Automata in Natural Language Processing

Jimmy Ma

Technical Report n^o0834, December 2008
revision 2002

VAUCANSON has been designed to satisfy the needs of the automaticians. There are, however, other fields where automata are widely used.
This prospective report will focus on linguistics, and more precisely the automata needs of the natural language processing community since the applications differ from our usual ones. This way, we will be able to assess our range of applicability and usability.
First, we will explore the different domains and detail the various needs. Then, we will be able to evaluate whether or not VAUCANSON is usable for those domains and, finally, discuss some potential work leads.

Jusqu'à présent, VAUCANSON s'adressait essentiellement à la communauté des automaticiens. Cependant, il existe d'autres domaines où les automates sont utilisés.
Dans ce rapport prospectif, nous nous orienterons vers la communauté des linguistes et, plus précisément, vers le domaine du traitement automatique des langues naturelles car les besoins de ce domaine diffèrent de nos utilisations habituelles.
Nous observerons diverses spécialités et dresserons un bilan des besoins en automates. Ainsi, nous pourrons évaluer l'adéquation de VAUCANSON pour ce domaine et en dégager des pistes d'évolutions éventuelles pour le projet.

Keywords
Automata, Natural Language Processing

Laboratoire de Recherche et Développement de l'Epita
14-16, rue Voltaire – F-94276 Le Kremlin-Bicêtre cedex – France
Tél. +33 1 53 14 59 47 – Fax. +33 1 53 14 59 22
jimmy.ma@lrde.epita.fr – http://www.lrde.epita.fr/

Copying this document

Contents

Chapter 1

Introduction

The VAUCANSON project is a finite state machine manipulation platform. It has been designed to satisfy the needs of the automaticians in terms of performance and genericity. The goal is to provide a complete framework that would fit any use of automata. There are, however, other fields where automata are widely used.

Natural Language Processing is a field that is part of both computer sciences and linguistics. It aims to process natural languages automatically with the less human supervision possible. A lot of its domains use finite-state machines with requirements that differ from our usual needs.

Automata are seen as an efficient representation of data such as morphological rules or lexicons. They outperform in terms of speed and space the previous representations. In practice, those applications have shown to be stable and quite easy to implement and to setup (Kaplan and Kay, 1994). With those ideas in mind, computational linguistics have invested some efforts into the automata theory.

Since the begin, VAUCANSON always insisted on its generic aspects. So far, we have done a lot to improve it, with the latter work to integrate a new data structure (Lazzara, 2008), we have fixed a lot of issues. By focusing on natural language processing, we intend to assess our range of applicability and usability. However, in recent work, some important performance loss were established. This problem is currently being resolved and therefore, no benchmarking will be done here. This report is more to be treated as a preliminary work that would be realised before upgrading VAUCANSON into a natural language processing platform.

First, we will briefly introduce some general notions about automata and transducers that will be used later on. Then, we will explore the different domains and detail the various needs. This will cover most fields of the linguistics, from morphology to syntax with some semantics. Finally, we will evaluate whether or not VAUCANSON is usable for those domains and discuss some potential future work leads.

Chapter 2

Automata theory

Our goal in this chapter is to introduce the automata theory for the needs of linguistics. Therefore, in the following sections and in the rest of the paper, we will only consider the use of finite-state machines.

This section will not resume the whole theory behind each mathematical object that will be introduced. It will briefly define each item to give markers to the reader before the next chapter.

2.1 Boolean Automata

2.1.1 Definitions

An automaton is represented by the 5-tuple $\langle Q, \Sigma, \delta, I, F \rangle$ where:

- Q is a set of states,
- Σ is a finite set of symbols, also called the *alphabet* of the language recognized by the automaton,
- δ is a transition function:
  $\delta : Q \times \Sigma \rightarrow Q$,
- I is a set of states of Q called *initial states*,
- F is a set of states of Q called *accepting states*.

A less formal definition of δ would be: Δ is a set of transitions between a pair of states of Q with a label that is a letter of Σ.

An automaton is accepting a word whenever we have reached an accepting state and the input is empty.

Deterministic Finite Automata (DFA)

A DFA is a kind of automata where each state has an outgoing transition for each letter of the alphabet. For example, Figure 2.1 is an deterministic finite automaton. Notice that in a DFA, there can only be one initial state.

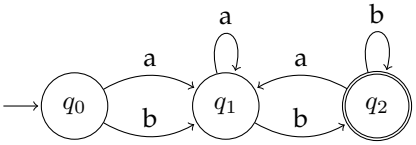


Figure 2.1: A deterministic finite automaton.

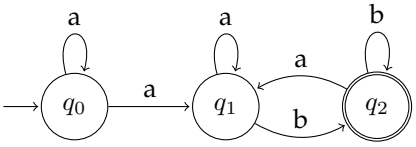


Figure 2.2: A nondeterministic finite automata.

Nondeterministic Finite Automata (NFA)

A NFA does not need to have for each state an outgoing transition for each letter of the alphabet. Furthermore, it can even have for a state more than one outgoing transition labeled by the same letter. For example, Figure 2.2 is a nondeterministic finite automaton.

Nondeterministic Finite Automata with ε-transitions (ε-NFA)

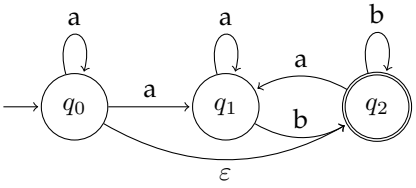


Figure 2.3: A nondeterministic finite automata with ε-transitions.

