New Version of a Translater for a Natural Language Study

Line Jakubiec-Jamet

New Version of a Translater for a Natural Language Study

Line Jakubiec-Jamet
LIS - UMR 7020 CNRS, Aix Marseille University
Parc Scientifique et Technologique de Luminy
163, avenue de Luminy - Case 901, F-13288 Marseille Cedex 9, France
line.jakubiec@lis-lab.fr

October 2nd, 2020

Abstract

This paper presents a tool that is used in a natural language processing study. A first version of a translater has been developped yet but here is an amelioration of this program that translates Latex trees to Coq trees. This tool is included into a study devoted to the formalization and to the analysis of sentences in the Coq system. The analysis is based on a hierarchy of types (that represents an ontology) for type-checking the conceptual well-formedness of sentences. In this study, we investigated how to exploit the particular features of the Coq type system for natural language. The tool presented here is a part of this study and it is a translater that allows to generate automatically a Coq description from a tree described in a Latex file.

1 Introduction

1.1 Overview

The study presented in this paper is motivated by a project in natural language analysis. As the project needs tools for faciliting the use of our work in Coq, the translater aims to hide the logical aspects of the Coq specifications which can be unsuitable for a linguistic analysis. A first version of this translater has been developped yet [1] but this report presents an amelioration of this Java program that translates Latex trees to Coq trees. The first version of the translater needed to be improved [2]. This version is divided into four steps : processing the file, analysing the data in the file, using the information, and writing the .v file. In the first step the program creates a temporary file containing only the tree part of the LaTeX file. Then the program parses this temporary file to extract the nodes and insert them into a list. This list of nodes is then processed using various functions. And finally, the .v file is generated accordingly.

Here we recall as in [2], the subject of the underlying study for natural language processing. The Figure 1 gives an overview of the tool based on the Coq type-checking. It is not yet fully implemented but it allows to explain the motivations of our work.
The application takes as input a natural language sentence and a latex description of a tree. On one hand, in this sentence, the words will be tagged (the parser implemented in Java allows to

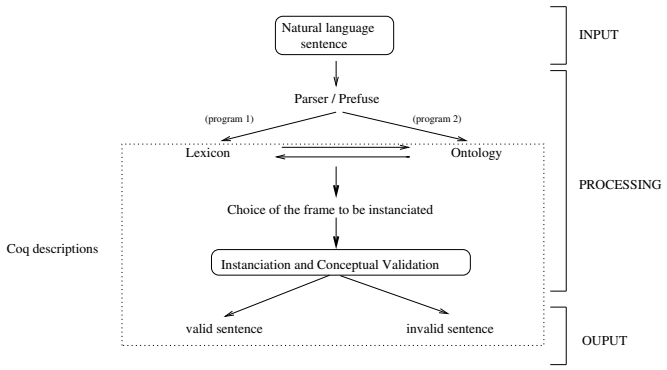

Figure 1: Overview of the application based on a Coq analysis

interactively tag verbs the user wants to classify according to an ontology to take into account). On the other hand, the tree represents a hierarchy of concepts and it is used during the analysis of the sentence. This tree is an ontology as the one depicted in the figure 2.

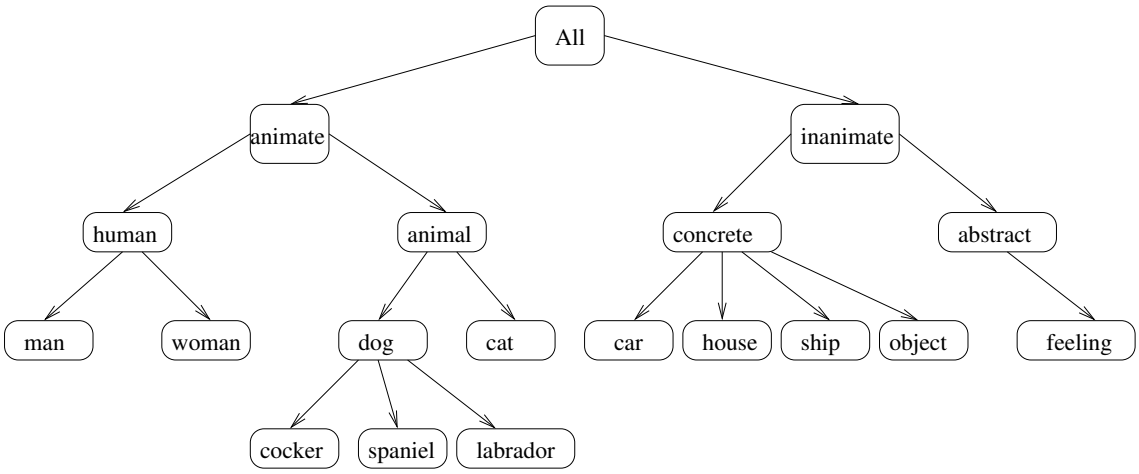

Figure 2: A hierarchy of conceptual types

This classification ($program1$ in the above figure) provides a lexicon of verbs in Coq (for instance, *bark* is tagged as $dog \rightarrow Prop$ depending on the choice of the user). The Coq description of the ontology can be automatically obtained by the $program2$. Thus, the $program2$ part in the figure, generates from a latex file, the corresponding Coq file that describes the coercion between types and also the semantic predicates corresponding to the ontology ($is\_dog : animal \rightarrow Prop$ for instance). Then the user chooses the Coq generic frame to be instanciated and the conceptual validation is performed as it has been depicted in this paper. If the representation of the sentence could be type-checked into Coq, the application outputs that the sentence is semantically valid ;

