A multilingual semantic dependency parser for texts of legislation in the domain of Value Added Tax (VAT)

Dan Thorin

Master Thesis
Lund University - November, 2008

Abstract

Semantic parsing is a rapidly growing sub-field of computational linguistics, driven by the seemingly endless potential of natural language processing (NLP) in combination with an increasing demand for applications guiding and aiding humans in a variety of language related tasks. As a pilot study, this project investigated whether it was possible to create a classifier that correctly detected and labeled semantic elements from legal texts concerning Value Added Tax (VAT), and if it was possible in three different natural languages: English, Danish and Swedish. By detecting the semantic elements, the essential meaning of a particular sentence could be derived. This is a feature that could be used as an artificial semantic text scanner for applications taking input from small or large databases of written text. The input corpora were transformed from running text into a set of predicate-argument structures, modeling the semantics of each sentence. The results of the project are promising. Almost all of the target predicates and around 80% of the arguments were correctly detected and labeled. To demonstrate the performance of the classifiers, a semantic classifier program was implemented, visualizing the way each sentence was transformed into predicate-argument statements. Not surprisingly, it showed that short sentences with fairly distinct arguments were generally easily transformed, while the arguments of longer sentences comprising several predicates, were less accurately determined.

Acknowledgments

I would like to thank Pierre Nugues of Lund University for his sincere supervision and tutoring throughout the eight months required to finish this thesis project. It has been a great experience working with you, and you've always been able to help me on short notice when facing various problems or difficult decisions. I would also like to thank Richard Johansson of Lund University for providing me with the WekaGlue class, used in order to invoke the semantic models and use them to classify each instance of my corpora. The WekaGlue class formed the backbone to my semantic classifier demonstration program, and spared me numerous additional hours from an already immense task.

Contents

1 Introduction

Ever since the dawn of the computer age in the 1940s, philosophers and futurists have envisioned and anticipated computers completely or partially controlled through written or spoken natural language. A computer with such an interface would be able to take instructions in a form common to humans, enabling a more straight-forward interaction requiring less learning and adaptation on the user's part. However, the task of creating such a computer has proved to be anything but simple.

Over the years, the computer has often been compared to the human brain. Contradictory however, language, that comes very natural to human beings, comes very unnatural to computers. The explanation is that even though the computer and the brain can perform some very similar tasks, they essentially work in two very different ways. The computer needs definite and unambiguous information to work its way to a conclusion and really has no other ways to infer missing information. The human brain, on the other hand, fills out missing data by consulting its big knowledge base that is its memory, based on experience about the real world and how it functions. Natural language is very indefinite indeed, relying vastly on omitted information that requires an advanced interpreter to decipher the message, but it is also this lack of determination that makes language so versatile and efficient as a means of communication between humans. In essence, communicating through natural language enables us to take linguistic shortcuts and still being able to make a clear statement with a distinct meaning.

Computational linguistics, an offspring of linguistics and computer science, focuses on designing mathematical models and algorithms enabling the automatic processing of natural language using a computer. When research began in the United States in the early 1950s, it was widely assumed that computers' outstanding ability to perform arithmetic calculations quite easily could be ported to various applications of natural language processing (NLP) (Hutchins, 1999). Some of the first tasks included efforts to automatically translate German post-WWII documents and Russian scientific reports into English. However, the results proved poor and it was soon realized that machine translation (MT) was far more complex than what was originally expected. In fact, the influential and once of MT very enthusiastic Israeli philosopher Yehoshua Bar-Hillel (1960) even came up with a proof for the non-feasibility of automated MT, claiming that translators in their work constantly make use of high-level knowledge of the world or the document being translated. As such information simply was out of reach for automated MT systems, the aspirations were reduced to a vision of natural language translation as a mere symbiosis between human translator and computer.

Research continued on a smaller scale through the 1970s, focusing on computer-assisted translation. The paradigm at the time included a non-satisfactory MT process that was preceded, intertwined and/or followed by manual simplification or correction. In the mid 1970s however, the wind finally started to change as the Commission of the European Communities acquired SYSTRAN, an at

the time acclaimed English-French machine translation system, which spurted later versions capable of translating other languages as well. Activity in the field of MT again rose in the early 80s as new ideas for research were introduced and the sources of funding increased, mainly provided by the European Union and various computer companies. This eventually led to the introduction of commercial machine translation systems on the market.

In the modern era of NLP, new techniques for coping with the daunting complexity of language have emerged. Instead of the rule-based systems of the early days, focus has now shifted towards statistical corpus-based methods (Gildea and Palmer, 2002). Catalyzed by the exponential growth of computing power in the last 15 years, these stochastic techniques are able to address some of the previously unsurpassable problems of for instance linguistic ambiguity, quantification, anaphora, aspect, modality and idiomatic expressions as well as other linguistic complexities. However, an inevitable bottleneck for the development of NLP models using statistical techniques is that it takes vast amounts of manually annotated examples, which are produced by the gruesome work of human annotators (Johansson and Nugues, 2006). The completion of satisfactory such corpora takes time, and has to be done for each new linguistic phenomenon that is to be captured as well as for each new language. However, a possible rectification to the problem of complete linguistic phenomenon coverage might be the deployment of a method called automatic frame detection (Giuglea and Moschitti, 2006), where templates are assigned to seen phenomena that subsequently can be ported to unseen ones, thus expanding the scope and increasing the robustness of the semantic parser. A number of databases of annotated examples are currently being developed and distributed freely for English, most notably PropBank, NomBank and FrameNet. Sources of this type for other languages are unfortunately still scarce.

The rise of the Internet also brought NLP renewed potential as Internet forms a free and accessible platform for sharing information and making business between people from different parts of the world. With a growing user base and a maturing research field, a great number of new types of applications found its way onto the market. Spelling and grammar checkers, information retrieval, speech dictation, speech synthesis, voice control of various devices, interactive voice response systems, conversational agents and the like are becoming common factors of our everyday lives.

2 Theory

2.1 Semantics

The meaning of language is commonly referred to as semantics and is its own field of study, resting on the pillars of philosophy and linguistics. Semantics require some sort of mental store that holds a finite set of words. In humans, this lexicon of words is not completely static as we constantly learn new words and forget old ones (Saeed pp 10, 2003). It is clear though, that at any given time, at least for our native language, we hold thousands of words in our lexicon, all referring to entities in the physical or abstract world.

An important aspect of natural language is the fact that a finite set of words can be combined in a virtually infinite number of ways. Humans do this every day as our communication is based on the production of sentences we have never uttered or heard before. The term for this sort of linguistic production is generative grammar, which was invented and popularized in the 1960s by the prominent American linguist and philosopher Noam Chomsky. The generative grammar is built on the conviction that humans possess a language modality that makes use of recursive rules to form new sentences. This must be true because there is no way we could store all different word combinations in a semantic mental lexicon, all at one time. Instead, new sentences are improvised on the fly, and interpreted by the listener on the fly, according the set of rules that the generative grammar describes. A sentence that is correctly formed according to the generative grammar of a certain language is said to be *compositional* (Saeed, 2003). This implies that the overall meaning of the sentence is determined by the meaning of its decomposing parts and the way in which they are combined.

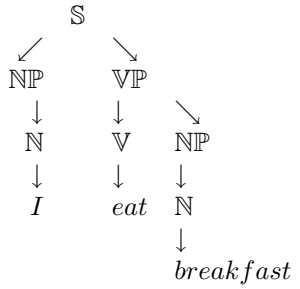
2.2 Syntactic formalisms

Syntactic formalisms are theories about how the composition of language can be described in a universal way. Chomsky's groundbreaking book *Syntactic Structures* (1957) altered the field of linguistics completely. At that time, the dominant theories described how mere words of a language could be combined in a veritable manner. Chomsky addressed the question of composition in a different way, forming a formal high-level language.

2.2.1 Constituent-based grammars

Chomsky's theories modeled composition on the basis of a tree structure. The representation starts in a root node, the so called sentence node. From the sentence node a number of sub-nodes, or leaves, unfold. These are described by the lexical category, or part of speech (POS), that the particular phrase consists of, for instance a noun phrase (NP), a verb phrase (VP), a prepositional phrase (PP) or a subordinate clause (SC) (Jackendoff pp 74, 1994). These second-level leaves can in turn be branched out into even more sub-leaves representing

Table 1: *A constituent-based tree representation*



either another POS phrase or just a POS, if at the end of the tree. Finally, when the whole syntactic structure of the sentence has been modeled, each word is projected under its corresponding POS.

By creating models using the constituent based grammar, linguists can define the syntax of a language in a formal way. This constituent based grammar makes it possible to state simple rules of what is a well formed sentence or phrase, and what isn't. Such a model of a sentence can look like

S → NP, VP

NP → N

VP → V, NP

which models for instance the short sentence *I eat breakfast*. The syntactic tree describing the constituting parts of the formal representation can be seen in Table 1.

When describing more complex sentences, the model might need additional rules, like the following set.

S → NP, VP

NP → Art, N

VP → V, NP

NP → Art, A, N

NP → Art, A, N, PP

PP → P, NP

These rules make up a formal representation of for instance the sentence *My mother is a respectful woman with good beliefs.* Of course, these rules are able to describe shorter sentences, but the formalism also offers a way to describe recursion in a language, which implies there is no definite limit to how many phrases a sentence can consist of. For instance, the sentence

> *Harry said that Amy thinks that Sam predicted that Mildred would believe that Beth is a genius* (Jackendoff pp 74, 1994)

is a perfectly correct syntactic sentence, which is modeled using only the following rules.

S→NP, VP

NP→N

VP→V, SC

SC→THAT, S

In this case, the word *that* constitutes the bridge that connects the current phrase to the next. Notice how the sentence starts all over again when the sub-ordinate clause is being used.

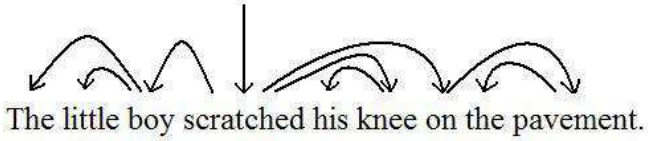
The most prominent type of products that syntactic formalisms have given birth to are applications like spelling and grammar checkers.

2.2.2 Dependency-based grammars

Whereas the traditional constituent-based parsing method can be useful when trying to formalize the syntax of a language, dependency parsing presents a different approach, taking into account the various semantic roles of the sentence. The two formalisms have been compared individually in a study by Johansson and Nugues (2007), where they obtained similar results. However, there is reason to believe that a semantic classifier based on the conjunction of constituent-based grammars and dependency-based grammars would be able to perform better, thanks to the more semantic-oriented approach. This standpoint has been augmented by Hacioglu (2004), who with a constituent-/dependency-based semantic classifier was able to quite easily outperform the then state-of-the-art scores of the constituent-based semantic role labelers submitted to the CoNLL shared task conference of that year.

