Structuring natural language data by learning rewriting rules

Guillaume Cleuziou, Lionel Martin, and Christel Vrain

LIFO, Laboratoire d'Informatique Fondamentale d'Orléans
Rue Léonard de Vinci B.P. 6759
45067 Orléans cedex2 - FRANCE
{Guillaume.Cleuziou,Lionel.Martin,Christel.Vrain}@univ-orleans.fr

Abstract. The discovery of relationships between concepts is a crucial point in ontology learning (OL). In most cases, OL is achieved from a collection of domain-specific texts, describing the concepts of the domain and their relationships. A natural way to represent the description associated to a particular text is to use a structured term (or tree). We present a method for learning transformation rules, rewriting natural language texts into trees, where the input examples are couples (*text, tree*). The learning process produces an ordered set of rules such that, applying these rules to a *text* gives the corresponding *tree*.

1 Introduction

The work presented in this paper has been motivated by a French project (ACI Biotim http://www-rocq.inria.fr/imedia/biotim/) in the field of Biodiversity. The task we address aims at semi-automatically building an ontology of the domain from corpora describing flora.

The term *ontology* has various definitions in various domains. From a practical point of view, an ontology can be defined as a quadruple $O = (C, R, A, Top)$ where C is a set of concepts, R is a set of relations, A is a set of axioms and Top is the highest-level concept [SB03]. The set R contains relations between concepts, as for example, the binary relation *partof* relating the concepts *hand* and *human*. Usually we distinguish taxonomic and non-taxonomic relations: taxonomic relations are used to organize information with generalization/specialization (or hyponymy) relationships in a "ISA hierarchy"; non-taxonomic relations are any other relations such as synonymy, meronymy, antonymy, attribute-of, possession, causality, ...

Ontology learning refers to extracting one of these elements from input data. This task has been addressed in several research areas. Ontology learning systems extract their knowledge from different types of sources, such as structured data (databases, existing ontologies, ...) or semi-structured data (dictionaries, XML documents, ...). One of the problems is to learn from unstructured data (domain-specific natural language texts). A quite natural formalism for structuring texts is first-order logics (usually logic programs), thus allowing the use of Inductive Logic Programming for different tasks, as for instance Text Categorization, Information Extraction or Parser Acquisition [Coh95,JSR99,Moo96]. This

usually leads to a two-step process: a syntactic analysis of the texts, followed by the learning task. Nevertheless, in our application, the corpora is specific (long descriptions of flore without verbs) making difficult the use of classical syntactic parsers. For instance, the following example is the beginning of the description of the plant called "Pulchranthus variegatus":

" Subshrubs or shrubs, 0.5-2 m tall. Stems terete with red, exfoliating bark. Leaves: petioles 3-13 mm long; blades elliptic-lanceolate, 13-26 × 5-9 cm, glabrous, the apex acuminate-cuspidate. Inflorescences terminal, racemes or panicles, 4-15 cm long, green, the flowers 2-many per node; peduncle 10-15 mm long; bracts small, narrowly triangular, 2.5-3 x 0.5 mm; pedicels lacking to short, 1.5 mm long; bracteoles 1.5-2 mm long. ..."

This text describes different concepts (stem, bark, leaf, ...) and various relations: part-of relations (bark is a part of a stem, flower is a part of inflorescences, ...) and attribute-value relations (stem is terete, bark is red, petiole is 3-13 mm long...).

All this information can be represented into a tree (term), the leaves (constants) are elements of the text. For example, the term

partOf(desc(stem, terete), desc(bark, [red, exfoliating]))

could be a representation of information associated with the sentence *"Stems terete with red, exfoliating bark"*. The detailed formal language used in our work is presented in Section 3.

Given a set of sentences and their corresponding terms (manually built), our goal is to produce a set of rules able to rewrite a sentence into a term. The corpora shows that in many cases, some simple regular structures can be automatically discovered, these structures are based on the punctuation and the syntactical categories of words. For example, when a noun is immediately followed by an adjective, then the adjective describes the noun; when two descriptions are separated by "," or ", *with*", then the second description is about a concept which is a part of the concept of the first description. This short example also shows that a preprocessing step is required: the initial text is transformed into a list of elements (words, punctuation), each element is tagged, using a part-of-speech (POS) tagger; this preprocessing is done in most existing ontology learners.

Some works have already adressed this task: [MPS02] proposes a survey of methods relying either on statistics or predefined patterns, [SM06] is based on cooccurrences with verb phrases, [Yam01] uses a n-grams representation and [Ait02] uses ILP techniques to characterize specific relations. [Bri93] proposes a transformation-based approach for parsing text into binary trees.