otherwise, it returns that the sentence is not valid. The sentence is semantically invalid according to the ontology under consideration.
This application is motivated from the fact that there are few resources in french about the semantics of words. Several linguistic resources rely on a classification of words where each class of verbs has abstract properties. In LVF[3], verbs are described as semantics classes whose scope is defined by syntax. These classes are generic and each of them gathers properties of verbs. For example, there is the class dedicated to the communication verbs. This latter is divided into 4 semantic categories (for french language) :

1. human, animal (to shout, to speak)
2. human (to say something)
3. human (to show)
4. figurative sense

Then these categories are subdivided into syntactical sub-classes which describe the use cases of verbs (the verb can be used with a subject and a complement, it can be transitive or intransitive and so on). Several studies based on LVF have been proposed [4], [5] and [6]. and they provide many linguistic descriptions not yet taken into account in our case-study.

1.2 Natural language studies based on type theory

From a logical point of view, using type theory for analyse natural language is not new. During the last fifteen years, the use of type theoretic methods for describing natural langage syntax and semantics has gained more and more popularity, resulting in the development of software relying on these methods. First, these type theoretic methods can be constantly improved with advances in the field of logic. Secondly, they depend on the developement of linguistic resources such as lexicons or dictionaries. In this context and from our point of view, the challenge for natural langage processing is twofold : the frameworks dedicated to linguistic analysis have to focus on the syntax-semantics interface ; and they have to combine linguistic resources and natural language processing programs. Among the most significant achievements, let us mention the works on categorial grammars which provide the integration of syntax and semantics in the same framework as it is described in [7] and in [8]. Moreover, categorial grammars are lexicalized that means that all items in the lexicon are typed. Thereby, although they describe syntactical rules, they also preserve the compositional aspect of Montague semantics [9]. The most well-known categorial grammars are those based on Lambek-Calculus [10]. Due to the Curry-Howard isomorphism, typed terms are proofs in logic which includes Lambek-Calculus[1]. Although many studies have already showed that it is natural to associate a syntactic term to a semantic type [15] [16] [17], our approach implemented in Coq, the Calculus of Inductive Constructions, aims to focus on the generality of definitions leading to reusable methodologies dedicated to semantic analysis of natural language. In Coq, few investigations have been performed for natural language processing. In [18], the author has developed an algorithm that produces natural language sentences from proofs described in a mathematical language. Later, for the case of categorial grammars, [19] gave a Coq

[1]There are new approaches that extend Lambek-Calculus, in particular in linear logic because it provides an efficient representation of proofs [11] [12] [13] and specific tools have been developed in this field as for example [14].

formalization of the Lambek-Calculus and of an extension as multimodal grammars; they prove several theorems as completeness and consistency of multimodal logic. More recently, in [20], the author explains how modern type theories provide a wide semantic coverage of linguistic features. Powerful typing mechanisms that have been implemented in proof systems can be employed in the field of linguistic semantics as it is described in [21] and [22]. Following these ideas, this work shows how straightforwardly use specific features of Coq language for semantic analysis [23] . It aims at showing how :

1. define formal models for representing sentences by taking advantage of the Coq type system and its particularly rich language (polymorphism, higher-order logic, coercion mechanism, module system),

2. propose natural and general specifications of sentences that can be checked for a conceptual analysis based on types and coercion mechanism,

3. take advantage of type-checking algorithms involved into the Coq system.

1.3 The tool

The tool presented here was the subject of a student internship [24]. The final objective is to determine whether a sentence in French is correct by looking at its meaning. To do this, we must use ontologies that allow us to give us a hierarchy of types and thus determine whether a sentence makes sense or not. For example, the sentence "to eat a stone" would not make sense because in the ontology we have chosen, "stone" would be in the category of non-edible things. But only edible things can be eaten. The first part of the course focuses on the graphic and logical representation of trees. These trees represent concept hierarchies used in the project on natural language semantics. Some tools allowing the graphical representation of trees (in general) will be studied. The project aims to study the way these trees are represented in these tools and then to translate this representation in a logical language (useful to perform further semantic analysis of sentences in natural language). In other words, it is a matter of writing a program which, given the graphical representation of a tree, will produce automatically its logical representation. This program can be developed in Java or C and the resulting target code must be compilable in the logical tool (that it will not be necessary to study in details for the internship). Only a sub-part of this tool will be presented for the purposes of the internship, in particular to ensure that there are no compilation errors when the logical representation of the tree is generated by the translater.