When parsing a sentence semantically, the goal is to sort out the semantic dependencies of the sentence. A typical semantic dependency is the bond between a certain verb and the subject of a sentence, postulating the action and the entity that performs the action. Another semantic bond is that between the verb and the object of the sentence, giving information about what entity is affected by the action. Words describing another word, i.e adjectives or attributes, also make up a semantic dependency, as do conjunctions, modal verbs and the like. A typical representation of a dependency tree can be seen in Figure 1.

Figure 1: A dependency tree



When examining Figure 1, the reader notices how dependency tree grammars rank the lexical entities of the sentence by semantic significance. Thus, if more detail of an entity is desired, it's found only one step down the dependency ladder. This feature is an important reason to why incorporating dependency-based grammars in the semantic classifier is a sound decision. The semantic detail can easily be adjusted by including or excluding levels of dependency sub-trees. If a shallow semantic classifier is to be implemented, we settle for the closest dependencies from the predicate verb, while if more detail is preferred, additional dependencies might be included.

2.3 Predicate logic

Predicate logic, or first-order logic, is a formal way to describe the action taking place in a sentence. The syntactic elements used to make this sort of description are symbols representing objects and relations. An object is any entity that in some way is being affected by a relation, and that relation might in turn be affected by yet another relation. Distinctive relations, i.e relations that has only one possible context, are sometimes referred to as propositions. These, as well as other relations, can consist of verbs, nouns, adjectives or prepositions. The number of objects taken as arguments by the predicate is called the *arity* of the relation (Russell and Norvig pp 246, 2003). This number is fixed for propositions, as that sort of relation requires a predefined number of arguments in order to be logically coherent. Consider for instance the proposition *Father* and the arguments *George H.W. Bush* and *George W. Bush* implying that the former is the father of the latter. In predicate logic terms, this is expressed *Father(George H.W. Bush, George W. Bush)*. A predicate verb relation that holds several arguments is the sentence

Chris bought a bike from Stephen for £100

which is modeled as *Bought(Chris, a bike, Stephen, £100)*. In this case, as opposed to the case of propositions, the arity is non-fixed, but in order to have a significant semantic meaning, at least the first two arguments are needed. The first argument of any type of semantic relation is generally referred to as argument 0 (A0), the second as argument 1 (A1) and so forth.

The aspired objective of the final semantic classifier was defined as to a certain extent try to model the semantics of a sentence, using the described predicate logic architecture.

2.4 Machine learning

The adopted method for the model generation was machine learning techniques, which is a large sub field of artificial intelligence. Machine learning techniques are not only used in natural language processing, but also in numerous other applications ranging from search engines, medical diagnosis, computerized object recognition and robot locomotion. The machine learning used for this project was an inductive learning method, where data sets were used to detect general

patterns and draw grammatical conclusions, based on the semantic information of the VAT related texts. The final decision of which classifier to use for the model generation fell on the J48 tree classifier. It presented the best overall performance with respect to memory requirements, processing speed, size of output model and the self-evidently important classification score.

3 Research and progression

3.1 PropBank

PropBank is a large online database of semantic roles where verbs have been manually annotated in English. These verbs, called propositions, are predicates which are connected to arguments in an example-based fashion, forming predicate-argument relations. Each lexical example of a verb may hold various semantic interpretations, and these are defined by suffixes *.01*, *.02* and the like, in order to distinguish them from one another. The most general sense of the verb is assigned the lowest suffix and the rest of the senses are ranked accordingly. The arguments of each verb sense are assigned to *Arg0*, *Arg1*, *Arg2*, *Arg3* and possibly *Arg4* and *Arg5* where each argument number constitutes a certain semantic role of the argument that is connected to the verb. Generally, for instance, *Arg0* defines the semantic role of a subject or agent, and *Arg1* the object or patient of the current predicate.

3.2 The annual CoNLL competition

The research field of NLP has been refined and improved by shared competitions like the CoNLL conference, running annually since 1997. Each year one or several shared tasks, based on recent research, are presented. Participants are invited to implement their own solutions, which are evaluated and ranked.

In the year of 2008, the conference was held in Manchester, UK (Johansson, Màrquez, Meyers, Nivre & Surdeanu, 2008). The presented task was to implement semantic role labelers built on a representation of both constituent and dependency based grammars, as introduced with promising results by Hacioglu (2004). This was the motivation behind the decision to pursue a similar type of representation to be used as backbone architecture for the semantic classifier described in this report.

4 Task definition

4.1 Objective

The objective of the thesis was to construct a multilingual model for automatic semantic extraction from random texts concerning businesses' Value Added Tax (VAT) obligations. The requested languages were Danish, Swedish and English and the preferred output was of predicate logic structure.

When capturing the semantics of a sentence, the crucial phenomena to detect and classify are the action that the main verb states, as well as the participating entities, the arguments. These are commonly, but not exclusively, the subject and object tied to that verb. As mentioned earlier, the main verb is generally referred to as the predicate of the sentence, so the goal for each sentence is to pinpoint the predicate-argument structure. Modifying arguments might also exist, like for instance a negation flipping the meaning of the statement. Other examples include the modal verbs *should* and *must*, which give information about the importance that the main verb entails. The success rate of detecting such modifying arguments is almost equally crucial to detecting the overall predicate-argument structure.

4.2 Possible future application

A future goal, for which this project was requested, is the aspirations to use the presented methodology as the architecture behind an artificial text scanner. Such a text scanner could be used to automatically extract semantic information from VAT legislation documents, lifting a substantial burden off the backs of accountants currently reading these texts manually. Nielsen, Simonsen and Larsen (2007) have described a way to build a model of Danish VAT rules using the Web Ontology Language (OWL) editor Protégé-OWL and description logics. The idea is that the output of the semantic classifier, the predicate-argument representations of the sentences, could be ported to description logic expressions which in turn form an ontology, automatically building up a dynamic database consisting of up-to-date VAT laws. The VAT law database could subsequently be used as backbone to a computer program that helps business personnel make the right decisions concerning VAT when dealing with various types of commercial transactions.

5 Method

5.1 Overall methodology

The project was carried out using a statistical inductive machine-learning method. Rules and patterns were extracted from a corpus, a text prepared with various linguistic information. Already existing semantic corpora tend to be too general, since capturing every semantic element of a sentence calls for extreme elaboration and is usually the work of several linguists over many years. Thus, using old corpora for this task would produce insufficient results in the specific case of VAT. Therefore, completely new corpora was built, one for each of the three languages.

Each word in the texts was processed by the means of POS taggers and dependency parsers, generating information about base forms (lemmas), word classes (part-of-speech, POS), role dependencies and grammatical functions. Also, manual semantic annotation was applied to the chosen semantic elements. Included for each sentence were the predicates and their arguments. The semantic scope was reduced to addressing the particular nomenclature of VAT legislation. Verbs considered were those regarding the semantic frames buy, sell, tax, charge, deliver, send, provide, market, pay, account and issue, and their equivalents in Danish and Swedish. Theme modifying arguments like negation, manner, time, phrase clauses and the like, were also annotated.

To detect certain linguistic patterns in the texts, features were extracted from the corpus. These features were then passed on to a data mining program, which generated the models. The feature set was chosen on the basis of studies by Johansson and Nugues (2007) and determined for instance whether a word is a member of a sentence written in active or passive tense (the voice), the semantic parent or child(ren) of that word, the lemma or the POS of the current word.

5.2 Choosing VAT texts

Producing an efficient semantic dependency parser unfortunately isn't something that can be done solely by automation. It requires quite extensive linguistic preparatory work, in the sense that a corpus has to be built based on manual annotation. In order to produce a semantic dependency parser capable of parsing sentences concerning VAT legislation, texts in the specific domain of VAT first need to be acquired. These texts will work as a platform for the three corpora, forming a boundary limiting the annotation workload.

The main focus was to obtain texts that to a great extent mimicked the texts that the parser, once finished, would be able to process. Therefore, it was crucial that the texts contained a lot of examples of commercial nature, where for instance somebody would sell or buy something from another for a certain price and reason. Sentences containing VAT handling guidelines or legal paragraphs were also favorable, since they represent the main target of the final product. Hence, tax authorities in Denmark, Sweden and England were contacted, in

Table 2: *VAT texts chosen as platforms on which to build the corpora of each language*

Danish	Swedish	English
Moms - fakturering, regnskab mv.	Moms vid utrikeshandel	The VAT guide
Moms ved EU-varehandel		
På vej mod egen virksomhed		

order to obtain a set of suitable texts to start off from. The final decision fell on three texts for Danish, one for Swedish and one for English. The three Danish texts were subsequently merged into one single corpus. The final word count of the texts was approximately 34000 for the Danish text, 28000 for the Swedish text and 45000 for the English text. Mutual for the selected texts were the fact that they *did not* hold the force of law, but were to be regarded merely as guidelines in the domain of VAT, addressed primarily to small businesses in a case-by-case fashion. Nevertheless, being guidelines issued by tax authorities in each respective country, a high degree of trustworthiness can be expected.

5.3 The annotation process

The platform building the foundation of a semantic dependency parser is its semantic annotation. These annotations constitutes the linguistic examples from which the machine learning algorithms draws a substantial part of its input. From it, it builds the statistical framework that the model will use to identify and classify each word in the text.

The annotation for the corpora, as indicated earlier, was carried out manually. For this reason, and for the fact that providing clear statistics takes a lot of annotation, this was the single event posing the heaviest workload on the project. Going through each text sentence by sentence, all in all 1590 predicates and their argument were annotated, distributed to 585 English, 565 Danish and 430 Swedish. A few semantic annotation tools were briefly tested in order to facilitate the annotation, but they all turned out quite useless. For this reason, the annotation was conducted entirely by hand.

It is very important to make sure that the statistics derived from the annotation data is as coherent as possible, otherwise there is a risk that the final model will contain contradictions, confusing the semantic dependency parser. Self-evidently, this will lead to a less capable end product. Therefore when annotating, having consistency in mind is a key factor. An annotation chart or template is useful to minimize the risk of unconsciously changing annotation rationale in the midst of the process. See section Appendices for the complete annotation template that was used throughout the course of the annotation process.