A ε-NFA is a NFA with an additional possibility for a label. It can have transitions without letters, they are marked with an ε and they are called ε-transitions. For example, Figure 2.3 is a nondeterministic finite automaton. An ε-transition is a transition that can be taken without any letter, therefore, in Figure 2.3, given one letter when one is in the state q_0, one can also take the transitions of the state q_2.

Pushdown Automata

A pushdown automaton is a variation of a finite-state automaton, the idea is to combine the use of a stack. It is formally defined as a 6-tuple $\langle Q, \Sigma, \Gamma, T, I, F \rangle$ where:

- Q is a set of states,
- Σ is a finite set of symbols, also called the *alphabet* of the language recognized by the automaton,

- I is a set of states of Q called *initial states*,
- F is a set of states of Q called *accept states*,
- Γ is a stack that can holds letter of the alphabet Σ,
- T is a transition function:
  $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to \mathcal{P}(Q)$.

In addition, when taking a transition, a letter can be pushed or popped if it matches the preceding condition. As to the evaluation, a pushdown is accepting a word when: it ends on an accepting state, its input is empty and its stack is also empty.

Pushdown automata are more powerful since they can recognize languages such as $a^n b^n$, see Figure 2.4.

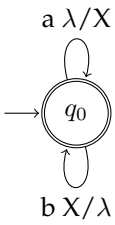


Figure 2.4: A pushdown automaton that recognize the language $a^n b^n$.

2.1.2 Algorithms

Among all the algorithms that exists on automata, we will focus on two in particular: determinization and minimization. They are the most used algorithms in the linguistic field.

Determinization

With regards to computation time, it is generally better to have deterministic automata. However, their construction is not always intuitive with regards the the current problem. The following will be very useful in those cases.

Any automaton $\mathcal{A}$ is equivalent to an automaton $\mathcal{B}$ that is both complete and deterministic. If $\mathcal{A}$ is finite with n states, we can build $\mathcal{B}$ so that it has at most 2^n states.

In *Éléments de théorie des automates* (Sakarovitch, 2003), there is an algorithm to determinize a boolean automaton by using the subset construction. It consist in computing the accessible part of $\mathcal{B}$ gradually. Figure 2.6 shows an example of an automaton and the result of its determinization.

Minimization

Concerning memory space consumption, it is better to have a minimized automaton. For any automaton, we can as well build its minimal equivalent, the algorithm was detailed by John E. Hopcroft (Hopcroft and Ullman, 1979).

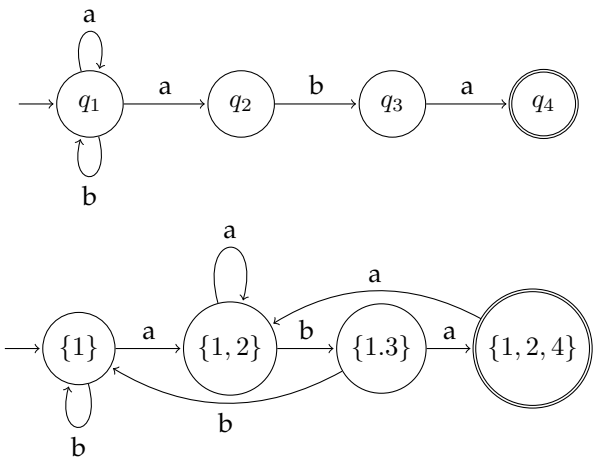


Figure 2.5: An automaton and its deterministic version (below).

2.2 Transducers

In this section, we will only consider the use of sequential transducers.

2.2.1 Definitions

A sequential string-to-string transducer is a 7-tuple $\langle Q, i, F, \Sigma, \Delta, \delta, \sigma \rangle$ where:

- Q is a set of states,
- i is the initial state,
- F is a set of states of Q call the final states,
- Σ and Δ are finite sets corresponding respectively to the input and the output alphabets of the transducer,
- δ is the state transition function
  $\delta : Q \times \Sigma \to Q$,
- σ is the output function
  $\sigma : Q \times \Sigma \to \Delta^*$.

The intuitive approach is to consider a transducer as an automaton that reads an input tape and can write on a output tape. It has one additional property, it is not capable of going backward on any of the tapes.
Figure 2.7 shows a transducer that converts the word *aaa* to *bbb* and *abb* to *baa*.

2.2.2 Algorithms

Transducers face problems that are similar to those of the automata with regards to space and time. The solutions are in fact very similar, on transducer, we also have a determinization (Mohri, 1996b) and a minimization (Mohri, 1996a) algorithms with similar properties.

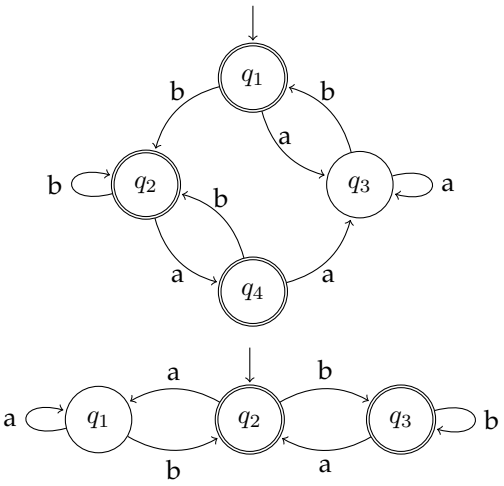


Figure 2.6: An deterministic automaton and its minimized version (below).

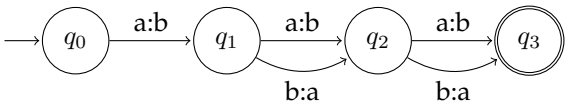


Figure 2.7: A sample transducer.