1.4 Organisation of the paper

The paper is organized as follows. Section 2 briefly introduces Coq by focusing on used aspects. The section 3 rapidly browse latex and in particular the package named forest. Section 4 deals with a formalization of the underlying ontology and with a semantic representation of simple sentences. In the section 4.3, we generalize our approach by specifying generic models for other sentences and we show how to use them. Then, in the section 5, the paper explains the work realised during the intership (this is the new part of the work). Then, in the conclusion, we further discuss the case-study and we highlight our perspectives.

2 An Overview of Coq

The Coq system [25] is a specification and proof system developed in the LogiCal project at Laboratoire de Recherche en Informatique (CNRS and University of Paris-Sud) and LIX (INRIA-Futurs and Ecole Polytechnique). Coq's language relies on a higher-order typed λ-calculus, the Calculus of Constructions [26] [27] enriched with inductive and co-inductive definitions [28] [29]. Coq's logic is a constructive logic and it is based on the *propositions-as-types* correspondence, the Curry-Howard isomorphism, that states a proposition is a type and a proof is a term inhabiting this type. This correspondence provides an elegant unifying framework where type-checking is proof-checking. The Coq system is tactic oriented and it allows to interactively develop proofs. The system is organized around a small kernel (the theory) extended by libraries. Moreover, it includes many user contributions.

Coq developments can be splitted into various parameterized modules. Thus, several developments can share modules that, being compiled once and for all, are loaded fast. Moreover, sections allow to organise modules in a structured way. In Coq, any user's term must be classified according to a type. There are two sorts of types : logical propositions are of sort $Prop$ and mathematical collections are of sort Set[2]. Polymorphic terms are parameterized with respect to terms of sort $Prop$ or Set. By defining them into a section, one can obtain reusable developments in which generic specifications has been already typed-checked. However, when instanciating these abstract specifications, types can be cumbersome. The implicit parameter mechanism allows to automatically infer arguments from the definition's context.

Moreover, Coq terms are organised according to a type hierarchy and they can be typed using the coercion subtyping system. This latter corresponds to an inheritance graph mechanism that allows to inject Coq hierarchy typed terms into another hierarchy's type. Technically, a term of type t is also of type t_1 (where t and t_1 are of types $Prop$ or Set for example) if there exists a coercion between t_1 and t, defined in Coq as :

```
Coercion c : t1 >-> t.
```

This declaration expresses the construction denoted by c as a coercion between $t1$ and t. Roughly speaking and for the need of the study, the coercion c indicates that t_1 is a subtype of t.

3 An Overview of Latex

3.1 Overview

Latex [30] is widely used in academic publications, in particular for the publication of scientific documents in mathematics and computer science. It is is a document preparation system where files are created with conventions by users for describing page layout but the latex page does not correspond to what you see on it. It is opposed to the formatted text found in "What You See

[2]Actually, this distinction is not necessary but it makes the system less confusing for the user. However, it is significant when extracting programs from proofs (a mechanism relying on the constructive aspect of Coq's logic). But this feature is not used here.

Is What You Get" word processors. The user defines the structure of a document (book, article, report, slide, letter and so on) and he can markup his text to stylise it (italics, bold, large, footnotesize for instance). The possiblities are numerous because the user can also use packages of all kinds (graphics, mathematics, tables, pictures and so on) as libraries or even make his own packages. Then, the latex file is compiled to produce an output file (pdf or dvi).

For instance, here is a latex file that structures the document to produce from a compilation instruction :

```
\documentclass{article}
\title{Title of the article}
\author{Author of the article}
\date{Date of the article}
\begin{document}
\maketitle

\begin{abstract}
\end{abstract}

\section{Title of section 1}
\subsection{Title of subsection 1.1}
\subsection{Title of subsection 1.2}

\section{Title of section 2}
\subsection{Title of subsection 2.1}
\subsection{Title of subsection 2.2}
\end{document}
```

From the pdflatex compilation instruction, we obtain the final document :

Title of the article

Author of the article

Date of the article

Abstract

1 Title of section 1

1.1 Title of subsection 1.1

1.2 Title of subsection 1.2

2 Title of section 2

2.1 Title of subsection 2.1

2.2 Title of subsection 2.2

In summary, not only Latex is a markup language that allows to write documents that are automatically formatted, but it also provides various packages adding new editing features. The package we are interested in here is the Forest package which allows us to write trees inside a Latex document. It is these trees that later, we want to parse and rewrite so that they can be interpreted by Coq. There are of course other packages for writing trees in LaTeX, such as Qtree.