The annotation process was based on the rationale of PropBank, focusing on predicate verbs as shown in the table. Using XML as mark-up language, the relevant predicates were identified and assigned suitable labels. Once the

predicate had been dealt with, its particular arguments were identified and marked in a similar manner. Finally, any modifiers relevant to the meaning of the sentence were annotated.

```
<arg arg0=”provider” id="a0021">We</arg>are committed to
<target type="supply" id="t0011">providing</target>
<arg arg2=”benefactive” id="a0022">newly registered
businesses</arg>with<arg arg1=”goods-services" id="a0023">
the option(s) of their choice</arg>
<arg argM=”timeframe” id="a0024">within three
months of<target type="receiving" id="t0012">
receiving</target>
<arg arg1=”thing_received-request" id="a0025">
their request(s)</arg></arg>.
```

As can be noticed in the example, some arguments were annotated in a hierarchical structure, so that a general description was followed by a more specific one (as in *goods*→ *services* and *thing_received*→ *request*). This procedure was used in order to provide better tagging accuracy to the subsequent parser. For consistency, a template was used for these hierarchies as well.

Subsequently, the annotation files were altered using a DTD and a Java program. The reason for this was that a more convenient structure was desired, one that would enable more linguistic data to be added to each word without giving up on readability. Therefore, the format was transformed into a matrix, were each row corresponded to a new word and each column corresponded to the different linguistic data, with which the corpus was to be deployed. The annotation data was put in the right-most columns, were the first column held the predicates only, the second column held for each sentence the arguments of the first predicate of that sentence, the third column held the arguments of the second predicate of that sentence, and so on.

5.4 Tokenization and structuring

The input text was tokenized and organized using a tokenization script written in Perl. Each word was separated from any punctuation character (colons, commas, full stops, quotes, exclamation marks, question marks etc.) and put on a new line. The matrix-like organization creates a lucid structure were each linguistic feature later is to be given its own column. This structuring is compliant to the one used by participants of the CoNLL shared task of 2008 (Johansson et al., 2008). However, for sake of simplicity, hyphenated words were not split as in CoNLL-2008.

Tokenization of the input text is very crucial for the outcome of the corpus. All subsequent stages of processing depend on the tokenization. The linguistic tools that later will be applied to the corpus need properly separated words and sentences, otherwise they will produce significant errors. Even though the Perl

script being used is capable of normalizing most of the text, some of the errors are hard to get by. It is for instance important that each sentence ends with a punctuation character and an empty line. Headlines usually don't end with a punctuation and are in addition difficult to cover automatically. Therefore, these punctuations were applied manually. Other examples include bulleted lists that were adjusted to natural sentences, and unidentified characters that were deleted or accordingly exchanged.

5.5 Part-of-speech tagging

POS tagging is used to obtain grammatical information about the words in a text. The basic objective for POS taggers is to decide the word class of each word, but the tagging can also be more fine-grained, for instance by determining the case of the word (i.e genitive or nominative). Furthermore, different POS taggers use different tag sets, complicating the compatibility between different linguistic corpus-processing tools. The tag set of the POS tagger depends on the tag set that was used for the treebank corpus that the tagger was trained on. Traditionally, the POS seen in Table 3 are determined and tagged.

Table 3: *Part-of-speech classes traditionally classified*

Verb	Noun
Adjective	Adverb
Conjunction	Interjection
Preposition	Pronoun

5.5.1 Swedish

The Granska POS tagger of KTH, Stockholm, implemented especially for the Swedish language, was the ideal choice for the Swedish POS tagging. Not only does it provide good POS tagging results, it also includes lemmatization. The POS tagging of the Swedish corpus was performed in Unix environment under Windows, running the Unix terminal emulator Cygwin. The following query was used to execute the tagger.

```
> ./tagg -NBIa [input_text] > [output_text]
```

The flag -NBIa states that N; interpret new line to indicate end of sentence or paragraph, B; print lemma, I; don't print words' info and a; tag new words ambiguously. The tagger comes with numerous other options that can be included via flags, but none of them were of gain to the POS/lemmatization task. For the conveyance to the dependency parsing next to be applied, the output format of the POS tagger was eventually adjusted using a Perl script.

5.5.2 Danish and English

The POS tagging of the Danish and English corpora was performed using the Stanford NLP Group Part-of-Speech tagger, version 1.5.1. It can be downloaded

free of charge from the web page of the Stanford NLP Group and comes with pre-trained models for English, Arabic, German and Chinese. For English, the Penn Treebank tag set is used, which is convenient, since the dependency parsing to follow go by the same tag set.

Unlike the Swedish Granska POS tagger, the Stanford POS tagger is highly trainable which means that it can handle every language, as long as the proper training data is provided. Training, however, takes memory, so the authors of the POS tagger recommend that at least one gigabyte is used for this task.

Since the tagger didn't include a Danish POS model, the first step was to find a suitable Danish training corpus. The CoNLL conference held in 2006 distributed, at the time of writing, the training data used by participants of that year's shared task. A modified version of the Danish Dependency Treebank (DDT), originally stemming from the PAROLE corpus, was selected. The data is considered open source and has been adjusted to the standards of CoNLL, making it good choice for this project. However, as the POS tagger didn't entirely accommodate the format of the DDT, a Perl script was written in order to discriminate the word and the POS tag from the remaining linguistic data.

The training itself was executed using the following query.

```
>java -mx1g -classpath postagger-2006-05-21.jar
edu.stanford.nlp.tagger.maxent.Train -prop [properties_file]
-model danish_ddt_trained_bidirectional -file
[training_corpus]
```

The query states that one gigabyte of memory is being used (-mx1g) and that the Java program is to be executed in training mode using a properties file and a training corpus, producing model files with the prefix *danish_ddt_trained_bidirectional*. Via the properties file, also used when tagging, numerous options can be modified. One of the adjustable training options is the architecture of the generated model, which is by default set to *bidirectional*.

With both an English and a Danish POS tagging model available, the final step was the actual tagging of the two corpora. It was done using the following line.

```
>java -mx1g -classpath postagger-2006-05-21.jar
edu.stanford.nlp.tagger.maxent.Test -prop
properties.txt -model [model prefix] -file [output_file]
```

Prior to execution, the tokenization option of the properties file was changed to *false*. The text was already tokenized and it was very important not to alter the alignment of each row. Otherwise, the task of reconnecting the tagged and parsed corpus with the manual annotation would be immensely and unnecessarily complicated.

5.6 Lemmatization

Lemmatization is applied to the corpus in order to group morphological variants of a word together under one single headword. For example, the words *am*, *was*, *are*, *is*, *were*, and *been* are all morphological variants of the base form *be*. Other examples are *laughing*, *laugh*, *laughs*, *laughed* and the base form *laugh* or *serpent*, *serpents* with *serpent* as lemma. The reason lemmatization is generally used in various NLP techniques is that the linguistic patterns of the sentences more accurately can be tracked once the lexical complexity has been reduced. The lemmatization process for this project was conducted using freely distributed lemmatization tools.

5.6.1 Swedish

The Swedish corpus was lemmatized in association with the POS tagging, using the Granska POS tagger. For details, see section 5.5.1.

5.6.2 Danish and English

For Danish and English, the downloadable command line CST lemmatizer of Copenhagen University was used for lemmatization, and the free online demo version of the lemmatizer was used to create a frequency lexicon. This frequency lexicon was subsequently modified into a look-up lexicon to be used when lemmatizing the actual corpora.

In addition to Danish and English, the lemmatizer supports Icelandic, German and Dutch. However, while the distribution of the lemmatizer is free of charge, the various word lists that are needed to generate the lemmas are distributed under license. The prices of these word lists varies depending on who is the intended user. For example, the Danish word list called STO, with more than 80000 unique words and their morphological forms, costs $15000 for companies acquiring the lemmatizer for commercial purposes, and $100 for governmental institutes acquiring it for sake of research. Given that the project described in this report was financially unsupported, a workaround strategy had to be implemented, which incorporated both the command line CST lemmatizer and the online CST demo version.

The Center of Speech Technology of Copenhagen University provides free online demo versions of their various NLP tools, which for Danish includes the STO word list. Since the tools are only demos, the amount of data that can be processed at any one time is limited. However, a password to unlock the demos can be acquired free of charge, enabling the user to run much larger texts.

Prior to lemmatization, the online lemmatizer automatically tokenizes the text, which was highly undesired in this matter. The input text was already tokenized and if this was tinkered with, the alignment to the semantic annotation would be unsynchronized, which would require extensive manual work at a later stage. For this reason, the online lemmatizer could only be used to produce a lemma frequency lexicon. Since the frequency lexicon was generated from the exact words of the corpus, it therefore contained every word in the corpus - the

perfect substitute for the too expensive word list described earlier. Hence, the command line version of CST could be used in conjunction with a customized free look-up lexicon based on the non-free word list.

The frequency lexicon consisted of four columns and a row for each unique word in the corpus, sorted by quantity. A Perl-script was written to turn the frequency list into a look-up corpus consisting of a word column and a lemma column sorted alphabetically. These two files were used with the lemmatizer to produce a binary dictionary.

The command line query

```
> cstlemma.exe -D -cFB -N <freq file> -nNFNB -i
<look-up corpus> -o dictionary.txt
```

executes the lemmatizer in create binary dictionary mode (*-D*). The look-up corpus format, which coincides with the subsequent output format of the lemmatizer when run in lemmatization mode, was set to full form followed by the base form, the lemma (*-cFB*), separated by a tab. The structure of the frequency file was regulated by the flag *-nNFNB* and the dictionary output name was set by *-o dictionary.txt*.

As a last preliminary step, an empty inflection pattern file, a flex file, was created. Inflection rules in linguistics describe the lexical morphology of a language. To clarify, consider the word *boys*. It can be divided into the word stem *boy* (the lexeme), which is a noun, and the functional suffix *s*, forming plural. Thus, the inflection rule for plural for nouns of English is NOUN-*s* (in general). A flex file is very useful when the lemmatizer cannot find a certain word in its look-up dictionary. In such a case, the lemmatizer would turn to the flex file in order to make an informed estimation. However, since the binary dictionary was generated on the basis of a look-up corpus containing *every single word* of the corpus being lemmatized, there wasn't a need for a flex file. The lemmatizer requires a flex file by default, but there is no need for it to contain any information.

Once an empty flex file had been created, the preparatory work was concluded and the actual lemmatization could commence. The query

```
>cstlemma.exe -L -c$w\t$b1[[$b?]~1$B]\t$t\n -i
<input file> -I$w\t$t\n -o <output file> -d
<binary dictionary> -f <flex file> -t
```

executes the lemmatizer in lemmatization mode -L. The format string -c ($w\t$b1[[$b?]~1$B]\t$t\n) states that the generated output format is set to word ($w), tab (\t), lemma ($b1, [[$b?]~1$B] = check binary dictionary first, if lemma not found, use flex file), tab (\t), POS tag ($t) and new line (\n). -I tells the lemmatizer that the structure of the input file is word, tab, POS tag and new line.

