FI MU

Faculty of Informatics
Masaryk University

Proceedings of the Third Learning Language in Logic Workshop

by

Luboš Popelínský
Miloslav Nepil

FI MU Report Series FIMU-RS-2001-08

September 2001

Luboš Popelínský
Miloslav Nepil (Eds.)

Learning Language in Logic

Proceedings of the Third
Learning Language in Logic Workshop
Strasbourg, France, 8–9 September 2001

Preface

The 3rd Learning Language in Logic (LLL) workshop is the follow-up of the previous LLL workshops held in 1999 in Bled, Slovenia, and in 2000 in Lisboa, Portugal. This year the LLL workshop takes place as a joint workshop of ILP'2001 conference. We would like to thank program committee for their help in reviewing submissions and to authors of all submitted papers for their work. Thanks also to Nicolas Lachiche, the local chair of the ILP/LLL, for his assistance, and Eva Žáčková for her help with the preparation of this volume.

Luboš Popelínský
Miloslav Nepil
Editors

Brno, August 2001

Program committee

Pieter Adriaans (Syllogic and University of Amsterdam, the Netherlands)
James Cussens (University of York, UK)
Martin Eineborg (University of Stockholm, Sweden)
Tomaž Erjavec (Institute Jozef Stefan, Slovenia)
Suresh Manandhar (University of York, UK)
Claire Nédellec (LRI, University of Paris-Sud, France)
Guenter Neumann (DFKI, Saarbrücken, Germany)
Luboš Popelínský (Masaryk University in Brno, Czechia) (chair)
Stefan Wrobel (University of Magdeburg, Germany)

Organization

Nicolas Lachiche (LSIIT Strasbourg, France)

Invited Speaker

Dan Roth (University of Illinois, USA)

Support

LLL 2001 is financially supported by the Network of Excellence in Inductive Logic Programming ILPnet2 funded under the European Union's INCO program.

Table of Contents

A Rule-Based Tagger Development Framework

Zoltán Alexin[1], Péter Leipold[2],
János Csirik[1], Károly Bibok[3], and Tibor Gyimóthy[2]

[1] Department of Applied Informatics, University of Szeged,
Árpád tér 2. POB 652, H-6701 Szeged,
alexin@inf.u-szeged.hu, csirik@inf.u-szeged.hu
[2] Research Group on Artificial Intelligence,
Hungarian Academy of Sciences,
Aradi vértanuk tere 1, H-6720 Szeged,
pleipold@freemail.hu, gyimothy@inf.u-szeged.hu
[3] Slavic Institute, University of Szeged,
Egyetem utca 6, H-6720 Szeged,
kbibok@lit.u-szeged.hu

Abstract. POS (Part-of-speech) tagging is an important step in natural language processing because identically written words may have different meanings. Part-of-speech tagging is the procedure during which the correct morphological annotation (the correct tag) for an ambiguous word is selected. Computer programs able to do the process automatically are called POS *taggers*. In this paper the RTDF (a Rule-based Tagger Development Framework) is presented that is capable identifying general tagging rules using different machine learning tools given a suitably large training data set. The framework can combine the learned tagging rules, and evaluate the resulted taggers. The authors are participants of a project for developing a medium sized learning corpus for Hungarian. The corpus contains 1 million words and — among others — can serve as a suitable medium on which the previously developed POS-taggers can be tested. During the project the morphological analyzer for Hungarian has been thoroughly investigated and the MSD encoding has been refined. The development of the corpus is going on and will be completed at the end of 2001. [1]

1 Introduction

Part-of-speech (POS) *tagging* is one of the first stages in natural language related processing (e.g., parsing, information extraction). There are many ambiguous words in each natural language.[2] During the morphological analysis the pos-

[1] This paper has been partially supported by the IKTA-027/2000 project of Hungarian Ministry of Education.

[2] Like *rule* in English, which can be a verb or a noun, *hét* in Hungarian can be a numeral (*seven*) or a noun (*week*).

sible tags for a given word can be determined. POS tagging means *syntactic* disambiguation of the text not regarding to semantics.[3]

There are many different approaches to the POS tagging (statistical, neural-network, HMM, rule-based, etc.) [1, 2, 5]. Developing a POS tagger requires a lot of human efforts. First, each learning tool needs a suitably large annotated corpus, a training data set. Second, interfacing between the natural language corpus and the learning tools: generating input data files in a specific format then parsing the result files, finally extracting tagging rules from them. Third, testing the working tagger, determining the possible causes of errors also need lots of systematic human work.

In this paper a framework called (RTDF) is presented that can efficiently support the development of taggers. Its main features are:

- *supports manual tagging*
  (a user friendly environment has been implemented for manual annotations of corpora; different morpho-syntactic coding schemes can be used)
- *natural language independent representation of the annotated corpora*
  (a unified prolog based model for different natural languages)
- *flexible morpho-syntactic encoding of words*
  (several coding schemes are allowed)
- *a special learning model[6]*
  (the key element of this model is introducing *ambiguity-classes*; this model have been successfully applied for different machine learning methods)
- *make use of different learning algorithms*
  (the RTDF system is able to generate training data for different learning tools like C 4.5[8] and Progol[7])
- *generating taggers from the results of the learning methods*
  (the system can parse the result files, extract tagging rules, combine the rule sets with each other, and produce a working tagger)
- *an efficient user interface for the evaluation of taggers*
  (The system highlights the *faulty* tags by using test corpora)
- *portable implementation*

In [6], the authors presented several promising tagger construction methods. For the time being, there was no suitably large annotated corpus for Hungarian. In order to develop a medium sized natural language corpus the authors proposed a 2 years long project formed of the Department of Informatics at the University of Szeged[4] and the MorphoLogic Ltd.[5] in Budapest. The IKTA 27/2000 project begun in the autumn of 2000, and the 1 million words long annotated corpus is expected to be ready at the end of 2001. Although the corpus is designed for

[3] POS tagging, for example, cannot distinguish between the two meanings of the *bug* noun: an insect or a mistake in a computer program.

[4] former partner in "ILP2"

[5] former industrial partner in "ILP2"

being a learning database it can be a source for many other computer linguistic research.

The official morpho-syntactic encoding of the corpus is the Hungarian version of MSD[4]. During the project the HuMor[6] (Hungarian Morphological parser) was reevaluated. All possible annotations for each text word produced by HuMor was manually checked against the academic dictionaries. Since many corrections are made, at the end of the project a newer version of the Hungarian parser will be produced.

The corpus is produced in XML and encoded by using the TEIXLITE[7] DTD scheme. Texts were collected from five different areas of recent Hungarian written language (fiction, newspapers, law texts, computer oriented texts, teenage compositions). Most texts are from 1999. Each word in the corpus is tagged by its morpho-syntanctic labels, then disambiguated. Annotations of ambiguous words are augmented by manually determined *reasons* (sets of positive or negative literals, in fact clause bodies), that are valid in the disambiguation. Literals are composed of certain predicates by that the ortograhpies, basewords or MSD codes can be checked at some context position. The development of the corpus is done by linguists and has not been finished yet. The first (partial) results are expected at the end of 2001. This paper presents a software framework that is used in preparing the annotated corpus and generating POS-taggers.

In the Section 2 the logical structure of the tagging development framework is presented. Section 3 deals with implementation issues and Section 4 is a summary.

2 Architecture of the RTDF System

The logical structure of the RTDF system is presented in Figure 1. The figure shows the process of a tagger development. Ovals represent data files, while rectangles are program modules.

The corpus stands in the focus of the RTDF system. Each word and punctuation mark are labeled by one or more MSD codes. These codes represents the meaning of that word, so an ambiguous word has more than one code. RTDF allows modifying them, in those cases when the morphological analyzer (program) result in incorrect number of codes or wrong codes.

The representation of the corpus allows storing the selected *good* tag for each ambiguous word (*the result of disambiguation*) in the text. Therefore corpora can be used as training or test databases.

In RTDF a new corpus can be created, existing corpora can be loaded, edited and saved. With the *editor module* the corpus (the text and the tags) can be modified. Manual annotations (choosing good tag) for words is also allowed.

[6] product of the MorphoLogic Ltd. see `http://www.morphologic.hu`
[7] http://etext.lib.virginia.edu/tei/uvatei12.html

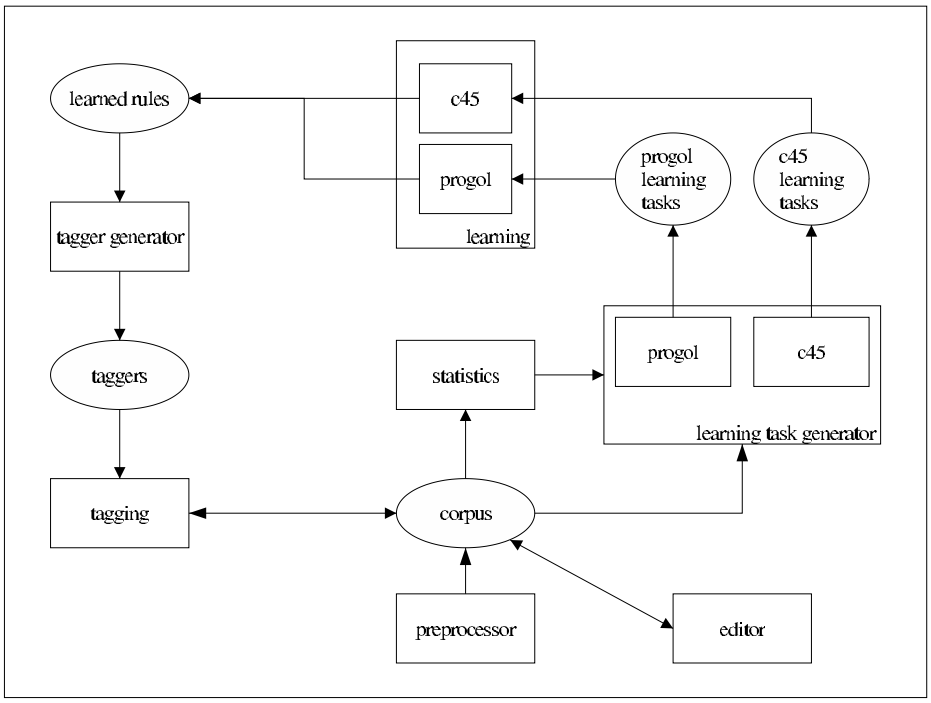


Fig. 1. The structure chart of the Development System

2.1 The Learning model

The key to presented learning model is introduction of *ambiguity classes*. Two words are in the same ambiguity class if and only if their morpho-syntactic labels are the same, see [6]. After that separate learning tasks are generated for each ambiguity class. This can be a solution when we have large number of training examples let us say more hundred thousand.

In the MULTEXT-East project [4] the MSD (Morpho-Syntactic) encoding system was extended to several east European languages, among others, to Hungarian. Since then MSD is widely accepted as a syntactic labeling system for words in many European languages. A code string represents all syntactic attributes of words like type, gender, number, case, definiteness, etc.

For example the Hungarian word *asztalnak* will get the `Nc-sg------` code, which means: noun, common, single, genitive. Those attributes that are not present or not applicable are denoted by hyphens. The first position is reserved for the main categories of words. The detailed description of the MSD encoding can be found in [4]. The main categories of words can be seen in Figure 2. The word *„hideg"(cold)* is tagged to `Afp-sn-------`. (The trailing hyphens are cut in practice to `Afp-sn`). This means that the word is an *adjective* having the following attributes: *qualificative, positive, singular, nominative.*

Category	Code	Category	Code
adjective	A	particle	Q
conjunction	C	adverb	R
determiner	D	adposition	S
interjection	I	article	T
numeral	M	verb	V
noun	N	residual	X
pronoun	P	abbreviation	Y

Fig. 2. The main categories of words in MSD

In Hungarian the `[Afc-sn, Nc-sn]` is a frequent ambiguity class. For the `[Afc-sn, Nc-sn]` class we learn a $choose\_Afc-sn\_Nc-sn(-T,+L,+R)$ predicate where L is the left context, R is the right context and T stands for the good tag. There are no restrictions on number of elements in ambiguity classes. Learning algorithms may determine several tagging rules for each ambiguity class.

Separate learning tasks are created for each ambiguity class. Extracting the learning examples for a specific ambiguity class from the training corpus is automated. RTDF is able to read in the result files and produce the sets of Prolog rules equivalent to the learnt decision trees. Tagging rule-sets are obtained by appending rules for each ambiguity class.

One can see a big circle in the Figure 1, which starts from the corpus, and finishes there too. This represents the process of generating taggers. The *learning task generator* module builds the input files for a rule-based learning tool, such as C 4.5 or Progol. The *learning* module runs these tools and extracts the invented rules from their output. The *tagger generator* module creates the taggers from the learned rules. Finally the *tagging* module runs the newly invented tagging rules on the corpus. The tagging module can evaluate different taggers by comparing the output of the tagger and the stored good tag with one another.

2.2 Modules of RTDF

In the following, we present the main modules of the RTDF.

Preprocessor The task of this module is to create a corpus from simple text files. The first step is to slice up the text into words and punctuation marks, and determine sentence boundaries. The second step is to make a morphological analyzis on each word, and label the words by their possible morpho-syntactic tags. Since disambiguation is done in a later phase the corpus will contain an *empty* tag in the place of the good tag.

Editor The editor allows the user modifying the possible tags and the good tag. By means of this module the errors made in the morphological parsing phase

can be corrected. Next important function is the manual tagging. It is necessary when making learning (training) database. The morphological corrections can be saved for later use.

Statistics This module is a collection of procedures computing statistical data. Statistics either can be displayed for the user or can be used in learning. Some examples:

- Count words, sentences and ambiguity classes.
- Compute the number of occurrences of particular tags.
- Compute the number of occurrences of ambiguity classes.
- Compute distribution of good tags relative to an ambiguity class.

Learning task generator This module is a collection of submodules (*drivers*) for each learning tool. Each driver can create learning tasks for one particular learning tool. Input to this module is the corpus text, from which it builds the necessary data files for the learning tool. Corpora can be used as training data if only if they contain disambiguation information. For the time being there are drivers for C 4.5[8] and Progol[7].

Learning This module is responsible for running the learning tool for the selected learning tasks, and extracting the invented rules from the output files. After extracting the invented rules from the output files rules are collected in a rule database. Rulesets can be saved and loaded as necessary.

Tagger generator This module builds the taggers, that can be used to disambiguate corpora automatically. Rulesets are created from the learnt rules by appending rules for each distinct ambiguity class. Each tagger contains the following parts:

- one or more *rulesets*
- a combination method

A ruleset is a set of the invented rules for several ambiguity classes. The rules may not necessarily come from the same learning task, however for one particular ambiguity class rules must be originated from the same learning task. When the user creates a ruleset he or she may flag out rules related to some ambiguity classes. Rulesets can be incomplete, i.e. may not contain rules for each ambiguity class. Rulesets coming from different learning tasks can be combined in a single tagger. Different combination methods are discussed in [5].
Combination methods implemented in RTDF are listed below:

- Sequencing. The rulesets are sorted. First disambiguate the whole corpus with the first ruleset then the remaining ambiguities with the second ruleset, then the third, etc.

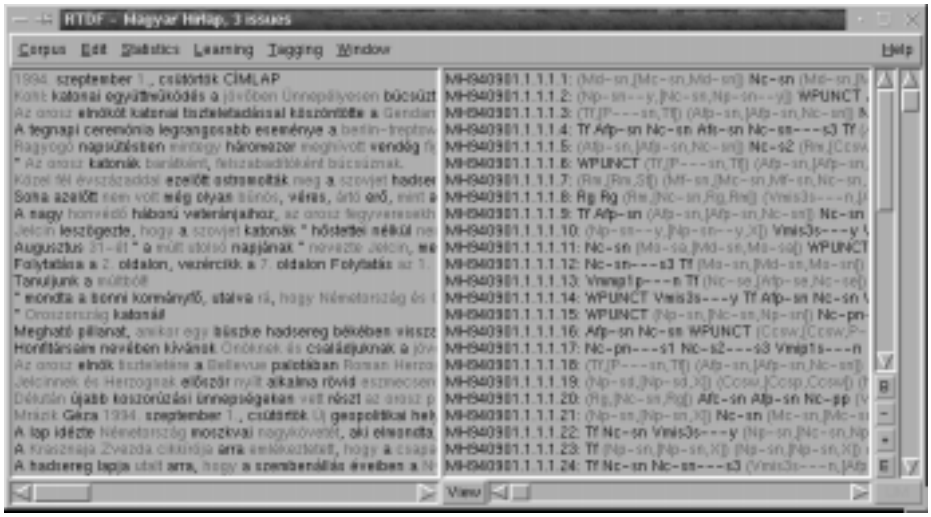


Fig. 3. The Main Window of the RTDF System — a Screenshot

– Voting. Ask each ruleset, and choose the tag which was elected most times.

The learned rules computes the good tag depending on the left and right contexts. Unfortunately, these rules may fail in those cases when there are ambiguous words in the left or right contexts. There are several possibilities to solve this problem.

On solution is that we will not compute the good tag, it will remain in an *undecided* state. Later in a second phase it can be resolved. Other solution is to enumerate all possible solutions for the labeling – together with the left and right contexts – and (somehow) choose the most probable one. Current version of RTDF implements the first solution.

Tagging This module can apply a tagger to the corpus. It can be used for either disambiguate an (ambiguous) text, or to evaluate a tagger. The latter case works if the user applies the tagger on an annotated corpus. The module compares computed good tags and the stored ones with one another. The module computes the accuracy of the tagger on the given test database.

3 Implementation

The RTDF system is implemented in SICStus prolog 3.8.1 and the graphical user interface was written in Tcl/Tk 8.2. A screenshot can be seen in Figure 3.