3.2 The Forest package

As mentionned, this study uses hierarchy of concepts (or hierarchy of types in Coq) for natural language analysis. So, it is necessary to manage the construction of trees in order to facilitate the Coq analysis of sentences. As Latex is commonly used in research, we decided to choose it for constructing trees. The user can depicted a tree in Latex and then he can submitted it to the tool for translating the Latex description into the Coq description. We have chosen the forest package for managing trees. This package provides a mechanism for drawing trees as for example the hierarchy of conceptuel types depicted in the Figure 1. This facility allows the encoding of trees using brackets and names for the nodes of the tree (as identifiers and in particular, conceptual types in the example). The package also provides many tree-formatting options and the possiblity to decorate trees in many ways but these characteristics are not used in this work.

In particular, as mentionned in the distribution of latex, the main features of this package are :

- a packing algorithm which can produce very compact trees,
- a user-friendly interface consisting of the familiar bracket encoding of trees plus the key-value interface to option-setting,
- many tree-formatting options, with control over option values of individual nodes and mechanisms for their manipulation,
- the possibility to decorate the tree,
- an externalization mechanism sensitive to code-changes.

Using the forest package, it is possible to describe such a tree :

```
\documentclass{article}
\usepackage{forest}
\begin{document}
\begin{forest}
 [ node 1
        [node 1.1]
        [node 1.2]
        [node 1.3]
 ]
\end{forest}
\end{document}
```

that represents a tree with one node labelled 1 that has got three children respectively labelled node 1.1, node 1.2 and node 1.3. This kind of latex description will be submitted to the translater

in order to automatically generate the formalisation of this description in Coq, where nodes will represent types.

4 The case study implemented in the Coq system

4.1 Ontology as Concept Hierarchy

This section is devoted to the presentation of the ontology used to determine the well-formedness of sentences. Let us note that the "concepts as types" representation is largely used in the knowledge representation since it provides an appropriate interface for semantic processing.
Figure 2 presents a conceptual hierarchy which organises concepts according a world we want to refer for our study. The verification of the sentence's conceptual well-formedness will be based on this hierarchy. It describes a simple world from which it is possible to analyse the meaning of sentences. This world is organized from *animate* and *inanimate* notions. For example, an *animal* is classified into the *animate* concept while a *car* is *inanimate* and can be considered as *concrete* as well. In this tree, each node is labelled by a conceptual information that can be specified as a type. Thus, for organizing this information, it is natural to consider the subtyping principle. In typed definitions, a subtype may appear wherever an element of the super type is expected. For example, in our verb's semantic representation (Section 4.2.1), a type t_1 is compatible with a type t, if t_1 is a subtype of t. Consequently, a semantic representation parameterized with t will be valid for parameters of type t_1 and for all those of the lower part.
In Coq, we declare the conceptual types (*all*, *animate*...) as logical propositions. Then, we use the coercion mechanism for describing the relations between types, as follows :

```
Coercion animate_is_all    : animate >-> all.
Coercion animal_is_animate : animal >-> animate.
Coercion dog_is_animal     : dog >-> animal)
Coercion cat_is_animal     : cat >-> animal)
      ......
```

Each coercion creates a path between two nodes of the tree. The whole list of coercions is ordered and it defines the conceptual tree depicted in Figure 2. Coq detects ambiguous paths during the creation of the tree and, it verifies the *uniform inheritance condition* because at most one path must be declared between two nodes. Let us remark that, in Coq, the conceptual tree is straightforwardly implemented, in a natural way. Therefore, Coq automatically verifies that the hierarchy of types is well-formed. Moreover, the conceptual analysis will be parametrizable by this kind of hierarchy : in general, the ontology depends on the field under consideration and it is interesting to be able to change the ontology according the needs.

4.2 Coq Semantic Representation of Sentences

This section proposes a semantic formalization of simple sentences. Since a sentence is composed of elements that must be compatible with each other, we first deal with their representation and their types. In particular, we focus on the verb's description because the sentence's representation is specified according to the verb's domain of use. Then we describe a generic model for sentences which involve a verb and its subject. Finally, we show, by instanciation of the model, how sentences are represented.

The verb has a central role in the sentence, as it has been developed in the Tesnière's linguistic theory [31] which defines the valence of verbs. Let us mention that, this characterisation of sentences have been widely used in the dependency grammars as it is described in [32].

4.2.1 Lexicon of Verbs

For typing the verb's representation, we use the conceptual types described in Figure 2. For each verb, we have to choose the best label (best for the user who build the lexicon that is to say the most likely concept for the domain under consideration) which, in general, corresponds to the lower type in the hierarchy. For example, the verb *to bark* can reasonably be used for all the dogs. So, it is declared in the lexicon as a logical proposition by the unary predicate *bark* as follows :

```
Parameter bark : dog -> Prop.
```

Let us notice that verbs can have different meanings and so, different lexical entries have to be considered. Once the hierarchy of types is defined, the verbs have to be described on these types. So, we can need to consider several declarations in Coq for the same verb. For example in french, the verb *bark* can be used for humans : *Paul barks.* (in the sense that Paul inveighs against someone or something) is an acceptable sentence. Here, we introduce a new input in our lexicon as :

```
Parameter bark2 : human -> Prop.
```

This is not really cumbersome because dictionnaries of verbs are built from all the possible situations (see Section **??**) and then, the Coq representation is very concise.