5.7 Dependency parsing

The MaltParser of Växjö University (Hall, Nilsson & Nivre, 2006) was chosen for the dependency parsing of all three languages. The parser is data-driven and a model can easily be induced by training the parser on a tree bank corpus such as the SUC corpus. This makes the parser very versatile, since as long as the user is in possession of a suitable tree bank corpus, any language can be efficiently parsed using this dependency parser. Malt parser is distributed for free from the Växjö University website.

5.7.1 Swedish

Since the Granska POS-tagger was implemented using the SUC tag set, all subsequent steps in the processing chain had to follow the same sort of tags. However, the Swedish model bundled with the distribution of Malt parser was induced using the Swedish Talbanken corpus. The tag set of this tree bank differs from the tag set used by the SUC corpus. Hence, the preliminary step was to re-train the parser, so that the induced model would recognize the input tags. A training set of the SUC corpus was downloaded from the website of the 2006 CoNLL competition (CoNLL-X).

As was the case with the POS tagging, the dependency parsing was conducted in Unix environment. To initiate the parser in training mode, the following query was entered.

```
> java -mx1g -jar malt.jar -c [model] -i [training_corpus]
-m learn
```

Using this query, the parser will produce a model called model.mco from the provided training corpus. Note that it is very important that the training corpus doesn't contain sentences longer than 300 words. If so, an error due to insufficient memory will occur. Stunningly, though distributed for the sole purpose of training, the training set provided by CoNLL-X exhibited such an error.

Once a model using the SUC tag set had been induced, the following step, of course, was to use the model to perform the dependency parsing. The line

```
>java -mx1g -jar malt.jar -c [model] -i [input] -o [output]
-m parse
```

will invoke the parser in parsing mode resulting in an output that can be seen in Table 5.

Table 4: *The appearance of the Swedish corpus after the POS tagging and the dependency parsing had been applied*

Nr	W	L	POS	POS	INFO (omitted)	DEP	FUNC
1	Om	om	PP	PP	_	3	adv
2	du	du	PN	PN	_	1	pr
3	handlar	handla	VB	VB	_	0	ROOT
4	med	med	PP	PP	_	3	adv
5	andra	annan	JJ	JJ	_	6	det
6	länder	land	NN	NN	_	4	pr
7	.	.	MAD	MAD	_	3	ip

5.7.2 Danish

A model to parse Danish texts is not provided for in the Malt-parser bundle. However, since a Danish tree bank corpus (the DDT corpus) had already been acquired for the previous POS tagging task, that same tree bank could be used to induce a dependency parsing model. The methodology for training and parsing described earlier was then applied in order to complete the Danish VAT corpus.

5.7.3 English

English is one of the two languages that Malt-parser supports straight away. The provided model has been trained on the Penn tree bank, which is compliant with the POS tagger that was previously used. Thus for English, no additional training had to take place prior to parsing.

5.8 Feature extraction

Having completed the corpora of the three languages, features could now be derived from the grammatical data. Features are utilized to prepare an argument/relation format file (.arff) that machine learning algorithms later will induce models of semantics from. This way, distinctive phenomena occurring in the VAT texts can be captured, which will guide the parser when classifying the various semantic objects.

The methodology and the features that were chosen for this task were based on research by Johansson and Nugues (2007), who implemented two different semantic analyzers to evaluate how the constituent based grammar approach compared to the dependency based approach. The analyzers were implemented in four steps, where the first step was responsible for target identification and the second step classified the targets as members of a frame category. After that, the third step handled the identification of arguments of each of the targets, and the fourth and final step classified each found argument to its most suitable semantic frame.

Since the feature extraction was conducted by implementing a Java program, virtually any conceivable feature could be selected. When choosing features to

implement, one strive for maximum gain while trying to maintain computational simplicity, and try to avoid less rational features. Thus, having a look at the corpora to detect patterns and trying to evaluate, by means of rational thinking, which features could improve each of the four steps, is a good way to go. This way, some of the most influential features were implemented, for instance the argumentTargetFrame feature of the argument classification, the targetSentence feature of the argument identification and the targetBoolean feature of the target classification. All the mentioned features were implemented to bridge the gap between the four steps, so that the conclusion of the previous step wasn't lost, but used to improve the performance of the next step. Astonishingly, the targetSentence feature gave a 25% boost to the argument identification.

5.9 Model generation

The machine learning procedure was carried out using Weka, a Java based freeware from the University of Waikato, New Zeeland. It incorporates a number of various machine learning algorithms, based on different architectures, like for instance tree based and rule based decisioning. Weka also includes filters that can be used to quickly remove, add or in other ways manipulate attributes and instances of the input data. Supervised filters automatically choose for example which features of a data set have the most impact on the performance, while unsupervised filters enables completely manual manipulation, i.e. removing all instances matching a given regular expression. Thanks to this, the user can easily conduct a quick personal research to discriminate poorly performing attributes, and thus enhance the overall performance of the classifier.

First and foremost, the architecture of the classifier had to be determined. This was done by evaluating what type of classifier gave the best results, while at the same time remaining as manageable as possible.

Support Vector Machines (SVMs) were one of the two architectures evaluated, particularly the heuristic of Sequential Minimal Optimization (SMO). SMO is a way to train SVMs using, as the data set grows, a linear amount of memory. Even so, for the data sets used in this project, SMO was generally found to require more memory than the 1.6 Gb available, forcing Weka to crash. This was unfortunate because SMO proved to perform better than all other evaluated classifiers, if the number of attributes and instances was reduced. Yet another drawback, however, was that the output model of SMO functions often ended up very large, seldom less than a couple of hundred Mb in size. The generation times were also outrageous; up to an hour or more, with no definite guarantee of scoring better than less computationally complex architectures.

The second architecture that was evaluated was two different types of tree based architectures; J48 and Id3. Id3 was found completely inept when it came to classifying the somewhat unbalanced input data. J48 scored better, but still had difficulties handling unbalanced input data where the outcome was either true or false, as in the case of target and argument identification. When the model generation was performed on the complete data set, the ratio

of instances with the outcome false, as opposed to those with the outcome true, were approximately a hundred to one. This led the classifier to classify all instances as false. For this reason, the target identification data set was reduced to hold instances that were either a target or a lexical equivalent of any target. The same procedure was applied to the argument identification. Lexical equivalents could for instance be the noun *(the) purchase* which is distinct from the target verb *(to) purchase*. Reducing the data set, the target classification for each of the three languages was satisfactory, but the argument identification data set needed further reducements, in order to balance the outcome. Hence, one third to one fifth of the lexical equivalents were kept in the data set for the final model generation. The decision tree based J48 was selected as classifier of the final model on the premise of its combination of low generation times, small model sizes and good overall parsing performances.

Last, a feature evaluation was conducted, which led to the conclusion that all features except the position, voice and path features were to be kept in the data sets. This decision was based on the fact that the mentioned features provided little improvement, only about 1%. At the same time, unlike all other features, the position, voice and path features were derived from manually annotated semantic data, which would be unavailable to the final classifier up front. Thus, keeping these features would immensely complicate the model utilization, requiring features to be extracted on the fly as the semantic dependencies were discovered.

5.10 End classifier implementation

The four models; the target identification, target classification, argument identification and argument classification models for each of the three language were ultimately used in a single end program, the semantic classifier, designed to demonstrate the capabilities of the models. The program was implemented in Java, with the three corpora utilized as test input texts. Of course, the semantic annotation was cut from the corpora prior to demonstration. In order to initiate and use the models, a Java class called WekaGlue was incorporated. It was originally implemented by Richard Johansson of Lund University, who granted the permission to use it for this cause, but was slightly modified to accommodate for unknown input tokens.

On invoking the semantic classifier, the user was asked to decide which of the three languages he or she preferred to demonstrate. Subsequently, the relevant four models were initiated, as well as the relevant arff data sets. The data sets were used to set the possible attributes of an instance, to which the input word and its attributes would be mapped. In case one of the attributes of an input word was unrecognizable, that attribute was altered to "UKN", enabling the classifier to keep running regardless of input. The same features used to generate the models were then extracted from the input corpus and stored in memory, from which the data was brought up sentence by sentence. Iterating the sentence, each word was evaluated by the classifier to detect possible target words. Once the classifier responded true to a certain word, that word were then

classified by the target classification model to assign a category label. To benefit from the result of the previous classifier, each new step was preceded by adding the outcome of previous step to the attributes of that word, which thus copied the inter-locking features that were used when generating the models. Next, the sentence was iterated all over again, this time to look for possible arguments of the found target. When an argument was detected and inter-locking features had been added, the final argument classification model assigned the most likely label of that argument. Targets, arguments and modifiers were stored in a data structure that allowed quick and comprehensible access to each found instance. Eventually, the results were written to an output file containing each sentence and a proposed predicate-argument statement along with contingent modifiers, to some extent modeling the semantics of the sentence.

6 Results

The results of each of the three languages were evaluated based on the performance scores obtained from each of the four classification steps described earlier. Common for all the test sessions were the fact that they were carried out using 10-fold cross validation. Running this evaluation mode, the data set is split into ten equally sized portions. The classifier is subsequently trained on nine tenths of the data set, leaving the final tenth for testing. The procedure is repeated ten times until all the ten portions have acted as test set once. The chosen evaluation method is a fair and efficient way to work out a balanced performance score based on the complete data set.

The cross validation returns a set of different scores. Most notable is the F1 score, which is comprised of a harmonic mean of the precision and the recall score, where the best possible score is 1 and the worst is 0. The F1 score, as well as the precision and the recall scores, are listed for each classifier outcome. Thus, one can evaluate the score of each class independently. Furthermore, a confusion matrix gives detailed statistics about the number of instances of each specific class that was classified as members of what classes, correctly or incorrectly. This information can be used to draw conclusions about classification difficulties and how to correct errors in the input data. One might for instance find that the feature extractor is unable to handle certain linguistic scenarios in the text, thus generating misleading features.

6.1 Danish

6.1.1 Target identification

Table 5: *Danish target identification*

Summary		
Correctly classified instances	562	85.4103%
Incorrectly classified instances	96	14.5897%

Accuracy by class			
Precision	Recall	F1	Class
0	0	0	False
0.854	1	0.921	True

Confusion matrix		
a	b	←classified as
0	96	a = false
0	562	b = true

The F1 score of class *true* was 0.92 for the Danish target identification, but none of the 96 instances of the lexical equivalents of target words were actually correctly labeled.