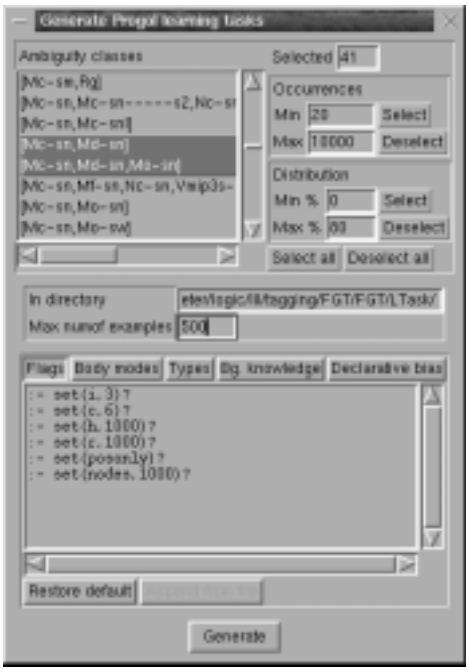


Fig. 4. The Progol Learning Task Generator — a Screenshot

3.1 Corpus representation

For using in RTDF the XML corpora are converted to Prolog format. Each sentence is represented by a $s(+ID, +Words)$ fact, where ID is the unique sentence identifier and $Words$ denotes the list of words in the sentence. List elements contain the words, and their basewords, MSD labels and manually entered literals called *tagging info*.

In Figure 3 the main screen of the RTDF is shown. The largest part of the screen is occupied by the corpus text. On the left side the natural language text is displayed, while the morpho- syntactic labels are shown on the right side. Each row contains one sentence. The corpus is colored. Blue colored words are the ambiguous words, black ones are the normal unambiguous words. Red color denotes unknown words, that have no morphological label.

3.2 Major Modules

Learning task generator

Progol A screenshot of the Progol learning task generator is shown in Figure 4. User can select those ambiguity classes, for which he or she wants to create a learning task. The framework helps to choose the needed ambiguity classes:

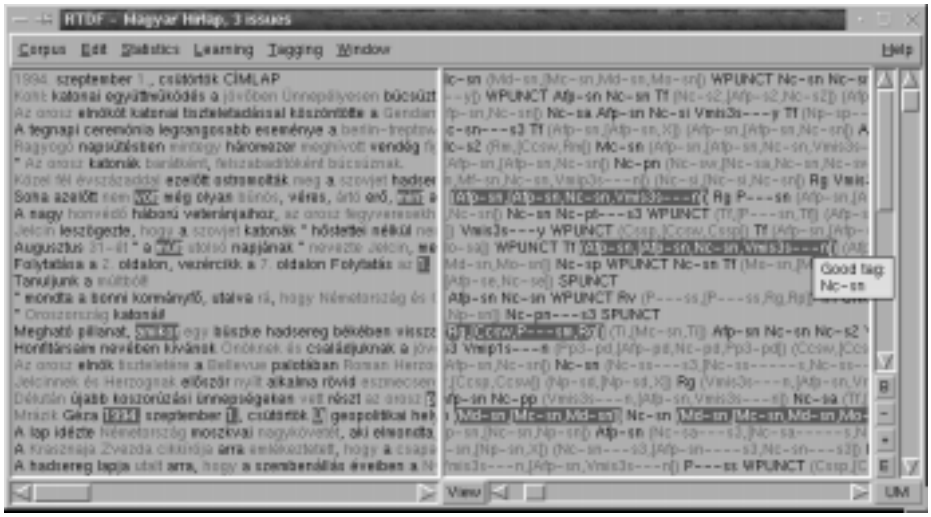


Fig. 5. After Tagging — a Screenshot

1. User can select/deselect those ambiguity classes, whose occurrence numbers lies between two specified integers. 2. Select or deselect those classes, where the most frequent good tag's occurrence number lies between two specified numbers in percent of all occurrences.

Tagging Tagging can be done simply by choosing a tagger. After disambiguating the test database RTDF calculates the following results: all tokens in the test database, all ambiguities, remaining ambiguities, tags disambiguated, correctly tagged words, incorrectly tagged words, relative accuracy and per-word accuracy. All accuracies based on the manually entered good tags stored in the test database.

It is possible to mark those ambiguous words which were disambiguated incorrectly. An example can be seen in Figure 5. Words with inverse colors are those which were tagged incorrectly. If user moves the mouse over such a word, the system displays the good tag.

Currently, RTDF is tested on a part of the TELRI corpus[3]. The results obtained are the same as reported in [6]. When combining a Progol and a C 4.5 tagger the learned rule-sets reach the 93.5% per-word-accuracy and 88.8% relative accuracy (well resolved ambiguities relative to resolved ambiguities). We hope that by applying the new 1 million words long training corpus and exploiting the added logical formuli, eventually by manually correcting the learned rules this accuracy can be slightly increased, up to 95–96%.

4 Summary

A flexible, easy-of-use tagger development framework is shown. The framework can be used to edit and correct corpus files, and run learning tools. For the time being the C 4.5 and Progol are supported. RTDF can generate input files for the learning tool, generate background batches, run background learning batches, finally reading in the results and producing tagging rules. Tagging rules can be combined by chaining rule-sets one after another. More sophisticated combinations are under development.

The above mentioned environment is used in an ongoing project for the development of a 1 million words long Hungarian natural language corpus. The corpus has been designed especially for being a learning database for POS-taggers. Parallel to the development of the corpus, the Hungarian MSD encoding scheme is inspected and minor corrections has been made. The Hungarian morphological analyzer software (HuMor) has also been validated. Human originated background knowledge is included in the corpus especially for the goals of POS-tagging.

References

[1] J. Cussens. Part-of-speech tagging using Progol. In N. Lavrač and S. Džeroski, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *LNAI*, pages 93–108. Springer, Sept. 17–20 1997.

[2] M. Eineborg and N. Lindberg. ILP in Part-of-speech Tagging — An Overview. In *Learning Language and Logic*, volume 1925, pages 157–169, 2000.

[3] T. Erjavec, A. Lawson, and L. R. (eds.). East meets west: A compendium of multilingual resources, 1998. CD-ROM, produced and distributed by TELRI Association e.V., ISBN:3-922641-46-6.

[4] T. Erjavec and M. Monachini. Specifications and notation for lexicon encoding. Technical report, Research Institute for Linguistics, Hungarian Academy of Sciences, 1997. COPERNICUS Project 106 MULTEXT-East.

[5] H. v. Halteren, J. Zavrel, and W. Daelemans. Improving data driven wordclass tagging by system combination. In *Proc. of COLING-ACL'98, Montreal, Canada*, pages 491–497, 1998.

[6] T. Horváth, Z. Alexin, T. Gyimóthy, and S. Wrobel. Application of different learning methods to Hungarian part-of-speech tagging. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *LNAI*, pages 128–139, Berlin, June 24–27 1999. Springer.

[7] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[8] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. QUINLAN86.

Identification of reversible dependency tree languages

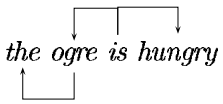
Jérôme Besombes and Jean-Yves Marion

Loria, B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France.
{Jerome.Besombes,Jean-Yves.Marion}@loria.fr,
WWW home page: http://www.loria.fr/~{besombes,marionjy}

Abstract. We investigate learning dependency grammar from positive data, in Gold's identification in the limit model. Examples are dependency trees. For this, we introduce reversible lexical dependency grammars which generate a significant class of languages. We have demonstrated that reversible dependency languages are learnable. We provide a $O(n^2)$-time, in the example size, algorithm.

1 An identification paradigm

From Tesniére [13] seminal study, and from ideas of Mel'čuk [10] and Vergne [14], we propose a two tier communication process between two speakers, see Figure 1. Jean transmits a sentence to Marie. At the first stage, Jean generates a structural sentence, like the following dependency tree



Then, Jean transforms it into a linear phrase, *the ogre is hungry*, and send it to Marie.

Now, Marie has to inverse the two tier process of Jean. For this, she has (i) to recover the structural sentence from the linear sentence (η^{-1}), (ii) to build/update/improve her grammar in order to understand the message, and to generate other messages (θ^{-1}).

In the setting of natural language learning, parsers perform the first task of Marie. Roughly speaking, parsing corresponds to inverse η, that is to compute η^{-1}. For example, the syntactic robust parser developed by Giguet and Vergne[1] computes dependency trees from texts. It is worth noticing that robust parsing is made by tagging and by chunking, without the help of a formal grammar.

Then, the results of those parsers are dependency trees. We are investigating identification of dependency tree languages. The data available are positive examples of a language, that is a sequence of dependency trees. Our hypothesis is

[1] See http://users.info.unicaen.fr/~jvergne, see also Sleator and Temperley english parser [12].

that the computation of θ is reversible, that is the inputs can always be deduced from the outputs. So, identification corresponds to inverse θ. For this, we give a sufficient condition of reversibility on the grammars (G_0).

We show that the class of reversible dependency tree languages is efficiently identifiable in Gold's model [5]. That is, given a finite sequence of examples of a reversible language, we determine a reversible grammar that generates it. We refer to [7] for further explanations.

Our study leans on the work of Angluin [1] on learning languages produced by deterministic and reversibles finite automaton. Further results are described in Mäkinen survey [9]. It is also closely related to Sakakibara [11] work on reversible context free grammars and to Kanazawa [8] work on rigid categorial grammars.

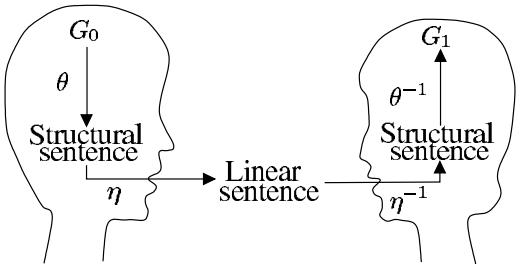


Fig. 1. A two tier communication process

2 Lexical dependency grammar

Following Dikovsky and Modina [3], we present a class of projective dependency grammars which was introduced by Hays [6] and Gaifman [4].

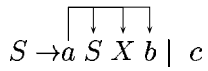
A *lexical dependency grammar* (LDG) Γ is a quadruplet $\langle \Sigma, N, P, S \rangle$, where Σ is the set of terminal symbols, N is the set of non-terminal symbols, $S \in N$ is the start symbol, and P is the set of productions. Each production is of the form $X \to U_1 \ldots U_p\ a\ V_1 \ldots V_q$, where $X \in N$, each U_i and V_j are in $\Sigma \cup N$. The terminal symbol a is called the *head* of the production. In other words, the head is the root of the flat tree formed by the production right handside. Actually, if we forget dependencies, we just deal with context free grammars.

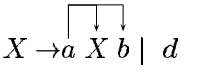
Example 1. Two examples of LDG

1. The grammar $\Gamma_0 = \langle \{a, b\}, \{S\}, P, S \rangle$ where P consists

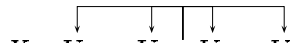
$$S \to a\ S\ b \mid a\ b$$

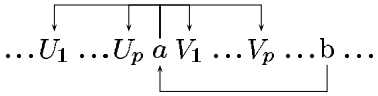
2. The grammar Γ_1=$\langle\{a,b,c\},\{S,X\},P,S\rangle$ where P is



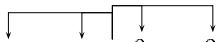


Partial dependency trees are recursively defined as follows.

1. S is a partial dependency tree generated by Γ.
2. If  is a partial dependency tree generated by Γ, and if  is a production of Γ, then

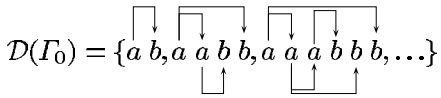


is a partial dependency tree generated by Γ.

When it is convenient, we shall write  for  .

A *dependency tree* generated by a LDG Γ is a partial dependency tree of Γ in which all nodes are terminal symbols. The language $\mathcal{D}(\Gamma)$ is the set of all dependency trees generated by Γ.

Example 2. The language generated by Γ_0 of Example 1 is



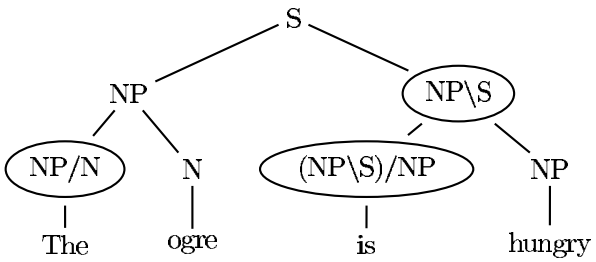
Without dependencies, we recognizes the context free language $\{a^n b^n / n > 0\}$.

3 Relations with categorial grammar and CFG

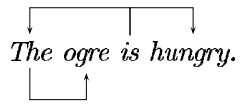
Following the example above, define $\mathcal{L}(\Gamma)$ as the set of words which are obtained by removing the dependencies of trees in $\mathcal{D}(\Gamma)$. We immediatly see that if Γ is a LDG, then $\mathcal{L}(\Gamma)$ is a context free language. Consequently, langages recognized by LDG and by context free grammar (CFG) are equivalent at the level of the words. From a CFG, it is not possible to retrieve dependencies. That is why LDG and CFG are said weakly equivalent.

Bar-Hillel et al. [2] established that categorial grammars (CG) are weakly equivalent to CFG. So, CG are also weakly equivalent to LDG. However, CG are strongly equivalent to LDG in the following sense. From a LDG, we can construct a CG which respect dependencies and conversely. We won't go further on this subject, but we merely give an example.

Example 3. Consider the following parse tree of the phrase "The ogre is hungry" in CG. We transform the parse tree into a dependecy tree by choosing the heads as the functor nodes. These nodes are encircled.



We obtain the tree :



Notice that induced dependency tree doesn't provide a natural analysis because "the" is the head of "ogre", unlike the LDG approach.

4 Reversible LDG

A LDG grammar Γ is *reversible* if the conditions **R1**, **R2** and **R3** of Figure 2 are satisfied. The class of *reversible dependency tree languages* is the class of languages generated by reversible LDG.

5 The learning algorithm

The learning algorithm works as follows. The input is a finite set H of positive examples which are dependency trees. Define $\mathtt{TG}(H)$ as the grammar constructed from all productions outputed by $\mathtt{TG}(w, S)$, for each $w \in H$. The function TG is described in Figure 3.

Stage 0 $G_0 = \mathtt{TG}(H)$.
Stage n+1 The grammar G_{n+1} is defined by applying one of the rule of Figure 2.

The process terminates at some stage m because the number of grammar productions decreases at each stage. So, put $\phi(H) = G_m$.

R1 If $X \rightarrow \mathbf{U}\ a\ \mathbf{V}$ and if $Y \rightarrow \mathbf{U}\ a\ \mathbf{V}$, then $X = Y$.

R2 If $X \rightarrow \boldsymbol{\alpha}\ Y \boldsymbol{\beta}\ a\ \boldsymbol{\gamma}$ and if $X \rightarrow \boldsymbol{\alpha}\ Z \boldsymbol{\beta}\ a\ \boldsymbol{\gamma}$,

then $Y = Z$, where $Y, Z \in N$.

R3 If $X \rightarrow \boldsymbol{\alpha}\ \mathrm{a}\ \boldsymbol{\beta}\ Y\ \boldsymbol{\gamma}$ and if $X \rightarrow \boldsymbol{\alpha}\ \mathrm{a}\ \boldsymbol{\beta}\ Z\ \boldsymbol{\gamma}$,

then $Y = Z$, where $Y, Z \in N$.

Fig. 2. Reversibility conditions and reduction rules.

Function TG(w, X)
Inputs : w is a dependency tree,
X is a non-terminal symbol.
if $w \in \Sigma$
then Output $X \rightarrow w$

if $w = u_1 \ldots u_p\ \mathrm{a}\ v_1 \ldots v_q$
then Take new $U_1, \ldots, U_p$ and $V_1, \ldots, V_q$

Output $X \rightarrow U_1 \ldots U_p\ \mathrm{a}\ V_1 \ldots V_q$
for $i = 1$ **to** p **do** TG(u_i, U_i)
for $i = 1$ **to** q **do** TG(v_i, V_i)

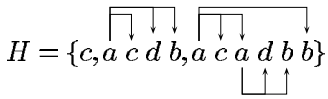
Fig. 3. TG(w, X) recognizes exactly w, i.e. $\mathcal{D}(\mathrm{TG}(w, X)) = \{w\}$

Theorem 1. *ϕ learns the class of reversible dependency tree languages.*

In other words, suppose that $\mathcal{L}$ is a reversible dependency tree language. We claim that there is a finite set $\mathbf{CS}(\mathcal{L})$ such that for each finite set H, if $\mathbf{CS}(\mathcal{L}) \subset H \subset \mathcal{L}$, then the LDG $\phi(H)$ generates exactly $\mathcal{L}$, i.e. $\mathcal{D}(\phi(H)) = \mathcal{L}$. The learning algorithm is incremental and runs in quadratic time in the size of the examples. Our algorithm is implementing as a Java prototype which is accessible from `http://www.loria.fr/~besombes`. The full demonstration is also available as a tachnical report, on the same web page.

Example 4.

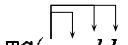
1. We illustrate the learning algorithm above from the set of samples



Fisrt, we compute $G_0 = \mathtt{TG}(H)$ whose productions are

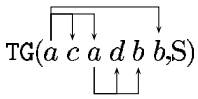
TG(c,S)

$$S \to c$$



TG(a c d b,S)

$$S \to a\ X_1\ X_2\ X_3$$
$$X_1 \to c$$
$$X_2 \to d$$
$$X_3 \to b$$



TG(a c a d b b,S)

$$S \to a\ X_4\ X_5\ X_6$$
$$X_4 \to c$$

$$X_5 \to a\ X_7\ X_8$$
$$X_6 \to b$$
$$X_7 \to d$$
$$X_8 \to b$$

Second, we apply **R1**. So we identify $S = X_1 = X_4$, $X_2 = X_7$, and $X_3 = X_6 = X_8$. Then, we have

$$S \to c$$

$$S \to a\ S\ X_2\ X_3$$

$$X_2 \to d$$

$$X_3 \to b$$

$$S \to a\ S\ X_5\ X_3$$

$$X_5 \to a\ X_2\ X_3$$

Now, we apply **R3** on S rules, by merging $X_2 = X_5$.

$$S \to c$$

$$X_2 \to d$$

$$X_3 \to b$$

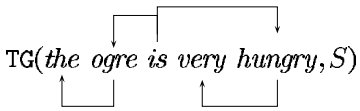
$$S \to a\ S\ X_2\ X_3$$

$$X_2 \to a\ X_2\ X_3$$

We see that the final grammar is LDG equivalent to Γ_1 of Example 1.