Our approach can be compared with [Hea92] which proposes to use a pattern-based approach to extract hyponymy/hyperonymy relations from texts. [Hea92] proposes to use patterns like for instance :

NP_0 *such as* $\{NP_1, NP_2..., (and/or)\}$ NP_n

to infer that $hyponymy(NP_0, NP_i)$ for $i = 1..n$. In Hasti [SB04], patterns are also used for building ontology. In these works, the user has to define the patterns. The particularity of our approach is that we propose to learn such rules automatically, from a set of examples. In [Bos00], transfer rules are learned in

a bi-lingual translation perspective: rules are produced from pairs of structured terms. These rules are mainly based on the structure of both terms, i.e. on the non-terminal symbols occurring in each term. In our approach, rules are learned from pairs $(sentence, term)$ where the sentence is a list of (tagged) words, i.e. a term which is not sufficiently structured to apply the approach of [Bos00].

For this reason, our method can be applied to any type of relation. The learning process presented in this paper is simple: it is an iterative method, from the inner structure to the outer structure, building new examples for each iteration. We propose a divide and conquer approach guided by the structure and based on a least general generalization principle, applied independently on different parts of couples $(sentence, term)$. In that sense, this work is a preliminary one, and we plane to explore deeply the search space, taking into account the sequential aspect of the data. From an ILP perspective, this point is a challenging problem.

2 Introductive example

To introduce the learning problem, let us consider the three following sentences:

s_1: *"Stems terete to quadrangular, with swollen nodes"*
s_2: *"Bracts imbricate, the margins toothed"*
s_3: *"Corolla curved, the lobes subequal to dimorphic"*

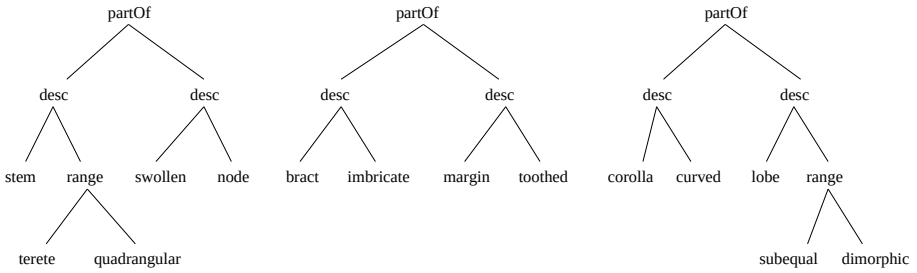
We propose to associate to each sentence a term (or tree), representing information which can help to build an ontology. For example, the first sentence presents two concepts (stem and node), where node is a part of stem; these concepts are described by:

- stems are terete to quadrangular,thus introducing a range of values
- nodes are swollen

In order to represent this information, we can associate to the sentence s_1 the following term:

$term_1$: *partOf(desc(stem, range(terete, quadrangular)), desc(swollen, node))*

In the same way, we can produce $term_2$ and $term_3$ from sentences s_2 and s_3. We have chosen here very similar examples to introduce the learning process and so the corresponding terms are very similar. These terms are represented by the following trees:



Our goal is to build rules such that, applying them on a sentence builds the corresponding term. The rewriting process is based on the grammatical categories of elements in sentences, so we need a mapping from the set of elements in a sentence to a set of possible tags. We use TreeTagger [Tre] to perform this mapping and the possible tags are those proposed by TreeTagger : **nn** (noun, common singular), **jj** (adjective, general), **vvn** (verb, past participle), ..., a detailed list is proposed in [Tre]. We have added the tag **pct** for punctuation.

Then a sentence can be viewed as a list of terms; the previous examples s_1, s_2 and s_3 are respectively represented by the lists $list_1$, $list_2$ and $list_3$:

$list_1$: [nn(stem),jj(terete),to(to),jj(quadrangular),pct(virg),in(with), jj(swollen),nn(node)]
$list_2$: [nn(bract),jj(imbricate),pct(virg),dt(the),nn(margin),vvn(toothed)]
$list_3$: [nn(corolla),vvn(curved),pct(virg),dt(the),nn(lobe),jj(subequal), to(to),jj(dimorphic)]

The goal is then to learn rules, rewriting such lists into the corresponding terms. The input of the process is a set of couples ($list_i$, $term_i$), and we propose to learn these rules by a generalization process. However, the terms are usually too much different to be generalized. For this reason, we consider all their sub-terms with their corresponding subtrees in the generalization process: this requires to be able to extract the sub-list associated to a sub-term, this point is detailed in Section 3.2.

In our example, we can get for instance the two following couples (sub-list, sub-term):

([jj(terete),to(to),jj(quadrangular)], range(terete, quadrangular))

and

([jj(subequal),to(to),jj(dimorphic)], range(subequal, dimorphic))

which can be generalized into

([jj(X),to(to),jj(Y)], range(X, Y)) (rule 1)

The last couple can be considered as a rule, producing a term from a list of terms (we propose in the following section a detailed definition of rewriting rules). Then, applying such a rule to a list of terms consists in replacing a part of the list matching the left-hand side of the rule by the corresponding right-hand side.