4.2.2 Sentences as logical expressions

Let us consider the sentence : *A dog is barking.* It can be represented by the logical expression : $\exists x, bark(x) \wedge is\_dog(x)$ where the unary predicate is_dog characterizes the semantics of the subject *dog*. The following predicates introduce in Coq semantic representation for some agents used later in the paper :

```
Parameter is_animate : all -> Prop.
Parameter is_animal  : animate -> Prop.
Parameter is_human   : animate -> Prop.
Parameter is_dog     : animal -> Prop.
Parameter is_cocker  : dog -> Prop.
```

Finally, the sentence : *A dog is barking* merely can be represented in Coq as follows :

```
Definition a_dog_is_barking :=
  exists x, bark(x) /\ is_dog(x).
```

where the implicit argument mechanism automatically synthesizes the type of x as *dog* (from the first predicate of the definition).

4.2.3 Towards Generic Models

The kind of sentences we study is composed of a verb and a subject. In this part, we show how to generalize this kind of sentences by specifying a general frame in Coq that states : $\exists x, verb0(x) \wedge is\_something(x)$, where $verb0$ (that stands for the verb) and $is\_something$ (that stands for the subject) are polymorphic predicates respectively parameterized on $A1$ and A of sort $Prop$, with $A1$ subtype of A (to ensure the compatibility between words). The complete specification in Coq is given below, inside a section :

```
Section General_frame_v0.
     (** Local parameters of the section **)
  Variables (A A1:Prop)
            (A1A : A1 -> A).
     (** Declaration of the subtype **)
  Coercion A1A : A1 >-> A.
     (** Declaration of the predicates **)
  Variables (verb0 : A1 -> Prop)
            (is_something : A -> Prop).
     (** Definition of the generic model **)
  Definition frame_verb0 :=
     exists c, verb0 c /\ is_something c.
End General_frame_v0.
```

In the definition $frame\_verb0$, c is implicitly of type $A1$ and the coercion $A1A$ which converts the type $A1$ to A is implicitly applied on c into $is\_something(c)$. Outside the section, the local context of the definition is discharged. This means that A, $A1$, $A1A$, $verb0$ and $is\_something$ appear as parameters of the definition $frame\_verb0$.
So, $frame\_verb0$ depends on two types, on a coercion which states a subtyping relation and on two predicates which respectively stand for a verb and a subject. It is a generic representation for this kind of simple sentences due to polymorphism (from the parameters A and $A1$) and higher-order (from the $verb0$ and $is\_something$ predicates).

4.2.4 Type-checking is Well-formedness Checking

By instanciation of the generic model $frame\_verb0$, we can define the semantic representation of the sentence *A dog is barking* as :

```
Definition a_dog_is_barking :=
    (frame_verb0 dog_is_animal bark is_dog).
```

The instanciation of the parameters A and $A1$ can be omitted due to the implicit synthesis. The coercion *dog_is_animal* instanciates $A1A$ in the generic model ; *bark*, which is defined on *dog*, instanciates $verb0$ and *is_dog*, which is defined on *animal*, instanciates $is\_something$. So, the compatibility of words is ensured by the coercion that states *dog* is a subtype of *animal*.

In a similar way, the sentence *A cocker is barking* is formalized as :

```
Definition a_cocker_is_barking :=
           (frame_verb0 cocker_is_dog bark is_cocker).
```

But the checking of *A cat is barking* :

```
Definition a_cat_is_barking :=
           (frame_verb0 cat_is_animal bark is_cat).
```

is rejected by the system because the term *bark* has type $dog \to Prop$, while it is expected from *cat_is_animal* (the coercion that states *cat* is a subtype of *animal*) to have type $cat \to Prop$. This encoding leads to simple conceptual representation of sentences. Coq performs the type-checking of these instanciations and so, it establishes the sentence's well-formedness.

4.3 Generic Models Based on Verb's use

In this section, we propose two other generic frames for representing the sentences. The first one describes sentences which are composed of a verb, a subject and a complement. The second frame defines sentences where the parameters of the verb depends on each others. This latter is interesting is the case-study because it emphasizes the dependent aspect of typing for natural language processing.