6.1.2 Target classification

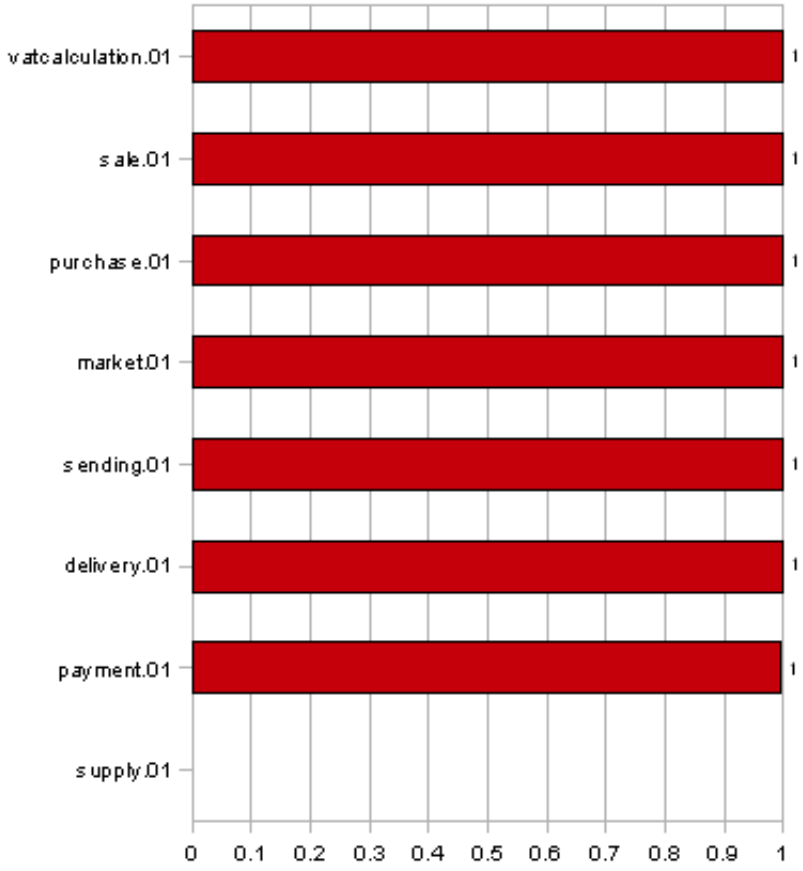
Table 6: *Danish target classification*

Summary		
Correctly classified instances	561	99.8221 %
Incorrectly classified instances	1	0.1779 %

Confusion matrix								
a	b	c	d	e	f	g	h	←classified as
8	0	0	0	0	0	0	0	a = market.01
0	23	0	0	0	0	0	0	b = sending.01
0	0	29	0	0	0	0	0	c = delivery.01
0	0	0	140	0	0	0	0	d = vatcalculation.01
0	0	0	0	144	0	0	0	e = sale.01
0	0	0	0	0	155	0	0	f = payment.01
0	0	0	0	0	0	62	0	g = purchase.01
0	0	0	0	0	1	0	0	h = supply.01

The one instance of *supply.01* was falsely labeled *payment.01*. All other instances were correctly classified, thus returning a near perfect overall result of 99.8%.

Figure 2: *Danish target classification (F1 scores)*



Only one instance of the Danish target classification data set was misclassified, yielding perfect F1 scores for all target words except *supply.01*.

6.1.3 Argument identification

Table 7: *Danish argument identification*

Summary		
Correctly classified instances	3385	82.1004 %
Incorrectly classified instances	738	17.8996 %

Accuracy by class			
Precision	Recall	F1	Class
0.957	0.741	0.835	False
0.699	0.948	0.804	True

Confusion matrix		
a	b	←classified as
1868	654	a = false
84	1517	b = true

Danish argument identification returned a good overall result of 82.1%, and the F1 scores for both classes were satisfying. The recall of the class *true* was a fair bit higher than for the class *false*, only misclassifying 84 out of 1517 (5%) compared to 654 out of 1868 (35%).

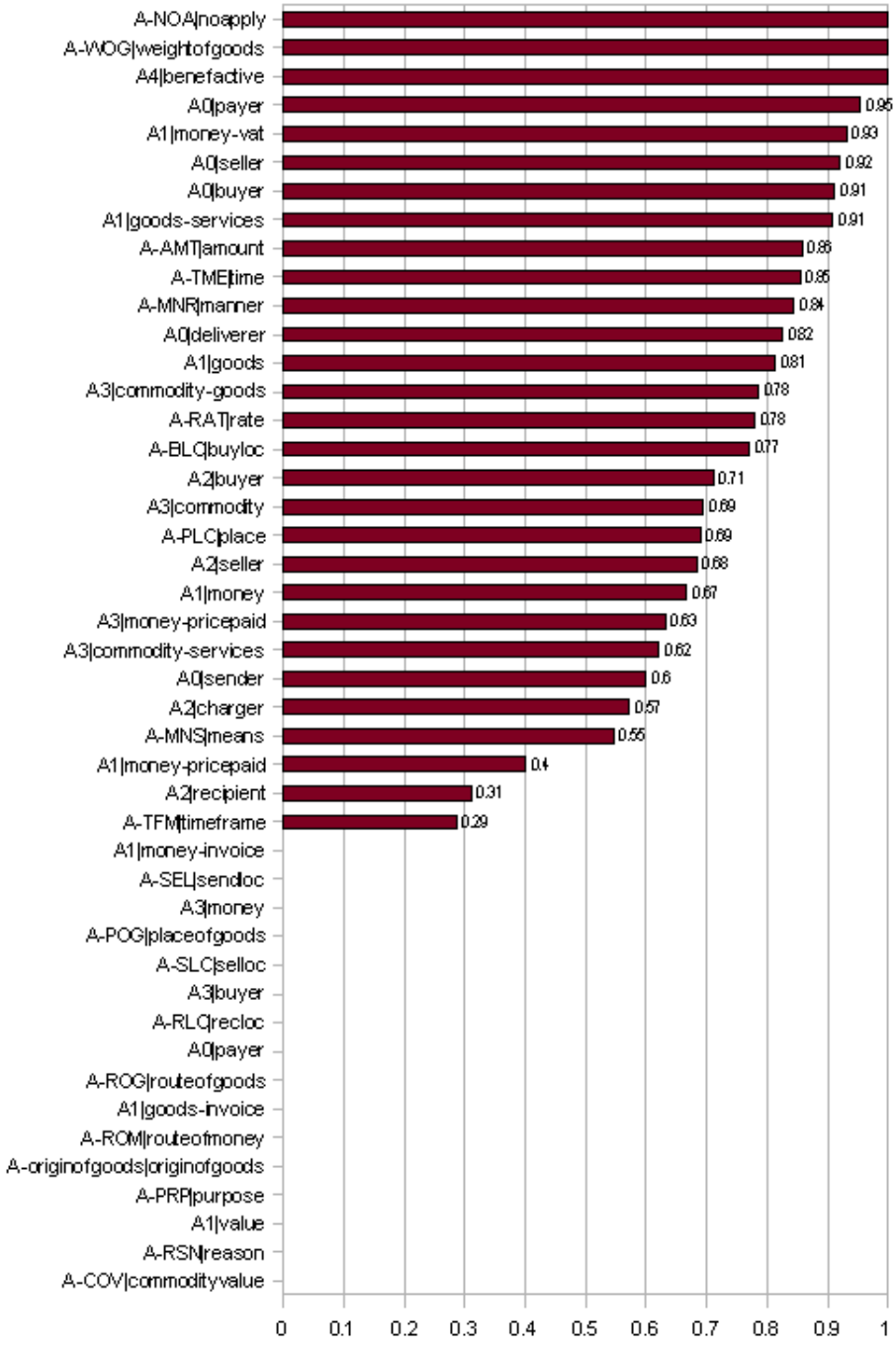
6.1.4 Argument classification

Table 8: *Danish argument classification*

Summary		
Correctly classified instances	1316	82.1986 %
Incorrectly classified instances	285	17.8014 %

The argument labeling of Danish was accurate overall. 82.2% of the total instances was correctly classified.

Figure 3: *Danish argument classification (F1 scores)*



The F1 scores of Danish classification showed that more than half of the instances were well classified. However, 16 different labels were never correctly classified, the likely explanation being that these labels were seldom seen in the text, thus drowning amongst more likely labels.

6.2 Swedish

6.2.1 Target identification

Table 9: *Swedish target identification*

Summary		
Correctly classified instances	429	98.6207 %
Incorrectly classified instances	6	1.3793 %

Accuracy by class			
Precision	Recall	F1	Class
0	0	0	False
0.986	1	0.993	True

Confusion matrix		
a	b	←classified as
0	6	a = false
0	429	b = true

The same pattern as of the Danish target identification was seen in the Swedish one. None of the *false* instances were correctly classified. A probable explanation is the unbalanced data set, as well as too few differences between instances of *false* and *true*.

6.2.2 Target classification

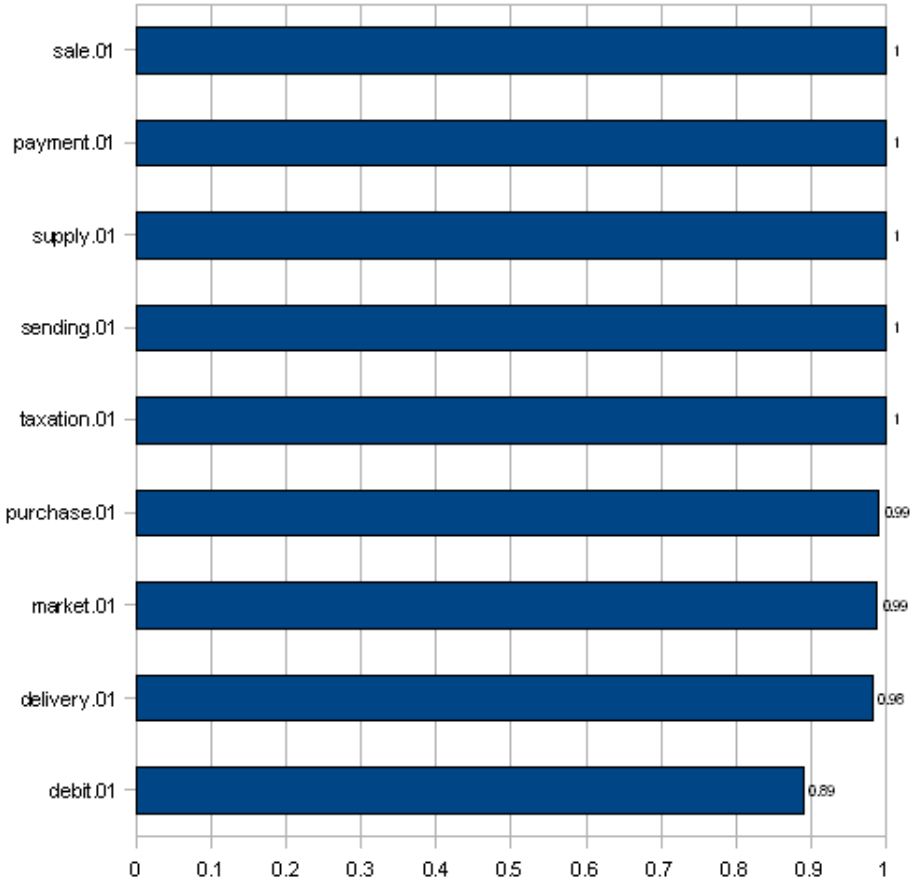
Table 10: *Swedish target classification*