2. $H = \{$ *the ogre is very hungry*, *the ogre is hungry*, *the ogre is very very hungry*$\}$,

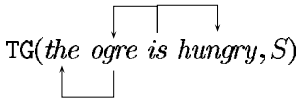
The productions of $G_0 = \texttt{TG}(H)$ are



$S \rightarrow X_1\ is\ X_2$

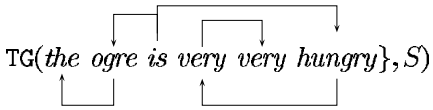
$X_1 \rightarrow X_3\ ogre$

$X_2 \rightarrow X_4\ hungry$
$X_3 \rightarrow the$
$X_4 \rightarrow very$



$S \rightarrow X_5\ is\ X_6$

$X_5 \rightarrow X_7\ ogre$
$X_6 \rightarrow hungry$
$X_7 \rightarrow the$



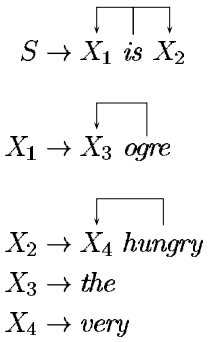
$S \rightarrow X_8\ is\ X_9$

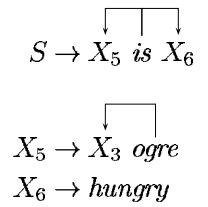
$X_8 \rightarrow X_{10}\ ogre$

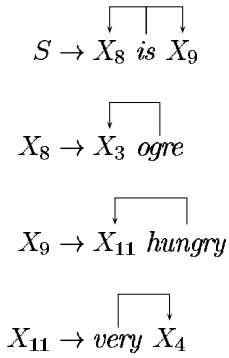
$X_9 \rightarrow X_{11}\ hungry$
$X_{10} \rightarrow the$

$X_{11} \rightarrow very\ X_{12}$
$X_{12} \rightarrow very$

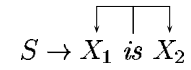
Second, we apply **R1** to identify $X_4 = X_{12}$, $X_3 = X_7 = X_{10}$.

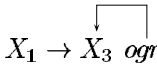


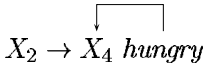




We apply **R1** to identify $X_1 = X_8$ and $X_5 = X_1$.

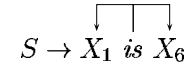




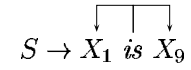


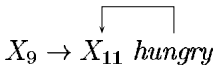
$X_3 \to the$

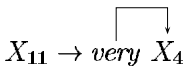
$X_4 \to very$



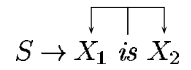
$X_6 \to hungry$

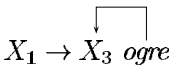


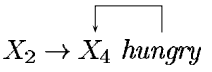


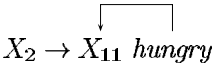


Now, we merge $X_2 = X_6 = X_9$ by aplying **R3** on S rules.









$X_2 \rightarrow \textit{hungry}$
$X_3 \rightarrow \textit{the}$
$X_4 \rightarrow \textit{very}$

$X_{11} \rightarrow \overset{\downarrow\ \downarrow}{\textit{very}\ X_4}$

Lastly, we apply **R2** on X_2 rules by merging $X_4 = X_{11}$. We obtain the final grammar:

$S \rightarrow \overset{\downarrow\ \ \downarrow\ \ \downarrow}{X_1\ \textit{is}\ X_2}$

$X_1 \rightarrow \overset{\downarrow\ \ \downarrow}{X_3\ \textit{ogre}}$

$X_2 \rightarrow \overset{\downarrow\ \ \downarrow}{X_4\ \textit{hungry}}$
$X_2 \rightarrow \textit{hungry}$
$X_3 \rightarrow \textit{the}$
$X_4 \rightarrow \textit{very}$

$X_4 \rightarrow \overset{\downarrow\ \ \downarrow}{\textit{very}\ X_4}$

6 Related works

- Sakakibara [11] gives a learning algorithm to infer reversible context free languages from skelton parse trees. The definition of reversible grammar is very similar to ours. However, we distinguish between both productions $X \rightarrow \overset{\downarrow\ \ \downarrow\ \ \downarrow\ \ \downarrow}{Y\ a\ b\ Z}$ and $X \rightarrow \overset{\downarrow\ \ \downarrow\ \ \downarrow\ \ \downarrow}{Y\ a\ b\ Z}$, unlike [11] in which they are considered identical.
- Kanazawa [8] studies inference of several classes of categorial grammars from functor structures, based on counting the number of categories associated to a terminal symbol. It is not difficult to faithfully translate rigid grammar in reversible dependency grammars.

The defect of learning from structures is that examples usually depend on the implicit grammar that we have to guess. It appears that it is not the case in our approach beacause we deal with tree languages, and so is seemingly more natural

7 Future works

The concept of reversibility, which takes its root in [1], has yet to be explored. In passing, notice that there are universal reversible Turing machine[2] So, we see that the "reversible computation hypothesis" does not decrease expresivity. In particular, we are investing reversible categorial grammar, and reversible Lambek calculus.

References

1. Dana Angluin. Inference of reversible languages. *Journal of the ACM*, 29:741–765, 1982.
2. Y. Bar-Hillel, C. Gaifman, and E. Shamir. On categorial and phrase structure grammars. *Bulletin of Research Council of Israel*, F(9):1–16, 1960.
3. Alexander Dikovsky and Larissa Modina. Dependencies on the other side of the curtain. *Traitement automatique des langues*, 41(1):67–96, 2000.
4. H. Gaifman. Dependency systems and phrase structure systems. *Information and Control*, 8(3):304–337, 1965.
5. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
6. D.G. Hays. Grouping and dependency theories. In *National symp. on machine translation*, 1961.
7. J. Jain, D. Osherson, J. Royer, and A. Sharma. *Systems that learn*. MIT press, 1999.
8. Makoto Kanazawa. *Learnable classes of categorial works*. CSLI, 1998.
9. E. Mäkinen. Inferring regular languages by merging nonterminals. Technical report, University of Tampere, 1997.
10. Igor Mel'čuk. *Dependency Syntax: Theory and Practice*. The SUNY Press, 1988.
11. Y. Sakakibara. Efficient learning of context free grammars from positive structural examples. *Information and Computation*, 97:23–60, 1992.
12. D. Sleator and D. Temperley. Parsing english with a link grammar. Technical report, Carnegie Mellon University Computer Science, 1991.
13. Lucien Tesnière. *Eléments de syntaxe structurale*. Klincksieck, 1959.
14. Jacques Vergne. Etude et modélisation de la syntaxe des langues à l'aide de l'ordinateur. Analyse syntaxique automatique non combinatoire, Sept 1999. Habilitation Thesis.

[2] See `http://www.ai.mit.edu/~cvieri/reversible.html`

Learning Rigid Lambek Grammars and Minimalist Grammars from Structured Sentences*

Roberto Bonato[1] and Christian Retoré[2]

[1] IRISA-INRIA, Campus Universitaire de Beaulieu, 35042 Rennes Cedex, France
rbonato@irisa.fr
[2] IRIN, Université de Nantes, BP 92208, 44322 Nantes Cedex 03, France
retore@irisa.fr

Abstract. We present an extension of Buszkowski's learning algorithm for categorial grammars to rigid Lambek grammars and then for minimalist categorial grammars. The Kanazawa proof of the convergence in the Gold sense is simplified and extended to these new algorithms. We thus show that this technique based on principal type algorithm and type unification is quite general and applies to learning issues for different type logical grammars, which are larger, linguistically more accurate and closer to semantics.

1 Presentation

Learning lexicalized grammar This papers deals with automated syntax learning, also known as grammatical inference. The grammars we consider are lexicalized: their rules are universal, do not depend on the language, hence a grammar is completely defined by a map called the lexicon which associates to each word a finite set of objects, here types, which define the word syntactic behavior. In such grammar formalisms, acquiring a grammar consists in finding which types should be associated with a word in order that the lexicon generates the examples (computational linguistics does not usually consider counter examples). It should be observed that grammar formalisms that allow for a learning algorithm are rare.

Gold model There are various models for grammatical inference. Here we follow the model proposed by Gold [8]: *identification in the limit from positive examples only*. According to this model, each time the learner meets a new example, she generalizes her current grammar hypothesis to a grammar in the class which generates all the examples met so far. The learning process is said to be convergent if, whenever the set of examples is an enumeration of a language generated by a grammar in the class, the learner finds the correct grammar after a finite number of examples.

* This work is part of the INRIA *Action de Recherche Coopérative* GRACQ, Categorial Grammar Acquisition

Psycholinguistic aspects If one thinks, but this can be discussed, that a grammatical learning algorithm should roughly follow what is known of human language acquisition, this model is rather accurate: indeed, as claimed by Gleitman, Lieberman or Pinker in [7], it seems that only positive examples are used in the process of children's first language acquisition. Another fact that we do take into account is that learning proceed on structures, and it is known that children use structures to learn (they are provided by intonation, semantic contents etc.).

Nevertheless our kind of algorithm relying on generalization by unification departs from the actual process of learning: children do not go through an increasing sequence of languages till they reach the correct one, they rather go through languages which hardly intersect the language to be learnt.

Tools for automated grammar construction We are presently implementing such algorithms, using Objective CaML with a Tcl/Tk interface, and proceeding with XML files for representing structured data — directed acyclic graphs which embed all the various structures we are using. The learning algorithms are not too difficult to implement, and rather efficient.

A positive point is that we are able to learn classes of languages relevant for natural language syntax, which as far as we know, have not yet been addressed with respect to learning issues. The algorithms are quite fast. Nevertheless this is due to the fact that we learn from structures, but from structures which are too rich to be available from corpora, even tagged corpora. However it is necessary to learn from structures; when learning from plain strings of words nothing would prevent the grammar learnt from generating sentences like *Adam ate an apple* with the structure *(Adam (ate an)) apple*, which is not what we are looking for.

Another positive point is that we deal with lexicalized grammar. The grammatical words, which are the ones associated with the most sophisticated syntactic behavior are in a finite, fixed number: pronouns, prepositions and such are not invented by speakers, hence in a real application they should be already present in the lexicon, and the words which remain to be learnt are always simple words like nouns, adjectives, verbs. This advantage is minimized by the fact these grammars are not robust, and consequently have a small empirical coverage.

Previous work This work mainly relies on the first learning algorithm for categorial grammars of Buszkowski and Penn [4] which was shown to be convergent in the Gold sense by Kanazawa [10] — we use the simplified version due to the first author [2].

In order to learn a class of lexicalized grammars with a hope to be convergent, one has to bound the maximal number of types that the lexicon associates to a given word: otherwise each time a new word is seen the algorithm could provide it with a new type perfectly ad hoc for this particular sentence, and nothing would be learnt. The base case obviously is when the lexicon maps each word to one type: such grammars are called *rigid grammars* — and this limits their generative capacity.

The pioneering paper [4] provides an algorithm called RG for learning efficiently rigid AB grammars from structures — where AB grammars is a class of categorial grammars which only contains residuation rules. RG relies on a typing algorithm (an

easy adaptation of the principal type algorithm for simply typed λ-calculus) and unification (with was shown to be a general tool for grammatical inference by Nicolas [16]).

Kanazawa [10] proved the convergence of RG, and extended the result in two directions that can be combined. One direction was to learn k-valued AB grammars by testing all the possibilities of unifying a type with one of the k types, and the other was to learn from strings by trying all possible parse structures on a given string. The convergence of these compatible extensions was shown, but the complexity of the learning algorithms becomes hardly tractable.

Aims of the paper Compared to Kanazawa, we extend the RG algorithm in an orthogonal direction: we moved to Lambek categorial grammars which are obtained by adding two rules to the two rules of AB grammars, and, in the same style we moved to the richer deductive system and grammar defined with Lecomte [13] which enables a representation of Stabler's minimalist grammars [18] a formalization of Chomsky's minimalist program [5]. In both cases we wish to learn more realistic classes of languages without losing the efficiency of the original algorithm. Furthermore the methods of Kanazawa for extending our results to k valued grammars or for learning from strings probably apply here as well.

Little is known regarding the class of rigid Lambek languages: all its languages are context-free and some are not regular, but it does not include all regular languages — learnable classes are always transversal to the Chomsky hierarchy [8]. We do think, from the examples we tried, e.g. example 1 in the paper, that rigid Lambek grammars are more expressive than rigid AB grammars — although when unrestricted both AB grammars and Lambek grammars exactly describe context free languages. Regarding minimalist grammars, we know that rigid minimalist grammars do contain non context free languages like the language $a^n b^n c^n d^n e^n$ (Stabler, private communication). Furthermore minimalist grammars take into account sophisticated syntactic constructions, and enjoy polynomial parsing.

As it well-known, learning is a cornerstone in Chomskyan linguistics. Language acquisition is viewed as parameter setting in the universal grammar — hence few examples are needed to acquire the syntax. These parameters, like feature strength, are explicit in minimalist grammars, and this could yield a more efficient learning strategy.

Another motivation is semantics. If the lexicon includes simply-typed λ-terms depicting word semantics, there is a straightforward mapping from syntactic analyses to semantics [15]. As real learning makes use of some semantic information, the possibility to learn Lambek grammars should improve syntax learning from sentences enriched with semantic information as suggested for instance in [19]. Although the syntax/semantics interface is less simple for minimalist grammars [13], it is also computable, and this interface could be exploited as well.

Regarding learning theory we think that the RG algorithm and the simple proof of convergence of [10, 2] could lead to a general result on learnability which can be applied to most type logical grammars.

2 Categorial grammars: BCG, LCG, MCG

In categorial grammars, the objects that the lexicon associates to words to rule their syntactic behavior are *types* (also called *categories* or *formulae*):

$$Tp \quad ::= \quad Pr \quad | \quad Tp/Tp \quad | \quad Tp\backslash Tp$$

The set Pr is the set of base categories which must contain a distinguished category s (sentence) and usually contains categories np (noun phrase) and n (noun), and possibly vp (verb phrase), pp (prepositional phrase). Stated informally, an expression e is of type B/A (resp. $A\backslash B$) when it needs to be followed (resp. preceded) by an expression e' of type A to obtain an expression of type B.

Given $a \in \Sigma$ (the set of words) and $A \in Tp$, we write $G : a \mapsto A$ (G *assigns* A *to* a), whenever A is one of the types associated to the word a. If for all $a \in \Sigma$ G assigns at most one type to a, G is said to be **rigid.**

Given a grammar/lexicon G, a sequence[1] of words $w_1 \dots w_n \in \Sigma^+$ is said to be of type $T \in Tp$ whenever there exists for every i in $[1, n]$ a type $T_i \in Tp$ such that $T_1 \cdots T_n \vdash T$, where $\vdash$ is the following relation between finite sequences of types and types:

Definition 1. *The binary relation* $\vdash$ *between* Tp^+ *and* Tp *is the smallest relation such that, for all* $A, B \in Tp$ *and all* $\Gamma, \Delta \in Tp^+$*:*

$[ID]$ $A \vdash A$
$[/E]$ *if* $\Gamma \vdash A$ *and* $\Delta \vdash A\backslash B$*, then* $\Gamma, \Delta \vdash B$ *(Forward application or modus ponens);*
$[\backslash E]$ *if* $\Gamma \vdash B/A$ *and* $\Delta \vdash A$*, then* $\Gamma, \Delta \vdash B$ *(Backward application, modus ponens);*
$[\backslash I]$ *if* $A, \Gamma \vdash B$ *then* $\Gamma \vdash A\backslash B$ *(*$\backslash$ *hypothetical reasoning, introduction);*
$[/I]$ *if* $\Gamma, A \vdash B$ *then* $\Gamma \vdash B/A$ *(/ hypothetical reasoning, introduction);*

As the $\vdash$ symbol suggests it is a deduction relation which can be depicted by natural deduction trees. [2]

$$A\ [ID] \qquad \frac{\begin{array}{cc}\vdots & \vdots\\ A/B & B\end{array}}{A}\,[/E] \qquad \frac{\begin{array}{cc}\vdots & \vdots\\ B & B\backslash A\end{array}}{A}\,[\backslash E] \qquad \frac{\begin{array}{c}[B]\\ \vdots\\ A\end{array}}{A/B}\,[/I] \qquad \frac{\begin{array}{c}[B]\\ \vdots\\ A\end{array}}{B\backslash A}\,[\backslash I]$$

Note: in [/I] and [\I] rules the cancelled or discharged hypothesis is always the rightmost and the leftmost uncanceled hypothesis, respectively, and there must remain at least one other uncanceled hypothesis.

The grammars defined using this deductive system are Lambek grammars (**LCG**) from [12] while the ones which only use $[\backslash R]$ and $[/E]$ are called AB (Ajdukiewicz Bar-Hillel) grammars or basic categorial grammars (**BCG**) from [1] (see [3] for a survey). The learning algorithms defined so far [4, 10] only handles BCGs.

[1] As usual, given a set U, U^* stands for the finite sequences of elements in U and U^+ for the finite non empty sequences of elements in U

[2] The only difference with usual natural deduction is that the order of premises matters, and that exactly one hypothesis is cancelled in an introduction rule. This can be observed from the fact that, in the previous definition there are no rule *if* $\Gamma, A, B, \Delta \vdash C$ *then* $\Gamma, B, A, \Delta \vdash C$ and no rule *if* $\Gamma, A, B, \Delta \vdash C$ *then* $\Gamma, A, B, \Delta \vdash C$.

Example 1.

If one consider the rigid grammar/lexicon besides, then the sentences *Guarda passare il treno* (*(s)he is looking the train passing by*) and *Cosa guarda passare* (*What is (s)he looking passing by?*) belong to the generated language of the LCG. With BCG, only the first one is generated.

$$\begin{array}{r} cosa \mapsto S/(S/np) \\ guarda \mapsto S/np \\ passare \mapsto vp/np \\ il \mapsto np/n \\ treno \mapsto n \end{array}$$

The languages generated by AB categorial grammars exactly are the context-free languages, and the languages generated by Lambek categorial grammars also are exactly context-free languages. Nevertheless the LCG generate much richer structure languages than BCG, see below.