Let us consider the sentence *A woman likes cars*. In this case, the verb *to like* may be of type $animate \rightarrow inanimate \rightarrow Prop$, where *animate* stands for the type of the subject and *inanimate* for the type of the complement[3]. Similarly to the representation given in Section 4.2, we can specify :

```
Definition a_woman_likes_cars :=
   (frame_verb1 woman_is_human
          car_is_concrete like is_woman is_car).
```

where the model $frame\_verb1$ is obtained from the generic definition depicted below :

```
Section General_frame_v1.
  Variables (A A1 B B1:Prop)
            (A1A : A1 -> A)
            (B1B : B1 -> B).
  Coercion A1A : A1 >-> A.
  Coercion B1B : B1 >-> B.
  Variables (verb1 : A1 -> B1 -> Prop)
            (is_something1 : A -> Prop)
            (is_something2 : B -> Prop).
  Definition frame_verb1 :=
      exists x, exists y, verb1 x y /\
             is_something1 x /\ is_something2 y.
End General_frame_v1.
```

As in the previous frame, coercions has been defined between $A1$ and A and between $B1$ and B. The predicates $is\_something1$ and $is\_something2$ respectively stand for the subject and the complement. So the instanciation of $frame\_verb1$ is realized using the two coercions *woman_is_human* and *car_is_concrete*, the verb *like* and the predicates *is_woman* and *is_car*.

The second model describes a particular case where sentences are composed of a verb, a subject and a complement as well, but the typing of the verb is more binding. For example, let us consider the verb $confuse$. Only a human can $confuse$ two things that must represent the same concept in the hierarchy (for instance, a man confuse two dog breeds, two particular cars and so on ; but he cannot confuse a car and a dog). So, the type of the generic relation ($verb2$ in the hereunder definition) that stands for $confuse$ is $human \rightarrow A \rightarrow A \rightarrow Prop$ where A gives the general concept to be used but the two subtypes of A can be different althought they are from the same concept.

This is specified in the next Coq definition inside a section :

```
Section General_frame_v2.
  Variable A A1 A2 : Prop.
  Variable A1A : A1->A.
  Coercion A1A : A1>->A.
  Variable A2A : A2->A.
```

[3]But, more generally, the type of the verb *to like* in the lexicon is $animate \rightarrow all \rightarrow Prop$ because the type of the complement must be as general as possible.

```
  Coercion A2A : A2>->A.
  Variables (verb2 : human-> A ->  A -> Prop)
            (is_something1 : A1 -> Prop)
            (is_something2 : A2 -> Prop).
  Definition frame_verb2 : Prop :=
     exists h:human, exists a1:A1, exists a2 :A2,
     verb2 h a1 a2 /\ is_human h /\
            is_something1 a1 /\ is_something2 a2.
End General_frame_v2.
```

The two types from A are $A1$ and $A2$; the variable $verb2$ sets that the first argument is a human while the following are from A.
As already described, due to implicit synthesis, the application ($verb2$ h $a1$ $a2$) is actually ($verb2$ h ($A1A$ $a1$) ($A2A$ $a2$)), for generating the type A from $A1$ and from $A2$. The instanciation of this frame is similar to the previous paragraphs and it is not given here.

5 The translater

5.1 Basic Java program

As in [1], the new version of the translater is implemented in Java. We chose to program it in the Java language because Java is one of the most popular programming languages (and students are very comfortable with it). It also takes as input a latex file (.tex) and it produces as output a Coq file (.v), using a parsing analysing. These two files contains the description of a tree, the first is given using the forest package of latex and the second corresponds to the Coq formalisation of the tree.

The program is composed of a main Java class and a node class. The node class does not contain any methods and only serves as a structure. It has three attributes :

- name, the name of the node,
- parent, the parent node of the node,
- children, a list of nodes representing the children of the node.

The constructor is very simple : we define the name of the node and its father as parameters. Then we initialise children, and add this new node to the list of children of its father (if any). The heart of the program is located in the main class and works in two stages : firstly, a graph is created from the forest tree (the graph is only a simple list of adjacencies of nodes). This graph creation is performed by the createGraphFromFile() method which takes as parameter the latex file given as input. In a second step, the different fields of the Coq file (.v) are created using a simple path in width on this previously created tree. Other basic features are used as for instance the reading of the file and the concatenation of the different lines to produce the output file but it is not really significant for describing the translater. To use it, we just need ti specify the path to the .tex file as input to the program and then, the translater will automatically generate the corresponding Coq file.

5.2 New version

This part describes the development of the translation program from Latex files to Coq files in the framework of our study. In addition to simplifying the Java code, the new translator allows :

- to have a program able to read real latex files (not just files containing only the trees),
- to have a program able to processing files containing several trees.

Moreover, special care has been paid to this new version of the translater because even the most poorly written files is translatable.

5.2.1 Parser

We create a LatexParser class that parses a latex file. This class actually has only one main function which is the parse() function which parses the file and retrieve the different trees (ontologies) it finds in it. It has a constructor and a close() function that respectively create and close a Bufferedreader (a class which simplifies reading text from a character input stream, here the latex file).