Summary		
Correctly classified instances	426	99.3007 %
Incorrectly classified instances	3	0.6993 %

Confusion matrix									
a	b	c	d	e	f	g	h	i	←classified as
40	0	0	0	0	0	0	0	0	a = taxation.01
0	113	0	0	0	0	0	0	0	b = market.01
0	1	4	0	0	0	0	0	0	c = debit.01
0	0	0	21	0	0	0	0	0	d = sending.01
0	1	0	0	28	0	0	0	0	e = delivery.01
0	0	0	0	0	63	0	0	0	f = sale.01
0	0	0	0	0	0	74	0	0	g = payment.01
0	1	0	0	0	0	0	48	0	h = purchase.01
0	0	0	0	0	0	0	0	35	i = supply.01

The only errors of the target classification of Swedish were one instance each of *debit.01*, *delivery.01* and *purchase.01*, which were all misclassified as *market.01*. Thus, good F1 scores were obtained.

Figure 4: *Swedish target classification (F1 scores)*



6.2.3 Argument identification

Table 11: *Swedish argument identification*

Summary		
Correctly classified instances	3534	81.5601 %
Incorrectly classified instances	799	18.4399 %

Accuracy by class			
Precision	Recall	F1	Class
0.914	0.825	0.867	False
0.626	0.79	0.699	True

Confusion matrix		
a	b	←classified as
2607	553	a = false
246	927	b = true

The argument identification of Swedish obtained an F1 score of 0.7 for the class *true*. Overall, 81.6% of the total instances were correctly classified.

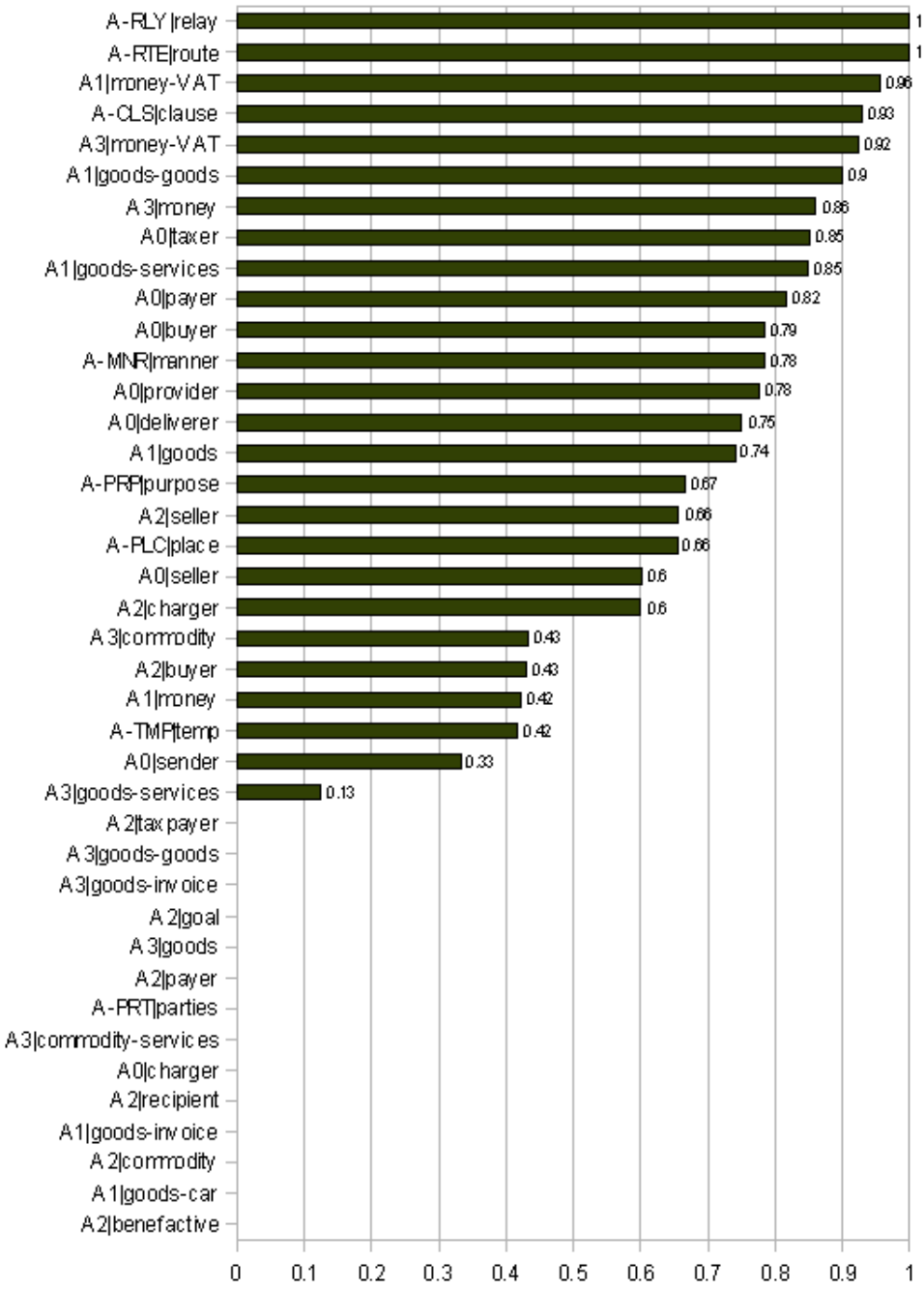
6.2.4 Argument classification

Table 12: *Swedish argument classification*

Summary		
Correctly classified instances	847	72.208 %
Incorrectly classified instances	326	27.792 %

The argument classification of Swedish was a bit worse than that of Danish, getting the label right in 72.2% of all cases.

Figure 5: *Swedish argument classification (F1 scores)*



Two labels, modifiers *A-RLY|relay* and *A-RTE|route*, obtained a perfect F1

score of 1. 14 labels were never correctly classified.

6.3 English

6.3.1 Target identification

Table 13: *English target identification*

Summary		
Correctly classified instances	1216	88.8889 %
Incorrectly classified instances	152	11.1111 %

Accuracy by class			
Precision	Recall	F1	Class
0.919	0.884	0.901	False
0.852	0.896	0.874	True

Confusion matrix		
a	b	←classified as
691	91	a = false
61	525	b = true

The F1 scores of both classes were very high, differentiating English to Swedish and Danish, where the *false* class received an F1 score of 0.

6.3.2 Target classification

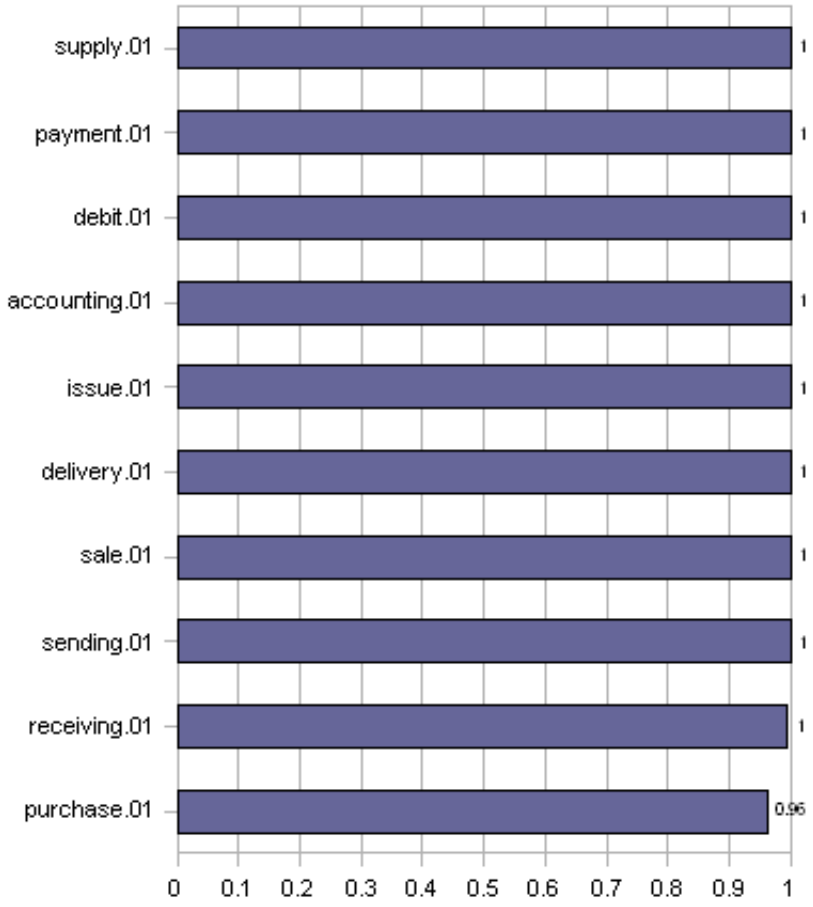
Table 14: *English target classification*

Summary		
Correctly classified instances	585	99.8294 %
Incorrectly classified instances	1	0.1706 %

Confusion matrix										
a	b	c	d	e	f	g	h	i	j	←classified as
81	0	0	0	0	0	0	0	0	0	a = issue.01
0	110	0	0	0	0	0	0	0	0	b = receiving.01
0	0	3	0	0	0	0	0	0	0	c = delivery.01
0	0	0	37	0	0	0	0	0	0	d = sending.01
0	0	0	0	65	0	0	0	0	0	e = debit.01
0	0	0	0	0	81	0	0	0	0	f = payment.01
0	0	0	0	0	0	29	0	0	0	g = sale.01
0	1	0	0	0	0	0	13	0	0	h = purchase.01
0	0	0	0	0	0	0	0	66	0	i = accounting.01
0	0	0	0	0	0	0	0	0	100	j = supply.01

The target classification of English was almost impeccable, only misclassifying one single instance of *purchase.01*.

Figure 6: *English target classification (F1 scores)*



6.3.3 Argument identification

Table 15: *English argument identification*

Summary		
Correctly classified instances	4175	86.2782 %
Incorrectly classified instances	664	13.7218 %

Accuracy by class			
Precision	Recall	F1	Class
0.927	0.874	0.9	False
0.735	0.835	0.782	True

Confusion matrix		
a	b	←classified as
2983	429	a = false
235	1192	b = true

The argument identification of English was better than that of Swedish, and on par with that of Danish, scoring an F1 of 0.9 for *false* and 0.78 for *true*.

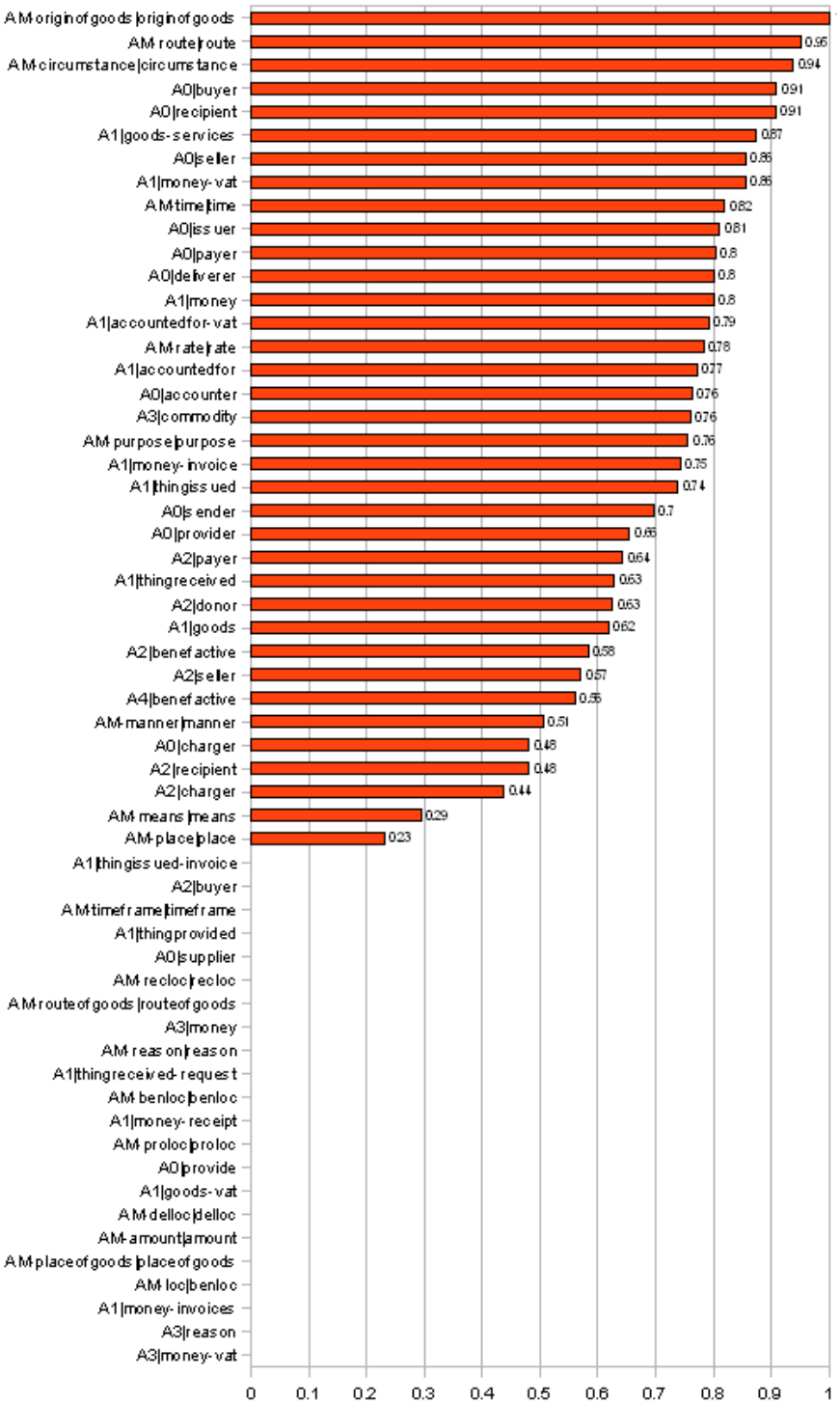
6.3.4 Argument classification

Table 16: *English argument classification*

Summary		
Correctly classified instances	1006	70.4975 %
Incorrectly classified instances	421	29.5025 %

Argument classification, however, was a little worse than that of Swedish, and even more worse than that of Danish. However, the number of different labels of English was also higher. Again, when turning to the individual F1 scores, one find that distinctive modifiers and common arguments score high.

Figure 7: *English argument classification (F1 scores)*



6.4 Probable explanation for certain parsing errors

When examining the target identification performance of Danish and Swedish, one finds that unlike the case of the English target identification, none of the non-target instances were correctly classified. Since the objective of the target identification models was to part target words from their lexical non-target equivalents, a likely explanation for this error is that the non-targets of Danish and Swedish simply are inseparable from the actual targets. Indeed, when examining the annotated corpora, one notices that all of the lexical equivalents were verbs. The lexical equivalents of English were generally nouns. This difference leads to the conclusion that the target identification errors derive from committed manual errors in the annotation.

6.5 Semantic classifier demonstration results

6.5.1 The good

Figure 8: The good

```
Sentence:
 Företagaren Sven i Sverige köper en programmeringstjänst för 100000 kr av datakonsulten Irene i
 Irland.
Predicate-argument conversion:
 köper[purchase.01](Sven[A0|buyer], programmeringstjänst[A1|goods-services], Irene[A2|seller], 100000[A3|money])
Modifiers:
 Sverige[A-PLC|place]

Sentence:
 Du køber en vare i Tyskland.
Predicate-argument conversion:
 køber[purchase.01](Du[A0|buyer], vare[A1|goods])
Modifiers:
 Tyskland[A-PLC|place]

Sentence:
 Du skal som udgangspunkt også momsregistreres, hvis du sælger tjenesteydelser.
Predicate-argument conversion:
 sælger[sale.01](Du[A0|seller], tjenesteydelser[A1|goods-services])
Modifiers:
 som[A-MNR|manner]
 hvis[A-TME|time]

Sentence:
 In the same way, VAT charged to you on your business purchases is
 your input tax.
Predicate-argument conversion:
 charged[debit.01](VAT[A1|money-vat], you[A2|payer], purchases[A3|commodity])
Modifiers:
 way[AM-manner|manner]

Sentence:
 How do I account for VAT on my supplies?
Predicate-argument conversion:
 account[accounting.01](I[A0|charger], VAT[A1|accountedfor-vat], supplies[A3|commodity])
```

Fairly short sentences containing a predicate and quite typical arguments were generally correctly parsed, including typical modifiers modifying the context of the sentence.

6.5.2 The fair

Figure 9: The fair

```
Sentence:
 Det är särskilt viktigt att du kan styrka exporten vid hämtköp, dvs i
 de fall du levererat en vara i Sverige eller i ett annat EG
 - land till en näringsidkare som själv för ut varan till ett land
 utanför EG.
Predicate-argument conversion:
 levererat[delivery.01](du[A0|deliverer], de[A1|goods], land[A2|goal])
Modifiers:
 Sverige[A-PLC|place]

Sentence:
 Det innebär att organisationerna måste betala moms på inköp i Sverige om varorna levereras
 i Sverige.
Predicate-argument conversion:
 betala[payment.01](moms[A1|money-VAT], inköp[A3|commodity])
Modifiers:
 Sverige[A-PLC|place]

Sentence:
 Du skal momsregistreres, så snart du regner med at sælge for mere end
 50000 kr inden for en 12-måneders periode.
Predicate-argument conversion:
 sælge[sale.01](Du[A0|seller], kr[A1|money], for[A3|commodity])
Modifiers:
 inden[A-TFM|timeframe]

Sentence:
 Du skal dog ikke momsregistreres, hvis du kun sælger ud af dine egne
 private ejendele.
Predicate-argument conversion:
 sælger[sale.01](Du[A0|seller], ejendele[A1|goods], private[A2|buyer])
Modifiers:
 hvis[A-TME|time]
 ud[A-ROG|routeofgoods]
 ikke[A-NOA|noapply]

Sentence:
 If you make taxable supplies of goods or services to a customer for which
 you are not paid, you may be able to reclaim relief from
 VAT on the bad debts.
Predicate-argument conversion:
 paid[payment.01](you[A0|payer], supplies[A1|goods], goods[A3|commodity])
```

In case the sentence comprised several probable arguments, the precision was a little bit weaker, but still acceptable.

6.5.3 The not so good

Figure 10: The not so good

```
Sentence:
 Det innebär att organisationerna måste betala moms på inköp i Sverige om varorna levereras
 i Sverige.
Predicate-argument conversion:
 betala[payment.01](, moms[A1|money-VAT], inköp[A3|commodity])
Modifiers:
 Sverige[A-PLC|place]

Predicate-argument conversion:
 levereras[delivery.01](, moms[A1|money-VAT], inköp[A3|commodity])
Modifiers:
 Sverige[A-PLC|place]

Sentence:
 Erhvervelsesmoms er betegnelsen for den moms, der skal beregnes og betales af momsregistrerede
 virksomheder, der køber varer i andre EU-lande.
Predicate-argument conversion:
 beregnes[vatcalculation.01](virksomheder[A0|payer], Erhvervelsesmoms[A1|money-vat], der[A3|commodity])
Modifiers:
 EU-lande[A-PLC|place]

Predicate-argument conversion:
 betales[payment.01](virksomheder[A0|payer], Erhvervelsesmoms[A1|money-vat], der[A3|commodity])
Modifiers:
 EU-lande[A-PLC|place]

Predicate-argument conversion:
 køber[purchase.01](, Erhvervelsesmoms[A1|money-vat], virksomheder[A2|seller], der[A3|commodity])
Modifiers:
 EU-lande[A-PLC|place]

Sentence:
 We are committed to providing newly registered businesses with the option ( s )
 of their choice within three months of receiving their request ( s ).
Predicate-argument conversion:
 providing[supply.01](We[A0|provider], option[A1|goods], businesses[A2|benefactive])
Modifiers:
 months[AM-timeframe|timeframe]

Predicate-argument conversion:
 receiving[receiving.01](We[A0|recipient], option[A1|goods], businesses[A4|benefactive])
Modifiers:
 months[AM-timeframe|timeframe]
```

A very difficult semantic case to capture was when the sentence included several predicates of the same kind but with different arguments. In this case the arguments generally were assigned the same labels, which was usually incorrect. Also, some of the A0s went undetected.

7 Constraints and proposed improvements

Of course, since the semantic classifier was only a pilot project carried out to investigate the feasibility of this architecture, there are a number of possible improvements that could be implemented to future semantic classifiers.

For example, anaphoric referencing is not supported in this version of the semantic classifier. An anaphoric reference is for instance the word *"it"*, referring to an already mentioned entity. The semantic meaning of *it* has generally been declared in a previous sentence, and thus, to improve fluency, that information can be omitted in the current sentence. This, however, means that a classifier designed to process the text sentence by sentence without the help of anaphoric referencing, is unable to catch the full semantics of the text. It will probably be capable of finding *it*, but that doesn't yield much information about the real semantics of that argument for the sentence being processed.

Also, to improve the semantic details that can be caught, a classifier equipped with a deeper hierarchy would be able to return more useful information. The current semantic classifier is implemented with a two-step semantic hierarchy, as in for instance *A1/goods-car*, where *goods* is the more general description and *car* is a more specific one. However, since that hierarchy is hard coded into the model, there is no real hierarchy classification going on. To improve this, the classification tags can be split up, and each level of detail is classified independently.

Furthermore, the feature set extracted from the corpora could be further expanded and enhanced, which would most certainly improve information gain. A feature that could be added is for instance the notion of transitive (i.e *kick*, *sell*, *drive*) and intransitive verbs (i.e *lie*, *stand*, *walk*). The former could, however not compelling, hold an object that is the receiver of the described act, whereas the latter never include an object, thus eliminating the possibility that an A1 is incorrectly added to the predicate-logic clause of the sentence. A project investigating the gain of each individual feature added to the feature set could prove substantial to the performance of a subsequent version of the semantic classifier.

The implementation also enables the possibility to add further semantic detail to each argument. The annotation of the arguments was attached to the dependency head of each word chain qualifying as an argument, but the algorithm of the semantic classifier currently only takes the head of an argument word chain into account. While the semantic information obtained from the head of an argument chain sometimes is sufficient, most noun arguments, for instance, would be better depicted if their determiners and adjectives were included. This could be accomplished by adding the dependency sub-tree of the head argument to the complete argument that is passed on to the predicate-argument model of the sentence.

Another possibly substantial improvement could be obtained if the dependency representation was used for yet another task. The current implementation of the semantic classifier algorithm iterates, once a target word has been found, the sentence from the first word to the last. This is an approach generally re-

ferred to as a *local model* (Haghighi, Manning & Toutanova, 2005), where the argument labels are assigned individually, without knowledge of the labels of other words or the dependencies between them. This produces a lot of errors as the first detected argument, suitable of constituting an A0, A1, A2 and so on, is permanently stored in its argument category as long as the sentence is still being processed. Thus, if an argument is captured too early, there is no way of rectifying it later on. The solution, however, could be to alter the iteration algorithm in a similar manner to that of a *joint model* (Haghighi et al., 2005). A joint model considers the label dependencies of words, and labels all words simultaneously. For Haghighi et al., this method reduced the number of errors of all arguments by 17%, while the number of errors of core arguments (A0, A1, A2 etc.) were reduced a staggering 37%, compared to the gold-standard parse trees of PropBank. This could prove to be a huge performance booster if implemented to the semantic classifier described in this report, especially regarding very long sentences and sentences with several targets.

Since the reattachment of the manual annotation corpus to the POS-tagged, dependency parsed and lemmatized corpus without the manual annotation was really time consuming and tedious due to unavoidable alignment problems, the semantic corpus completion would benefit from a chronological switch. Instead of first completing the annotation, detach it from the corpus, applying the automatic syntactic/dependency tools to the corpora, and finally reconnect it all, the alignment problems would have been avoided altogether if the manual annotation was performed and applied directly to the complete syntactic/dependence corpus.

Also, a final point to clarify is the fact that the annotation conducted in this project by no means sufficiently can support a large-scale semantic classifier in the field of VAT. The approximately ten different target types that were considered for each language are not even close to covering the full scope of VAT. However, the annotated targets of this project do show the potential of the overall architecture, which was the goal of this pilot study.

8 Conclusion

The objective of the project was to investigate the feasibility of a semantic classifier based on a joint representation of constituents and dependencies for the domain of VAT capable of transforming written text of three different languages into predicate-argument structures. This had never been done before. The results that were obtained are very promising in that all target predicates are almost unmistakeably distinguished, as well as some of their arguments and modifiers. The demonstration shows that for short sentences, the performance is very good, not seldom capturing the complete semantic statement. However, as the size of the sentence grows, so does the number of errors. To rectify this, a proposed modification to the algorithm of the semantic classifier could be implemented, so that arguments are parsed in order of dependency proximity from its target, instead of in order of appearance within the sentence. This would likely boost the performance when several target predicates are detected in the same sentence. The demonstration, however, also indicates the need of more complete corpora, enabling the capability of capturing even more semantic elements, thus modeling the full semantics with greater accuracy.

To sum it up, it is beyond question that the results obtained from this pilot study grant the feasibility of the described methodology. By implementing some of the proposed improvements and investigate even further enhancements, a semantic classifier, built around an architecture of machine learning models induced from a constituent- and dependency-based corpus complete with manually applied semantic annotations, could lead to very useful outcomes.

9 References

Andersen, Jesper; Elsborg, Ebbe; Henglein, Fritz; Simonsen, Jakob Grue and Stefansen, Christian (2003). *Compositional Specification of Commercial Contracts.*

Babko-Malaya, Olga (2005). *Propbank Annotation Guidelines.*

Bar-Hillel, Yehoshua (1960). *The present status of automatic translation of languages.* Advances in Computers 1, 91-163.

Chomsky, Noam (1957). *Syntactic Structures.* Mouton and Co.

Clark, Alexander (2003). *Machine Learning Approaches to Shallow Discourse Parsing: A Literature Review.*

Dalianis, Hercules and Jongejan, Bart (2006). *Hand-crafted versus Machine-learned Inflectional Rules: The Euroling-SiteSeeker Stemmer and CST's Lemmatiser.* In Proceedings of LREC 2006, Genova, May 2006, pp. 663 – 666.

Dinesh, Nikhil; Joshi, Aravind; Lee, Alan and Prasad, Rashmi (2006). *Complexity of Dependencies in Discourse: Are Dependencies in Discourse More Complex than in Syntax?* In Proceedings of the TLT 2006, pp. 79-90, 2006.

Engel, Ralf (2006). *SPIN: A Semantic Parser for Spoken Dialog Systems.*

Gildea, Daniel and Palmer, Martha (2002). *The Necessity of Parsing for Predicate Argument Recognition.* In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, July 2002, pp. 239-246.

Giuglea, Ana-Maria and Moschitti, Alessandro (2006). *Semantic Role Labeling via FrameNet, VerbNet and PropBank.* In Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL, pp. 929–936 2006.

Hacioglu, Kadri (2004). *Semantic Role Labeling Using Dependency Trees.*

Haghighi, Aria; Manning, Christopher and Toutanova, Kristina (2005). *Joint Learning Improves Semantic Role Labeling.*

Hall, Johan; Nilsson, Jens and Nivre, Joakim (2006). *MaltParser: A*

Data-Driven Parser-Generator for Dependency Parsing. In Proceedings of the fifth international conference on Language Resources and Evaluation (LREC2006), May 24-26, 2006, Genoa, Italy, pp. 2216-2219

Hutchins, John (1999*). Retrospect and prospect in computer-based translation.*

Jackendoff, Ray S. (1994). *Patterns in the Mind.* BasicBooks, A Member of the Perseus Books Group.

Johansson, Richard; Màrquez, Lluís; Meyers, Adam; Nivre, Joakim and Surdeanu, Mihai (2008). *The CoNLL 2008 Shared Task on Joint Parsing of Syntactic and Semantic Dependencies.*

Johansson, Richard and Nugues, Pierre (2006). *A FrameNet-based Semantic Role Labeler for Swedish.* In proceedings of the COLING/ACL 2006, p 436-443.

Johansson, Richard and Nugues, Pierre (2007). *Syntactic Representations Considered for Frame-semantic Analysis.*

Kate, Rohit J.; Wah Wong, Yuk; Ge, Ruifang and Mooney, Raymond J. (2004). *Learning Transformation Rules for Semantic Parsing.*

Kingsbury, Paul and Palmer, Martha (2002). *From TreeBank to PropBank.* In Third International Conference on Language Resources and Evaluation, LREC-02, Las Palmas, Canary Islands, Spain, May 28- June 3, 2002.

Kingsbury, Paul and Palmer, Martha (2003). *PropBank: the Next Level of TreeBank.*

Larsen, Ken Fris; Nielsen, Morten Ib and Simonsen, Jakob Grue (2007). *Tutorial on Modeling VAT rules using OWL-DL.*

Meyers, Adam (2007). *Annotation Guidelines for NomBank Noun Argument Structure for PropBank.*

Russell, Stuart J. and Norvig, Peter (2003). *Artificial Intelligence - A Modern Approach (Second Edition).* Pearson Educational, Inc.

Saeed, John I. (2003). *Semantics.* Blackwell Publishing.

Toutanova, Kristina and Wen-tau Yih, Scott (2007). *Automatic Semantic Role Labeling.*

A Appendices

Figure 11: The annotation template

PropBank	Target (eng)	Target (dan)	Target (swe)	A0	A1	A2	A3	A4
buy.01	purchase	købe	köpa	buyer	goods	seller	money	benefactive
	buy							
	acquire							
sell.01	sell	sælge	sälja	seller	goods	buyer	money	benefactive
tax.01			beskatta	taxer	money	taxpayer	commodity	benefactive
			skatta					
charge.01	charge		debitera	charger	money	payer	commodity	benefactive
deliver.01	deliver	leverer	leverera	deliverer	goods	recipient	money	benefactive
send.01	send	sende	skicka	sender	goods	recipient	money	benefactive
			sända					
provide.01	provide	formidle	tillhandahålla	provider	goods	benefactive	money	
	offer							
	supply							
market.01		handle	handla	seller	goods	buyer	money	benefactive
			omsätta					
pay.01	pay	betale	betala	payer	money	charger	commodity	benefactive
			deklarera					
		beregne						
account.01	account			accounter	accounted_for	benefactive	commodity	
issue.01	issue			issuer	thing_issued	recipient	commodity	
	receive			recipient	thing_received	donor	money	benefactive