Minimalist categorial grammars (MCG) In [13] the minimalist grammars of [18] have defined similarly, by using a lexicon which ranges over more sophisticated types. There is no room to present there intuitively, but formally they can be defined a mere variation of categorial grammars. The set of primitive categories is larger: it includes primitive categories but also movement features, like case, wh etc. which can be strong or weak[3] Types are extended with a commutative and associative product:

$$Tp_m \quad ::= \quad Pr \quad | \quad Tp_m/Tp_m \quad | \quad Tp_m\backslash Tp_m \quad | \quad Tp_m \otimes Tp_m$$

Instead of sequences of types, contexts are sets of types and the deductive system is replaced with the following one, where every formula is labeled by a sequence of words:[4]

$[ID]$ $x : A \vdash x : A$

$[AX]$ $\vdash w : A$ whenever $w \mapsto A$ is in the lexicon.

$[/E]$ if $\Gamma \vdash x : A$ and $\Delta \vdash y : A\backslash B$, then $\{\Gamma, \Delta\} \vdash xy : B$

$[\backslash E]$ if $\Gamma \vdash y : B/A$ and $\Delta \vdash x : A$, then $\{\Gamma, \Delta\} \vdash yx : B$

$[\otimes E]$ if $\Delta \vdash u : A \otimes B$ and $\Gamma[\{x : A, y : B\}] \vdash t(x,y) : C$ then $\Gamma, \Delta \vdash t(x := u_1, y := u_2) : C$ with, depending on A either $u_1 = u_2 = u$ or $u_1 = (u)$ and $u_2 = u$.

The labels can be computed from the proof. The way to read them is a bit puzzling, but agrees with the copy theory of minimalism. Every word sequence w has a phonological part $/w/$ and a semantic part (w) and when they are superimposed, this is simply denoted by w. The reason for this double sequence (semantic and syntactic) is to obtain correct semantic representation in a parameterized view of language variation.

To read such a sentence, one should only keep the phonological part and forget items which are met several times, the repetition being traces: for instance *Petrus Mariam amat Mariam* is phonologically read as */Petrus/ /Mariam/ /amat/* and semantically read as *(Petrus) (Mariam) (amat)* while *Peter (Mary) loves Mary* is phonologically read as

[3] For instance in VSO languages the accusative case provided by the verb is weak, while it strong in SOV languages.

[4] In fact if one wants a real logical system, then it is more complex. It involves structured contexts with a commutative comma and a non commutative comma, and the possibility to replace non commutative commas by commutative ones. However a large part of this formalism can be handled by the little system we give here.

/Peter/ /loves/ /Mary/ and semantically read as *(Peter) (Mary) (loves)*. If we consider the assignments $Mary \mapsto d \otimes k$ and $loves \mapsto K\backslash vp/d$ we obtain that *(Mary) loves Mary* is a vp, but if case k was strong we would obtain that *Mary loves Mary* is a vp as in SOV languages.

$$\dfrac{\vdash Mary : d \otimes k \qquad \dfrac{y : K \vdash y : K \qquad \dfrac{\dfrac{\vdash loves : K\backslash vp/d \quad x : d \vdash x : d}{x : d \vdash loves\ x : k \backslash vp}[/E]}{y : K; x : d \vdash y\ loves\ x : vp}[\backslash E]}{y : K, x : d \vdash y\ loves\ x : vp}}{\vdash (Mary)\ loves\ Mary : vp}[\otimes E]$$

The grammar is said to generate a sentence $w_1 \dots w_n$ whenever the lexicon contains assignments $w_i \mapsto T_i$ such that $T_1, \dots, T_n \vdash /w_1/ \dots /w_n/ : s$. These grammars have been shown to generate exactly the same languages as Multiple Context-Free grammars, which go beyond context free grammars and even TAG languages. [9, 14].

2.1 Proofs as Parse Structures

As said above, parse structures are essential to a grammar system. As Tiede [20], we define them as *normal* proof trees that are proof trees in which an introduction rule yielding $A\backslash B$ or B/A is never followed by an elimination rule whose argument is A. Every proof tree can be normalized in linear time to a normal one, and there are finitely many normal proof tree with the same hypotheses and conclusion.

Thus the structure underlying a sentence $a_1 \cdots a_n \in \Sigma^+$ generated by a Lambek grammar G is one of the finitely many normal proof trees of the deduction $A_1, \dots, A_n \vdash s$, with $G : a_i \mapsto A_i$. Figure 1 displays the proof tree of the sentence 'he likes her' in a grammar G such that G : he $\mapsto s/(np\backslash s)$, him $\mapsto (s/np)\backslash s$, likes $\mapsto (np\backslash s)/np$.

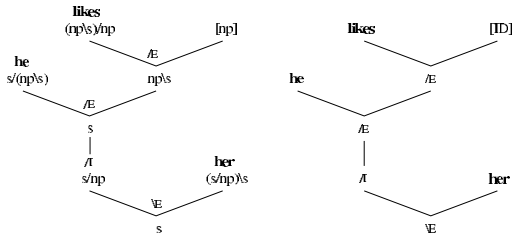


Fig. 1. Proof tree for a sentence and the corresponding proof-tree structure.

Given a Lambek grammar G, a *proof-tree structure* over its alphabet Σ is a unary-binary branching tree whose leaf nodes are labeled by either $[ID]$ ("discharged" leaf nodes) or symbols of Σ and whose internal nodes are labeled by either $[\backslash E]$, $[/E]$, $[\backslash I]$,

or $[/I]$. Thus, a proof tree structure of a sentence is obtained from a proof tree by removing the types which label the nodes. We write Σ^P for the *structure language* that is the set of proof-tree structures over Σ, and $\mathrm{PL}(G)$ for the *(proof-tree) structure language* of G that is the set of structures generated by G. In order to distinguish $\mathrm{L}(G)$, the language of G, from $\mathrm{PL}(G)$, its structure language, the former is called the *string language* of G. The *yield* of a proof-tree structure T is the string of symbols $a_1, \ldots, a_n \in \Sigma^+$ labelling the undischarged leaf nodes of T, from left to right in this order. The yield of T is denoted $\mathit{yield}(T)$. Note that $\mathrm{L}(G) = \{\mathit{yield}(T) \mid T \in \mathrm{PL}(G)\}$.

By taking a proof tree as the structure of a sentences generated by Lambek grammars, Tiede in [20] proved some important results about their strong generative capacity. Firstly, a proof in the Lambek calculus is really a tree — as opposed to a proof in intuitionistic logic. Indeed, as he observed there is no need to specify which introduction rule $[I/]$ or $[\backslash I]$ did cancel a leaf in the course of a derivation: this information can be reconstructed from the leaves to the root, since it is always the left the leftmost or right most uncanceled hypothesis. Secondly, as opposed to a previous notion of parse structures for Lambek grammars [3], not every bracketing is produced by the finitely many normal proof trees which parse a given sentence. Thirdly, even though Lambek grammars and BCGs both exactly generate context-free languages, the former can generate proper context-free tree languages while the latter, as context free grammars only generate regular tree languages.

Parse structures for minimalist categorial grammars For the same reason we take proof trees as parse structure for MCG. Observe that there as well subtrees correspond to constituents.

3 Learning Framework: Gold's Model from Structured Examples

This sections recalls the basic formal notions of the learning model we use, Gold's model of learning [8], also called *identification in the limit from positive data*. We are given a universe T (for us the set of parse structures) and a class Ω of finite descriptions of subsets of T (for us categorial grammars), with a naming function N which maps each element of Ω to the subset of T that it generates. The learning question is given a subset S of T (positive examples), to find an element $G \in \Omega$ such that $N(G) = S$. The learning function we are looking for is a partial function φ from finite sequences of elements in T (or in S, since it is a partial function) to Ω which converges in the sense of Gold: whenever $S = N(G)$ for some $G \in \Omega$, then for any enumeration $(e_i)_{i\in\mathbb{N}}$ of S, there exists an $p \in \mathbb{N}$ such that $\forall n > p$ one has $\phi(e_1, \ldots, e_n) = G'$ and $N(G') = N(G)$.

Stated informally, we can think of a learning function as a formal model of the cognitive process (successful or unsuccessful) by which a learner conjectures that a given finite set of positive samples of a language are generated by a certain grammar. The convergence simply means that after a finite number of examples, the hypothesis made by the learner is a grammar equivalent to the correct one.

In Gold's model, we will say that a class of grammars is *learnable* when for *each* language generated by its grammars there exists a learning function which converges

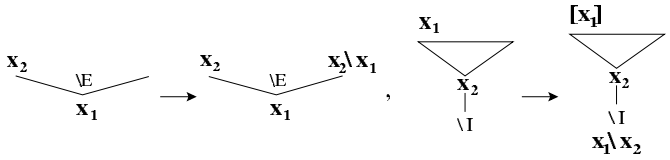


Fig. 2. Typing rules

to one correct underlying grammar; it is called *effectively learnable* if that function is computable.

4 An Algorithm for Learning Rigid Lambek Grammars

In the present section we describe an extension of Buszkowski's RG algorithm [4] for learning rigid BCGs from structured positive examples to rigid Lambek grammars, and rigid minimalist grammars. We also sketch the first author's convergence proof [2] inspired from [10].

4.1 Argument Nodes and Typing Algorithm

Our learning algorithm is based on a process of labeling for the nodes of a set of proof-tree structures. We introduce here the notion of *argument node* for a normal form proof tree. Sometimes we will use the same notation to indicate a node and its type label, since the graphical representation of trees avoids any confusion.

Definition 2. *Let T be a normal form proof-tree structure. Let's define inductively the set $Arg(T)$ of argument nodes of T. There are four cases to consider:*

- *T is a single node, which is the only member of $Arg(T)$;*
- *$T \equiv [\backslash E](T_1, T_2)$, then $Arg(T) = \{Root(T)\} \cup Arg(T_1) \cup Arg(T_2) - Root(T_2)$;*
- *$T \equiv [/E](T_1, T_2)$, then $Arg(T) = \{Root(T)\} \cup Arg(T_1) \cup Arg(T_2) - Root(T_1)$;*
- *$T \equiv [\backslash I](T_1)$ or $T \equiv [/I](T_1)$, then $Arg(T) = Arg(T_1)$.*

Applying the principal type algorithm from λ-calculus to Lambek proofs, as in [20, 2], one obtains the following result on argument nodes:

Proposition 1. *Let T be a well formed normal form proof-tree structure. If each argument node is labeled, then any other node in T can be labeled with one and only one type.*

This proposition allow to define the notion of principal parse:

Definition 3. *A* principal parse *of a proof-tree structure T is a partial parse tree $\mathcal{T}$ of T, such that for any other partial parse tree $\mathcal{T}'$ of T, there exists a substitution σ such that, if a node of T is labeled by type A in $\mathcal{T}$, it's labeled by $\sigma(A)$ in $\mathcal{T}'$.*

To compute in linear time the principal parse of any well-formed normal proof-tree structure, provide argument nodes with distinct variables, and then apply the rules in figure 2 (the $/E$ and $/I$ cases are symmetrical).

4.2 RLG (Rigid Lambek Grammar) Algorithm

- **input:** a finite set D of *normal* proof-trees (since all proof trees normalize)
- **output:** a rigid Lambek grammar G such that $D \subset \mathrm{PL}(G)$, if there is one.

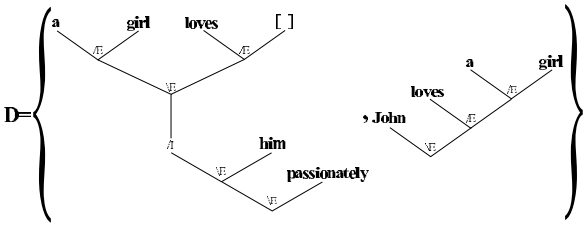


Fig. 3. The set D of positive structured samples for RLG algorithm.

Step 1. Assign a type to each node of the structure in D as follows:

- Assign s to each root node;
- assign distinct variables to the argument nodes (see Fig. 4);
- compute types for the remaining nodes as after definition 3 (Fig. 5).

Step 2. Collect the types assigned to the leaf nodes into a grammar $GF(D)$ called the *general form* induced by D. One has $GF(D) : c \mapsto A$ if and only if the previous step assigns A to a leaf node labeled by symbol c.

$$GF(D) : \mathbf{passionately} \mapsto x_1\backslash s \qquad \mathbf{him} \mapsto (x_2/x_5)\backslash x_1$$
$$\mathbf{a} \mapsto x_3/x_4, x_7/x_8 \qquad \mathbf{girl} \mapsto x_4, x_8$$
$$\mathbf{loves} \mapsto (x_3\backslash x_2)/x_5, (x_6\backslash s)/x_7 \qquad \mathbf{John} \mapsto x_6$$

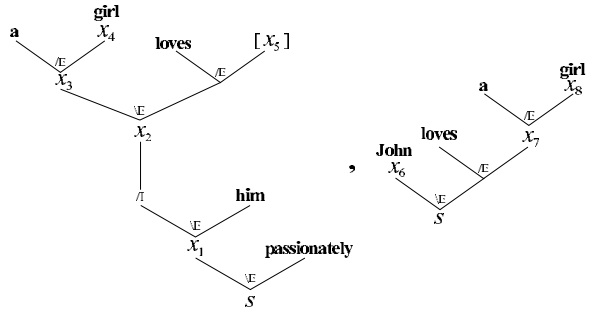


Fig. 4. Set D after labeling argument nodes.

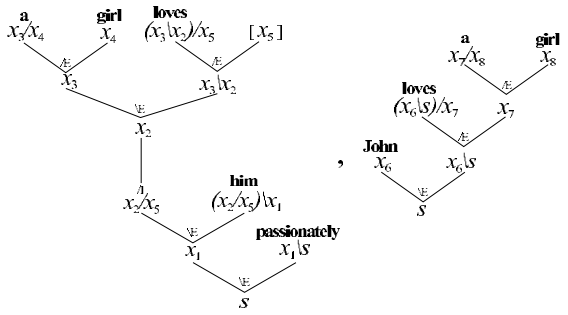


Fig. 5. Set D after computing the label for every node.

Step 3. Unify the types assigned to the same symbol. Let $\mathcal{A} = \{\{A \mid GF(D) : c \mapsto A\} \mid c \in dom(GF(D))\}$, and compute the most general unifier $\sigma = mgu(\mathcal{A})$. There are various well-known algorithms for unification (see [11] for a survey), so a most general unifier of a finite set of types can be computed in linear time. The algorithm fails if unification fails.

$$\sigma = \{x_7 \mapsto x_3, x_8 \mapsto x_4, x_6 \mapsto x_3, x_2 \mapsto s, x_5 \mapsto x_3\}$$

Step 4. Let $RLG(D) = \sigma[GF(D)]$.

$$\begin{array}{lll} RLG(D): & \mathbf{passionately} \mapsto x_1\backslash s & \mathbf{him} \mapsto (s/x_3)\backslash x_1 \\ & \mathbf{a} \mapsto x_3/x_4 & \mathbf{girl} \mapsto x_4 \\ & \mathbf{loves} \mapsto (x_3\backslash s)/x_3 & \mathbf{John} \mapsto x_3 \end{array}$$

Our algorithm is based on the *principal parse algorithm* described in the previous section, which has been proved to be correct and terminate, and the unification algorithm. The result is, intuitively, the *most general* rigid Lambek Grammar which can generate all the proof tree structures appearing in the input sequence.

Theorem 1. *Let φ_{RLG} be the learning function for the grammar system $\langle G_{rigid}, \Sigma^P, \mathrm{PL}\rangle$ defined by $\varphi_{RLG}(\langle T_0, \ldots, T_n\rangle) \simeq RLG(\{T_0, \ldots, T_n\})$. Then φ_{RLG} learns $\mathcal{G}_{rigid}$ from structures.*

We have no room for the complete proof which can be found in [2], we only state the key lemmas below. Given two non necessarily rigid LCGs, let us write $G \sqsubset G'$ whenever there exists a substitution σ such that $\sigma(G) \subset G'$ — that is every type assignment of G is a type assignment of $\sigma(G)$. Given a finite set D of examples, let us call $GF(D)$ the non rigid grammar obtained by collecting all the principal types associated to the words by the examples.

1. Given a grammar G are finitely many grammars H such that $H \sqsubset G$.
2. If $G \sqsubset G'$ then $\mathrm{PL}(G) \subset \mathrm{PL}(G')$.
3. If $D \subset \mathrm{PL}(G)$ then $GF(D) \sqsubset G$.

4. If $GF(D) \sqsubset G$ then $D \subset \mathrm{PL}(G)$.
5. If $RLG(D)$ exists and $RLG(D) \subset G$ then $D \subset FL(RLG(D))$.
6. If $D \subset \mathrm{PL}(G)$ then $RLG(D)$ exists and $RLG(D) \sqsubset G$.

Given these lemmas, it is not difficult to see that $D \subset D' \subset \mathrm{PL}(G)$ entails that $RLG(D), RLG(D'$ exists and $RLG(D) \sqsubset RLG(D') \sqsubset G$. Because of the first item a correct grammar is necessarily met in finitely many steps.

When RLG is applied successively to a sequence of increasing set of proof-tree structures $D_0 \subset D_1 \subset D_2 \subset \cdots$, it is more efficient to make use of the previous value $RLG(D_{i-1})$ to compute the current value $RLG(D_i)$.

It is easy to see that φ_{RLG} can be implemented to run in linear time if positive structured samples are in normal form. As said earlier both the principal type algorithm for Lambek proof trees [20] and efficient unification algorithms [11] are linear.

Learning MCG Rigid minimalist grammars admit a similar algorithm RMG. Here are the differences. To compute the principal type when an $[\otimes]$ rule is met, the subproof sp with conclusion $A \otimes B$ should be typed after the other subproof has been typed, hence the value of $A \otimes B$ is known for typing sp. Whenever a set of examples D is included in $\mathrm{PL}(G)$ then there also exists a substitution σ such that $\sigma(GF(D)) \subset G$. The unification of the collected types is more difficult, because the connective $\otimes$ is associative and commutative, but it has been shown by Fages that unification modulo associativity and commutativity is decidable, even in the presence of other connectives [11]; the difference is that there does not exists a most general unifier, but a finite set of minimal unifiers $(\sigma_i)_{i \in [1,p]}$, and for at least one index $i_0 \in [1,p]$, $\sigma = \tau\sigma_{i_0}$. We define $RMG(D)$ to be $\sigma_{i_0}(GF(D))$. Thus $\tau(RMG(D)) = \tau\sigma_{i_0}(GF(D)) = \sigma(GF(D)) \subset GF(D)$ and $RMG(D) \sqsubset G$. Thus the argument for the convergence of RLG also applies to RMG.

