal{A} =< \Sigma, Q, F, \delta, q_0 >$
  $Q = \{q_0\}$
  $i = 1$
  Candifates = Candifates $\cup\{\langle 0, a\rangle : a\Sigma^* \cap S_+ \neq \emptyset\}$
  while Candifates $\neq \emptyset$ do
    $\langle k, a\rangle$ = Choose(Candifates)
    Candifates = Candifates $-\{\langle k, a\rangle\}$
    if $\exists q_j \in Q$ : Compatible$(k, a, j, \mathcal{A}, S)$ then
      $\delta(q_k, a) = q_j$
    else
      $Q = Q \cup \{q_i\}$
      $\delta(q_k, a) = q_i$
      Candifates = Candifates $\cup\{\langle i, a\rangle : a\Sigma^* \cap L_A(q_i)^{-1}S_+ \neq \emptyset\}$
      $i++$
    end if
  end while
  for all $q_i \in Q$ do
    if $\lambda \in (L_A(q_i))^{-1}S_+$ then
      $F = F \cup \{q_i\}$
    end if
  end for
```

Algorithm 1 depends on the choice of function *Chose*. Provided it is a deterministic function (such as one that choses the minimal $\langle i, a\rangle$ in lexicographic order), convergence is insured.

## 3 Inferring Typed Automata

Between the additional information one may have about the language to be inferred, we consider here that it is not only about inclusion (all strings should belong to such language), but also structural. In many cases information about how strings are meant to belong. Typically this can be dealt with by typing.

**Algorithm 2** Compatible($i, q, j, \dashv, S$)

```
Require: $S = (S_+, S_-)$
  $\delta(q_i, a) = q_j$
  for all $q_k \in Q$ do
    if unparsed$(S_+, G, A_k) \subseteq \{\lambda\} \vee$unparsed$(S_+, G, A_k) \cap$unparsed$(S_, G, A_k) \neq \emptyset$ cest a dire
    $(L(q_k))^{-1})S_+ \subseteq \{\lambda\} \vee (L(q_k))^{-1})S_+ \cap (L(q_k))^{-1})S_- \neq \emptyset$ then
      return false
    end if
  end for
  return true
```

Type theory has been used in different areas of computer science to help formalize and restrict classes, definitions, etc . Here quote ...

## 3.1 Type Automaton

**Definition 5.** *A type automaton $\mathcal{T}$ is defined by $< \Sigma, Q, q_0, F, \delta, S, \sigma >$ where:*

- *$\Sigma$ is an alphabet*
- *$Q$ is a finite set of states*
- *$q_0$ is the initial state*
- *$\delta : Q \times \Sigma \to Q$ is the transition function*
- *$S$ is a set of elements called sorts*
- *$\sigma : Q \to S$ is the typing (total) function*
- *$F \subset Q$ is the set of final states such that $\forall q_1, q_2 \in Q,\ \sigma(q_1) = \sigma(q_2) \implies (q_1 \in F \iff q_2 \in F)$*

It should be noticed that a type automaton is a dfa with types. The intended meaning of the type automaton is to describe how strings are constructed. Final states are used to indicate which are the sorts of the objects in the language. Classically, $L(\mathcal{T})$, the language defined by a type automaton $\mathcal{T}$, is the set of all words $w$ in $\Sigma^*$ such that $\delta(q_0, w) = q$ with $q \in F$. We define the extension of $\sigma$ to words such that for all $w \in Pref(L(\mathcal{T}))$, $\sigma(w) = \sigma(\delta(q_0, w))$.