Such a rule will have to be applied to lists of terms such as $list_1$. So, it will be necessary to decompose such a list into 3 lists l_1, l' and l_2 such that l' matches with the left-hand list of the rule and then, l' is replaced by the right-hand term of the rule. For the example $list_1$, a possible decomposition is

l_1 = [nn(stem)],
l' = [jj(terete),to(to),jj(quadrangular)]
l_2 = [pct(virg),in(with), jj(swollen),nn(node)]

and applying the rule 1 to $list_1$ gives the list:

$list'_1$: [nn(stem),range(terete, quadrangular),pct(virg),in(with), jj(swollen),nn(node)]

In the next section, we introduce special symbols $\diamond_i$ instead of variables $X, Y, \ldots$. Then, the right-hand term of the rule 1 will be written $range(\diamond_1, \diamond_2)$ and we will have to extract the a sub-list from l' (in our example [*terete*, *quadrangular*]) such that symbols $\diamond_1$ and $\diamond_2$ are replaced by the terms belonging to this sub-list.

We can notice that, applying the previous rule to any list from the initial lists $list_1$, $list_2$ or $list_3$, produces only expected sub-terms. Conversely, the rule

([nn(X),jj(Y)], desc(X, Y))

could also be considered but it produces some unexpected terms since it can be applied to the sub-list [nn(stem),jj(terete)] of the sentence s_1, producing the term $desc(stem, terete)$ which is not a sub-term of $term_1$. For this reason, this rule is not acceptable at this level (in this paper, we require that rules are 100% correct).

Once an acceptable rule is produced, we propose to apply it to all the sub-lists of the positive examples: if we apply the previous rule to the couple ($list_1$, $term_1$), we obtain the couple

([nn(stem),range(terete,quadrangular),pct(virg),in(with),jj(swollen), nn(node)], $term_1$)

Then we get a new set of positive examples and we propose to continue this process until either all the couples have the form $([term], term)$ or no more rule can be learned. In our example, we can expect that, after learning and applying some rules, $list_1$, $list_2$ and $list_3$ will be rewritten respectively into

$list'_1$: [desc(stem,range(terete,quadrangular)),
pct(virg),in(with), desc(swollen,node)]

$list'_2$: [desc(bract,imbricate),
pct(virg),dt(the), desc(margin,toothed)]

$list'_3$: [desc(corolla,curved),
pct(virg),dt(the), desc(lobe, range(subequal, dimorphic))]

These terms are similar in their structure but in order to generalize the examples $(list'_1, term_1)$, $(list'_2, term_2)$, $(list'_3, term_3)$, we have to introduce a more general form of rewriting rules, generalizing the three previous examples allowing either "*the*" or "*with*" between two descriptions in a rule producing a *partOf* term. These problems also arise with the following example:

([nn(bract),**jj**(imbricate)], desc(bract, imbricate))

and

([nn(margin),**vvn**(toothed)], desc(margin, toothed))

where two different tags are possible for the second word.

This process is formally detailed in the following sections.

3 Definitions and languages specification

As mentioned in the previous section, the examples used by the learning method are couples (*list*, *term*) where *list* is a list of term. In this section we introduce

some definitions and we present the language specifications used in our application, for both terms in the initial lists of terms and terms representing intended information to extract.

3.1 Terms and lists

We first recall some definitions and notations for terms and lists, see [CDG+97] for more details.

A regular tree language is defined by a ranked alphabet $(\mathcal{F}, arity)$ where $\mathcal{F}$ is a finite set of symbols and $arity$ a function from $\mathcal{F}$ to $\mathbb{N}$, which indicates the arity of a symbol. Given a set of variables $\mathcal{X}$, terms are inductively defined by: a symbol of arity 0 is a term, a variable of $\mathcal{X}$ is a term, if f is a symbol of arity n and $t_1, ..., t_n$ are terms, then $f(t_1, ..., t_n)$ is a term. For any $i \in [1..n]$, t_i is a sub-term of $f(t_1, ..., t_n)$ and any sub-term of t_i is a sub-term of $f(t_1, ..., t_n)$. Given a term $t = f(t_1, ..., t_n)$, we define $top(t) = f$.

A context is a term $C[\diamond]$ containing a special variable $\diamond$ which occurs just once in that term, it marks an empty place. Throughout, the substitution of $\diamond$ by a term u is written $C[u]$.

Lists are usually terms build with a 2-ary symbol (*cons*) and a 0-ary symbol (ϵ). In this paper, lists of terms are written with square brackets in order to distinguish terms from lists of terms.