5 Conclusion

Meanwhile we implement some algorithms in the family we presented in Objective CaML with input as DAGs in XML, and interface in Tcl/Tk, we would like to address the following questions:

- What is a good notion of structure for learning syntax from structures, that is a notion of structure which yields good grammars, but is rather computable from corpora?
- How can one automatically obtain parse structure from more realistic examples that can actually be produced by taggers and robust parsers?
- How can we take Montague like semantics into account, first for Lambek grammars, and then for minimalist grammars?
- What are the language classes of rigid grammars?
- Can we used the fact that types for minimalist grammars are of a given shape to avoid associative commutative unification and bound the complexity?
- Does there exists a general result stating that every type logical grammars enjoying certain properties is learnable from structures?

References

1. Y. Bar-Hillel. A quasi arithmetical notation for syntactic description. *Language*, 29:47–58, 1953.
2. Roberto Bonato. A study on learnability of rigid Lambek grammars. Tesi di Laurea & Mémoire de D.E.A, Università di Verona & Université Rennes 1, 2000. available at http://www.irisa.fr/aida/aida-new/Fmembre_rbonato.html.
3. Wojciech Buszkowski. Mathematical linguistics and proof theory. In van Benthem and ter Meulen [21], chapter 12, pages 683–736.
4. Wojciech Buszkowski and Gerald Penn. Categorial grammars determined from linguistic data by unification. *Studia Logica*, 49:431–454, 1990.
5. Noam Chomsky. *The minimalist program*. MIT Press, Cambridge, MA, 1995.
6. Philippe de Groote, Glyn Morrill, and Christian Retoré, editors. *Logical Aspects of Computational Linguistics, LACL'2001*, volume 2014 of *LNCS/LNAI*. Springer-Verlag, 2001.
7. L.R. Gleitman and M. Liberman, editors. *An invitation to cognitive sciences, Vol. 1: Language*. MIT Press, 1995.
8. E.M. Gold. Language identification in the limit. *Information and control*, 10:447–474, 1967.
9. Henk Harkema. A characterisation of minimalist languages. In de Groote et al. [6], pages 193–211.
10. Makoto Kanazawa. *Learnable classes of categorial grammars*. Studies in Logic, Language and Information. FoLLI & CSLI (distributed by Cambridge University Press), 1998. published version of a 1994 Ph.D. thesis (Stanford).
11. Claude Kirchner and Hélène Kirchner. *Rewriting, Solving, Proving*. LORIA, 2000. Book draft available from http://www.loria.fr/~ckirchne.
12. Joachim Lambek. The mathematics of sentence structure. *American mathematical monthly*, 65:154–169, 1958.
13. Alain Lecomte and Christian Retoré. Extending Lambek grammars: a logical account of minimalist grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics, ACL 2001*, pages 354–361, Toulouse, July 2001. ACL.
14. Jens Michaelis. Transforming linear context-free rewriting systems into minimalist grammars. In de Groote et al. [6], pages 228–244.
15. Michael Moortgat. Categorial type logic. In van Benthem and ter Meulen [21], chapter 2, pages 93–177.
16. Jacques Nicolas. Grammatical inference as unification. Rapport de Recherche RR-3632, INRIA, 1999. http://www.inria.fr/RRRT/publications-eng.html.
17. Christian Retoré, editor. *Logical Aspects of Computational Linguistics, LACL'96*, volume 1328 of *LNCS/LNAI*. Springer-Verlag, 1997.
18. Edward Stabler. Derivational minimalism. In Retoré [17], pages 68–95.
19. Isabelle Tellier. Meaning helps learning syntax. In *Fourth International Colloquium on Grammatical Inference, ICG'98*, 1998.
20. Hans-Jörg Tiede. Lambek calculus proofs and tree automata. In Michael Moortgat, editor, *Logical Aspects of Computational Linguistics, LACL'98, selected papers*, volume 2014 of *LNCS/LNAI*. Springer-Verlag, 2001.
21. J. van Benthem and A. ter Meulen, editors. *Handbook of Logic and Language*. North-Holland Elsevier, Amsterdam, 1997.

From Logic to Grammars via Types*

Daniela Dudau-Sofronie, Isabelle Tellier, Marc Tommasi

LIFL-Grappa Team and Université Charles de Gaulle-lille3
59 653 Villeneuve d'Ascq Cedex, FRANCE
dudau@lifl.fr, tellier,tommasi@univ-lille3.fr
http://www.grappa.univ-lille3.fr

Abstract. This paper investigates the inference of Categorial Grammars from a new perspective. To learn such grammars, Kanazawa's approach consists in providing, as input, information about the structure of derivation trees in the target grammar. But this information is hardly arguable as relevant data from a psycholinguistic point of view. We propose instead to provide information about the *semantic type* associated with the words used. These types are considered as general semantic knowledge and their availability is argued. A new learning algorithm from types is given and discussed.

1 Introduction

Learning a foreign language from texts is like trying to decipher hieroglyphs without the Rosetta stone: it is an unreachable challenge. Without the indications about their meaning found in the stone, hieroglyphs would probably still remain mysterious. This paper can be interpreted as a tentative explanation of how semantics can help syntax learning.

Categorial Grammars are well known for their formalized connection with semantics ([Mon74], [DWP81]). They provide a good compromise between formalism and linguistic expressivity ([OBW88]). Previous works have studied the learnability of such grammars ([Adr92], [Kan98], [MO98]) but neither of them uses the syntax/semantics interface to help the syntactic learning process.

Links between Kanazawa's learning strategy and semantic information have been shown in [Tel99]. This first approach is still not satisfactory as it does not avoid combinatorial explosion. This paper is a new way of considering learning Categorial Grammars from semantic knowledge. The original contribution consists in the use of *semantic types* associated with words.

Types are very general information, allowing to distinguish facts from entities and from properties satisfied by entities. Most knowledge representation languages use this notion, and it ususally seems possible to deduce types from lexical semantics. A new learning algorithm taking as input data both syntactically correct sentences and the corresponding sequences of types is explained.

Sections 2 and 3 are preliminaries introducing the class of Grammars we want to learn and the nature of the information admitted as input to the learning

* this work was supported by the ARC INRIA project GRACQ

process. Section 4 exposes the heart of our proposition, illustrated with a detailed example, and criticizes its limitations.

The conclusion argues about the cognitive plausibility of the data provided to this algorithm, in comparision with the data usually used in other learning algorithms. Perspective issues are also evoked.

2 Grammatical Inference

The Problem of *Grammatical Inference* from positive examples (or: from texts) consists in the design of algorithms able to identify a formal grammar from a sample of sentences it generates. This problem is a rough formal approximation of how children manage to learn the grammar of their mother language ([WC80]). From a theoretical point of view, identifying a formal grammar from texts is very difficult since regular (and therefore context-free) grammars are not learnable from texts in usual learning models ([Gol67,Val84]).

2.1 Categorial Grammars

In the following, we will use formal grammars belonging to the class of *AB-Categorial Grammars* ([HGS60]). In this formalism, every member of the vocabulary (every word) is associated with a finite set of categories, expressing its combinatorial power. The set C' of every possible category is built from a finite set C of basic categories in the following way: C' is the smallest set satisfying $C \subset C'$ and for every $X \in C'$ and $Y \in C'$ then $(X/Y) \in C'$ and $(Y\backslash X) \in C'$[1].

Syntactic rules are reduced to the following rewriting schemas (where "." denotes the concatenation operation): for every category X and Y in C'

- **FA** (Forward Application) : $(X/Y).Y \rightarrow X$;
- **BA** (Backward Application) : $Y.(Y\backslash X) \rightarrow X$.

These schemas justify the fractional notations of the operators / and \. Categories of the form X/Y (resp. $Y\backslash X$) can be considered as *oriented functors* expecting an argument of category Y on their right (resp. on their left) and providing a result of category X. Given a vocabulary Σ, a Categorial Grammar G is defined by an axiomatic category S and an association between words in Σ and categories in C'. The language recognized by G is the set of finite concatenations of elements of Σ for which there exists an assignment of categories that can be *rewritten* with the schemas into S. The class of (string) languages that are recognizable by Categorial Grammars is the class of context-free languages.

Example 1 (A basic Categorial Grammar). Let us define a Categorial Grammar for the analysis of a small subset of natural language whose vocabulary is {a, man, woman, John, runs, walks, fast}. The set of basic categories is $C = \{S, T, CN\}$ where S is the axiomatic category of sentences, T stands for

[1] For a sake of clarity, parentheses are omitted in non ambiguous expressions

term and CN means *common noun*. The assignment of categories to words is:
John: T; man, woman: CN; runs, walks: $(T\backslash S)$;
a: $((S/(T\backslash S))/CN)$; fast: $((T\backslash S)\backslash(T\backslash S))$.

This grammar allows to recognize sentences like : *"a man runs"* and *"John runs fast"* because:

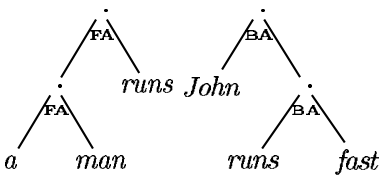
$$\begin{array}{llll} a & man & runs & \text{(twice with the } \mathbf{FA} \text{ rule)} \\ ((S/(T\backslash S))/CN) \;.\; \text{CN} & \;.\; (T\backslash S) & \rightarrow (S/(T\backslash S)) \;.\; (T\backslash S) & \rightarrow S \\ John & runs & fast & \text{(twice with the } \mathbf{BA} \text{ rule)} \\ T \;.\; (T\backslash S) & \;.\; ((T\backslash S)\backslash(T\backslash S)) & \rightarrow T \;.\; (T\backslash S) & \rightarrow S \end{array}$$

2.2 Grammatical Inference of Categorial Grammars

Learning a Categorial Grammar consists in identifying the categories assigned to each word of its vocabulary. The rewriting schemas are supposed to be known.

The formal learnability of Categorial Grammars has been studied in a variant of the PAC model ([Adr92]) and in Gold's model ([Kan96,Kan98]). The most powerful result obtained uses the notion of *Structural Example*. A Structural Example is derived from a syntactic analysis structure by deleting intermediate categories while preserving the terminal symbols and the reduction schemas used.

Example 2. The Structural Examples dervived from analyses of Example 1 are:



Kanazawa has proved that the class $\mathcal{G}_k$ of Categorial Grammars assigning at most k different categories with any given word is learnable in Gold's model from positive Structural Examples ([Kan96,Kan98]). Basically, the learning strategy, extending an algorithm earlier proposed by [BP90], relies on the unification of variable categories assigned to each nodes of the Structural Examples. When $k = 1$, the grammars are called "rigid" and they can be efficiently learned. When $k > 1$, the learnability is NP-hard ([Flo00]).

The main problem in this approach is that Structural Examples are hardly arguable as relevant data from a psycholinguistic point of view. The learnability result from Structural Examples can be extended to a learnability result from texts, but in this case the algorithm first needs to enumerate every possible tree structure corresponding with every string of words provided as example, and thus becomes exponential. Efficiency and cognitive plausibility don't seem to go together well.

Our purpose is also to learn Categorial Grammars, but from a new kind of input data, more informative than texts and more relevant than Structural Examples, i.e. *sequences of types*. Types will be associated with words and can be interpreted as semantic information.

3 Semantic information

3.1 A typing system

A well known interest of Categorial Grammars is their connection with semantics. This connection, first formalized by Montague ([Mon74,DWP81]) is inspired by the Principle of Compositionality ([Jan97]). One of its consequences is a strong correspondence between syntactic categories and semantic types. These types are what we propose to use in the learning process.

We will not define here a full semantic language, as types can be associated with very different kinds of semantic representation (the next subsection will provide clues to extract types from a usual semantic language). For us, the only semantic information needed will be a typing system making a basic distinction between *entities* and *facts* and allowing to express the types of *functors*, among which are the *predicates*. This typing system is a un-intensional version of Montague's intensional logic. The set Θ of possible types is defined by:

- elementary types : $e \in \Theta$ (type of entities) and $t \in \Theta$ (type of truth values) are the elementary types of Θ;
- Θ is the smallest set including every elementary type and satisfying : if $u \in \Theta$ and $v \in \Theta$ then $\langle u, v \rangle \in \Theta$ (the composed type $\langle u, v \rangle$ is the type of functors taking an argument of type u and providing a result of type v).

These types combine following rewriting rules similar with the ones of 2.1.

- **TF** (Type Forward) : $\langle u, v \rangle \ . \ u \rightarrow v$;
- **TB** (Type Backward) : $u \ . \ \langle u, v \rangle \rightarrow v$.

Example 3. For the grammar of Example 1, the types of the words are:
John: e; man, woman, runs, walks: $\langle e,t \rangle$; a: $\langle \langle e,t \rangle, \langle \langle e,t \rangle, t \rangle \rangle$; fast: $\langle \langle e,t \rangle, \langle e,t \rangle \rangle$.
These types express that "John" denotes an entity; "man", "woman", "runs" and "walks" characterize one-place predicates and "fast" is a one-place predicate modifier. The type of "a" comes from its semantic translation in Montague's system (see 3.2).

The main difference between categories in C' and types in Θ is that the *direction* of a functor-argument application in categories is indicated by the operator used (/ or \), whereas this direction is lost in types (as both are replaced by a ","). The functor-argument application of types is commutative, whereas it is not for categories. Note that for a given type containing n elementary types, there are 2^{n-1} possible ways to replace each of its $n-1$ "," by either / or \.

Another difference is that in our typing system, the set of elementary types is reduced to $\{e,t\}$, whereas it just needs to be finite in Categorial Grammars.

3.2 From logic to types

Before entering the learning process, let us slightly deepen the links between semantics and types. As a matter of fact it seems possible, under conditions, to

deduce the type corresponding with a word from its meaning representation in a Montague-like system.

The typing system we use is perfectly well adapted to the language of first order predicate logic extented with typed-lambda calculus (corresponding with un-intensional Montague's intensional logic).

Example 4. The semantic translation of the words of our little grammar in this logical language are the following:

- John : $John'$ (the prime symbol distinguishes logical constants from words)
- man : $\lambda x.man'_1(x)$; woman : $\lambda x.woman'_1(x)$
- runs : $\lambda x.run'_1(x)$; walks : $\lambda x.walk'_1(x)$
- a : $\lambda P_1.\lambda Q_1.\exists x[P_1(x) \wedge Q_1(x)]$
- fast : $\lambda P_1.\lambda x.[P_1(x)]$

The types associated with these words are easily deductible from their logical translation. An algorithm working in this case has been implemented.

Nevertheless, the reader should note that we need to impose some syntactic rules (not restrictive in the language of the logic) in order to derive types from logical formulas. For the language evoked, they include the following:

- atomic symbols noted x, y, etc. and isolated logical constants are of type e;
- symbols denoting predicates or functions must be used *in extension*, i.e. displaying their arity and all their arguments: for example, a two-place predicate P will appear as $\lambda x \lambda y.P_2(x)(y)$;
- formulas associated with words must be close, to avoid free variables

In fact, typed languages are usually defined with rules taking into account conditions on types. Types are thus deductible from formulas of these languages if the formulas themselves can be analyzed without ambiguity. The precise conditions allowing type deduction in the general case are being explored.

4 A new learning algorithm from types

We adopt here a *semantic-based theory of syntax learning*, i.e. we consider that the capacity of acquiring a grammar is conditioned by the ability to build a representation of the situation described. In previous semantic-based methods of learning ([HW75,And77,Lan82,Hil83,Fel98]), word meanings are supposed to be already known when the grammatical inference mechanism starts. We make here the smoother hypothesis that the crucial information to be extracted from the environment is the *semantic type of words*. Recognizing that a word represents an entity or a property satisfied by a (or several) entity(ies) is the prerequisite of our learning system. Section 3.2 suggests that types can be deduced from semantic representations, and are thus less informative.

4.1 Layout of the algorithm

The input of our learning algorithm is a sample of couples composed of a syntactically correct sentence and a corresponding sequence of types. The purpose is to build a rigid Categorial Grammar able to generate this sample. The missing information is the *direction* of the functor/argument applications, as each type of the form $\langle u, v\rangle$ can combine with a type u placed either on its left (with a **TB** rule), or on its right (with a **TF** rule). In the first case, the type $\langle u, v\rangle$ syntactically behaves like $u\backslash v$ with the rule **BA**, in the second one like v/u with the rule **FA** in the Categorial Grammar formalism. The identification of the operators / and \ will be processed in two steps: a parsing step and a step to get categories from types.

Parsing types We assume that every sentence given as input is syntactically correct and thus that the associated sequence of types can be reduced using the **TB** and **TF** rules to the type of truth values t. The first step of the algorithm is to find such a reduction.

Example 5.

a	*man*	*runs*
$\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$	$\langle e,t\rangle$	$\langle e,t\rangle$

Only **TF** can apply as a first reduction and combines the first two types into the type $\langle\langle e,t\rangle,t\rangle$. Again using **TF**, the final reduction leads to t. See Figure 1.

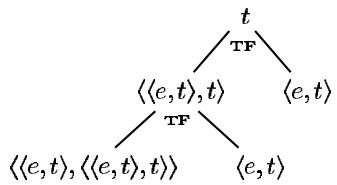


Fig. 1. A parse tree for types.

This reduction can be viewed as a parse in a context free grammar. In this setting, **TF** : $\langle u, v\rangle.u \to v$ and **TB** : $u.\langle u, v\rangle \to v$ are interpreted as schemes of rules where u and v are instantiated by types in the input sequence. As every instantiated rule fitting one of these schemes is in Chomsky Normal Form, it is possible to adapt a standard parsing algorithm to find the parse on types. In this paper, we modify the Cocke-Younger-Kasami (CYK) algorithm.