2.3 Probabilistic Automata and Hidden Markov Models

2.3.1 Hidden Markov Models

A discrete Hidden Markov Model (HMM) is defined by the 5-tuple $\langle Q, \Sigma, A, B, \iota \rangle$ where:

- Q is a set of states,
- Σ is a finite set of symbols, also called the *alphabet*,
- A is mapping defining the probability of each transition:
  $A : Q \times Q \to [0, 1]$,
- B is mapping defining the emission probability of each letter on each state:
  $B : Q \times \Sigma \to [0, 1]$,
- ι is mapping defining the initial probability of each state:
  $\iota : Q \to [0, 1]$.

The following constraints must be satisfied:

- $\forall q \in Q, \sum_{q' \in Q} A(q, q') = 1$
- $\forall q \in Q, \sum_{a \in \Sigma} B(q, a) = 1$

- $\sum_{q \in Q} \iota(q) = 1$

A discrete Hidden Markov Model with transition emission (HMMT) is a 5-tuple $\langle Q, \Sigma, A, B, \iota \rangle$ where:

- Q is a set of states,
- Σ is a finite set of symbols, also called the *alphabet*,
- A is mapping defining the probability of each transition:
  $A : Q \times Q \to [0, 1]$,
- B is mapping defining the emission probability of each letter on each state:
  $B : Q \times \Sigma \to [0, 1]$,
- ι is mapping defining the initial probability of each state:
  $\iota : Q \to [0, 1]$.

The following constraints must be satisfied:

- $\forall q \in Q, \sum_{q' \in Q} A(q, q') = 1$
- $\forall q \in Q, \sum_{a \in \Sigma} B(q, a) = \begin{cases} 1 & \text{if } A(q, q') > 0 \\ 0 & \text{otherwise} \end{cases}$
- $\sum_{q \in Q} \iota(q) = 1$

On a HMM, the letter emission are made on the states whereas the are made on the transitions on a HMMT. Along any path on a HMM, the emission process is Markovian since the probability of emitting a letter on a given state only depends on that state.

2.3.2 Probabilistic Automata

A probabilistic automata is defined by the 5-tuple $\langle Q, \Sigma, \phi, \iota, \tau \rangle$ where:

- Q is a set of states,
- Σ is a finite set of symbols, also called the *alphabet*,
- ϕ is mapping defining the transition probability function:
  $\phi : Q \times \Sigma \times Q \to [0, 1]$,
- ι is mapping defining the initial probability of each state:
  $\iota : Q \to [0, 1]$,
- τ is mapping defining the final probability of each state:
  $\tau : Q \to [0, 1]$.

The following constraints must be satisfied:

- $\sum_{q \in Q} \iota(q) = 1$
- $\forall q \in Q, \tau(q) + \sum_{a \in \Sigma} \sum_{q' \in Q} \phi(q, a, q') = 1$
- $\forall q \in Q, P_{A_q}(\Sigma^*) = \sum_{q'} \phi(q, \Sigma^*, q')\tau(q') > 0$

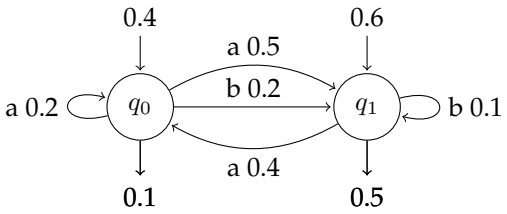


Figure 2.8: A probabilistic automaton.

For example, Figure 2.8 is a probabilistic automata.

A probabilistic automaton with no final probabilities (PNFA) is a probabilistic automata where the set of final states is empty. Let $A = \langle Q, \Sigma, \phi, \iota, \tau \rangle$ be a probabilistic automaton, A is a PNFA when $\forall q \in Q, \tau(q) = 0$.

2.3.3 Linking Hidden Markov Models and Probabilistic Automata with no final probabilities (Dupont et al., 2005)

It has been proven that probabilistic automata with no final probabilities are equivalent to hidden Markov models.

For all PNFA, there exists an equivalent HMMT with the same number of states. Similarly, for all HMMT $M = \langle Q, \Sigma, A, B, \iota \rangle$, there exists an equivalent HMM $M' = \langle Q, \Sigma, A, B, \iota \rangle$. However, in this case, the number of states of M' is less or equal to $|Q|^2$. And finally, for all HMM, there exists an equivalent PNFA.

It has been proven that it was possible to simulate a HMM with n states by a PNFA with n states. It is possible to simulate a PNFA by a HMM, however, in this case, the number of states is polynomially related.

Chapter 3

Natural Language Processing

3.1 What is Natural Language Processing

Natural language processing is a field of linguistics and computer science which focuses on processing natural language.

We define as a natural language a human spoken language, in opposition to artificial languages such as computer languages C or Cobol. In other words, a natural language is nothing more than a spoken language such as French, English or Inuktitut.

The main goal of this field is to make human languages automatically processable. Therefore, it implies finding techniques to convert an utterance which can be either spoken or written, into formal data. Formal data are a representation of that utterance that can be processed using a computer and with no or minimal supervision.

Computer science has been aiming at processing natural language since its early ages. During the Cold War, in order to intercept and to process Russian messages quickly, the US Government had a program of translation. Norbert Wiener explained part of it in 1948 (Wiener, 1948), and a few years later Bar Hillel joined his work. Even though they seemed very optimistic, their results have been detailed in a report concluding that it will never work (Pierce and Carroll, 1966). Since then, research in that field have slowed down considerably.