Given n terms $t_1, ..., t_n$, the list containing $t_1, ..., t_n$ is denoted by $[t_1, ..., t_n]$, n is the size of l. Let l be the list $[t_1, ..., t_n]$, $l(i)$ denotes the i-th term t_i. The concatenation of two lists $l_1 = [a_1, ..., a_n]$ and $l_2 = [b_1, ..., b_p]$ is written $l_1.l_2 = [a_1, ..., a_n, b_1, ...b_p]$.

3.2 Definitions and notations

Let l be the list $[t_1, ..., t_n]$, a *sub-list* of l is a list $[l(i_1), ..., l(i_k)]$ with $i_1 < i_2 < ... < i_k$. We write $S_k(l)$ the set of sub-lists of l with size k, and $S_*(l)$ the set of sub-lists of l of any size.

Given a list $l = [t_1, ..., t_n]$, the list l' is a *part of the list* l if l can be written as a concatenation $l = l_1.l'.l_2$ (l_1 and l_2 are lists possibly empty). In this case, l' is a sub-list of l that can be written $l' = [l(i), l(i+1), ..., l(i+k)]$.

$\diamond$ ***Definition : k-context.*** A *k-context* is a term $C[[\diamond_1, ..., \diamond_k]]$ containing k special variables $\diamond_1, ..., \diamond_k$ which occur just once in that term, each one marks an empty place. Given a list of terms $l = [t_1, ..., t_k]$, the substitution of each $\diamond_i$ by the term t_i is written $C[l] = C[[t_1, ..., t_k]]$. The special variable $\diamond_i$ appears to the right-hand side of $\diamond_j$ iff $i > j$

Given a couple (l, t) (l is a list of terms and t is a term), the role of a k-context is to express that the term t can be obtained from a k-context $C[[\diamond_1, ..., \diamond_k]]$ and a sub-list $l_k \in S_k(l)$ such that $t = C[l_k]$. To learn rewriting rules, we have to find k-contexts that are common to different couples.

For instance, let us consider a couple (l, t) with $l = [a, b, c, d]$ and $t = f(b, h(d))$. If we consider the 2-context $C[[\diamond_1, \diamond_2]] = f(\diamond_1, h(\diamond_2))$, we can extract the sub-list $l' = [b, d]$ from l such that $t = C[l']$. Let us notice that another possible context is for instance the 1-context $C'[[\diamond_1]] = f(b, h(\diamond_1))$ with the sub-list $l'' = [d]$, such that $t = C'[l'']$.

Since for a couple (l, t) t is a possible 0-context, we propose a more restrictive definition:

$\diamond$ ***Definition : k-skeleton.*** A *k-skeleton* is a k-context $sk_k = C[[\diamond_1, ..., \diamond_k]]$ such that there are no other 0-ary symbols in sk_k than the $\diamond_i$. A term t is a *skeleton* if there exists k such that t is a k-skeleton (such a value k is unique).

For example, $f(a, g(\diamond_1, \diamond_2))$ is a 2-context but is not a 2-skeleton. On the other hand, $f(\diamond_1, g(\diamond_2, \diamond_3))$ is a 3-skeleton

In the following, rewriting rules are obtained from skeletons. This choice has been made in order to ensure that the information associated to term are all contained in the initial sentence. In a more general process, we could consider k-contexts to learn rewriting rules.

$\diamond$ ***Definition : consistency.*** A couple (l, t) (l is a list of terms and t is a term) is said to be *consistent* if there exists a unique couple (sk, sl) such that sk is a skeleton $C[[\diamond_1, ..., \diamond_k]]$, $sl \in S_k(l)$ and $C[sl] = t$.

The consistency condition for (l, t) ensures that the term t can be obtained from a skeleton and a sub-list of l. It ensures also that there exists only one way to obtain t from a skeleton and a sub-list. Then, the uniqueness of the skeleton and the sub-list allow to denote by $skel(l, t)$ and $sub(l, t)$ the associated skeleton and sub-list of a consistent couple (l, t).

The consistency condition is not necessary in a general rewriting-rule framework. It is required in the method presented here in order to reduce the search space for learning rules. In order to ensure the consistency condition:

- we can require that the order of words occurring in a term is the same that the order of words in the associated sentence,
- we can also require that each word occurring in a term occurs exactly one time in the associated sentence.

3.3 Rewriting rule

A rewriting rule allows to replace a part of a list of terms, by a new term, containing some terms of the list of sub-terms. This replacement is made under some conditions that are specified in the rule. For this reason, we propose the following general and formal definition:

$\diamond$ ***Definition : Rewriting rule.*** A *rewriting rule* is defined by an integer k and a triple $(Cond, Ext, T)$ that expresses how to apply it on a list l decomposed into three sub-lists $l = l_1.l'.l2$,

- $Cond$ is a condition which has to be satisfied by $(l_1, l', l2)$,

- Ext specifies conditions on sub-lits l'' with size k that can be extracted from l'. $Ext(l')$ denotes the set of sub-lists that can be extracted from l' by applying Ext
- $T = C[[\diamond_1, ..., \diamond_k]]$ is a k-context.