The whole process is the search for replacing the "," in type expressions to get categories. Possible values for "," are /, \ or "left unchanged" (a "," is left unchanged when the corresponding type is never in a functor place). To this aim, we identify every "," with a distinct variable. A reduction via **TF** or **TB** implies some equalities between categories and subcategories that must be propagated.

For instance, when one applies a **TF** : $\langle u, v\rangle.u \rightarrow v$ rule on *a man* associated with types $\langle\langle e\ \underline{x_1}\ t\rangle\ x_2\ \langle\langle e\ x_3\ t\rangle\ x_4\ t\rangle\rangle$ and $\langle e\ \underline{x_5}\ t\rangle$, the constraint $x_1 = x_5$ arises.

Hence, type reductions translate into variable equalities and a given parse leads to a system of equalities. We depict such equalities in Figure 2 by the underlined variables.

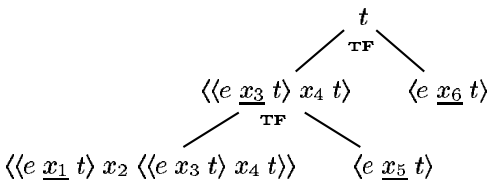


Fig. 2. Parsing types and introducing variable equalities. We underline variables that generate constraints to be fulfilled at each level of the parse tree.

Note that there could be more than one parse for a given input. The outcome of the parser is thus a set of parses each of which being described by a set of variable equalities. We will discuss complexity issues at the end of this section.

Getting categories Now given such contraints on types, we easily deduce categories and hence a Categorial Grammar that accepts the sequence of words as input. To this aim, we consider that **TB** and **TF** have **BA** and **FA** as a counterpart in the categorial grammar formalism. The association between types and categories is done in the following way:

- t is associated with S.
- any distinct type that never occurs at a functor place in the parse is associated with a new variable category.
- If **TF** : $\langle u\ x_i\ v\rangle.u \rightarrow v$ and u (resp. v) is associated with the category A (resp. B) then $x_i = /$ and $\langle u\ x_i\ v\rangle$ is associated with the category B/A.
- If **TB** : $u.\langle u\ x_i\ v\rangle \rightarrow v$ and u (resp. v) is associated with the category A (resp. B) then $x_i = \backslash$ and $\langle u\ x_i\ v\rangle$ is associated with the category $A\backslash B$.

Thus, while getting categories, we add equality constraints of the form $x_i = /$ or $x_i = \backslash$ for some i. The process is always guaranteed to give a set of categories because of the properties of the sequences in input.

Learning process An on-line and incremental learning algorithm can be designed using the strategy describe above. It consists in inferring equality constraints between variables and equality constraints fixing the value of variables from each input couple (in the example below, both are inferred at the same time). We also take into account that the target Categorial Grammar is rigid, that is every word is associated with at most one category.

Example 6. A first input is provided ...

a	*man*	*runs*
$\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$	$\langle e,t\rangle$	$\langle e,t\rangle$
$\langle\langle e\ x_1\ t\rangle\ x_2\ \langle\langle e\ x_3\ t\rangle\ x_4\ t\rangle\rangle$	$\langle e\ x_5\ t\rangle$	$\langle e\ x_6\ t\rangle$

leads to the system $\begin{array}{l} x_1 = x_5 \\ x_2 = / \\ x_3 = x_6 \\ x_4 = / \end{array}$.

At this point, the Categorial Grammar inferred consists in the associations: $a : (S/B)/A$; *man* : A; *runs* : B.

Consider a second input. Since every word has at most one category we consider the same variables in the type of words already encountered in a previous sentence. The set of variable equalities is enriched.

a	*woman*	*walks*
$\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$	$\langle e,t\rangle$	$\langle e,t\rangle$
$\langle\langle e\ x_1\ t\rangle\ x_2\ \langle\langle e\ x_3\ t\rangle\ x_4\ t\rangle\rangle$	$\langle e\ x_7\ t\rangle$	$\langle e\ x_8\ t\rangle$

$\begin{array}{l} x_1 = x_7 = x_5 \\ x_2 = / \\ x_3 = x_8 = x_6 \\ x_4 = / \end{array}$

Thus, *man* and *woman* (resp. *walks* and *runs*) are of the same category. Consider a third input. One can find that the set of variable equalities becomes:

a	*woman*	*walks*	*fast*
$\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$	$\langle e,t\rangle$	$\langle e,t\rangle$	$\langle\langle e,t\rangle,\langle e,t\rangle\rangle$
$\langle\langle e\ x_1\ t\rangle\ x_2\ \langle\langle e\ x_3\ t\rangle\ x_4\ t\rangle\rangle$	$\langle e\ x_7\ t\rangle$	$\langle e\ x_8\ t\rangle$	$\langle\langle e\ x_9\ t\rangle\ x_{10}\ \langle e\ x_{11}\ t\rangle\rangle$

$$\begin{array}{l} x_1 = x_7 = x_5 \\ x_2 = / \\ x_3 = x_8 = x_6 = x_{11} = x_9 \\ x_4 = / \\ x_{10} = \backslash \end{array}$$

Finally, if the last sentence is *"John walks"*, equality constraints propagate $x_8 = \backslash$ to several categories.

John	*walks*
e	$\langle e,t\rangle$
e	$\langle e\ x_8\ t\rangle$

$\begin{array}{l} x_1 = x_7 = x_5 \\ x_2 = / \\ x_3 = x_8 = x_6 = x_{11} = x_9 = \backslash \\ x_4 = / \\ x_{10} = \backslash \end{array}$.

To summerize, after renaming e in T and $\langle e\ x_1\ t\rangle$ in CN, the Categorial Grammar obtained consists in *John*: T; *man, woman*: CN; *runs, walks*: $T\backslash S$; a: $(S/(T\backslash S))/CN$; *fast*: $(T\backslash S)\backslash(S\backslash T)$. So, with this sample, we exactly obtain the grammar of Example 1. This grammar is able to recognize sentences like "John runs fast", not given as example. Some generalization has thus occurred.

In more complicated cases where more than one parse is possible on a given input, the algorithm must maintain a set of sets of constraints representing a set of candidate grammars.

Criticism A parse in a context free grammar in Chomsky Normal Form can be computed in polynomial time in the size of the input. This complexity result is stated for a given context free grammar. In our setting, we do not have a

context free grammar but a "set of possible ones" defined by schemes of rules. This changes a lot the complexity issues as illustrated by the following example:

Example 7. Let us consider the following input:

$a\ a \ldots a$	b	$a \ldots a$
$e\ e \ldots e$	$\langle e, \langle \ldots, \langle e, t \rangle \rangle \rangle$	$e \ldots e$

with n letters a on both sides of a b. It can be proved that there are $\binom{2n}{n}$ different Categorial Grammars recognizing this input. As a matter of fact, the type associated with b expects $2n$ arguments among which n are on its left and n are on its right. So, **TF** and **TB** must both be applied n times, no matter in which order. Thus, a parser that builds every possible parse runs in exponential time *relatively to the total number of "," in the input sequence of types.*

As an interpretation of this example, two interesting facts can be said. First, we wanted to provide more information as input than simple sentences in order to find linguistically relevant and computationally acceptable learning procedures. The process of finding structural data from sole string input data being too expensive we propose to add semantic information represented by types. It can be observed that even with the richer data of types provided as input, the problem is far from trivial in terms of time and space complexity.

Second, a naive approach consists in trying every possible assignment of a value in {"\", "/","−"} with every "," in the input sequence of types. For n "," in this sequence, there are 3^n different assignments. The worst case complexity of our approach is comparable with the complexity of this naive algorithm.

Nonetheless, the algorithm is not exponential in the number of words in the sentence given as input. The problem arises from the intrinsic number of possible solution grammars. But we are not interested in finding all acceptable Categorial Grammars (w.r.t. the input): only one is enough. Since the parser can work in polynomial time in the case where there is only one acceptable Categorial Grammar, there is still some hope to find relevant adaptations of our method. We will pursue these research directions in the near future.

4.2 Implementation details

We now describe a simplified scheme of our algorithm. The global structure is similar to CYK. We also follow as much as possible the notations in [HU79]. The algorithm builds a table. Rows and columns range from 1 to the length of the input. In the CYK algorithm, a cell (i, j) contains the non-terminals types that the subsequence of length j starting at position i reduces to. Here we add two modifications. First, there might be more than one parse. Therefore cells in the table are indexed by the starting position in the sequence, the length of the subsequence, the number of the parse. Second, we add some information in cells in order to propagate the equality constraints over variables and the partial parse of types called the *partial structure* (that is the parse of a subsequence of types). Hence, cells in the analysis table contains a (non-terminal) type, equality constraints over variables and a partial structure.

The input consists in sequences of words and types. Types are built with variables in place of ",".
The procedure `apply-tf(table[i][k][p],table[i+k][j-k][q],table[i][j])` tries to apply a **TF** rule to a subsequence starting at position i and of length j. The first cell corresponds to the *pth* parse of the subsequence of length k starting at position i and the second cell corresponds to the *qth* parse of the subsequence of length $j-k$ starting at position $i+k$. Basically, `apply-tf` checks that `table[i][k][p].Type` is of the form $\langle u_1\ x_i\ v\rangle$, `table[i+k][j-k][q].Type` is of the form u_2. Then, `apply-tf` searches for a substitution (a variable replacement) σ such that $\sigma(u_1) = \sigma(u_2)$. If it is possible, `apply-tf` adds a new cell `table[i][j][x]` in the list `table[i][j]` in the following way:

- `table[i][j][x].Type` is $\sigma(v)$;
- `table[i][j][x].Var` combines the substitution σ and the substitutions in `table[i+k][j-k][q].Var` and in `table[i][k][p].Var`;
- `table[i][j][x].Stru` is:
  TF(`table[i][k][p].Stru`, `table[i+k][j-k][q].Stru`).

The procedure `apply-tb` is of course similar to `apply-tf`. The main algorithm is given in algorithm 1.

5 Conclusion

Learning a Categorial Grammar means associating categories with words. Categories are built from basic categories and operators. Learning categories from strings of words seems impossible in reasonable time. So, richer input data need to be provided.

The approach developed so far by Buskowsky & Penn and Kanazawa consisted in providing Structural Examples, i.e. giving the nature of the operators / and \ and letting the basic categories to be learnt. Our approach is the exact dual, as it consists in providing the nature of the basic categories (under the form of types), but letting the operators to be learnt.

From a cognitive point of view, our choice seems more relevant, because the data we provide can be interpreted as coming from semantic information. If we admit the existence of a universal symbolic mental language ([Fod75]), types are related with this language and are thus language-independent. On the contrary, the operators used in categories are connected with *the word order* of a specific natural language and this linguistic parameter is not arguably innate.

The ability to identify types, i.e. for example to distinguish an entity from a predicate describing a property satisfied by entities can be compared with what psychologists call *categorization*. This very general ability does not anticipate on the way this semantic distinction will be grammaticallized in a particular natural language. For example, common nouns and intransitive verbs receive the

Algorithm 1 Parse algorithm

Input: $(w_1, \tau_1) \ldots (w_n, \tau_n)$ the sequence of words with the corresponding associated types built with variables $x_1, \ldots, x_p$.

```
 1: for i = 1 to n do
 2:   table[i][1][0].Type= τ_i; // the first partial type in cell ij
 3:   table[i][1][0].Var= ∅; // the first set of equality constraints in cell ij
 4:   table[i][1][0].Stru= w_i; // the first partial structure in cell ij
 5:   table[i][1].size-cell=1; // the number of elements in the cell ij
 6: end for
 7: for j = 2 to n do
 8:   for i = 1 to n − j + 1 do
 9:     table[i][j].size-cell=0;
10:     for k = 1 to j − 1 do
11:       for p = 1 to table[i][k].size-cell do
12:         for q = 1 to table[i+k][j-k].size-cell do
13:           apply-tf(table[i][k][p], table[i+k][j-k][q],table[i][j])
14:           apply-tb(table[i][k][p], table[i+k][j-k][q],table[i][j])
15:         end for
16:       end for
17:     end for
18:   end for
19: end for
```

Output: The global derivation table calculated.

same semantic type corresponding with a one-place predicate. But, in English, intransitive verbs can combine with individual terms (proper names) to provide a sentence whereas common nouns never appear at a functor place. So, both kinds of words are syntactically distinguished by their different combinatorial properties and the initially identical semantic types finally correspond with different syntactic categories.

The algorithm we propose here is able to identify any rigid Categorial Grammar. It still needs to be extended to k-valued Categorial Grammars. In fact, two sources of polymorphism should be distinguished: the case where a word must be associated with different semantic types (for example words like "and"), and the case where a word must be associated with different syntactic categories corresponding with the same semantic type (which must be the case for words like "a"). The first case should be handled as a natural extension of our learning strategy, where each semantically distinct instance of the word is treated as a new word, but the second case should be harder to deal with. Another extension to be explored is the case when only some of the types associated with words are known (for example, those associated with lexical words), whereas some others (those associated with grammatical words) remain unknown.

Finally, our implemantation still needs to be applied on real corpuses to compare its performance with other implemented methods.

References

[Adr92] P. W. Adriaans. *Language Learning from a Categorial Perspective.* PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, 1992.

[And77] J. R. Anderson. Induction of augmented transition networks. *Cognitive Science*, 1:125–157, 1977.

[BP90] W. Buszkowski and G. Penn. Categorial grammars determined from linguistic data by unification. *Studia Logica*, 49:431–454, 1990.

[DWP81] D. R. Dowty, R. E. Wall, and S. Peters. *Introduction to Montague Semantics.* Linguistics and Philosophy. Reidel, 1981.

[Fel98] J. A. Feldman. Real language learning. In *ICGI'98, 4th International Colloquium in Grammatical Inference*, pages 114–125, 1998.

[Flo00] Costa Florncio. On the complexity of consistent identification of some classes of structure languages. In *ICGI'2000, 5th International Colloquium on Grammatical Inference*, volume 1891 of *Lecture Notes in Artificial Intelligence*, pages 89–102. Springer Verlag, 2000.

[Fod75] J. Fodor. *The Language of Thought.* Harvester Press, 1975.

[Gol67] E.M. Gold. Language identification in the limit. *Inform. Control*, 10:447–474, 1967.

[HGS60] Y. Bar Hillel, C. Gaifman, and E. Shamir. On categorial and phrase structure grammars. *Bulletin of the Research Council of Israel*, 9F, 1960.

[Hil83] J. C. Hill. A computational model of language acquisition in the two-year-old. *Cognition and Brain Theory*, 3(6):287–317, 1983.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[HW75] H. Hamburger and K. Wexler. A mathematical theory of learning transformational grammar. *Journal of Mathematical Psychology*, 12:137–177, 1975.

[Jan97] T. M. V. Janssen. Compositionality. In J. V. Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 417–473. MIT Press, 1997.

[Kan96] Makoto Kanazawa. Identification in the limit of categorial grammars. *Journal of Logic, Language, and Information*, 5(2):115–155, 1996.

[Kan98] M. Kanazawa. *Learnable Classes of Categorial Grammars.* The European Association for Logic, Language and Information. CLSI Publications, 1998.

[Lan82] P. Langley. Language acquisition through error discovery. *Cognition and Brain Theory*, 5:211–255, 1982.

[MO98] T. Briscoe M. Osborne. Learning stochastic categorial grammars. In *CoNLL97: Computational Natural Language Learning*, pages 80–87, 1998.

[Mon74] R. Montague. *Formal Philosophy; Selected papers of Richard Montague.* Yale University Press, 1974.

[OBW88] Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, editors. *Categorial Grammars and Natural Language Structures.* D. Reidel Publishing Company, Dordrecht, 1988.

[Tel99] I. Tellier. Towards a semantic-based theory of language learning. In *12th Amsterdam Colloquium*, pages 217–222, 1999.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[WC80] K. Wexler and P. Culicover. *Formal Principles of Language Acquisition.* MIT Press, 1980.

Interactive Background Knowledge Acquisition for Inducing Differences among Documents

Chieko Nakabasami

Toyo University, 1-1-1 Izumino Itakura Oura Gunma 374-0193, Japan
chiekon@itakura.toyo.ac.jp

Abstract. This paper presents a case study in which an Inductive Logic Programming (ILP) technique is applied to natural language processing. Aleph, an ILP system, is used to induce differences among documents. A Case-Based Reasoning (CBR) system is proposed for the purpose of compiling the background knowledge inputted into Aleph. In the CBR system, lexical and syntactic information concerning words in close proximity to the target word(s) in training sentences is provided in order to infer new cases effectively. The compiled background knowledge is used to determine the semantic differences in documents that are written in natural language. Tentative experiments with this technique have produced encouraging results.

1 Introduction

This paper describes a method to induce differences among natural language documents using Aleph[18], a superseded version of Progol [15]. The background knowledge inputted into Aleph is generated by means of Case-Based Reasoning (CBR) with user interaction if necessary. Sophisticated background knowledge should be provided to efficiently process natural language documents since they are rich in representation. The goal of this study is to synthesize Inductive Logic Programming (ILP) and Natural Language Processing (NLP). The acquisition of background knowledge is one of the problems to be overcome. On the other hand, the acquisition and compilation of background knowledge have been reported to cause a bottleneck in knowledge engineering because they are very time-consuming tasks. The automatic generation of background knowledge provides a solution to the problem.

In this study, CBR is applied to obtain such knowledge. Though background knowledge is constructed with a great deal of user interaction during the initial phase of CBR, the frequency of interaction decreases as the information is processed by CBR. The system processes a sentence in turn with human interaction, and novel representations acquired are stored into cases so that they may be applied to future processes. The acquisition technique locates words that are in close proximity to target words, and it functions by matching syntactic and lexical patterns. Users validate the selected cases by judging them correct or incorrect. The CBR system enables users to add new appropriate cases and delete inappropriate ones.

The knowledge obtained includes discriminating features that have been extracted from the target documents and incorporated into the system. The knowledge that people want to acquire is not general knowledge concerning similar documents; instead, it is specific and distinctive knowledge contained only in the target document. In this paper, scientific manuscripts dealing with material design serve as the domain. The focus is on material design because collaborative engineering studies on the application of artificial intelligence have been conducted by metallurgy experts[16], and plenty of on-line manuscripts in this discipline are available.