There have been, however, some positive results. In 1971, Terry Winograd published his work on SHRDLU (Winograd, 1971). It is a program in which the user could interact using English sentences. By understanding some actions and facts told by the user, SHRLDU could deduce what to do and respond correctly. It was even able to perform some anaphoric inferences and ask confirmation about them (Figure 3.1). This program was one of the first to respond to user requests given in natural languages.

```
User  : How many things are on top of green cubes?
SHRDLU: I'm not sure what you mean by "on top of" in the phrase
        "on top of green cubes".
        Do you mean:
         1 - directly on the surface
         2 - anywhere on top of?
```

Figure 3.1: SHRDLU, sample conversation.

Some has continued in the trail of SHRDLU, however, nothing has really succeeded in this

approach. Mainly because too many data was hard coded and in order to extend it, the systems became too hard to maintain. Those approaches have then become subject to less investigations and most are back to studying various formalisms of the grammar.

In the forthcoming sections, we will detail some part of natural language processing that can rely on automata theory. The use of automata or transducers is often very appropriate since it mechanisms are often close to the way one may think the algorithms (Kaplan and Kay, 1994).

3.2 Lexicon

3.2.1 Context

In linguistics, a lexicon is the vocabulary of a person, group or language. It contains all the minimal grammatical elements of a language. In a sense, it represents a speaker's knowledge of the vocabulary. This is comparable to a dictionary, however, it does not necessarily include the definitions of the entries. Furthermore, it can carry only part of words such as suffixes.

```
did,V:PRET
do,N:s
do,V:INF
doe,N:s:p
does,V:P3s
does,N:p
done,V:PP
done,A
```

Figure 3.2: A sample lexicon.

Figure 3.2 shows a sample lexicon corresponding to the entry of the word *do* and some of its derivatives. There is the infinitive form, the past participle and the present. There is also some homonyms, the adjective *done* and also the noun *doe* and its plural. This formalism inspired by the DELAS (Courtois, 1985) is the one used in the English DELAS. The part before the comma correspond to the actual word and the second part is grammatical information on the entry. The "N" stands for "noun", "V" for "verb" and "A" for "adjectives". After what, there are additional information such as the form of the entry when it is a verb, its gender and plurality for a noun, etc.

This comes in handy when processing a text automatically because it allows automatic tagging of the text using only simple text comparison. However, the naive approach can be very expensive in terms of computing and memory usage.

3.2.2 From listings to automata

The naive approach to represent a lexicon is to simply list the entry in a file. The first optimisation would be to sort the entries alphabetically to fasten the lookup.

In his Ph.D Thesis, Dominique Revuz showed that efficient representation of lexicons were possible using finite-state automata (Revuz, 1991). The finite-state automata used in this case are generally boolean. Figure 3.3 displays the automaton that is equivalent to the lexicon of Figure 3.2. The building process is natural when considering how automata work.

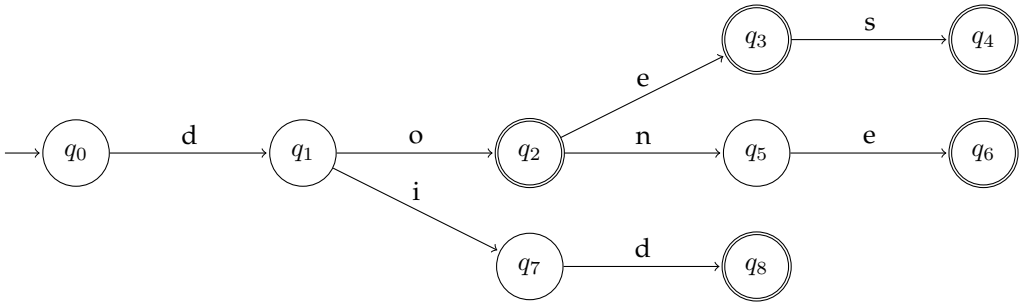


Figure 3.3: A lexicon represented by an automaton.

It is possible to reach a good compromise between the size of the automaton and the speed of the access. Aside from the representation chosen for the automaton, determinizing it can improve the access and minimizing it can reduce the number of states considerably. Those algorithms become useful when the automata are handcrafted. The building process of the automaton is in this case very natural, and the determinized and minimized automaton has good performances in terms of memory consumption and processing time.

In order to update the input with the appropriate tags, it is easier to switch from automata to transducers. Therefore, the idea would be to add the DELAS type information directly into the transducers. To do so, we need to take the automaton from Figure 3.3 and the information from Figure 3.2, and then, for each accepting state of the automaton, output the matching entry from the lexicon.

Lexicon	FDELAF	GDELAF	EDELAF
Initial size	21.2Mb	27.9Mb	3.6Mb
Automata size	1.2Mb	3.1Mb	470Kb

Table 3.1: Lexicon representations: memory consumption. (Mohri, 1996b)

Using efficient finite-state machine implementation can then improve the memory consumption by a factor up to 18 (Table 3.1). In this table, the lexicons used are the French, the German and the English DELAF. A DELAF is a lexicon of simple inflicted forms of words. Its mechanism is similar to the one of the DELAS introduced in subsection 3.2.1. The first line indicates the size of the DELAF in a text format as they are distributed over the internet. The second line is the size of the automaton that represent the same data.

3.3 Morphology and Phonology

3.3.1 Definition

Morphology is a field of the linguistics that focuses on the study of the internal structure of a word. It is the smallest units of syntax. This field analyses the word production rules that are common to the speakers of a language.