5.2.2 Creation of an automaton

The automaton describes the different methods of the LatexParser class. We have chosen to represent the "states" of the automaton by functions. Its states are described hereunder :

- State Q0 : in the initial state of this automaton, the file is read until either the end of the file is reached (in which case the final state is reached) or a backslash is read.
- State : in the second case, we continue to read the file and build a string by reading it. If a backslash is read again, it means that the command cannot be of the form "begin{forest}", and so the string is read again from scratch.
- State { and state begin : if the parser reads an opening bracket we go to the next state, and if the string read is "begin" we continue reading. Otherwise we return to the initial state.
- State } and state forest : similarly, after reading "begin", the parser constructs a new string; if it is "forest" we continue, otherwise we return to the initial state.
- State {} : once "begin{forest}" is read, it may be that the tree starts with "for tree = { something }", there is a state to handle this case.
- Parsing state : finally, as soon as the first bracket is read outside the braces, we finally enter the tree. What happens in the "parsing" state is equivalent to the "createGraphFromFile()" method of the first version of the program with one major difference, which is that there is a counter that increments and decrements respectively when we encounter an opening and a closing brace. The method uses the treatNode() function that is divided of four parts :
  1. Add the node to the ontology.
  2. Set the current node as the new parent.
  3. Reset the name of the current node.

4. Read the next character.

- State Q0 : Once the counter has reached zero, we return to the initial state. Parsing can then continue and handle any other trees in the file.

Finally, to use the translater, you have to go to the "tex2coq" folder, and compile the latex file with the command "tex2coq file.tex". The Coq file(s) will be generated in the same directory as the latex file.

6 Conclusion and Perspectives

The tranlater presented in this report is included into a work that aims at studying the capabilities of the Coq system, in the field of semantic representation and conceptual analysis for natural language processing. The relevance of our encoding has been motivated by the particular features of the Coq system. It provides an unifying framework with a rich type system that allows to straightforwardly specify the underlying conceptual tree as well as to analyse semantic representations. In this paper we have proposed :

1. a way for describing ontologies in Coq,
2. several reusable sentence's semantic representations,
3. a sentence's conceptual analysis by instanciation process.

In particular, for this analysis, we focused on several aspects of the Coq system :

1. polymorphism : the generic models of sentences are parametrized by types and they can be reused for specifying specific sentences.
2. higher-order : it allows to take as parameters relations and so it provides a good framework for developing general definitions. From these definitions, specific sentences are derived by instanciation.
3. modularity : the development is split into several sections. This contributes to lisibility and it allows an easy reuse of libraries.
4. coercion mechanism : the hierarchy of types is easily described in Coq and it is a good way for knowledge representation based on types.
5. implicit parameters : the implicit synthesis of some parameters greatly improves the readability of the definitions.

The case study proposed in this paper is being extended to several other modules. In particular, it requires the addition of the following :

1. extension of the verb's lexicon : the lexicon takes into account around 50 verbs. We plan to use the developments of LVF mentionned in the Section **??** for improving the semantic analysis (by specifying classes of verbs and general models that characterize the uses of verbs). This study will allow to integrate syntactical rules for introducing more linguistic information in our representations. From this specification, formal properties about sentence's representations could be established in order to validate linguistic rules in Coq.

2. representation of other sentences : once the lexicon will be extended, it will be possible to describe other generic models of sentences.

3. contextual analysis : for completing the semantic analysis, contextual representation will be necessary. This will allow to analyse texts and not just sentences.

Finally, the application depicted in the Figure 1 should be completed by adding more automatic processing : firstly, in the parser, the tagging of verbs has to be improved. This can be done using an underlying lexicon which includes linguistic properties about verbs (conjugaison patterns, lemmatization for example). Secondly, the interface between the lexicon and the ontology has to be developped. This process involves the programming of an interface that allows to manipulate verbs and concepts according to the domain of the sentences to be analysed. Thirdly, it should be relevant to give more explanation about the output when an invalid sentence has been detected. This is possible by retrieving from the Coq system the information about typing, in order to propose for the user, some accepted constructions of sentences.

Moreover, we plan to continue the development of tools for using our approach about the semantic analysis of sentences in an easier way. In the field of natural language processing, many resources, such as dictionaries, lexicons, texts in general are available. Their use sometimes requires a translation in an adequate language so that they can be exploited according to the needs of the users. In particular, many ressources are available in XML format (as for example [6]). So we are interested at developing a set of programs that translate XML format files into the target language (the Coq language for this study). The objective is to extract relevant information from dictionaries already coded in XML and then to represent them in logical language (in order to be able to exploit them for our underlying semantic analysis).

References

[1] S. Merrouch, *Développement de programmes Java pour l'analyse sémantique du langage naturel*, Stage LIS, Marseille, 2018.

[2] L. Jakubiec-Jamet, "A Translater from Latex Trees to Coq Trees for a Natural Language Study," LIS - UMR 7020 CNRS, Marseille, Tech. Rep. hal-03536652, 2019.