A concise text-classification method is not required for the target domain, but one that detects minor differences among documents should be explored. Researchers in this field want to identify the differences in experimental procedures explained in the target paper and compare them to those in similar papers because slight differences in experimental procedures affect the properties of materials. Relational representations are needed to find these differences since the main issue is not to acquire the meaning of each word but the structures composed from words appearing in the documents. ILP is thought to be a better method because it can construct a relationship among words in a document. Aleph, which contains the assembled background knowledge, introduces sophisticated differences as the manuscripts are compared. Aleph produces a representative body of knowledge contained in the target manuscripts that is missing in the other manuscripts.

The remainder of this paper is organized as follows. Section 2 defines knowledge representation of the background knowledge. Section 3 explains the CBR system for generating the background knowledge. Section 4 illustrates the mechanism by which Aleph induces the intended results. Section 5 shows the tentative results produced by Aleph. Section 6 mentions related works for applying ILP to NLP. Section 7 is the conclusion.

2 Forms of Background Knowledge

2.1 Predicate Definitions

The background knowledge applied to Aleph has the form of horn clauses. The predicate names and their arguments are defined in advance so that Aleph may efficiently construct understandable results. The predicates depend on the experimental papers on the subject of material design. They are as follows:

- result(*Paper*): The result of *Paper*
- has_focus(*Paper, Focus*): The *Paper* has the *Focus*.
- focus (*Focus, Content*): The content of the *Focus* is *Content*.
- action (*Paper, Action, Target*): The *Paper* describes the *Action* that is conducted for the *Target*.
- change (*Paper, Result, Target*): The *Paper* describes the *Result* that is achieved for the *Target*.

- purpose (*Paper, Purpose, Target*): The *Paper* describes the *Purpose* of processing the *Target*.
- show (*Paper, Result, Target*): The *Paper* describes the *Result* of processing the *Target*.
- detail (*Paper, Detail, Target*): The *Paper* describes the *Detail* of the *Target*.
- condition (*Paper, Proposition, Target*): The *Paper* describes under what condition the *Target* is processed. The *Proposition* represents a proposition used in the text.

2.2 Knowledge Representation of Sentences in Documents

The background knowledge for Aleph has been extracted from the sentences in the target documents according to the predicate definitions presented in section 2.1. This information is developed by using the CBR process. For example, the following representation (3) is constructed from sentence (1) via (2). The words in the sentences are tagged with a part-of-speech[3] such as (2).

```
(1) Feedstock material is melted in a high vacuum chamber
    by electron beam guns into water.
(2) [feedstock/nn,material/nn],is/vbz,melted/vbn,
    in/in,[a/dt,high/jj,vacuum/nn,chamber/nn],
    by/in,[electron/nn,beam/nn,guns/nns],
    into/in,[water/nn].
(3) action([melt],[feedstock, material]).
    condition([in],[high,vacuum,chamber]).
    condition([by],[electron,beam,guns]).
    condition([into],[water]).
```

In (2), the determiners are removed, and "nn," "vbz," "vbn," "dt," "jj," "in," and "nns" represent singular noun, singular present verb, past participle, determiner, adjective, proposition, and plural noun, respectively. The representations in (3) correspond to the predicates in section 2.1. The transformation presented above is explained in detail in section 3.

3 CBR Process for Construction of Background Knowledge

Case-Based Reasoning (CBR) is applied to obtain the background knowledge concerning the contents of the documents. In the CBR system, each case stored is represented with a *pred/3* predicate and composed of three arguments. These three have a list form of the predicate introduced in section 2.1, which deals with lexical and syntactic information. For instance, sentence (1) shown in section 2.2 is represented by a form such as (4), if it is stored as one of the training cases.

```
(4) pred([action,[melt],[feedstock,material]],[],[is/vbz,melted/vbn]).
```

```
pred([condition,[in],[high,vacuum,chamber]],[in/in],[]).
pred([condition,[by],[electron,beam,guns]],[by/in],[]).
pred([condition,[into],[water]],[into/in],[]).
```

The first argument of *pred/3* corresponds to one of the predicates introduced in section 2.1. As a second argument of *pred/3*, the lexical and syntactic information that appears in a previous position in a sentence is based on the third element of the first argument of *pred/3* (e.g., [feedstock, material]) and as the third element in the postposition. Hereafter, the term used for the third element of the first argument of *pred/3* is "pivot words." Empty lists are assigned when there is not a pair of keywords.

The CBR system locates the words nearest the target by matching syntactic and lexical patterns. The criterion for finding neighboring words is whether a word (or phrase) in novel sentences includes the word(s) in the pivot words and whether the part-of-speech of the word(s) in front of and behind the pivot words matches the part-of-speech in the original cases. For example, sentence (5) is translated into (7) because the word "material" in (5) is included among the pivot words of the first *pred/3* in (4). In addition to word matching and from the fact that the part-of-speech of the words behind "material" matches those of (4) (i.e., "vbz," "vbn"), it can be assumed that one of the neighboring cases of sentence (5) is (1). Then (7) is constructed from (5) via (6).

```
(5) The material is sampled for that experiment.
(6) [the/dt,material/nn],is/vbz,sampled/vbn,for/in,
    [that/dt,experiment/nn].
(7) action([sample],[material]).
```

The CBR system enables users to add new appropriate cases and delete inappropriate ones. The schema of the CBR system is sketched in Fig. 1.

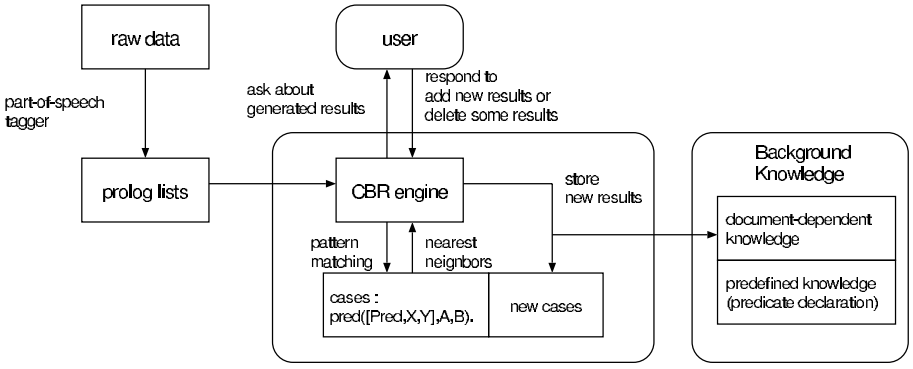


Fig. 1. Schema of the system

In the CBR system, about 120 cases are stored manually as initial cases. These cases are randomly extracted from among 630 manuscripts on the subject of the material design of superalloys. The system processes a sentence as humans interact, and the acquired novel representations are stored into cases so that they may be applied to future processes. The lexical and syntactic information around the pivot words is assigned automatically when the new case is stored. Though background knowledge is constructed with a great deal of user interaction during the initial phase of CBR, the frequency of interaction decreases as the information is processed by CBR.

4 Mechanism for Inducing Differences among Documents

The following data are prepared for Aleph:

- Positive example: A target paper (or papers) from which to extract information;
- Negative example: Other papers that have some content in common and some that differs from the target paper; and
- Background knowledge: Descriptions in terms of content for each example.

The three kinds of data are inputted into Aleph, and descriptions of the content of the target paper that is absent in the other related papers are developed. At first, researchers pick up the paper that they want to process and also choose other similar papers (e.g., the title words might include the same material name or type of mechanical testing). Each paper is given a unique identification (ID). Next, these papers are inputted into the CBR system one by one, and the system outputs the background knowledge used in Aleph. After that process, Aleph induces the description that differentiates the target papers from others. For example, the following files are prepared for Aleph. These file stems are identified as "alloy" because of the material design domain.

- alloy.f: Positive examples are written.

  ```
  result(p01).  result(p02).
  ```

- alloy.n: Negative examples are written.

  ```
  result(p03).
  ```

- alloy.b: The knowledge representation of the papers in both positive and negative examples, which has been constructed by the CBR system, is written.

In the example above, Aleph attempts to induce the content present in paper "p01" and "p02" but absent in paper "p03." The body of result/1 is hypothesized by means of the ILP technique. The hypothesis represents the differences between the positive and negative example papers.

5 Results

Tentative results are obtained using sample papers. Experts chose the papers carefully to ensure that the experimental methodology was similar. Three sets, each consisting of three papers, were chosen. The sections on the experimental methods in three papers are selected as sample documents. These documents are divided into positive and negative examples. Two documents are considered positive and one negative. The positive and negative documents are as follows:

Positive

```
Experimental (paper id: p01)
Five new alloy compositions were selected to analyze the potential
mechanical property improvements obtained when multiphase dispersions
of fine gamma, carbide, and oxide particles are combined in typical,
fully alloyed nickel-based superalloy compositions.

Experimental alloys and procedures (paper id: p02)
Alloy compositions, listed in Table 1, were selected to investigate
a wide range of refractory alloying additions and to permit statistical
analysis of results.
```

Negative

```
Experimental methods (paper id: p03)
GTD-111 specimens were machined into cylindrical shapes
with four different diameters of 10, 15, 25, and 35 mm
and a length of 30 mm.
```

Using these samples, Aleph induced the following descriptions. The "best clause" outputted by Aleph is used as a result clause, and this clause is in turn unified with facts in the background knowledge. As a result, facts that can be unified are obtained and embedded into sentences that are understandable to users. In the output parts, the result clauses are shown followed by the learning time.

Output

```
result(A) :- has_focus(A,B),purpose(B,C,D). (0.08 seconds)

The difference compared to other papers:

paper p02 describes in terms of [experimental,alloy]
The purpose of the [experimental,alloy] is to [investigate]
the [refractory,alloying,addition]

The difference compared to other papers:
```

```
paper p02 describes in terms of [experimental,alloy]
The purpose of the [experimental,alloy] is to [permit]
the [statistical,analysis]

The difference compared to other papers:

paper p01 describes in terms of [experimental]
The purpose of the [experimental] is to [analyze]
the [potential,mechanical,property,improvement]
```

Positive

```
Experiment (paper id: p04)
The crystallographic mode of propagation indicates
enhanced cracking resistance of the grain boundaries.
While this mode of propagation was observed in all
specimens tested at 1400 F, the amount of crystallographic
cracking varied. This transgranular fracture mode
at 1400 F appears to be a characteristic of the hafnium effect.

Experimental procedures (paper id: p05)
With the addition of hafnium to D.S. MAR-M200
in 1969, transverse grain boundary strength and
ductility were significantly improved.
Work on single crystals was discontinued at this time
because they offered no significant improvements
in properties over D.S. and they were more expensive.
```

Negative

```
Experimental methods (paper id: p06)
SC blades were used to inhibit grain nucleation on mold walls.
No significant investment mold-superalloy reactions
were observed as a result of the higher superheats
required for SC castings.
```

Output

```
result(A) :- has_focus(A,B),show(B,C,D). (0.19 seconds)

The difference compared to other papers:

paper p05 describes in terms of [single,crystal]
```

The observation of the [single,crystal] is to [improve]
the [mar,m200]

The difference compared to other papers:

paper p05 describes in terms of [single,crystal]
The observation of the [single,crystal] is to [improve]
the [ductility]

The difference compared to other papers:

paper p05 describes in terms of [single,crystal]
The observation of the [single,crystal] is to [offer]
the [no,significant,improvement]

The difference compared to other papers :

paper p04 describes in terms of [hafnium,addition]
The observation of the [hafnium,addition] is to [indicate]
the [enhanced,cracking,resistance]

The difference compared to other papers :

paper p04 describes in terms of [hafnium,addition]

The observation of the [hafnium,addition] is to [observe]
the [propagation]

Positive

Fatigue at high Delta-K at 700 C (paper id: p07)
Particles were observed on fracture surfaces and
there was evidence that intergranular crack growth
had involved fine-scale cavitation.
Striations with spacings close to the macroscopic
growth rate were present on transgranular areas.

Experimental procedure (paper id: p08)
Fatigue-crack-growth data were obtained by testing
compact-tension (CT) specimens cut from the forging
blanks such that specimens were equidistant from
the centre of the forging and the direction of
crack growth was radial.

Negative

```
Materials and experimental approach (paper id: p09)
The crack length was monitored during the precrack period
using the plastic replica technique. Several of the cracks
were naturally initiated at elevated temperature under the
actual strain controlled test conditions to better simulate
the actual crack growth process.
```

Output

```
result(A) :- has_focus(A,B),detail(B,C,D). (0.15 seconds)

The difference compared to other papers :

paper p07 describes in terms of [fatigue]
The detail of the [fatigue] is [high,delta-k] of the [700,deg,c]

The difference compared to other papers :

paper p07 describes in terms of [growth]
The detail of the [growth] is [striation] of the [spacing]
```

Through these experiments, 220 facts have been introduced by the system and 86 facts by a user.

6 Related Work

There are some studies although the search was not exhaustive, apply Progol to Natural Language Processing[5][6][10]. In these cited studies, Progol is used to develop rules for making words less ambiguous by tagging them according to the part of speech they represent. As in the other papers, this study uses context information found in close proximity to the target word, which we call "pivot words." In addition, ILP methods are applied to NLP applications such as conceptual clustering[11], transfer-rule learning[2], data mining[8][17], learning morphology[9][13][12], and learning phonetic rules[1]. A note by [7] mentions the European works on ILP. It has been proposed that ILP could be applied to semantic parsing as well as to the analysis of the syntactic aspects of language[19]. In [19], syntactic and semantic parsers are learned so that a natural language database query can be mapped directly into an executable Prolog query that will answer the question. Promising advantages are being claimed from the synthesis of ILP and NLP because of the knowledge that can be gained[14].

On the other hand, as for applying CBR method to acquire knowledge representation, [4] proposes a system, which is used to extract structural knowledge in terms of a specific domain. While the purpose of [4] is to assign a correct conceptual role to each word by making it less syntactically and semantically ambiguous, a goal of this study is to extract discriminating features from target documents by acquiring background knowledge from them for an ILP system such as Aleph.

7 Conclusions and Further Work

In this paper, a case study for inducing differences among natural language documents was reported. Aleph was used to develop explanations for the semantic differences that exist in a variety of documents. The background knowledge for Aleph is constructed using the CBR system. In a CBR system, lexical and syntactic information is examined to develop information that can help interpret novel sentences. The experiments that make use of cumulative background knowledge show promising results.

The number of interactions with users has not been mentioned at this point in the system's development because the system has only been evaluated with a small number of sentences. The decreasing rate of user interactions should be shown by graphs. In addition, some evaluation methods should be applied to the system though justifications of whether the results are correct or not depend on a subjective point of view from each domain expert.

In addition, how semantic representations are produced and the structure of background knowledge are subjects for future research. A sound synthesis of ILP and NLP should be carefully considered, and intelligent NLP applications should be developed on the basis of such a synthesis.

References

1. Alexin, Z., Csirik, J., Jelasity, M., Tóth, L.: Learning Phonetic Rules in a Speech Recognition System. Proc. of 7th International Conference on Inductive Logic Programming (ILP'97). (1997) 37–44
2. Boström, H.: Induction of Recursive Transfer Rules. Proc. of Learning Language in Logic (LLL) Workshop. (1999)
3. Brill, E.: A simple rule-based part of speech tagger. Proc. of the 3rd Conference on Applied Natural Language Processing (ACL'92). (1992)
4. Cardie, C.: A Case-Based Approarch to Knowledge Aquisition for Domain-Specific Sentence Analysis. Proc. of the 11th National Conference on Artificial Intelligence. (1993) 798–803
5. Cussens, J.: Part-of-Speech Tagging Using Progol. Proc. of 7th International Conference on Inductive Logic Programming (ILP'97). (1997) 93–108
6. Cussens, J.: Using Inductive Logic Programming for Natural Language Processing. Workshop Notes on Empirical Learning of Natural Language Tasks (ECML'97). (1997) 25–34

7. Cussens, J.: Notes on Inductive Logic Programming Methods in Natural Language Processing (European Work). Manuscript. (1998).
8. Dehaspe, L., Raedt, L. D.: Mining Association Rules in Multiple Relations. Proc. of 7th International Conference on Inductive Logic Programming (ILP'97). (1997) 125–132
9. Džeroski, S., Erjavec, T.: Induction of Slovene Nominal Paradigms. Proc. of 7th International Conference on Inductive Logic Programming (ILP'97). (1997) 141–148
10. Eineborg, M., Lindberg, N.: Induction of Constraint Grammar-Rules Using Progol. Proc. of 8th International Conference on Inductive Logic Programming (ILP'98). (1998) 116–124
11. Faure, D., Nédellec, C.: Aquisition of Semantic Knowledge using Machine Learning Methods: The System "ASIUM". Technical Report ICS-TR-88-16, LRI University. (1998)
12. Kazakov, D., Manandhar, S.: A Hybrid Approarch to Word Segmentation. Proc. of 8th International Conference on Inductive Logic Programming (ILP'98). (1998)
13. Manandhar, S., Džeroski, S., Erjavec, T.: Learning Multilingual Morphology with CLOG. Proc. of 8th International Conference on Inductive Logic Programming (ILP'98). (1998) 135–144
14. Mooney, R. J.: Inductive Logic Programming for Natural Language Processing. Proc. of 6th International Conference on Inductive Logic Programming (ILP'96). (1996) 3–22
15. Muggleton, S.: Inverse entailment and Progol. New Generation Computing Journal,13. (1992) 245–286
16. Nakabasami, C., Hoshimoto, K.: Extracting Knowledge from Technical Papers in Metallurgy on the basis of a Generative Lexicon. Proc. of the Conference Pacific Association for Computational Linguistics. (1999) 137–142
17. Raedt, L. D.: A Logical Database Mining Query Language. Proc. of 10th International Conference on Inductive Logic Programming (ILP2000). (2000) 78–92
18. Srinivasan, A. and Camacho, R.: The Aleph Manual. (1993) http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/
19. Zelle, J. M., Mooney, R. J.: Learning Semantic Grammars with Constructive Inductive Logic Programming. Proc. of the 11th National Conference on Artificial Intelligence. (1993) 817–822.