This is part of the neologism phenomenon. There is a wide range of rules:

- N+"s" for the plural (*dog*, *dogs*),
- V+"s" for the third form of the present (*read*, *reads*),
- V+"ed" for the past participle of most verbs (*match*, *matched*),
- V+"er" to designate the one who (*to hunt*, *the hunter*),
- Adj+"ly" to qualify a way of doing things (*slow*, *slowly*),
- Political figure+"ism" to refer the political party of this figure (*Bush*, *bushism*).

Some words may not figure in any dictionary but they are based on existing rules so most speakers will be able to understand their meaning easily.

Phonology is similar to morphology in a sense that it also studies the structure of a word, but it focus only on the spoken aspects.

3.3.2 Compiling rules in an automaton

The structures of the morphological rules is very similar to lexicons. Hence, the use of automata or transducers is also efficient in this case. The building process can as well be intuitive and the execution performances improved using the determinize and minimize algorithms.

TAGH is a project that describes the morphological rules for the German language using finite-state automata (Geyken and Hanneforth, 2005). They have achieved a recognition rate of 99% on the corpus of *Die ZEIT* which is a German political newspaper.

Researches at the Xerox Alto Research Center have showed that complete phonological rules could be easily described using a finite-state machine (Kaplan and Kay, 1994). Part of their conclusion was that by using finite-state transducers, the program required to perform the evaluation of a grammar is "simple almost to the point of triviality" without regards to the complexity of the grammar, the interpreter is robust to changes and the compiler is easy to implement.

Some translation system uses transducers to represent their intermediate data (Casacuberta et al., 2002). They building a lexical automaton that they are derivating into a phonological representation before any further processing.

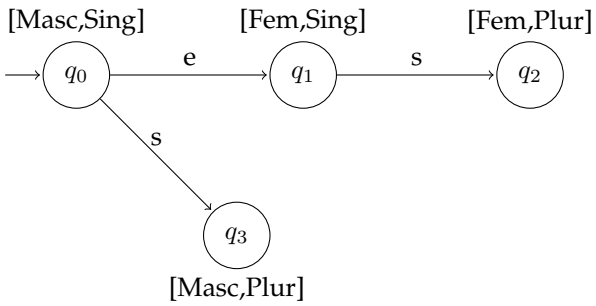


Figure 3.4: Sample of morphological inflection rules of adjectives in French.

Unitex (http://www-igm.univ-mlv.fr/~unitex/) is a program that manipulates transducers. It embeds transducers representing word inflections. Figure 3.4 is a sample automaton representing word inflections in French, they can be useful to combine with other lexical automaton in order to generate part of the lexicon.

3.4 Part-of-Speech Tagging

3.4.1 Context

Part-of-Speech tagging consists in associating grammatical categories to words. The most common categories are *noun*, *verb*, *adjective*, *adverb*, etc.

It is an early stage in natural language processing since most systems rely on its results to continue. It is also subject to a lot of ambiguities and can hardly be resolved without the context of each word. For instance, let us consider the sentence *I saw her duck*:

- *I* can either be a noun or a pronoun,
- *saw* can either be a noun or a verb,
- *her* can either be an adjective, a pronoun or a noun,
- *duck* can either be a noun or a verb.

The problem is that based on which category we assign to each word, the meaning can be different:

- "I have seen her animal which is a duck",
- "I have seen her when she was ducking",
- "I am used to cutting her animal which is a duck with a saw".

This example brings up another difficulty: there are not just one correct tagging for a sentence and each of them leads to a very different analysis of the sentence.

3.4.2 Tagging

The former solution to this problem was to rely only on the data provided by the lexicon. This is problematic because the ambiguities are never solved and the system has to handle them constantly. We will introduce here two solutions: morphological rules to detect a category and a system using hidden Markov model.

Hidden Markov Models are capable of providing good results (Halteren et al., 2001). Their main advantage is that they require only few resources and work since the system is quite easy to setup. Using only trigrams, the system is reaching up to 97.55% of correct answers on some corpus.

Mehrnoush Shamsfard proposed a system using both morphological rules and probabilistic techniques to solve the problem on Persian (Shamsfard and Fadaee, 2008). Their system is reaching the 100% of correctness on their database.

The results given by those probabilistic methods are good. The are totally comparable to the applications of the linguistics rules that are exhibited by a linguist after months of analysis. In terms of work, the amount is also comparable. The linguist has to work on a corpus for months to write emphasize the phenomenon. On the other side, the learning process can be long, on real corpus, it can take up months depending on the size of the n-grams that are considered. N-grams are succession of N words. They are commonly used with automatic systems to define the perspective of the learning process. It takes week when using bigrams and months when using trigrams. It is currently not conceivable to use quadrigrams. Probabilistic systems carries, however, one major step back. The lack of readability of the systems produced blocks most analysis. This is a main reason to why part of the linguistic community would rather work on rules.

3.5 Syntax

3.5.1 Context

Syntax is the study of the rules that govern the correctness of a sentence in a language. "A properly formulated grammar should determine unambiguously the set of grammatical sentences" (Chomsky, 1956). It brings the notion that not only syntax must allow to decide if a sentence is grammatical, it must also allow to define clearly if a sentence is not. It is a strong statement because it implies anything that is recognised by the grammar is a correct sentence and anything that is not recognized is incorrect. This idea, however, has not been reached yet because a natural language is so irregular that no system has successfully achieved that goal.

There are many other approaches that have been developed, some interesting ones rely on lexicon mostly. In the theory that are nowadays still active, we have Lexical-Functional Grammars (Bresnan, 2000), Lexicalized Tree Adjoining Grammars (Sarkar and Joshi, 1996) and Head-driven Phrase Structure Grammars (Pollard and Sag, 1988) In those formalisms, most information about the grammar is associated to the words, for instance, a verb knows whether or not it is transitive, etc.