**Definition 6.** *A type automaton $L(\mathcal{T}) =< \Sigma, Q, q_0, F, \delta, S, \sigma >$ is* minimal *if $\forall q, q' \in Q$ such that $q \neq q'$, $\exists w \in \Sigma^* :\ \sigma(\delta(q, w)) \neq \sigma(\delta(q', w))$*

**Definition 7.** *An* dfa *$\mathcal{A} =< \Sigma, Q, q_0, F, \delta >$ is* compatible *with a type automaton $\mathcal{T} =< \Sigma, Q_T, q_{0T}, F_T, \delta_T, S, \sigma >$, noted $\mathcal{A}$ is $\mathcal{T}$-compatible, if*

1. *$\forall u, v \in \mathrm{Pref}(L(\mathcal{A})),\ \delta(q_0, u) = \delta(q_0, v) \implies \sigma(u) = \sigma(v)$*
2. *$\forall u \in L(\mathcal{A}),\ u \in L(\mathcal{T})$*

The type automaton $\mathcal{T}$ might be too general relatively to a given dfa $\mathcal{A}$ so that only a part of this automaton is used to type $\mathcal{A}$. Therefore we can define the pruning of the type automaton $\mathcal{T}$ relatively to $\mathcal{A}$:

**Definition 8.** *Let $\mathcal{T} =< \Sigma, Q_T, q_{0T}, F_T, \delta_T, S_T, \sigma_T >$ be a type automaton and $\mathcal{A} =< \Sigma, Q, q_0, F, \delta >$ be a* dfa. *$\mathcal{T}_\mathcal{A} =< \Sigma, Q_{TA}, q_{0T}, F_{TA}, \delta_{TA}, S_{TA}, \sigma_T >$, the automaton obtained by pruning $\mathcal{T}$ relatively to $\mathcal{A}$ is defined by :*

- $Q_{\mathcal{T}\mathcal{A}} = \{q \in Q_\mathcal{T} : \exists u, v \in \Sigma^*,\ \delta_\mathcal{T}(q_{\mathcal{T}0}, u) = q \wedge uv \in L(\mathcal{A})$
- $F_{\mathcal{T}\mathcal{T}} = \{q \in F_T\ \exists u \in L(\mathcal{A}) : \delta_\mathcal{T}(q_{\mathcal{T}0}, u) = q$
- $\forall q \in Q_{\mathcal{T}\mathcal{A}} \forall a \Sigma \delta_{\mathcal{T}\mathcal{A}}(q, a) = q'$ *if* $\exists u, v \in \Sigma^*,\ \delta_\mathcal{T}(q_{\mathcal{T}0}, u) = q \wedge uav \in L(\mathcal{A})$
- $S_{\mathcal{T}\mathcal{A}} = \{s \in S_\mathcal{T} : \exists q \in Q_{TA},\ \sigma_\mathcal{T}(q) = s\}$
- *$\sigma_{\mathcal{T}\mathcal{A}}$ is the restriction of $\sigma_\mathcal{T}$ to $Q_{\mathcal{T}\mathcal{A}}$*

$\mathcal{T}_\mathcal{A}$ is a.....**Here a discussion about the discussion of this object should take place.**

**Proposition 1.** *$\mathcal{A} =< \Sigma, Q, q_0, F, \delta >$ is compatible with a minimal type automaton $\mathcal{T} =< \Sigma, Q_T, q_{0T}, F_T, \delta_T, S, \sigma >$ if and only if*

- $\forall u, v \in \mathrm{Pref}(L(\mathcal{A})),\ \delta(q_0, u) = \delta(q_0, v) \Rightarrow \delta_T(q_0, u) = \delta_T(q_0, v)$
- $\forall u \in L(\mathcal{A}),\ u \in L(\mathcal{T})$

*Proof.* A state in $\mathcal{T}$ has unique type, thus $\Leftarrow$.