This definition is very general. It will be illustrated in Section 4, after having defined a first order language.

Given a list of terms l, to apply a rewriting rule $r = (Cond, Ext, T)$, we first need to search for a decomposition $l = l_1.l'.l_2$ such that the condition $Cond$ is satisfied for $(l_1, l', l2)$. Then, given $l'' \in Ext(l')$, the list $l_1.[C[l'']].l_2$ can be produced from l, by applying r.

In this general context, given a list of terms l and a rewriting rule r, applying r to l may produces different results; we note the set of results $r(l)$. In the same way, we note $r^2(l) = \{r(l')|l' \in r(l)\}$, and so on. We note $r^*(l)$ the set $r^n(l)$ such that the rule r cannot be applied to any element of $r^n(l)$, if it exists.

The previous definition is very general, we do not propose a precise formalism to express conditions and extraction methods. In the next section, we detail a specific form of rule.

3.4 Languages specification

We specify here the language used in our experiments. Examples are couples $(list, term)$ associated to a sentence. In such couples, $list$ is a list of terms and $term$ represents information to extract from the sentence.

Initially, the list of terms is built from the elements in the sentence, associated to their corresponding syntactical tag. As mentioned above, TreeTagger has been used for the tagging task. The language for terms in the initial lists of terms is then specified as follows:

- all the syntactical elements (words, punctuations) are 0-ary symbols,
- any tag is a 1-ary symbol.

We have used the tagset proposed in the original English parameter files given with TreeTagger [Tre]. We have added a specific tag "dimension" for expression such as "$2 - 4 \times 2.5\ cm$".

Concerning the right-hand part of the examples, the language used has to represent conceptual information associated to a sentence, mainly the concepts, the attributes and values, and the "part of" relations. Moreover, some symbols have been introduced to handle lists.

We chose to use a detailed language; the symbols are (the arity is specified behind /) :

n/1 : the argument of this symbol is a concept,
attr/1 : the argument is an attribute,
val/1 : the argument is a value,
prec/1 : the argument is a precision (mainly expressed by adverbs),

att/3 : this symbol allows to build an attribute/value association; the arguments are an attribute, a value and a precision. We always put 3 arguments, even when some information is missing in the sentence. In this case, we use the 0-ary symbol e. For example, "mostly pilose" is represented by the term $att(prec(mostly), val(pilose), attr(e))$, the attribute is not indicated.

latt/2 : this symbol is used for lists of attribute/value associations, the first argument is made with the symbol *att* and the second is a list of attribute/value associations or e. We have chosen to systematically use this symbol, even for a single attribute/value association.

range/2 : allows to express range of values; this symbol may appear as an argument of *val*,

disj_jj/2 : express a disjunction of values (conjunction of values are expressed with lists),

nj/2 : this symbol is used for some noun-adjective association, such as "lower lobe" $(nj(lower, n(lobe)))$, "posterior lip" $(nj(posterior, n(lip)))$, ...

rj/2 : this symbol is used when a value is associated to an adverb, such as "mostly triangular" $(rj(mostly, triangular))$ or "slightly emarginate" $(rj($ $slightly$ $, emarginate))$, ...

desc/2 : this symbol allows to express that a list of attributes/values describes a particular part of the plant. The arguments are a concept and a list of attributes/values,

partOf/2 : this symbol expresses a "part of" relation; it has two arguments, one corresponding to a concept (build with symbols n or *desc*) and one corresponding to a precision,

conj_nn/2 : this symbol expresses conjunctions of concepts, it is used when different part-of relations are described in a sentence.

These symbols are illustrated on the following example: the initial sentence is :

"*Corolla tubular to funnelform, +/- arcuate , 2-lipped , mostly pilose, usually red, the posterior lip entire or slightly emarginate, the anterior lip 3-lobed*"

and the associated term is:

```
partOf(
  desc(n(corolla),
    latt(att(val(range(tubular,funnelform)), attr(e), prec(e)),
     latt(att(prec(+/-), val(arcuate), attr(e)),
      latt(att(val(2-lipped), attr(e), prec(e)),
       latt(att(prec(mostly),val(pilose), attr(e)),
        latt(att(prec(usually),val(red), attr(e)), e))))))),
  conj_nn(
    desc(nj(posterior,n(lip)),
      latt(att(val(disj_jj(entire,rj(slightly,emarginate))),
        attr(e), prec(e)),e)),
    desc(nj(anterior,n(lip)),
      latt(att(val(3-lobed), attr(e), prec(e)),e))),
  prec(e))
```

4 Learning method

As mentioned above, we propose to learn from an initial set of couples $\mathcal{C}_0 = \{(list_i^0, term_i)\}$. We require that any initial couple is consistent.