In the next section, we will focus mainly on part of Generative and Transformational Grammar (Chomsky, 1957) since it is the most appropriate one for automata.

3.5.2 Regular Grammars

Regular grammars are defined by $\langle \Sigma, V, P, S \rangle$ where:

- Σ is the alphabet of the language,
- V is a set of variables or *nonterminals*,
- P is a set of production or rewriting rules,
- S is symbol that represents the first rule to apply.

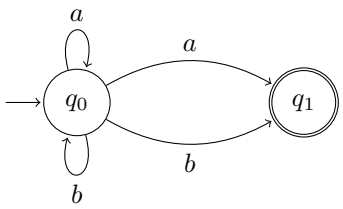
A rule is defined as follows, let α be a string of Σ^*, and let X, Y and Z be a triplets of nonterminals. Any rules of P can either be formed as:

- $X \rightarrow YZ$
- $X \rightarrow \alpha Y$
- $X \rightarrow Y\alpha$
- $X \rightarrow \alpha$

The regular grammars have a small power of expression since the are very restrictive. They can be used to handle small artificial languages or some restricted part of a natural language. They are, however, easy to use and totally equivalent to finite-state automata (Figure 3.5 for example).

3.5.3 Context Free Grammars

The context free grammars are similar to regular grammars but without the restrictions on the length of the rules.

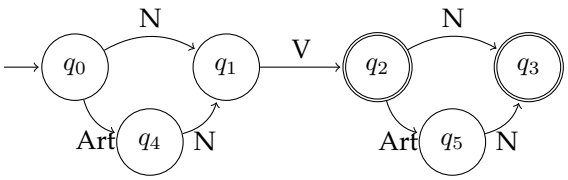


$\Sigma = a, b$
$V = \{S\}$
$P = \{S \to aS, S \to bS, S \to a, S \to b\}$

Figure 3.5: A regular grammar and its equivalent automaton.

They were described by Noam Chomsky in his early work (Chomsky, 1957). What was odd is that he described this grammar in order to prove that it was insufficient for natural language processing. However, in the years that followed, it had been very popular because it was the only implementable system that provided acceptable results.

The expressiveness of these grammars remain good enough as long as one do not try to describe the language with its whole complexity. Figure 3.6 is a simple context free grammar that can recognize basic sentences such as *the girl loves the music*, *a policeman risked his life*, *dog barks* or *dog loves bones*.



S → NP VP
NP → N
NP → Art N
VP → V
VP → V NP

Figure 3.6: A simple context free grammar and its equivalent automaton.

It is important to underline that context free grammars are not equivalent to finite-state automata. Even though Figure 3.6 shows that it can sometimes be represented by one, context free grammars are equivalent to pushdown automata which will not be explained here. This is because context free grammars can generate languages such as $a^n b^n$ which is not regular.

An other option is to consider Recursive Transition Network (Woods, 1970). They rely on the use of automata network and are equivalent to pushdown automata. Instead of using one automaton, we can have one for each rule and whenever we reach a rule on a transition, we evaluate this in the automaton corresponding to the rule. When we reach an accepting state in the sub-automaton, we go back the previous one and resume treatment. An equivalent to the automata on Figure 3.6 could be Figure 3.7.

3.6 Indexation

3.6.1 Context

When working with natural languages, it is common to handle a large corpus of texts. In order to work efficiently, it is mandatory to have fast access to the information required. The web is a good example of that, there are billions of texts and search engines are able to browse through them in only a few milliseconds to output pertinent information.

In order to achieve that, a technique called indexation is used. The basic idea is to create a database of a text containing all the word occurrences. Then, instead of searching through

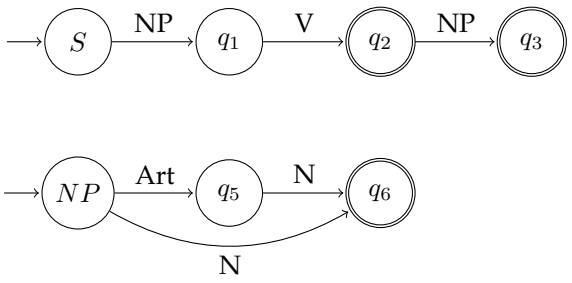


Figure 3.7: An automata network

the raw text data, searches are only done through the index which is a lot faster. A common problem is related to the storage and the access to the data.

3.6.2 Using Finite-State Transducers

Maxime Crochemore devised an algorithm using finite-state transducer that allows a representation in linear space of the index of a text (Crochemore, 1986). In this algorithm, summing the weights of the label during the recognition gives the first occurrence of a word. A specific path is required in order to retrieve all the other references.

This has been improved by Mehryar Mohri (Mohri, 1996b). He presented an algorithm that stores the list of the current recognized word on each node. The occurrences of a word is then retrieved in linear time and the storage space is still linear. Figure 3.8 shows a sample automaton of the text *aabba*. To get the beginning of the current word, one just have to subtract the length of the word to the list associated to the node.

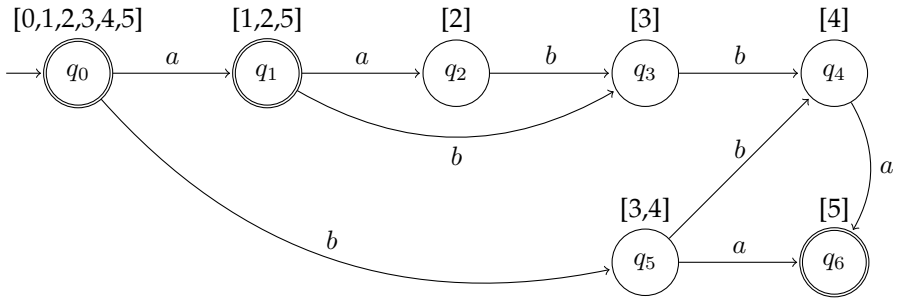


Figure 3.8: Indexation of the text *aabba* by an automaton. (Mohri, 1996b)

3.7 Translation

3.7.1 Context

Translation is a very complex task because it combines problems from almost all the stages of the linguistics.