[3] J. Dubois and F. Dubois-Charlier, *Les verbes français*. Larousse-Bordas, 1997.

[4] M. Silberztein, "La formalisation du dictionnaire LVF avec Nooj et ses applications pour l'analyse automatique de corpus," in *Empirie, Théorie, Exploitation : le travail de Jean Dubois sur les verbes français*, ser. Langages, no. 179-180, 2010, pp. 221–241.

[5] J. François, D. Le Pesant, and D. Leeman, "Classements syntactico-sémantiques des verbes français," in *Langue française*, vol. 153, 2007.

[6] F. Hadouche and G. Lapalme, "Une version électronique du LVF comparée avec d'autres ressources lexicales," in *Empirie, Théorie, Exploitation : le travail de Jean Dubois sur les verbes français*, ser. Langages, vol. 179-180, 2010, pp. 193–220.

[7] M. Moortgat, "Categorial Type Logics," in *Handbook of Logic and Language*, ser. North-Holland Elsevier-Amsterdam, J. Benthem and A. T. Meulen, Eds., 1996, pp. 93–177.

[8] C. Retoré, "Systèmes déductifs et traitement des langues, un panorama des grammaires catégorielles," *INRIA*, 2001.

[9] R. Montague, "The Collected Papers of Richard Montague," in *Yale University Press*, R. Thomason, Ed., 1974.

[10] J. Lambek, "The Mathematics of Sentence Structure," *American mathematical monthly*, vol. 65, pp. 154–169, 1958.

[11] P. De Groote and C. Rétoré, "Semantic Readings of Proof Nets," in *Formal Grammar*, O. D. Kruijff G., Morrill G., Ed. FoLLI, 1996, pp. 57–70.

[12] G. Perrier, "Labelled Proof Nets for the Syntax and Semantics of Natural Languages," in *Logic Journal of the IGPL*, vol. 7, 1999, pp. 629–654.

[13] A. Lecomte, "Grammaire et théorie de la preuve : une introduction," in *Traitement automatique des langues*, vol. 37, 1996, pp. 1–38.

[14] R. Moot, "A short Introduction to Grail," in *Proceedings of Methods for Modalities*, C. Areces and M. Rijke, Eds., 2001.

[15] A. Ranta, "GF: a Multilingual Grammar Formalism," in *Language and Linguistics Compass*, vol. 3, 2009.

[16] R. Moot and C. Rétoré, *The Logic of Categorial Grammars*, ser. FoLLI-LNCS. Springer, 2012.

[17] R. Muskens, "Type-logical Semantics," in *Routledge Encyclopedia of Philosophy Online*, E. Craig, Ed. Routledge, 2011.

[18] Y. Coscoy, "Explication textuelles de preuves pour le calcul des constructions inductives," Thèse d'université, Université de Nice-Sophia-Antipolis, 2000.

[19] H. Anoun, "Reasoning on Multimodal Logic with the Calculus of Inductive Constructions," in *Logic for Programming Artificial Intelligence Reasoning*, 2005.

[20] S. Chatzikyriakidis and Z. Luo, Eds., *Modern Pespectives in Type Theoretical Semantics*. Springer, 2017.

[21] Z. Luo, "Type-theoretical Semantics with Coercive Subtyping," in *SALT*, 2010.

[22] ——, "Common Nouns as Types," in *LACL*, 2012.

[23] L. Jakubiec-Jamet, *Natural Language Processing and Coq : a Case-study*, WIL-Workshop LICS, Reykjavic, Island, 2017.

[24] R. Rochas, *Développement de programmes Java ou C pour l'analyse sémantique du langage naturel*, Stage LIS, Marseille, 2019.

[25] *The Coq Proof Assistant. Reference manual, v8.4*, Coq Development Team, INRIA, 2014.

[26] T. Coquand, "Une théorie des constructions," Ph.D. dissertation, Université Paris 7, Janvier 1989.

[27] T. Coquand and G. Huet, "Constructions : A Higher Order Proof System for Mechanizing Mathematics," *EUROCAL 85, Linz Springer-Verlag LNCS 203*, 1985.

[28] C. Paulin-Mohring, "Inductive Definitions in the System Coq - Rules and Properties," in *Proceedings of the conference Typed Lambda Calculi and Applications*, ser. Lecture Notes in Computer Science, M. Bezem and J.-F. Groote, Eds., no. 664, 1993, research report, lIP research report 92-49.

[29] E. Giménez, "Un calcul de constructions infinies et son application à la vérification de systèmes communicants," Ph.D. dissertation, Ecole Normale Superieure de Lyon, 1996.

[30] L. Lamport, *A Document Preparation System*, Palo Alto, 1994.

[31] L. Tesnière, *Eléments de syntaxe structurale*. Klincksieck, Paris, 1959.

[32] S. Kahane, "Grammaires de dépendance formelles et théorie sens-texte," *TALN*, 2001.