Once a rule r has been learned, it is applied to any list $list_i^0$, if possible, giving new examples $\mathcal{C}_1 = \{(list_i^1, term_i)\}$, where $list_i^1$ is obtained by applying r to $list_i^0$ as many times as possible, otherwise $list_i^1 = list_i^0$. Then, this process is repeated to define $\mathcal{C}_2$ from $\mathcal{C}_1$, ...

In our framework, when a learned rule r is applied to a list l, we require $r^*(l)$ to exist and to be a singleton. This means that if there are different ways to apply r to a list l, the way r is applied has no importance and any way to apply r leads to the same result. In practice, such rules, if generated, will cover negative examples and therefore are rejected by the learning process; this is realized due to the definition of negative examples.

Moreover, the form of the learned rules ensures that any $\mathcal{C}_m$ contains only consistent examples.

For learning a rule, the idea is to use any subterm and their corresponding list of terms as positive examples. Then, at any step, the set of examples used in the learning process has to be defined from $\mathcal{C}_n = \{(list_i^n, term_i)\}$.

4.1 Positive and negative examples

Given a set of couples $\mathcal{C}_n = \{(list_i^n, term_i)\}$, we define the set of positive examples E_n^+ as the set of couple (l_i^k, t_i^k) such that t_i^k is a subterm of $term_i$ and l_i^k is the corresponding part of the list $list_i^n$. In order to automatically build E_n^+ from $\mathcal{C}_n$, this definition requires a function that maps any subterm to the corresponding part of the list. Given a couple $(list_i^n, term_i)$ and a subterm t_i^k from $term_i$, since the couple is consistent, there exists a unique skeleton $sk = C[[\diamond_1, ..., \diamond_k]]$, and a unique sub-list $sl \in S_k(list_i^n)$ such that $C[sl] = term_i$. Then, there exists a subterm $sk' = C'[[\diamond_j, ..., \diamond_{j+p}]]$ of sk, and the corresponding sub-list sl' such that $t_i^k = C'[sl']$. The sub-list sl' can be written $[list_i^n(j_1), list_i^n(j_2), ..., list_i^n(j_s)]$, we propose to choose $l_i^k = [list_i^n(j_1), list_i^n(j_1+1), ..., list_i^n(j_s)]$, which is the shortest part of list associated to t_i^k.

An example (l_i^k, t_i^k) is said to be *covered* by a rule r if $r(l_i^k)$ is unique and $r(l_i^k) = [t_i^k]$.

From the set of positive examples, we propose to define the set of negative examples E_n^- as the set of couples (l_i^-, t_i^-) such that l_i^- is a part of a list of l_j^k for $(l_j^k, t_j^k) \in E_n^+$ and $(l_i^-, t_i^-) \notin E_n^+$ (any term t_i^- such that$(l_i^-, t_i^-) \notin E_n^+$ can be chosen). The set of negative examples is then infinite, and in practice it is not generated. To ensure that no negative example is covered, it is sufficient to test whether each time a rule r can be applied on $list_i^n$, $(list_i^n, term_i) \in \mathcal{C}_n$, the term produced by the rule is a subterm of $term_i$.

4.2 Form of the learned rules

As mentioned above, a rewriting rule is a triple $(Cond, Ext, T)$, where T is a k-skeleton. We propose to write a rule (associated to a k-skeleton T) as :

$r = [list_0, (\diamond_1, Symb_1), list_1, \ldots (\diamond_n, Symb_n), list_n] \rightarrow T$

where: $list_i$ is a list of list of terms and $Symb_i$ is a list of symbols

We have to specify $Cond$ and Ext from this representation. Let l_1, l' and l_2, be lists of terms, $Cond$ is satisfied by $(l_1, l', l2)$ if l' can be written

$l' = ll_0.[t_1].ll_1.[t_2].ll_2. \ldots .[t_k].ll_k$

with $ll_i \in list_i$ and $top(t_i) \in Symb_i$. In this case, $[t_1, ..., t_k]$ belongs to $Ext(l')$.

In this representation of a rule, $list_i$ corresponds to the list of possible separators between terms occurring in the list and in the term. Let us notice that when the condition is satisfied for $(l_1, l', l2)$, it does not depend on l_1 nor l_2. In this context, the condition does not depend on the context of the list to be replaced.

Consider the following examples $(list_1, term_1)$ and $(list_2, term_2)$:

$list_1 = [dt(the), nn(stem), in(with), nn(anther), jj(2 - locular)]$

$term_1 = partOf(stem, desc(anther, 2 - locular)$

$list_2 = [nn(bract), pct(virg), dt(the), nn(margin), vvn(toothed)]$

$term_1 = partOf(bract, desc(margin, toothed)$

they are covered by the following rule:

$r = [[[dt(the)], []], (\diamond_1, [nn]), [[in(with)], [pct(virg), dt(the)]], (\diamond_2, [nn]),$
$[[]], (\diamond_3, [jj, vvn]), [[]]] \rightarrow partOf(\diamond_1, desc(\diamond_2, \diamond_3))$

The construction of this rule is illustrated in the following table:

	lst_1	lst_2	$\rightarrow r$
l_0	[dt(the)]	[]	[[dt(the)], []]
t_1	nn(stem)	nn(bract)	$\diamond_1 = nn(...)$
l_1	[in(with)]	[pct(virg),dt(the)]	[[in(with)], [pct(virg),dt(the)]]
t_2	nn(anther)	nn(margin)	$\diamond_2 = nn(...)$
l_2	[]	[]	[[]]
t_3	jj(2-locular)	vvn(toothed)	$\diamond_3 = jj(...)$ or $vvn(...)$
l_3	[]	[]	[[]]

4.3 The search space

Given a set of positive examples E_i^+ (and the associated set of negative examples E_i^-), the goal is to find a rule covering some positive examples and covering no negative ones, in a "divide-and-conquer" way. In this paper, we propose a simplified method, based on a decomposition of the set E_i^+ into a partition $G_1, \ldots, G_n$, where examples with the same skeleton are in the same group.

A rule is built by generalizing examples of a group. Given a group G_i and the associated skeleton $C[[\diamond_1, ..., \diamond_k]]$, it is possible to write:

$G_i = \{ll_0^j.[\diamond_1^j].ll_1^j.[\diamond_2^j]. \ldots .[\diamond_k^j].ll_k^j\}$, $j = 1..|G_i|$.

since examples are consistent (in the previous notation, we just write $\diamond_p^j$ in the term l, where the terms of $sub(l,t)$ occur, for an example (l,t) of the group). Then, we propose to build the rule:

$$r = [list_0, (\diamond_1, Symb_1), list_1, \ldots (\diamond_k, Symb_k), list_k] \rightarrow C[[\diamond_1, ..., \diamond_k]].$$

where $list_p = \cup_{j=1..|G_i|} ll_p^j$, $p = 0..k$, and $Symb_p = \cup_{j=1..|G_i|} top(\diamond_p^j)$, $p = 1..k$

If a rule covers a negative example, it is rejected. The search starts with groups for which the associated skeleton has the lowest depth.

4.4 Learning process

Our learning process differs from the usual divide-and-conquer methods: each time a rule is learned, t is applied on the set of positive examples. Each rule is then built from a particular set of positive examples. This choice is motivated by the general process of the transformation of a text into a term: rules are applied in the same order they are learned. When a rule is applied, some rules may have been applied before, then, when we start learning a rule, the rules previously learned have to be applied.

Formally, let r_i be the rule learned from E_i^+ (starting by r_0). We define $E_{i+1}^+ = r_i^*(E_i^+) = \{(r_i^*(list^+), term^+) | (list^+, term^+) \in E_i^+\}$.

The learning process stops when each example $(list^+, term^+) \in E_n^+$ is such that $list^+ = [term^+]$, or when any new rule covers some negative examples.

5 Experiments and conclusion

This approach has been applied in the field of botany, using a corpus on vascular plants of central French Guiana. We have used the description of 5 plants, corresponding to 54 texts, and producing 1115 initial positive examples. The method produces 49 rules covering 81,3% of the positive examples.

The preliminary results are very promising since many improvements can be done: we have used a simplified method for learning a rule; some of the uncovered positive examples could have been covered by splitting some of the groups G_j or by exploring in more details the search space. It could also be interesting to consider the context of the examples: some ambiguous cases could be solved by including in the rule, the category of elements preceding and following the examples. As mentioned above, we will focus in further works on the learning task: we proposed in this paper a least general generalization approach, learning more complex rules is an interesting ILP perspective.

Moreover, in some cases, the rules cannot be based only on categories of elements. Consider the examples "*corolla glandular, mauve or white*" and "*corolla blue, mauve or white*". In the first case, the disjunction concerns the words "mauve" and "white", in the second case it concerns the 3 colors. This situation could be treated by using additional information or by producing rules allowing different possible transformations from the same text.

References

[Ait02] J.S. Aitken. Learning Information Extraction Rules: An Inductive Logic Programming approach. In *Proceedings of the 15th European Conference on Artificial Intelligence*, pages 355–359, 2002. http://citeseer.ist.psu.edu/586553.html.

[Bos00] Henrik Boström. Induction of recursive transfer rules. In James Cussens and Saso Dzeroski, editors, *Learning Language in Logic*, volume 1925, pages 237–246. June 2000.

[Bri93] Eric Brill. Automatic grammar induction and parsing free text: A transformation-based approach. In *Meeting of the Association for Computational Linguistics*, pages 259–265, 1993.

[CDG+97] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 1997. release October, 1rst 2002.

[Coh95] W.W. Cohen. Learning to classify English text with ILP methods. In L. De Raedt, editor, *ILP95*, pages 3–24. DEPTCW, 1995.

[Hea92] Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the Fourteenth International Conference on Computational Linguistics*, pages 539–545, Nantes, France, July 1992.

[JSR99] Markus Junker, Michael Sintek, and Matthias Rinck. Learning for text categorization and information extraction with ILP. In James Cussens, editor, *Proceedings of the 1st Workshop on Learning Language in Logic*, pages 84–93, Bled, Slovenia, 1999.

[Moo96] R. J. Mooney. Inductive logic programming for natural language processing. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, volume 1314, pages 3–24. Springer-Verlag, 1996.

[MPS02] Alexander Maedche, Viktor Pekar, and Steffen Staab. Ontology learning part one - on discovering taxonomic relations from the web. In Ning Zhong, editor, *Web Intelligence*. Springer, 2002.

[SB03] Mehrnoush Shamsfard and Ahmad Abdollahzadeh Barforoush. The state of the art in ontology learning: a framework for comparison. *Knowl. Eng. Rev.*, 18(4):293–316, 2003.

[SB04] Mehrnoush Shamsfard and Ahmad Abdollahzadeh Barforoush. Learning ontologies from natural language texts. *Int. J. Hum.-Comput. Stud.*, 60(1):17–63, 2004.

[SM06] David Sanchez and Antonio Moreno. Discovering non-taxonomic relations from the Web. In *Proceedings of the 7th International Conference on Intelligent Data Engineering and Automated Learning*, Burgos, Spain, 2006. Springer.

[Tre] TC Projetc TreeTagger. http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html.

[Yam01] Takahira Yamaguchi. Acquiring conceptual relationships from domain-specific texts. In *Proceedings of the Second Workshop on Ontology Learning OL 2001*, Seattle, USA, 2001.

A An example of flore description: ANISACANTHUS

Branching herbs or subshrubs. Stems covered with brown or gray exfoliating bark. Leaves: petioles present or absent; blades linear to lanceolate, cystoliths present. Inflorescences spicate, racemose, or paniculate, the flowers secund or opposite, borne singly or several at inflorescence node; bracts and bracteoles mostly triangular to linear, usually caducous. Flowers: calyx 3-5-lobed, the lobes triangular to linear; corolla tubular to funnelform, arcuate, 2-lipped, mostly pilose, usually red, the posterior lip entire or slightly emarginate, the anterior lip 3-lobed; stamens 2, the anthers 2-locular, subequal, not mucronate or appendaged. Capsules subpyriform, slightly beaked. Seeds 2-4, homomorphic, flattened, each supported by curved retinaculum.

B Positive examples corresponding to the firt two sentences

ex([vvg(branching),nn(herbs),cc(or),nn(subshrubs)],
disj_nn(desc(latt(att(val(branching), attr(e), prec(e)), e), n(herbs)),
n(subshrubs))).

ex([nn(stems),vvn(covered),in(with),jj(brown),cc(or),jj(gray),
vvg(exfoliating),nn(bark)],
partOf(n(stems),desc(latt(att(val(disj_jj(brown, gray)), attr(e), prec(e)),
latt(att(val(exfoliating), attr(e), prec(e)), e)), n(bark)))).

C Examples of learned rules

rule 1 : [[[[]], ($\diamond_1$, [nn]), [[cc(and)]], ($\diamond_2$, [nn]), [[]]],
conj_nn($\diamond_1$, $\diamond_2$)]
for example applied to "bracts and bracteoles"

rule 2 : [[[[]], w($\diamond_1$, [dim]), [[]], w($\diamond_2$, [rb, nn, jj]), [[]]],
att(val($\diamond_1$, $\diamond_2$, prec(e))]
for example applied to "6-8 cm long", "15mm diam", ...

rule 3 : [[[[]], ($\diamond_1$, [vv, jj]), [[pct(virg)]], ($\diamond_2$, [jj]), [[pct(virg), cc(or)]],
($\diamond_3$, [vvg, jj]), [[]]],
disj_jj($\diamond_1$, disj_jj($\diamond_2$, $\diamond_3$))]
for example applied to "spicate, racemose, or paniculate".

rule 4 : [[[[]], ($\diamond_1$, [range, jj]), [[pct(virg)]], ($\diamond_2$, [att]), [[]]],
latt(att(val($\diamond_1$), attr(e), prec(e)), latt($\diamond_2$, e))]
for example applied to "triangular to linear, usually caducous". Let us notice that this rule is learned after some rules have been learned and applied to examples: a first rule has produced the term $range(triangular, linear)$, a second rule has produced the term $att(prec(usually), val(caducous), attr(e))$ from "usually caducous".