Conversily suppose we have $u, v \in \mathrm{Pref}(L(\mathcal{A}))$ such that $\delta(q_0, u) = \delta(q_0, v) \wedge \sigma_\mathcal{T}(u) = \sigma_\mathcal{T}(v) \wedge \delta_\mathcal{T}(q_{0\mathcal{T}}, u) \neq \delta_\mathcal{T}(q_{0\mathcal{T}}, v)$ Then as $\mathcal{T}$ is minimal $\exists w \in \Sigma^*$ such that $\sigma_\mathcal{T}(\delta_\mathcal{T}(q_{0\mathcal{T}}, uw)) \neq \sigma_\mathcal{T}(\delta_\mathcal{T}(q_{0\mathcal{T}}, vw)$. But as $\delta(q_0, u) = \delta(q_0, v)$, $\delta(q_0, uw) = \delta(q_0, vw)$. So in that case by compatibility $\sigma_\mathcal{T}(uw)) \neq \sigma_\mathcal{T}(vw)$, which is a contradiction. □

The definition of a $\mathcal{T}$-compatible automaton induces the definition of a $\mathcal{T}$-compatible language :

**Definition 9.** *A language $L$ is $\mathcal{T}$-compatible if there exists a $\mathcal{T}$-compatible* dfa *$\mathcal{A}$ such that $L(\mathcal{A}) = L$.*

It is possible to know if a dfa is compatible with a given type automaton as stated in the following property:

**Property 1.** *Let $\mathcal{T} =< \Sigma, Q_\mathcal{T}, q_{0\mathcal{T}}, F_\mathcal{T}, \delta_\mathcal{T}, S_\mathcal{T}, \sigma_\mathcal{T} >$ be a type automaton and $\mathcal{A} =< \Sigma, Q, q_0, F, \delta >$ be an* dfa. *It can be decided in $O(\|\mathcal{A}\|)$ time if $\mathcal{A}$ is $\mathcal{T}$-compatible.*

The proof of this property is given by algorithm 3. This algorithm constructs the product automaton of $\mathcal{A}$ and $\mathcal{T}$ (line1) and checks whether $L(\mathcal{A}) \subset L(\mathcal{T})$ (line 2) and whether all the words in $Pref(L(\mathcal{A}))$ leading to the same state in $\mathcal{A}$ have the same type (line 5).

## 3.2 Inference of Typed Languages

The problem of regular inference may be viewed as a search in the regular automata space. The search space has been characterized by Dupont *et al.* [DMV94] as a partition lattice. In this section, we propose an extension of this framework to typed languages inference.

Let $\mathcal{A} =< Q, \Sigma, \delta, q_0, F >$ be a dfa and $\pi$ a partition of $Q$ the set of states. This partition induces an equivalence relation on the states. The quotient automaton $\mathcal{A}/\pi =< Q_\pi, \Sigma, \delta_\pi, q_{\pi 0}, F_\pi >$ is defined as :

**Algorithm 3** The automata typing algorithm

```
1: compute $\mathcal{A} \times \mathcal{T}$
2: if there exists $(q, q')$ a state of $\mathcal{A} \times \mathcal{T}$ such that $(q, q') \in F \times Q' \backslash F'$ then
3:     return false
4: end if
5: if for all $(q, q_1)$ and $(q, q_2)$ two states of $\mathcal{A} \times \mathcal{T}$, $\sigma'(q_1) = \sigma'(q_2)$ then
6:     return true
7: else
8:     return false
9: end if
```

- $Q_\pi = Q/\pi$ is the set of equivalence classes defined by the partition $\pi$
- $\delta_\pi$ is a function $Q_\pi \times \Sigma \rightarrow 2^{Q_\pi}$ such that $\forall q_\pi, q'_\pi \in Q_\pi \; \forall a \in \Sigma$, $\delta_\pi(q_\pi, a) = q'_\pi \iff \exists q \in q_\pi \; \exists q' \in q'_\pi, \; \delta(q, a) = q'$
- $q_{\pi 0}$ is the equivalence class to which belongs $q_0$.
- $F_\pi$ is the set of equivalence classes to which belong at least one state $q$ such that $q \in F$.

We define the following relation on the set of all possible quotient automata of a deterministic automaton :

**Definition 10.** *Let $\mathcal{A} =< Q, \Sigma, \delta, q_0, F >$ be a deterministic automaton and $\pi_1$ and $\pi_2$ be two partitions on $Q$. We say that $\mathcal{A}/\pi_2$ derives from $\mathcal{A}/\pi_1$, noted $\mathcal{A}/\pi_2 \preceq \mathcal{A}/\pi_1$, if $\pi_1$ is finer than $\pi_2$.*

This relation is a partial order on the set of all possible deterministic quotient automata of a deterministic automaton and we have the following property [FB75]:

**Property 2.** *If an automaton $\mathcal{A}/\pi_j$ derives from an automaton $\mathcal{A}/\pi_i$ then $L(\mathcal{A}/\pi_i) \subseteq L(\mathcal{A}/\pi_j)$.*

**Definition 11 (Positive and negative $\mathcal{T}$-compatible sample set).** *Let $\mathcal{T}$ be a type automaton. A positive $\mathcal{T}$-compatible sample set $I_+$ of a $\mathcal{T}$-compatible language $L$ is a finite subset of $L$. A negative $\mathcal{T}$-compatible sample set $I_-$ is a finite subset of $L(\mathcal{T}) \backslash L$.*

**Definition 12 (Structural completeness).** *A sample set $I_+$ is structurally complete relatively to a* dfa *$\mathcal{A} =< Q, \Sigma, \delta, q_0, F >$ if and only if*

1. *$\forall q \in Q \; \forall a \in \Sigma, \; \delta(q, a) = q' \implies \exists u, v \in \Sigma^* : \delta(q_0, u) = q \wedge uav \in I_+$*
2. *$\forall q \in F \; \exists u \in I_+ : \delta(q_0, u) = q$*

**Definition 13 ($\mathcal{T}$-compatible Prefix Tree Acceptor).** *Let $I_+$ be a $\mathcal{T}$-compatible sample set and $\mathcal{T}$ a type automaton. The $\mathcal{T}$-compatible prefix tree acceptor of $I_+$ according to $\mathcal{T}$ is the smallest $\mathcal{T}$-compatible* dfa *accepting exactly $I_+$ in which each state has at most one predecessor.*

**Definition 14 (Prefix Tree Acceptor).** *Let $I_+$ be a sample set of some regular language $L$; the prefix tree acceptor of $I_+$, denoted* $\mathrm{PTA}(I_+)$ *is the smallest $\mathcal{T}$-compatible* dfa *accepting exactly $I_+$ in which each state has at most one predecessor.*

**Property 3.** *Let $L$ be a $\mathcal{T}$-compatible language for some type automaton $\mathcal{T}$. Then* $\mathrm{PTA}(I_+)$ *is $\mathcal{T}$-compatible.*

*Proof.* Trivial *????* □

We extend the definition of a partition to $\mathcal{T}$-compatible dfa.

**Definition 15 ($\mathcal{T}$-compatible partition).** *Let $\mathcal{T}$ be a type automaton and let $\mathcal{A}$ be a $\mathcal{T}$-compatible* dfa. *A partition $\pi$ of $Q$ is $\mathcal{T}$-compatible if $\mathcal{A}/\pi$ is $\mathcal{T}$-compatible.*

**Proposition 2.** *Let $\pi$ be a partition of $Q$. $\pi$ is $\mathcal{T}$-compatible iff*

1. *$Q'$ is deterministic*
2. *$\forall q, q' \in Q, q \in q_\pi$ and $q' \in q_\pi \implies \delta_\mathcal{T}(q_0, w) = \delta_\mathcal{T}(q_0, w)$*

*Ceci est une premiere passe : le but est de dire qu'il faut une correspondance de sommets. A revoir.*

*Proof.* NOT Trivial *????* □

The relation $\preceq$ is a partial order on the set of $\mathcal{T}$-compatible deterministic automata derived from a $\mathcal{T}$-compatible dfa $\mathcal{A}$. We consider the restriction of $\preceq$ to the set of $\mathcal{T}$-compatible deterministic quotient automata, denoted $\preceq_T$. Then we have the following proposition :

**Proposition 3.** *Let $\mathcal{A}_T$ be a $\mathcal{T}$-compatible* dfa. *The set of all $\mathcal{T}$-compatible quotient automata of $\mathcal{A}_T$ together with the partial order $\preceq_T$ is a finite lattice denoted $Lat_T(\mathcal{A}_T)$ for which $\mathcal{A}_T$ is the lowest upper bound and the type automaton $\mathcal{T}$ is the greatest lower bound.*

*Proof.* NOT Trivial *????* □

We can now define the problem of the inference of a $\mathcal{T}$-compatible regular language in the framework of $\mathcal{T}$-compatible deterministic automata lattice :

**Corollary 4.** *Let $\mathcal{T}$ a type automaton and $I_+$ be a sample set structurally complete relatively to some $\mathcal{T}$-compatible* dfa *$\mathcal{A}$; then $\mathcal{A}$ belongs to $Lat_T(PTA(I_+))$.*

*Proof.* NOT Trivial *????* □

## 4 Learning from Inclusion

## 5 Applications

We visit or revisit a number of applications where typing has been tried, with diverse formalisms, and propose the Type automaton for each case.

## 5.1 DNA family representation

$< \Sigma, Q, q_0, F, \delta, S, \sigma >$

- $\Sigma = \{A, G, C, U\}$
- $Q = \{0, 1, 2\}$
- 0 is the initial state
- $F = Q$ is the set of final states
- $\forall x \in \Sigma \delta(0, x) = 1, \delta(1, x) = 2, \delta(2, x) = 0$
- $S = \{0, 1, 2\}$
- $\forall q \in Q : \sigma(q) = q$

## 5.2 Many-Sorted Terms

We intend to model heaps of objects. Each object has a shape (square, circle) and a colour (red, blue). A heap is represented by a term ontop(object(blue,square),ontop(object(red, square),nothing)). We give the following type automaton:

- $\Sigma = \{ontop, nothing, red, blue, circle, square, object\}$
- $Q = \{0, 1, 2, 3, 4\}$
- 0 is the initial state
- $F = 4$
- $\delta(0, red) = 1$
- $\delta(0, blue) = 1$
- $\delta(0, square) = 2$
- $\delta(0, circle) = 2$
- $\delta(12, object) = 3$
- $\delta(0, nothing) = 4$
- $\delta(34, ontop) = 4$
- $S = \{0, 1, 2\}$
- $\forall q \in Q : \sigma(q) = q$

## 5.3 Natural Language

¡In tasks involving the learning of a language model, typing may be used to make a difference between syntactically correct sentences and semantically correct ones. An approximative hypothesis would be to consider that every semantically correct sentence is syntactically correct. Typing can thus reduce the search space by allowing us to consider only sentences that can be parsed through a very generasl grammar that could be used as a first filter.

Let us consider for example the alphabet

$$\Sigma = \{John, Mary, loves, the, woman\}$$

and the set of sorts $S = \{Start, D, N, V, PN\}$ that can be interpreted as determinant, noun, verb and proper noun. In this example *symbols* are english words and *words* are english sentences. A possible Type automaton is

- $\Sigma$
- $Q = \{0, 1, 2, 3, 4, 5\}$
- 0 is the initial state
- $F = 2, 5$
- $\sigma(1) = Start$, $\sigma(2) = PN$,$\sigma(3) = V$,$\sigma(4) = D$ and $\sigma(5) = N$
- $\delta(1, John) = 2; \delta(1, Mary) = 2; \delta(2, loves) = 3; \delta(3, John) = 2; \delta(3, Mary) = 2; \delta(3, the) = 4; \delta(4, woman) = 5$

From examples, the words (english sentences) *"John loves Mary"*, *"John loves the woman"* and *"John loves Mary loves Mary"* are correctly typed. The words *"the woman loves John"* and *"loves the Mary"* are not correctly typed. The type automaton representing the $\Gamma_1$-multi-signature S is shown on Figure 1. States correspond to sorts and edges are labeled by elements of the alphabet according to the $\Gamma_1$-multi-signature.

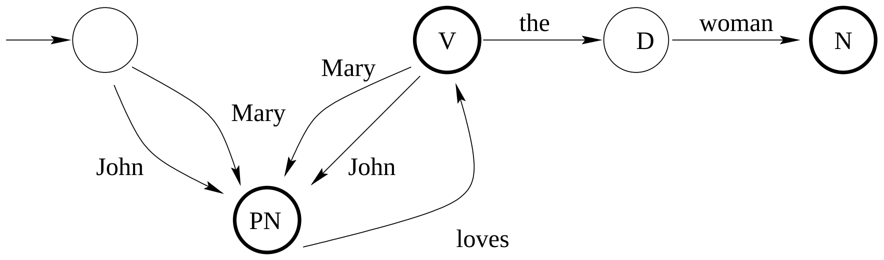

Figure 1: Type automaton

## 5.4 Loop-free automata

In [RST95], it is proved that any loop free dfa is equivalent to another automaton which has the following property : $\delta(q_0, u) = \delta(q_0, v) \implies |u| = |v|$.

As the result does not depend on the probabilities, it can be applied both to the dfa case and the PFA case. To obtain loop-free automata it is thus sufficient to restrict ourselves to the following typed automata

- $\Sigma$
- $Q = \mathbb{N}$
- 0 is the initial state
- $F = Q$
- $\forall x \in \Sigma \delta(i, x) = i + 1$
- $\forall q \in Q : \sigma(q) = q$

## References

[Ang87] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Control*, 39:337–350, 1987.

[BH01] M. Bernard and A. Habrard. Learning stochastic logic programs. International Conference on Inductive Logic Programming, Work in progress session, 2001.

[Bos96] H. Boström. Theory-Guided Induction of Logic Programs by Inference of Regular Languages. In *13th International Conference on Machine Learning*. Morgan Kaufmann, 1996.

[CO94] R. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *Grammatical Inference and Applications, ICGI'94*, number 862 in Lecture Notes in Artificial Intelligence, pages 139–150. Springer Verlag, 1994.

[dlH97] C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27:125–138, 1997.

[dlHB01] C. de la Higuera and M. Bernard. Apprentissage de programmes logiques par infrence grammaticale. *Revue d'Intelligence Artificielle*, 14(3):375–396, 2001.

[dlHOV96] C. de la Higuera, J. Oncina, and E. Vidal. Identification of DFA: data-dependent versus data-independent algorithm. In *Grammatical Inference: Learning Syntax from Sentences*, number 1147 in Lecture Notes in Artificial Intelligence, pages 313–325. Springer Verlag, 1996.

[DMV94] P. Dupont, L. Miclet, and E. Vidal. What is the search space of the regular inference? In *Grammatical Inference and Applications, ICGI '94*, number 862 in Lecture Notes in Artificial Intelligence, pages 25–37. Springer Verlag, 1994.

[FB75] K. S. Fu and T. L. Booth. Grammatical inference: Introduction and survey. part i and ii. *IEEE Transactions on Syst. Man. and Cybern.*, 5:59–72 and 409–423, 1975.

[Fer01] H. Ferna. Learning xml grammars. In P. Perner, editor, *Machine Learning and Data Mining in Pattern Recognition MLDM'01*, number 2123 in LNCS, pages 73–87. Springer-Verlag, 2001.

[GBE96] 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.*, Stanford, CA, 1996. AAAI Press.

[Gol67] E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[Gol78] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[LPP98] K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference*, number 1433 in Lecture Notes in Artificial Intelligence, pages 1–12. Springer-Verlag, 1998.

[Mug99] S. Muggleton. Inductive Logic Programming. In *The MIT Encyclopedia of the Cognitive Sciences (MITECS)*. MIT Press, 1999.

[OG92] J. Oncina and P. García. Identifying regular languages in polynomial time. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition*, volume 5 of *Series in Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific, 1992.

[RST95] D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. In *Proceedings of COLT 1995*, pages 31–40, 1995.

[Sak97] Y. Sakakibara. Recent advances of grammatical inference. *Theoretical Computer Science*, 185:15–45, 1997.

[SM00] Y. Sakakibara and H. Muramatsu. Learning context-free grammars from partially structured examples. In *Proceedings of 5th International Colloquium on Grammatical Inference (ICGI-2000)*, number 1891 in Lecture Notes in Artificial Intelligence, pages 229–240. Springer-Verlag, 2000.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the Association for Computing Machinery*, 27(11):1134–1142, 1984.

[YLT00] M. Young-Lai and F. W. Tompa. Stochastic grammatical inference of text database structure. *Machine Learning*, 40(2):111–137, 2000.