In semantics, there is the problem of lexical ambiguity. For instance, the word *bank* can be a noun but as a noun it has two distinct meanings: one is *a financial institution*, the other is *an edge of a river*. The problem is even more complicated in translation because it combines the ambiguities of the two languages at once.

Furthermore, we face the part-of-speech tagging problem. For instance, the word *bank* can be either a verb or a noun. It matters because it can lead to very different translation.

Some of the syntactic problems are related to the part-of-speech tagging. With the wrong category on a certain word, it is clearly not possible to produce the right translation. However, even with the correct category, some syntactic structures are not well formalized yet. For instance, connecting a relative clause to its correct reference can become tricky when there are multiple relative clauses that are intertwined.

Finally, there is the problem of alignments. Some words and there translation do not always have the same length, for instance *pomme de terre* in French and *potato* in English. This brings up the problem of compound words and there detection. This can be solved using a lexicon but some composition would rather be detected automatically because they are part of a regular construction, for example the use of *machine à + verb* in French.

3.7.2 Using Hidden Markov Models for Translation

The main idea for solving the alignment problems is to use two lexicons, one in each language, or to use hidden Markov models. Notice that the two can be combined.

An algorithm based on probabilistic transducer was proposed by Hiyan Alshawi (Alshawi et al., 1998). It builds up a translation in three steps, first it learns the alignment, using a bilingual corpus. Then, using the same corpus, it creates the transducer, and finally, this transducer outputs the translated version of the input.

N-Gram used	Recall (R)	Precision (P)	F-Measure (2*P*R / (P+R))
Phrase Unigram	43.7	80.3	56.6
Phrase Bigram	62.5	86.3	72.5
Phrase Trigram	65.5	85.5	74.2

Figure 3.9: Results of a Stochastic model applied to translation (Bangalore, 2000).

Srinivas Bangalore described an algorithm based on N-grams translation models (Bangalore, 2000). It uses the alignment algorithm that was introduced by H. Alshawi (Alshawi et al., 1998). Its representation is based on the Variable N-gram Stochastic Automaton (VNSA) (Riccardi et al., 1996). The idea is to acquire the lexical translations through alignments algorithms and then to produce a VNSA that will later be transformed to a Stochastic Transducer that is also able to reorder the output with regards to the language specifications. The results of this system are quite good Figure 3.9. The recall is the number of sentences correctly translated on the number of expected sentence and the precision is the number of sentences correctly translated on the number of sentences translated. The F-Measure represents a quality measure of the whole system. 74% for an automated translation is nowadays still considered pretty good.

Chapter 4

Using VAUCANSON in Natural Language Processing

Our goal in this chapter is to assess whether VAUCANSON is adequate to natural language processing or not. Through chapter 3, we have established the following types of finite-state machines can be useful to natural language processing: boolean automata and transducers, probabilistic automata, pushdown automata and recursive transition networks. We will then present how VAUCANSON can respond to those requirements.

4.1 The Genericity of VAUCANSON

4.1.1 Alphabets

A first challenge is to be able tho handle various alphabets, for instance, in order to process languages such as Japanese, Persian or Greek. There is, here, a real need for a support of a wide range of alphabets and encoding.

This was one goal of VAUCANSON, to be generic with regards to the alphabets. In theory, this was supported since the beginning, but recent work showed that it still lacked some functionalities. Indeed, it was not possible to change the representation of token such as the empty word for instance. However, this has been fixed by Vivien Delmon last year with a new Rational Expression Parser (Delmon, 2008) that can now handle the input/output part with all the complications implied by the use of a new alphabet.

4.1.2 Weights

The main difference implementation-wise between boolean automata and probabilistic automata is the handling of the weights.

VAUCANSON managed to keep that part generic as well. In fact, it handles any kind of weights as long as the user specifies how to deal with them through a C++ object. This relies on the ELEMENT design pattern that was introduced by Yann Régis-Gianas and Raphaël Poss (Régis-Gianas and Poss, 2003).

Boolean automata are available and stable in VAUCANSON. Considering that probabilistic automata are mainly automata with weights that are reals that belongs to the interval $[0, 1]$,

they are easily implementable in VAUCANSON. Since the real numbers are already available, it is mainly deriving them into bounded real numbers.

4.2 Pushdown Automata and Recursive Transition Networks

Pushdown Automata and Recursive Transition Networks (RTN) are not implemented in VAUCANSON. In order to do so, new automata types need to be added. They are not planned at the moment but they can easily be included.

For instance, VAUCANSON already handles the possibility to have transition labeled by rational expressions. To implement RTN, a lead might be to specialize an automata type to allow call to sub-automata when required.

For the pushdown automata, Michaël Cadilhac has provided a proof of concept that it is possible in VAUCANSON. From his work, the main part would be to specialize the evaluation algorithm in order to take into account the newly added stack.

4.3 Input/Output Data

A last point to consider is the input/output data format. The first ones introduced in VAUCANSON were designed for mathematical automata.

In his recent work, Florian Lesaint has established a new XML format to represent automata (Demaille et al., 2008). It aims to describe an automata without any regards to its kind. This format, even though it still needs to be extended, is prepared to apply itself to the field of natural language processing, therefore, VAUCANSON is ready to input/output automata for natural language processing.

Furthermore, compared to other automata representation, it presents the advantage to be in XML, which is easily readable as opposed to binary representation.

Chapter 5

Conclusion

Though this report, we have shown the interest of the linguistic community towards the Automata Theory. In natural language processing, automata can be used at all the stages. Automata can efficiently represent morphological and phonological rules. Furthermore, the required techniques that compare to others are simple to implement (Kaplan and Kay, 1994). They can also be used at various processing stage of translation (Casacuberta et al., 2002). It has been useful as well with regards to grammars and syntax, in Context Free Grammars (Chomsky, 1957) for instance. It has been shown that automata are great for indexation in terms of memory space consumption and computation time (Mohri, 1996b). Finally, it also has applications in translation (Bangalore, 2000), especially for solving the alignment problems (Alshawi et al., 1998).

The genericity of VAUCANSON should be good enough for the applications of the natural language processing field. It can handle all the alphabets that may be needed, it implements both automata and transducers on various semiring, from boolean to reals. The point it may be lacking is the implementation of functional pushdown automata, but it has been shown that their integration would not be impossible. At last thing it lacks in order to attract the linguistic community may be a graphical interface. However, this is as well a work in progress.

Chapter 6

Bibliography

Alshawi, H., Bangalore, S., and Douglas, S. (1998). Automatic acquisition of hierarchical transduction models for machine translation. In *In Proceedings of the 36th Annual Meeting Association for Computational Linguistics*, pages 41–47.

Bangalore, S. (2000). Stochastic Finite-State models for Spoken Language Machine Translation. In *In Proceedings of the Workshop on Embedded Machine Translation Systems*, pages 52–59.

Bresnan, J. (2000). *Lexical-Functional Syntax*. Blackwell Publishers.

Casacuberta, F., Vidal, E., and Vilar, J. M. (2002). Architectures for speech-to-speech translation using Finite-State models.

Chomsky, N. (1956). Three models for the description of language. *Information Theory, IEEE Transactions on*, 2(3):113–124.

Chomsky, N. (1957). *Syntactic Structures*. Mouton and Co, The Hague.

Courtois, B. (1985). Le sysème DELAS. *Rapport Technique du LADL*, 4.

Crochemore, M. (1986). Transducers and repetitions. *Theor. Comput. Sci.*, 45(1):63–86.

Delmon, V. (2008). Rational expression parser. Technical report, EPITA Research and Development Laboratory (LRDE).

Demaille, A., Duret-Lutz, A., Lesaint, F., Lombardy, S., Sakarovitch, J., and Terrones, F. (2008). An XML format proposal for the description of weighted automata, transducers, and regular expressions. In *Proceedings of the seventh international workshop on Finite-State Methods and Natural Language Processing (FSMNLP'08)*, Ispra, Italia.

Dupont, P., Fran c. D., and Esposito, Y. (2005). Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38(9):1349–1371.

Geyken, A. and Hanneforth, T. (2005). TAGH: A Complete Morphology for German Based on Weighted Finite State Automata. In *FSMNLP*, pages 55–66.

Halteren, H. V., Zavrel, J., and T, W. D. (2001). Improving accuracy in word class tagging through the combination of machine learning systems. *Computational Linguistics*, 27.

Hopcroft, J. E. and Ullman, J. D. (1979). Introduction to Automata Theory, Languages, and Computation.

Kaplan, R. M. and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, 20:331–378.

Lazzara, G. (2008). Boosting Vaucanson's genericity. Technical report, EPITA Research and Development Laboratory (LRDE).

Mohri, M. (1996a). Minimization of sequential transducers. *Lecture Notes in Computer Science*.

Mohri, M. (1996b). On Some Applications of Finite-State Automata Theory to Natural Language Processing. *Journal of Natural Language Engineering*, 2:1–20.

Pierce, J. R. and Carroll, J. B. (1966). *Language and Machines: Computers in Translation and Linguistics*. National Academy of Sciences/National Research Council, Washington, DC, USA.

Pollard, C. and Sag, I. A. (1988). *Information-based syntax and semantics: Vol. 1: fundamentals*. Center for the Study of Language and Information, Stanford, CA, USA.

Régis-Gianas, Y. and Poss, R. (2003). On orthogonal specialization in C++: dealing with efficiency and algebraic abstraction in Vaucanson. In Striegnitz, J. and Davis, K., editors, *Proceedings of the Parallel/High-performance Object-Oriented Scientific Computing (POOSC; in conjunction with ECOOP)*, number FZJ-ZAM-IB-2003-09 in John von Neumann Institute for Computing (NIC), Darmstadt, Germany.

Revuz, D. (1991). *Dictionnaires et lexiques. Méthodes et algorithmes*. PhD thesis.

Riccardi, G., Pieraccini, R., and Bocchieri, E. (1996). Stochastic automata for language modeling. *Computer Speech and Language*, 10:265–293.

Sakarovitch, J. (2003). *Éléments de théorie des automates*.

Sarkar, A. and Joshi, A. (1996). Lexicalized tree adjoining grammars.

Shamsfard, M. and Fadaee, H. (2008). A Hybrid Morphology-Based POS Tagger for Persian. In (ELRA), E. L. R. A., editor, *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco.

Wiener, N. (1948). *Cybernetics: Or the Control and Communication in the Animal and the Machine*. MIT Press.

Winograd, T. (1971). Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. *Cognitive Psychology*, 3.

Woods, W. A. (1970). Transition Network Grammars for Natural Language Analysis. *Communications of the ACM*, 13(10):591–606.