Part-of-Speech Tagging by Means of Shallow Parsing, ILP and Active Learning

Miloslav Nepil, Luboš Popelínský and Eva Žáčková

Faculty of Informatics, Masaryk University
Botanická 68a, CZ-60200 Brno, Czech Republic
Email: {nepil,popel,glum}@fi.muni.cz

Abstract. In this paper we describe a part-of-speech tagger for Czech which exploits DIS shallow parser for Czech, manually-coded rules and inductive logic programming. The active learning method used resulted in the decrease in the number of training examples to label as well as in a shorter learning time without the decrease in recall or accuracy. Compared with the previous work [10], both recall and accuracy increased and the number of training examples to label decreased. The method was tested on ambiguities that are frequent in Czech. The accuracy reached was higher than 96% with recall higher than 95%.

1 Introduction

Morphological tagging in inflectional languages like Czech is quite a difficult problem that can be hardly solved without employing (semi-)automatic taggers. Several morphological taggers have been developed or proposed for Czech, based

Fig. 1. Example of ambiguous words
(oni) <vlastní> auto. *(They <own> a car.)*
Zničili jejich <vlastní> auto. *(They destroyed their <own> car.)*

on either statistical methods [5, 11] or machine learning techniques [12, 13]. A tagger that combines manualy coded rules with HMM tagger is described in [6]. Here we focus on part-of-speech (POS) tagging. In the two sentences in Fig. 1, the word form *own* can be either an adjective or a verb. Our goal is to find the correct POS for a given word form if we know the words in its context.

To build a tagger one usually needs a representative set of texts. The principal difficulty for Czech lies in the fact that annotated corpora (i.e. unambiguously tagged ones) are too small – compare 164 000 stems of Czech words that a morphological analyser of Czech is able to recognize (each of them can have a number of both prefixes and suffixes) with 132 000 different word forms in DESAM corpus [11]. On the other hand, it is easy to collect unannotated data. E.g. Czech National Corpus [2] contains over 140 million words. If we know how to choose the most informative examples from these data sources, annotation cost will be lower.

Active learning [3] is defined as any form of learning in which the learning program has some control over the choice of examples on which it is trained. Its usefulness has been already shown in many NLP tasks including text categorisation [8], information extraction [14], semantic parsing [14] and part-of-speech tagging [4].

In the previous work [10], we have described POS tagger for Czech that combined manually written rules with ILP and active learning. We showed that this method surpassed passive learning in much shorter learning time, in smaller number of training examples to label and in compactness of the resulting tagger (smaller number of rules). This improvement was reached without any decrease of recall or accuracy. However, the tagger was not usable for real-world tasks because both recall and accuracy were not high enough.

Here we describe a new POS tagger that employs DIS shallow parser [16]. DIS is able to, after finding a chunk (noun or verb phrase), partially disambiguate words in the chunk. Even though the recall of DIS is quite low, its usage in combination with ILP results in significant improvement of recall and accuracy. We tested our tagger on two subtasks of POS tagging, substantive-adjective ambiguity and pronoun-verb ambiguity.

The paper is organised as follows. In Section 2 we describe the method that is used in the POS tagger. We briefly describe DIS shallow parser, the active learning method used and the structure of disambiguation rules. Section 4 displays results reached with the tagger. We conclude with Section 6.

2 Description of the Method

Our method combines three different techniques: DIS shallow parser, manually-crafted rules, and rules induced automatically by means of active learning with ILP system Aleph. In operating mode, when tagging an unseen text, these three components are applied in sequence to determine the right tag of each word.

2.1 DIS – Shallow Parser for Czech

The purpose of traditional syntactic parsers is to recover complete and exact parses. Such parsers usually assume that the grammar is complete and the globally best parse can be found in the entire space of possible parses. As a result they do not work well for noisy data. On the contrary, the purpose of shallow parsers is only to recognize the nonrecursive kernels of essential phrases, so called chunks.

The core of DIS, the shallow parser for Czech, consists of 131 DCG rules which describe the most important chunks – noun, pronoun, prepositional and verb phrases. We prefered high accuracy to recall. This means that only those rules which displayed very high accuracy (preferably 100%) were incorporated into the DIS grammar. While a good algorithmic description of noun, pronoun and prepositional groups can be found, finding description of (compound) verb groups is

much more difficult. Thus, we have used DESAM corpus [11] as the source of learning data. The algorithm for learning verb rules [17,18] takes, as its input, annotated sentences from DESAM and works in three steps: finding verb chunks, generalisation of them and verb rule synthesis. Partial analysis is controlled by Prolog predicates which process the DCG rules. They also ensure correct analysis of discontinuous constituents of verb groups which is a significant problem of parsing of free word order languages like Czech.

An input of DIS takes some text ambiguously tagged by `ajka` morphological analyzer [15] and preprocessed by script which performs statical disambiguation (elimination of very rare tags). According to the partial analysis, some tags of words can be eliminated and thus we can obtain partially disambiguated output.

The precision of disambiguation carried out by DIS is quite high – about 99.6%. It is influenced by the manner the grammar rules were selected – it was supposed to be better not to recognize the group at all than to recognize it wrongly. On the other hand, it implies a lower recall that ranges from 60% to 68% .

2.2 Manually-Crafted Rules

As some ambiguities, remaining after the application of DIS, are easy to remove, we have written simple disambiguation rules for them. Only very precise rules which displayed 100% accuracy on the learning data were accepted. This correlates with our effort for achieving a high accuracy – it is better to leave some ambiguities unresolved than to assign a wrong tag to some words. The rules are designed to remove a certain tag, when the context of a given word satisfies a set of conditions. The general structure of disambiguation rules is:

```
remove(Left, Word, Right, Tag) :- <set of conditions>
```

where the variable `Word` contains a word form to be disambiguated, together with all its tags which remain possible, the variable `Tag` determines a corresponding tag which should be removed, and the variables `Left` and `Right` represent ambiguously tagged left and right contexts, respectively. Thus, if the set of conditions on the right hand side holds, the `Tag` will be removed from the set of possible tags for the given `Word`. Usually, the conditions have the form `cn(Ctx,Scope,Cond)` where `Ctx` represents a left or right context and `Cond` is a first-order condition testing the context `Ctx` narrowed by the term `Scope`. For example, the `Cond` can be a test concerning the presence of certain words, lemmata and/or tags within the given narrowed context. A shorter form of conditions is `cn(Pos,Cond)` where the variable `Pos` directly refers to the questioned position. An example of a rule is shown in Fig. 2. In this example, tag `k1` will be removed

Fig. 2. Example of a rule

```
remove(L,W,R,k1) :- cn(W, e(k1) & e(k2&g:G&n:N&c:C)),
                    cn(R, first(1), [e(k1&g:G&n:N&c:C)]).
```

from the set of possible tags for the word `W`, if `W` can be a substantive (`k1`), but

also an adjective (k2) with the same gender G, the number N and the case C as the neighbouring substantive on the right.

2.3 Active Learning with Aleph

To lower both the learning time and the annotation cost (e.g. the number of training examples to label), we employed active learning strategy. We used a committee of relational learners and combined manually-crafted rules with those induced automatically by the inductive logic programming (ILP) technique [9]. We investigated a variant of committee-based sampling [4]. An unlabelled example was presented to $C \geq 2$ classifiers, members of the committee. Each member of the committee is represented by a set of those rules which remove the same particular tag. E.g., when resolving noun/adjective k1/k2 ambiguity, we used a committee of two members: the first member was a set of rules for removing tag k1 and the second was a set of rules removing tag k2. When the members disagreed in label prediction (both tags were to be removed, or none of them was), the example was assumed to be informative and the actual label was requested from the teacher.

The learning set was randomly split into N samples of approximately the same size. At each learning step, the rules learned from the previous sample were used for disambiguation of the next block. In the case that an ambiguity remained, a human was asked for a correct label.

These labelled cases were used for learning with the ILP system *Aleph*[1]. For each kind of ambiguity, two sets of rules were learned by Aleph with default settings. The rules induced by Aleph have the same form as the manually-crafted ones. E.g., for k1/k2 ambiguity we first learn rules for removing k1 tag and then rules for removing k2 tag. Positive examples for removing, e.g., the tag k1, were those cases which had been labelled with a tag different from k1. On the contrary, negative examples for removing the tag k1 were the cases which had been unambiguously tagged as k1. Whereas the positive examples were generated only from the ambiguities labelled by the human, the negative ones were generated from the whole sample. The reason for generating so many negative examples was that we prefered correctness to completeness which, in other words, means precision to recall. Only DIS together with the manually-crafted rules were employed to label the first sample.

Some of the learned set of rules were overgeneral, i.e. some tags were incorrectly removed. To prevent this, we used a simple specialisation operator. The rules that covered more than 5% negative examples on the next training block (from all the examples covered by the rule) were removed. Then, the remaining newly induced rules were added to the old ones and they together were used for disambiguating the next sample.

[1] http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html

3 Data Source

We used a randomly chosen subset of Prague Dependency Treebank (PDT) corpus [5] that contained 41647 items (word positions) ambiguously annotated with `ajka` morphological analyser [15]. This means that each word was labeled with all possible tags for given word. We used a full tag set for Czech that contains about 1600 different tags. Approximately 52% of words had more than one tag, 14.9% of words contained at least two part-of-speech tags (different word category). The data set was split into 5 samples of approximately equal size. Four of them were used for learning, the fifth one for testing.

4 Results

We tested our method on two of the most frequent part-of-speech ambiguities in Czech, substantive-adjective ambiguity (15.8% of all ambiguities) and pronoun-verb ambiguities (8.2%). Results for these ambiguities are displayed in Table 1 and Table 2. The second, the third and the fourth columns contain a number of ambiguities at the beginning, remaining after application of DIS, and remaining after application of rules. Next two columns inform about the relative number of cases for which just one of two tags was removed (RECALL) and about ACCURACY, the relative number of correctly removed tags from all the tags removed. The first line shows the values obtained by using the manually-coded rules.

Table 1. Results for substantive-adjective ambiguity

	#ambiguities						# newly learned	Set of
Sample	before	DIS	rules	RECALL	#err.	ACCURACY	rules	rules
0.	182	65	63	65.4%	0	100.0%	6	pl1
1.	216	63	17	80.4%	2	99.0%	6	pl2
2.	257	92	47	81.7%	1	99.5%	3	pl3
3.	174	40	4	97.7%	1	99.4%	2	pl4
4.	160	52	0	100.0%	2	98.8%	-	-

Out of 182 ambiguous words 119 cases were correctly disambiguated and 63 ambiguities remained (recall 54.9%). The remaining 63 cases were labeled and used for learning two sets of rules with Aleph: one for removing tag `k1` (substantive) and one for removing the other tag `k2` (adjective). New set of rules, named pl1, contained the set of 5 manually-coded rules plus 6 newly learned rules. This set of rules was used for disambiguation of Sample 1. By the same process the set of rules pl2 was obtained. These rules were then used for disambiguation of Sample 2. etc. In the end, set pl4 contained 22 rules: 5 of them written manually and 17

learned with Aleph.

For the pronoun-verb ambiguity, six disambiguation rules were written manually. These rules display an accuracy of 100% again. The results produced by the tagger for this ambiguity are displayed in Table 2.

Table 2. Results for pronoun-verb ambiguity

Sample	#ambiguities before	DIS	rules	RECALL	#err.	ACCURACY	# newly learned rules	Set of rules
0.	93	83	36	61.3%	0	100.0%	8	pl1
1.	102	86	20	80.4%	1	98.8%	4	pl2
2.	91	74	8	91.2%	0	100.0%	2	pl3
3.	83	64	7	91.6%	3	96.1%	2	pl4
4.	91	76	2	97.8%	3	96.6%	-	-

5 Discussion

A comparison of the results between active and passive learning is shown in Tables 3 and 4. Passive learning was performed by the following way. First, DIS followed by manually-crafted rules were used to remove ambiguities in all samples 0,1,2,3,4. The remaining ambiguous words were labeled and all cases from samples 0,1,2,3 were used for learning rules with Aleph. The rules learned were tested on Sample 4. For the noun-adjective ambiguity, the active learning

Table 3. Substantive-adjective ambiguity: passive and active learning

	#examples to label	#rules learned	RECALL	ACCURACY
passive	250	21	95.0%	100.0%
active	131	17	100.0%	98.8%

method needed much less examples to label than passive learning – compare 131 to 258 examples. The number of induced rules was also smaller – 21 for passive learning, 17 for the active one. The training time of passive learning was 1 h 17 min which is almost 6-times longer than that for active learning. Recall of active learning is about 5% higher than that for passive learning, with very small decrease in accuracy – from 100.0% to 98.8%. This small decrease of accuracy may be explained by the fact that in passive learning Aleph knew all negative

examples from all samples. In active learning, negative examples were generated only from one sample.

For the pronoun-verb ambiguity, the recall and accuracy were very high again. The recall on the test sample reached 97.8% and it achieved the accuracy of

Table 4. Pronoun-verb ambiguity: passive and active learning

	#examples to label	#rules learned	RECALL	ACCURACY
passive	307	12	94.5%	97.7%
active	71	16	97.8%	96.6%

96.6%. A comparison of the results between active and passive learning is shown in Table 4. The number of labelled examples in active learning was more than 4-times smaller than that in passive learning and the learning time was again much smaller. The recall increased from 94.5 to 97.5% with a small decrease of accuracy. This may be explained the same way as in the previous experiment.

Although we haven't carried any detailed investigation concerning the linguistical plausibility of our automatically induced rules, we can say that some of them really captured interesting laws. However, most of the learned rules were quite overgeneral in the sense that it will not be difficult for a human to produce a counterexample. In spite of this fact, the accuracy of these rules on real language data was not so bad. Moreover, our language of disambiguation rules is fairly comprehensible, therefore, many learned rules could be easy refined by a human to improve their accuracy.

Another approach to iterative POS is described in [7]. A combination of both methods could be promising.

6 Conclusion

The goal of this work was to test usability of active learning in POS tagging in Czech. The reached results are promising. A number of examples to label is much smaller than in the case of passive learning. The enormous decrease of time is also important. When compared with the previous work [10], both recall and accuracy increased and the number of labelled examples even decreased.

From the point of view of POS tagging, the results are pretty preliminary. Namely, comparison with results reached with HMM taggers [6] should be performed. However, the high values of recall and accuracy achieved by the discussed technique allow one to use this method for some subtasks of POS tagging of Czech corpora.

Acknowledgement

We thank the referees for their comments and Sylvia Wong and Karel Pala for their assistance. This work was partially supported by ILPnet2 project.

References

1. Angluin D.: Queries and Concept Learning. *Machine Learning* 2, 4, April 1988, 319-342
2. Čermák F.: Czech National Corpus: A Case in Many Contexts. *Int. J. of Corpus Linguistics*, 2,2, 181-197, 1997.
3. Cohn D., Atlas L., Ladner R.: Improving Generalization with Active Learning. *Machine Learning*, 15, 201–221, 1994.
4. Dagan I., Engelson S.: Selective Sampling in Natural Language Learning. In *IJCAI-95 Workshop On New Approaches to Learning for Natural Language Processing*, 1995.
5. Hajič J., Hladká B.: Tagging Inflective Languages: Prediction of Morphological Categories for a Rich, Structured Tagset. In *Proceedings of EACL 1998.*
6. Hajič J., Krbec P., Květoň P., Oliva K., Petkevič V.: Serial Combination of Rules and Statistics: A Case Study in Czech Tagging. In *Proceedings of ACL/EACL 2001, Toulouse*, pages 260–267, 2001.
7. Jorge A., Lopes A.: Iterative Part-of-Spech Tagging. *Cussens J., Džeroski S.: Learning Language in Logic. LNCS 1925*, pages 170–183. Springer, 2000.
8. Liere R., Tadepalli P.: Active Learning with Commitees for Text Categorization. In *Proceedings of the 14th National Conference on Artificial Intelligence,Providence*, pages 591–596, 1997.
9. Muggleton S., De Raedt L.: Inductive Logic Programming: Theory And Methods. *J. of Logic Programming 1994:19,20:629-679.*
10. Nepil M., Popelínský L.: Part-of-Speech Tagging by Means of ILP and Active Learning. ECML'01 Workshop on Active Learning, Freiburg 2000.
11. Pala K., Rychlý P., and Smrž P.: DESAM – Annotated Corpus for Czech. In *Plášil F., Jeffery K.G.(eds.): Proceedings of SOFSEM'97, Milovy, Czech Republic. LNCS 1338*, pages 60–69. Springer, 1997.
12. Pavelek T., Popelínský L.: Mining Lemma Disambiguation Rules from Czech Corpora. In *Principles of Knowledge Discovery in Databases: Proceedings of PKDD'99 Conference, LNAI 1704*, pages 498–503. Springer, 1999.
13. Pavelek T., Popelínský L., Ptáčník T.: On Disambiguation in Czech Corpora. FIMU-RS-2000-07, Faculty of Informatics MU Brno 2000.
`http://www.fi.muni.cz/informatics/reports/rep2000/brief.html`
14. Thompson C.A., Califf M.E., Mooney R.J.: Active Learning for Natural Language Parsing and Information Extraction. In *Proceedings of 16th International Conference on Machine Learning*, Morgan Kaufmann, San Francisco, CA, pages 406–414, 1999.
15. Sedláček R., Smrž P.: Automatic Processing of Czech Inflectional and Derivative Morphology. In *Proceedings of the Fourth International Conference TSD 2001*, LNAI 1902, Pilsen, Czech Republic, September 2001, Springer-Verlag.
16. Smrž P., Žáčková E.: New Tools for Disambiguation of Czech Texts. In *Text, Speech and Dialogue: Proceedings of TSD'98 Workshop*, pages 129–134. Masaryk University, Brno 1998.

17. Žáčková E., Popelínský L.: Automatic Tagging of Compound Verb Groups in Czech Corpora. In *Text, Speech and Dialogue: Proceedings of TSD'2000 Workshop, LNAI.* Springer 2000.
18. Žáčková E., Popelínský L., Nepil M.: Recognition and tagging of compound verb groups in Czech. In *Proc. of the 2nd Workshop on Learning Language in Logic LLL-2000, Lisbon, 2000.*

Publications in the FI MU Report Series are in general accessible via WWW and anonymous FTP:

http://www.fi.muni.cz/informatics/reports/
ftp ftp.fi.muni.cz (cd pub/reports)

Copies may be also obtained by contacting:

Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic