Automatic semantic role labeling in a Dutch corpus

Master thesis

by *Gerwert Stevens*

Universiteit Utrecht

Automatic semantic role labeling in a Dutch corpus

Master thesis by Gerwert Stevens (0154474)
September 2006
Universiteit Utrecht, Faculty of arts
Master taal- en spraaktechnologie

First supervisor: Dr. P. Monachesi, Universiteit Utrecht
Co-supervisor: Dr. A. van den Bosch, Universiteit Tilburg

This thesis was typeset using LaTeX.

Contents

Abstract

In this thesis, an approach to automatic semantic role labeling (SRL) in Dutch corpora is presented. Although there has been an increasing interest in automatic SRL in recent years, previous research has focused mainly on English corpora. Adapting earlier research to the Dutch situation poses an interesting challenge. First and foremost, because the machine learning techniques used in previous research can not be applied to Dutch texts. This is due to the fact that there is no semantically annotated Dutch corpus available that can be used as training data.

In order to solve this problem, a novel approach to rule-based tagging based on Alpino dependency trees is proposed in the first part of this thesis. This approach has been implemented in a rule-based semantic argument tagger, called XARA. After a corpus has been tagged automatically by XARA, manual annotation can be performed relatively fast, since annotators only need to correct XARA's output instead of starting annotation from scratch. In the last part of this thesis, the training and evaluation of a learning system for SRL trained on such a manually corrected corpus is discussed.

The evaluation shows that both XARA and the learning system achieve satisfactory results: the rule-based system achieves an F-score of 50,79, the learning system an F-score of 65,29.

Acknowledgements

First of all, I wish thank my main supervisor, Dr. Paola Monachesi. Although she is involved in many projects at the moment, she was willing to supervise my work and guide the evolution from first draft to this final thesis by providing comprehensive comments and positive feedback.

Secondly, I want to thank my co-supervisor Dr. Antal van den Bosch, also a busy person, but always prepared to answer my questions. Furthermore, he introduced me to several helpful people at the ILK research group at Tilburg University, in particular Roser Morante.

I would also like to thank the researchers who were kind enough to answer questions I had by e-mail: Dr. Martha S. Palmer (University of Colorado), Robert S. Swier (University of Toronto), Xavier Carreras (Universitat Politècnica de Catalunya), Dr. Roser Morante (Universiteit Tilburg), Bertjan Busser (Universiteit Tilburg) and Dr. Gosse Bouma (Universiteit Groningen).

Finally, I would like to thank all the people who give meaning to my life: my family, Gram, Snoes and all my other friends. I would like to thank Jantine in particular. She was always there to answer my questions, provide moral support and remind me that there is more to life than semantic role labeling during the lonely summer days at the Trans. Bedankt!

Chapter 1

Introduction

1.1 Motivation

In this thesis I will explore the concept of semantic annotation of natural language texts in a computational linguistic context. I will look at how semantic roles are defined in the literature, which current (mainly English) projects deal with this subject and how knowledge acquired in previous projects can be used in a Dutch context.

Applications such as machine translation, information extraction and question answering could benefit from semantic annotation of text. I will give an example concerning question answering below to clarify the concept of semantic role labeling.

The question answering (QA) task is to answer questions posed in natural language automatically. In order to answer a question, the QA system extracts information from a large collection of text (its background knowledge). Imagine for example a QA system that extracts its background knowledge from an encyclopedia. A typical question for such a system would be "*When was John F. Kennedy assassinated?*". This question can be answered if the system is able to extract relevant semantic information from sentences in its background knowledge.

In order to determine which information is relevant for the question at hand, sentences need to be analyzed at a semantic level. Suppose the system attempts to answer the aforementioned question by extracting information from the sentence:

> "President Kennedy was assassinated in Dallas, Texas, at 12:30 p.m. CST on November 22, 1963, while on a political trip through Texas".

Then, the system needs to recognize the verb *assassinated* and identify its semantic arguments and modifiers. A relevant argument in this case is *President Kennedy*, a relevant modifier is *12:30 p.m. CST on November 22, 1963*.

Apart from recognizing arguments and modifiers, the system also needs to somehow categorize them. The modifier *12:30 p.m. CST on November 22, 1963* for example is a *temporal marker*; it answers a "when?" question and thus provides the correct answer to our question. Marking

semantic roles such as temporal markers is the SRL task.

Many interesting research projects on QA and other applications of SRL are running at the moment. I hope my thesis can be an inspiration for this kind of research and that it can even offer some new ideas.

1.2 Goals

Almost all previous research in the area of SRL has been focused on English texts. This is mainly because modern approaches to semantic annotation, like FrameNet (Palmer et al., 2005) and PropBank (Baket et al., 1998) emerged in the United States. Secondly because most automatic SRL systems require manually annotated training data, which is almost exclusively available for English. Moreover, many linguistic resources that can be helpful for the SRL task are only available in English. Examples of such resources are WordNet (Miller, 1995) and VerbNet (Kipper et al., 2000), however versions in other languages are being developed at the moment[1].

Since manual annotation of, for example, an entire encyclopedia is very laborious, information extraction systems must rely on automatically annotated data. Recently developed automatic SRL systems (Carreras and Màrquez, 2005) based on English corpora, achieved very encouraging results. This automatically raises the question whether the approach followed in the development of English semantic corpora and systems can be applied to other languages. At the moment, several projects aiming at the development of lexicons, annotated corpora and SRL tools for other languages are in progress. Probably the largest of these projects is SALSA (Burchardt et al., 2006). The SALSA corpus is a large corpus of German sentences (20.000 in the first release), manually annotated with role-semantic information based on FrameNet. Current projects involving other languages are Spanish FrameNet (Subirats and Petruck, 2003) and Japanese FrameNet (Ohara et al., 2004). For Dutch, no large scale project dedicated to semantic annotation has been started yet. However, in the following months, semantic annotation of Dutch texts is being explored for the first time within the Dutch Language Corpus Initiative (D-Coi) project. D-Coi[2] is a linguistically annotated corpus of written Dutch comprising 500 million words. The syntactic annotation scheme of the D-Coi corpus is the same as used in the Spoken Dutch Corpus (CGN) (Moortgat et al., 2000) and is based on dependency structures. Dependency structures are rich in information on the relation between words and therefore provide a good starting point for semantic annotation.

In this thesis I will explore automatic SRL methods that can be applied to Dutch corpora in general and the D-Doi corpus in particular. My goal is to annotate a small portion of the D-Coi corpus with semantic roles automatically by using TiMBL (Daelmans et. al., 2003), a memory based classifier. Memory based learning (MBL) is a supervised machine learning technique, based on the k-NN algorithm.

In order to apply machine learning techniques to unannotated data, training data is needed. At the start of my project, there was no Dutch corpus of sentences annotated with semantic

[1] Such as EuroWordNet (URL: `http://www.illc.uva.nl/EuroWordNet/`).
[2] URL: http://lands.let.ru.nl/projects/D-Coi/

roles available that could be used as training data. Creating a training corpus by hand is very time consuming and therefore too expensive for most projects. Therefore, in the first part of this thesis I will propose a method to annotate a dependency treebank semi-automatically: first the corpus will be annotated by a rule-based system, thereafter the output will be hand-corrected. In the second part of this thesis I will discuss the performance of a memory based classifier trained on this data. My hypothesis is that dependency relations can be of great value to the SRL task, both for rule-based systems and machine learning systems.

Not only is English the predominant language in earlier work, almost all previous research efforts have gone into SRL based on phrase-structure treebanks. Probably the main reason for this is that large hand-annotated corpora such PropBank are based on data annotated with simple constituent structure only. In this thesis I plan to show how valuable dependency treebanks can be for the SRL task and which properties of dependency structures are particularly useful.

1.3 Outline of this thesis

The remainder of this thesis is structured as follows:

Chapter 2 introduces the concept of semantic roles and discusses some approaches to the annotation of corpora with such roles automatically. The first part of the chapter is focused on how semantic roles are regarded in linguistic literature, the second part of the chapter provides an overview of classification methods for automatic semantic role annotation.

In chapter 3, I will discuss how information extracted from dependency structures can be used in the SRL task. I will present a heuristic strategy to map nodes in a dependency tree to PropBank argument labels. This strategy is implemented in a rule-based semantic role tagger (XARA) that is described in detail in chapter 4. Chapter 4 also describes the composition of a small corpus that was labeled by this tagger and the manual correction afterwards.

In chapter 5, the training procedure of a memory based semantic role classifier is described, as well as the classifier's performance on the aforementioned corpus. Furthermore, I will compare the performance of XARA to that of the memory based system and previously developed systems. The chapter concludes with a brief discussion on possible improvements of the learning system.

Finally, chapter 6 summarizes my findings, presents conclusions and gives some ideas for future research.

Chapter 2

Semantic role labeling

2.1 Introduction

The concept of semantic roles in linguistic theory is certainly not new. Traditionally, semantic roles are part of linking theory, a grammatical theory that describes the interaction between syntax and semantics. Semantic roles are used to assign meaning to syntactic constituents. The central question in linking theory is how these roles can be inferred from syntax.

What is relatively new though, is the *automatic* assignment of roles based on syntactic information. Today's modern parsers are able extract syntax from text accurately, which might have contributed to the renewed interest in semantic role labeling in recent years.

Historically, two types of semantic roles have been studied: abstract roles such as AGENT, PATIENT and INSTRUMENT, and roles more specific to a certain verb or class of verbs, like EATER for the verb *eat* (Gildea and Jurafsky, 2000; Pollard and Sag., 1987). In this chapter, I will first discuss abstract roles, followed by an introduction to the FrameNet and PropBank projects, which are based on more specific semantic roles. Hereafter I will turn to automatic semantic role labeling. I will introduce some machine learning techniques used for this task and discuss their pros and cons.

2.2 Generic semantic roles

The traditional form of semantic annotation is based on generic (sometimes called abstract) roles. These kind of roles are often referred to as *case frames*, *theta-roles* or *thematic roles* in linguistic literature. In the simplest form of lexical semantic representation only two roles - PROTO-AGENT and PROTO-PATIENT (Dowty, 1991) - are defined, but most theories define approximately ten roles. The most prominent theoretical accounts in this domain have been developed by Charles Fillmore (Fillmore, 1968), Ray Jackendoff (Jackendoff, 1990) and David Dowty (Dowty, 1991).

In their original form, the early proposals took thematic roles to have the following charac-

teristics (Davis, 2001):

- There is a small fixed set of thematic roles.
- Thematic roles are atomic (generally one role does not subsume another, though this might not be true of locational roles).
- Each argument of a verb is assigned exactly one thematic role.
- Thematic roles are uniquely assigned within a verb (e.g., only one argument can be dubbed AGENT).
- Thematic roles are non-relational (e.g., the presence of a PATIENT role in a verb does not imply the presence of an AGENT role as well).

Each of these characteristics of thematic roles presents certain problems. First of all: a small fixed set of thematic roles has never been agreed on and it seems unlikely that this will change. A second problem that has been discussed in the literature extensively, is the assumption that an argument is assigned exactly one role. For example, causative verbs in many languages will have arguments that act both as agents and patients.

Some of the original theories have been refined over the years to deal with one more of these problems. Below I will discuss some of the most influential theories briefly.

Fillmore (Fillmore, 1968) defined nine roles: AGENT, EXPERIENCER, INSTRUMENT, OBJECT, SOURCE, GOAL, LOCATION, TIME and PATH. In Fillmore's original theory, semantic roles were called *deep cases*. Central hypothesis in his theory is that there is a direct relation between deep cases and grammatical functions such as subject and object. Fillmore's later work on lexical semantics led to the conviction that a small fixed set of deep case roles was not sufficient to characterize the complementation properties of lexical items. This ultimately led to the theory of *frame semantics* which later evolved into the FrameNet project (see section 2.3.1).

Another influential theory on the subject of semantic roles was introduced by Ray Jackendoff. Jackendoff (Jackendoff, 1990) started with the set of thematic roles that was originally introduced by Gruber (Gruber, 1965): THEME, SOURCE, GOAL and AGENT. Jackendoff proposed several modifications and refinements to this role inventory, based on a new formalism which he called *conceptual semantics*.

According to Jackendoff, the meaning of a linguistic expression can be represented by a *conceptual structure*, composed of *conceptual constituents*. A conceptual constituent comprises one or more atomic semantic primitives. For every major constituent in the syntactic structure there has to be a corresponding constituent in the conceptual structure. The mapping between syntactic and conceptual structures is governed by *correspondence rules*.

With his framework, Jackendoff argues against a syntax-centered view of generative grammar he calls *syntactocentrism*. Jackendoff points out that phonology (PF, Phonetic Form) and meaning (LF, Logical Form) have always been treated as if they are derived from syntax. Instead, Jackendoff puts conceptual structure at one end, and phonological structure at the other, with syntax in the middle.

Dowty looked at semantic roles from a different perspective. He stated that "there is in fact a notable absence in consensus about what thematic roles are" (Dowty, 1991). This problem was also noted by other researchers in the field. One solution that was argued for, was to assume individual thematic roles for each verb, rather than verb-independent roles. For example, a BUILDER role for the verb *build* instead of the more abstract AGENT role. Dowty rejected this idea "for it ignores precisely the semantic generalization of role-type across verbs that gives the notion its interest" (Dowty, 1991).

In his theory, Dowty focused on the problem of *argument selection*. Argument selection deals with the principles that determine which semantic arguments of a verb are expressed by which grammatical relation (e.g., subject, object). Dowty argued that thematic roles should not be treated as discrete categories, but more like prototypical concepts. He defined two *proto roles*: PROTO-AGENT (P-Agent) and PROTO-PATIENT (P-Patient). Each of these proto-roles can be characterized by a set of *entailments* (properties). For example, the Proto-Agent role is characterized by:

1. volitional involvement in the event or state
2. sentience
3. causing an event of change of state in another participant
4. movement (relative to the position of another participant)
5. exists independently of the event named by the verb

A verb's argument can have several entailments. With the help of the proto-roles and their entailments, Dowty developed the *argument selection principle*. This principle states that the argument for which the predicate entails the greatest number of Proto-Agent properties will be lexicalised as the subject of the predicate; the argument having the greatest number of Proto-patient entailments will be lexicalised as direct object.

Although Dowty defined only two roles, he notes that traditional role types can be reformulated in terms of proto-role entailments. For example, EXPERIENCER is sentience (2) without volition (1) or causation (3). Characterizing role types in terms of entailments makes Dowty's system flexible in the sense that no fixed role set needs to be defined. This is an interesting aspect of Dowty's theory, because not only is there no agreement about what semantic roles are within the linguistic community, there is also no consensus about a how many and which roles there are. This problem can be solved if a more flexible approach to role labeling is chosen, such as Dowty's proposal. However, the main weakness of Dowty's theory is probably that it lacks a proper theoretical foundation which explains the exact nature an origin of lexical entailments (Wagner, 2005).

A theory that influenced modern lexical resources like VerbNet (Kipper et al., 2000) and PropBank (Palmer et al., 2005) was developed by Beth Levin. Levin (Levin, 1993) argues that syntactic frames are a direct reflection of the underlying semantics. She defined verb classes based on the ability of a verb to occur or not occur in pairs of syntactic frames that are in some sense meaning preserving (diathesis alternations). VerbNet is a lexical resource in which the original set of Levin classes has been further subdivided into additional subclasses

which are more syntactically and semantically coherent. Arguments of VerbNet classes consist of thematic labels from a set of 20: AGENT, PATIENT, THEME, etc.

Some of the aspects of the work mentioned here, most notably Levin's and Fillmore's work, have been a starting point for recently developed approaches. Modern approaches to semantic role labeling that are particular relevant in computational linguistics are *verb specific* roles - the PropBank approach - and roles specific to a certain *semantic frame* - the FrameNet approach.

2.3 Domain specific semantic roles

One of the earlier approaches to verb specific roles was *situation semantics* on which HPSG is based (Pollard and Sag., 1987). Another "specific" labeling theory, the theory of frame semantics, dates back to Fillmore's theory of case frames (Fillmore, 1968). In this thesis I will denote both the frame specific and verb specific approaches as *domain specific*.

Research on domain specific semantic roles in the last decade mainly revolved around the PropBank and FrameNet projects. I will call these approaches domain specific, because they do not assume a fixed set of general roles, instead they assume role sets that are specific either with respect to a certain verb or to the concept a certain verb represents.

Besides the fact that the FrameNet and PropBank approaches to role labeling are specific, they share in my opinion another important property: they provide a more natural and intuitive way of capturing semantics than the theories discussed in the previous section. I contribute this to the fact that the FrameNet and PropBank projects are not focused on extending the theoretical linguistic framework of linking theory, instead they aim at developing an annotation scheme which allows for human intuition.

Palmer et al. (2005, p. 4) formulate the object of the PropBank project as follows: "Our objective with the Proposition Bank is not a theoretical account of how and why syntactic alternation takes place, but rather to provide a useful level of representation and a corpus of annotated data to enable empirical study of these issues". The FrameNet project is described by Baket et al. (1998, p. 1) as: "The primary emphasis of the project is the encoding, by humans, of semantic knowledge in machine-readable form. The intuition of the lexicographers is guided by and constrained by the results of corpus-based research using high-performance software tools.".

2.3.1 Framenet

The Berkeley FrameNet[1] (Baket et al., 1998) project is creating an on-line lexical resource for English, based on frame semantics and supported by corpus evidence.

The key concept in the FrameNet method of annotation is a *semantic frame*. A semantic frame can be described as a representation of an object, event or situation. Each frame has its own

[1]URL: http://framenet.icsi.berkeley.edu

set of roles. For example, the roles defined for the frame RESEARCH are FIELD, QUESTION, RESEARCHER and TOPIC. Frames are evoked by verbs that are semantically related to the frame. For example, the frame RESEARCH is evoked by *investigate* and *research*. Roles in FrameNet are called frame elements (FEs), the frame-evoking words are called lexical units (LUs). Figure 2.1 shows the (simplified) definition of the frame DRESSING.

Definition
A WEARER puts on an item of CLOTHING, which then occupies the body location.

Core elements:
CLOTHING: Clothing is used for the item of clothing that the WEARER ends up wearing.
WEARER: The individual that puts CLOTHING on their body.

Non-core elements:
BODY_LOCATION, DEPICTIVE, DURATION, MANNER, MEANS, PLACE, PURPOSE, REASON, RESULT, TIME.

Lexical units
don.v, dress up.v, dress.v, get on.v, pull on.v, put on.v, slip on.v, throw on.v

Figure 2.1: Definition of the DRESSING frame

Frame elements are classified in terms of how central they are to a particular frame (Ruppenhofer et al., 2005). Three levels of semantic roles can be distinguished:

1. core elements
2. peripheral elements
3. extrathematic elements

Core elements instantiate required roles, furthermore, core elements make the frame unique from other frames.

Peripheral FEs mark notions such as Time, Place and manner, i.e. the act as modifiers. They do not uniquely characterize a frame and are not mandatory.

Extrathematic frame elements situate an event against a backdrop of another state of affairs. An example of an extrathematic frame element is ITERATION:

(2.1) The ferry that Kenneth was on was hijacked [twice ITERATION].

The example below shows a complete sentence annotated with roles from the frame DESIRING. The frame evoking LU in this example is the verb *wishes*.

(2.2) [Susan *EXPERIENCER*] [really *DEGREE*] **wishes** [that you 'd listen to her *EVENT*]

The FrameNet lexical resource consists of a database with more than 625 semantic frame entries and 135.000 annotated example sentences. The corpus from which the example sentences of the first FrameNet release were taken is The British National Corpus (`http://info.ox.ac.uk/bnc/`), a 100-million word collection of samples of written and spoken language from a wide range of sources, designed to represent a wide cross-section of current British English, both spoken and written. For the second release, the LDC North American Newswire corpora in American English (`http://www.cs.vassar.edu/~ide/anc/`) was added.

Four processing steps are required to produce the FrameNet database. Each step is performed by participants with a certain role.

- **Preparation** - Generating initial descriptions of semantic and syntactic patterns for use in corpus queries and annotation. This step is performed by *vanguards*.
- **Subcorpus extraction** - Extracting good example sentences. In this step a representative collection of sentences based on the Vanguards work is produced by software tools.
- **Annotation** - Marking (by hand) the constituents of interest. This is done by *annotators* using annotation software and tagsets derived from the Frame Database.
- **Entry writing** - Building a database of lexical semantic representations based on the annotations and other data. A person responsible for this task is called *rearguard*. He reviews the skeletal lexical record created by the vanguard and builds the final frame descriptions using special software tools.

Frames in FrameNet are organized in a large frame ontology. A frame can have a subframe which inherits roles while possibly adding some of their own. The MANIPULATION frame for example, inherits from the INTENTIONALLY_AFFECT frame; a role that is added to the latter is BODYPART_OF_AGENT [2].

FrameNet is an ongoing research project. At the time of writing, the FrameNet database contained more than 8.900 lexical units, more than 6.100 of which are fully annotated, in more than 625 semantic frames, exemplified in more than 135.000 annotated sentences. The lexical database has gone through three releases.

Active research on related projects seeks to produce comparable frame-semantic lexicons for other languages, e.g. the German Salsa project (Burchardt et al., 2006), and to devise means of automatically labeling corpora with semantic information.

[2] Described as "the part of the Agent's body being used to manipulate the Entity. While the default is for an Agent to use his/her hands, other body parts may be specified."

2.3.2 PropBank

The Proposition Bank (PropBank) project aims at adding a layer of semantic annotation to the Penn Treebank (Marcus et al., 1994). Contrary to FrameNet, PropBank annotation is based on verb-specific roles: every verb in the PropBank lexicon has its own *frame file*. A frame file contains specific role sets for every word-sense of the verb. Verbs are called *predicates* in PropBank, the combination of a predicate and its arguments its called a *proposition*.

In PropBank[3], core arguments are numbered from ARG0 to ARG5 and are therefore called *numbered arguments*. Numbered arguments are specific for each verb sense. In addition to the numbered arguments, verbs can take a set of general arguments: ARGMs (verb modifiers). ARGMs are not verb specific, they can be compared to non-core elements in FrameNet for that manner.

The use of numbered arguments in PropBank was chosen as a middle ground between many theoretical theories and because numbered arguments can be mapped consistently onto any theory of argument structure (Palmer et al., 2005). In order to label verbs consistently, verb labeling in PropBank is based on Levin classes of verbs. To see how this works, let's first look back at Fillmore's theory.

According to Fillmore, there is a relation between theta roles (deep cases) and grammatical functions, e.g. the role of the subject of an transitive non-passive verb generally corresponds to the AGENT role, the direct object to the PATIENT role:

(2.3) [John $_{Subject,Arg0,Agent}$] **tastes** [the fruit $_{direct\_object,Arg1,Patient}$]

However, the grammatical function of the PATIENT role can change as a result of so called *diathesis alternations*. Diathesis alternations are changes in the way verbal arguments are grammatically expressed:

(2.4) Middle Alternation
[The fruit $_{Subject,Arg1,Patient}$] **tastes** lovely.

In example 2.3 the direct object fills the ARG1 role, whereas in example 2.4 the ARG1 role of *taste* is filled by the subject.

In Levin's verb classification (Levin, 1993), verbs sharing the same diathesis alternations share the same Levin class and thus the same argument structure. In PropBank, an effort was made to ensure that verbs belonging to the same class are given consistent role labels.

An example of a PropBank frame definition for the verb *wonder* is given in figure 2.2. The verb *wonder* takes two core arguments: ARG0 and ARG1. In addition it (like any other verb) can take any number of ARGMs. An overview of available ARGMs is shown in table 2.1.

The following example sentence (2.5 was taken from the PropBank corpus and shows how a complete proposition is annotated:

[3]Although *PropBank* is the name of a semantically annotated corpus, the term is also used to refer to the corpus' annotation method

Roleset wonder.01 "think about, ponder"
ARG0: thinker
ARG1: thought

Figure 2.2: Definition of one of the role sets for the predicate *wonder*

Table 2.1: Subtypes of ArgM

Label	**Description**
ArgM-LOC	Location
ArgM-EXT	Extent
ArgM-DIS	Discourse connectives
ArgM-ADV	General purpose
ArgM-NEG	Negation mark
ArgM-MOD	Modal verb
ArgM-CAU	Cause
ArgM-TMP	Time
ArgM-PNC	Purpose
ArgM-MNR	Manner
ArgM-DIR	Direction

(2.5) [They $_{ARG1}$] are [n't $_{ARGM-NEG}$] [accepted $_{REL}$] [everywhere $_{ARGM-LOC}$], [however $_{ARGM-DIS}$].

The PropBank development process is divided into two parts: framing and annotation.

- The framing process starts with examining a sample of sentences from the corpus containing the verb under consideration. The instances are then grouped into one or more major senses, each major sense is turned into a single frameset. The framing process was a substantial time investment: framing a given lexeme/sense pair took about 10-15 minutes.
- Annotation was started by running a rule-based argument tagger on the corpus, which achieved 83% accuracy on the pilot data. The output of the tagger is corrected by hand. (This is basically the same approach I will follow in my work). Annotation of the PropBank corpus was a two-pass process: each verb was annotated by two annotators followed by an adjudication phase to resolve differences between the two initial passes.

Like in the FrameNet project, PropBank related research is now turning towards the annotation of non-English corpora. At the moment, projects aiming at the annotation of Arabic

and Spanish corpora are in progress[4].

2.4 Comparison of role labeling approaches

Since my goal is to annotate a small Dutch corpus with semantic roles, I had to make the decision which annotation method to use. Because of the abstract nature of PropBank labels, I chose the PropBank method of annotation. In section 3.2 I will motivate this choice in more detail. In this section I will provide a short overview of the major differences between PropBank and FrameNet annotation.

FrameNet and PropBank are quite similar in terms of their goals: they both aim at providing a semantic annotation layer for corpora. There are however some clear differences in their methodologies:

- FrameNet roles are frame specific whereas PropBank roles are verb sense specific. That is, FrameNet is based on hierarchically structured semantic classes of verbs, PropBank lacks such a generalization across verbs (although PropBank arguments structures are based on Levin classes).
- In PropBank there is much less emphasis on the definition of the semantics of frame entries; although it is possible to derive semantic information through a mapping to VerbNet. FrameNet on the other hand, provides semantic information for every frame in natural language. Moreover, FrameNet labels directly reflect the semantics of arguments in contrast with the more abstract method of numbering arguments in PropBank. For example, the initiator of the action in the communication frame is called COMMUNICATOR, whereas this role in PropBank would be called ARG0. The neutral way of labeling in PropBank makes automatic annotation easier, whereas the specific labeling of FrameNet is beneficial for human readability.
- PropBank addresses only verbs, whereas in FrameNet nouns and adjectives are considered as well. Strictly speaking, PropBank is a method for labeling predicate-argument structures only, while FrameNet offers a more general approach to semantic role labeling. Current automatic SRL systems concentrate solely on verbs though.
- FrameNet does not annotate a complete corpus, but aims at providing a set of example sentences (albeit a considerable set: 135.000 sentences). PropBank on the other hand is aimed at providing training data for statistical systems by annotating parts of a complete corpus: the Penn Treebank (Marcus et al., 1994). This means that "difficult cases" can be avoided in the FrameNet project, whereas in the PropBank approach an attempt is made to annotate every linguistic phenomenon that occurs in the corpus. In (Monachesi and Trapman, 2006) this is called a *corpus-driven* approach, whereas the FrameNet method is characterized as *concepts driven*.

[4]These projects were pointed out to me by participating researchers. Since work on these project has just been started, there are no references available yet

Table 2.2 illustrates the differences in argument labeling between PropBank and FrameNet. In particular, it exemplifies the abstract labeling of PropBank versus the concrete labeling of FrameNet.

Table 2.2: Comparison of FrameNet and PropBank argument labels

PropBank		FrameNet
buy	*sell*	*Commerce*
Arg0: buyer	Arg0: seller	Buyer
Arg1: thing bought	Arg1: thing sold	Seller
Arg2: seller	Arg2: buyer	Payment
Arg3: price paid	Arg3: price paid	Goods
Arg4: benefactive	Arg4: benefactive	Rate/Unit

For thorough analysis of the differences between the PropBank, FrameNet and Salsa annotation practices, I would like to refer to Ellsworth et al. (2004).

2.5 Combining FrameNet and PropBank

Although I will use the PropBank method to annotate a small portion of the D-Coi corpus in my work, a final decision on an annotation scheme for the entire corpus is still to be made. Both FrameNet and PropBank have their pros and cons, making the decision to use either one difficult. A possible solution to this problem is to develop a "merging approach", in which the best of both worlds is combined.

Very relevant in a Dutch context is the work of Monachesi and Trapman (Monachesi and Trapman, 2006). In their paper, they seek an answer to the question which semantic annotation scheme is best suited for the D-Coi project. One of the options they propose reconciles the PropBank approach to role labeling with the generalization properties of FrameNet. Their main rationale for using neither FrameNet or PropBank exclusively, is that they don't want to give up FrameNet's classification of frames in an ontology and benefit from PropBank transparent labeling at the same time. The method they propose is to reduce FrameNet frames to a simpler form in which the set of frame elements is replaced by a set of comparable PropBank arguments. This can be achieved by selecting the most common arguments from PropBank role sets of predicates that share the same Levin class and diathesis alternations as the verbs in the FrameNet frame. Such new role sets can be determined automatically using an algorithm that calculates the intersection of the FrameNet and Levin classification, instead of manual role assignment for every individual verb. In several examples they show that their approach is feasible, although some issues concerning language specific phenomena (such as the Dutch middle construction) need to be resolved in order to annotate a complete corpus with this method.

In another paper, written by Giuglea and Moschitti (Giuglea and Moschitti, 2004), a similar combination strategy is used. But instead of Levin classes, they use VerbNet classes, a refinement of Levin's scheme. They attempt to define a mapping between the VerbNet class of a

predicate and one or more FrameNet frames. For example, the VerbNet class JUDGMENT can be mapped onto the FrameNet frames REWARD AND PUNISHMENTS, JUDGMENT COMMUNICATION, SENTENCING AMONGST, etc. They performed the mapping by following the simple heuristic that combines a VerbNet classes and FrameNet frames that have the most verbs in common.

These two approaches can be categorized as *merging approaches*. Another method of combining different resources is to *align* them. In an alignment method, resources are developed separately, but some sort of mapping between resources is provided. A project in which this approach is employed is Omega[5]. Omega is a 120.000 node ontology in which different language resources, such as WordNet, FrameNet and PropBank are synthesized. Alignment methods such as Omega however don't provide a method to annotate a corpus, making them less relevant for the SRL task.

2.6 Automatic SRL

In the previous sections I discussed the concept of semantic roles and two prominent approaches to annotate such roles in large corpora. I will now turn to the task of assigning semantic roles to text automatically. Since corpora can contain huge amounts of text, e.g. the final release of the D-Coi corpus will contain 500 million words, semantic annotation is only feasible if it can be done (semi-) automatically.

Classification algorithms are amongst the most popular methods of automatic role assignment (sometimes called *shallow semantic parsing*). In classification systems, text chunks are classified as a semantic argument or as a none-semantic argument (null argument). Semantic arguments can than be further classified into a set of argument labels. Often, classification is implemented as a supervised machine learning algorithm.

Classification can be defined as creating a mapping between a set of features and a set of predefined classes and is a popular technique in datamining. Features are properties of the items to be classified, in the case of shallow semantic parsing, generally information from a syntactic parse is used. Some commonly used features in SRL are: phrase type (e.g. NP), voice (active/passive), governing theory (indicates whether the constituent is a subject or object of the verb) and head word of the phrase.

Classifiers are often implemented in two phases (Dunham, 2003):

1. A model is created by evaluating a training database. Each entry in this database contains a set of features and a class assignment. This information is used to create a model that classifies the training data as accurately as possible. This phase is called the *training phase*.

2. The model developed in the previous step can then be used to classify entries in a target database, often called the test or evaluation data. This phase is often called *evaluation phase*.

[5]URL: `http://omega.isi.edu/doc/verb_frames.html`

Normally, training data is created with the help of domain experts who assign classifications to the data manually. In the case of semantic role classification for example, training data consists of manually annotated sentences.

A wide range of classification algorithms have been developed and many of these have been applied to the SRL task in previous research. In the next sections I will discuss the following methods in more detail:

- **Probability Estimation** (Gildea and Jurafsky, 2002) - This is a classic statistical approach, first applied to semantic role classification by Gildea and Jurafsky.
- **Support Vector Machines (SVMs)** (Vapnik, 1995) - SVMs have been shown to perform well on text classification tasks. They can handle an extremely large number of interacting or overlapping features with strong generalization properties. Drawback is that the theory behind SVMs is relatively complicated.
- **Memory based learning (MBL)** (van den Bosch et al., 2004) - MBL is a direct descendant of the classical k-Nearest Neighbor (k-NN) approach. k-NN however is used for classification of numeric data. By using different data-structures and applying a variety of speed optimizations, the Induction of Linguistic Knowledge (ILK) research group at Tilburg University adapted existing k-NN algorithms so that they can be used for natural language processing (NLP) applications. The result is TiMBL, a software tool which I used in my work (see chapter 4). A drawback to MBL that it is known to be sensitive to the chosen features and algorithm parameters.

I will first discuss classification based on probability distributions, because this was the first method applied to the SRL task. Next SVM classification is discussed. I discuss SVMs, because this method has been particularly successful in the SRL task. Finally, in section 2.6.3, I will introduce the memory based learning approach, a method that I will employ in this work.

2.6.1 Probability estimation

The foundations for automatic semantic role labeling for FrameNet were laid in 2002 by Gildea and Jurafsky (Gildea and Jurafsky, 2002). Their system is based on statistical classifiers and is capable of assigning FrameNet roles to syntactically parsed sentences automatically. Later, the system was adapted to task of labeling PropBank argument structures by Gildea and Palmer (Gildea and Palmer, 2002).

In spite of the pioneering nature of their research, they achieved quite impressive results: 82% accuracy in identifying the semantic role of pre-segmented constituents, 65% precision and 61% recall at the more difficult task of simultaneously segmenting constituents and identifying their semantic role[6].

[6]See chapter 5 for a detailed description of the precision and recall metrics

Gildea and Jurafsky used posterior probability distributions for the classification task. Their probability model considers the question of finding the boundaries of frame elements separately from the question of finding the correct label for a frame element, although similar features are use for both tasks.

Given a constituent with feature vector $\vec{v}$, $P(r|\vec{v})$ is the probability that this constituent fills semantic role r. For example, suppose we use the features *gov* (governing), *voice* and *pt* (phrase type). Then, it would be possible to calculate the probabilities of the possible semantic roles by counting the number of times each role appears in combination with these features, and dividing that by the total number of times the combination of features appears:

$$P(r|gov, voice, pt) = \frac{\#(r, gov, voice, pt)}{\#(gov, voice, pt)}$$

A problem with this method is that if certain feature combinations are observed in the training data a small number of times, this method results in a poor probability estimate. This can occur when features are used that can take a large number of values. To overcome this problem, a classifier can be build by combining a variety of subsets of features in which each subset is assigned a weight λ. These feature sets can be combined in a variety of ways.

The simplest combination method is linear interpolation, which simply averages the probabilities given by each of the distributions. For example:

$$P(r|constituent) = \lambda_1 P(r|voice) + \lambda_2 P(r|position)$$

where $\sum_i \lambda_i = 1$. In this example the features *voice* and *position* are combined to form one probability.

By using linear interpolation and equal values for λ, the Gildea and Jurafsky system labeled 79,5% of the roles correctly. There are however more sophisticated methods available for choosing interpolation weights, such as the Expectation Maximization (EM) algorithm. Gildea and Jurafsky experimented with 7 different methods: equal linear interpolation, EM linear interpolation, geometric mean, back off linear interpolation, back off geometric mean and assigning the most common role to a set of features. Their best performance on held out test data was achieved using a linear interpolation model in combination with the EM algorithm to calculate the weights of different feature vectors. Using basic features sets (combinations of phrase type, government, voice, position and target word), their final system performed at 80,4% accuracy.

2.6.2 Support Vector Machines (SVMs)

In previous research, SVMs - a relatively new classification method - performed well on text classification tasks (Pradhan et al., 2005), which is probably why many researchers have used them for semantic role classification.

Training instances in SVM classification are represented as sparse feature vectors in a high

dimensional space. Suppose training instances belong either to positive or negative class as follows:

$$(\vec{x_1}, y_1), \ldots, (\vec{x_n}, y_n)$$
$$x_i \in \mathcal{R}^n, y_i \in \{+1, -1\}$$

$\vec{x}_i$ is a n dimensional feature vector of the i-th sample: $\vec{x}_i = (f_1, \ldots, f_n) \in \mathcal{R}^n$, y_i is a scalar value that specifies the class (positive (+1) or negative (-1)) of i-th data. Classification can be described as building the function $f : \mathcal{R}^n \rightarrow \{\pm 1\}$ by going through a learning process, under the assumption that new examples were generated from the same unknown probability distribution $P(\vec{x}, y)$ (Kudo, 2003).

A SVM classifier tries to separate positive from negative examples by finding a hyperplane

$$(\vec{w} \cdot \vec{x}) + b = 0, w, x \in \mathcal{R}^n, b \in \mathcal{R}$$

that divides the training data in two groups. SVMs try optimize parameters w and b to find the optimal solution. Optimal means finding the hyperplane separating training examples with the maximal margin. Margin can be defined informally by the distance between the hyperplane and the boundaries in which it can move without any misclassification.

A problem with using SVMs for the SRL task, is that SVMs are binary classifiers, i.e. they can only differentiate between two classes. Two approaches have been developed to overcome this problem in SRL (Pradhan et al., 2005):

- **Pairwise approach** - A separate binary classifier is trained for each of the class pairs and their outputs are combined to predict the classes. The total number of classifiers required for this approach is $\frac{N(N-1)}{2}$.
- **One versus all (OVA) approach** - n classifiers are trained for a n-class problem. Each classifier can discriminate between a particular class and the set of all other classes.

Another problem is that kernel-based systems such as SVMs, exhibit classification speeds which are substantially lower than those of other machine learning algorithms (Kudo, 2003). A commonly adapted approach to reduce training time, is to filter out the instances that have a very high probability of being null.

2.6.3 Memory Based Learning (MBL)

Memory based learning can be described as reasoning on the basis of similarity of new situations to earlier encountered situations (Daelmans et. al., 2003). The *learning component* of a MBL system is memory based: all training examples are stored in memory. The other component of a MBL system, the *performance* component, is similarity-based and performs the actual classification.

During training, training instances are loaded into memory. An instance consists of a vector containing feature-value pairs and a class assignment. During classification, unseen examples are compared to instances in the training data. This comparison is done using a *distance metric* $\Delta(X, Y)$. The class assignment is based on the k-nearest neighbors algorithm: the most common class amongst the k most similar training instances is chosen. In case of a tie among categories, a tie breaking resolution method is used.

Different distance metrics can be used in MBL. The most common of which is the *overlap metric*:

$$\Delta(X, Y) = \sum_{i=1}^{n} \delta(x_i, y_i)$$

$$\delta(x_i, y_i) = \begin{cases} abs(\frac{x_i - y_i}{max_i - min_i}) & \text{if numeric, else} \\ 0 & \text{if } x_i = y_i \\ 1 & \text{if } x_i \neq y_i \end{cases}$$

A k-NN algorithm with this metric is called IB1. In the overlap metric, all features have the same weight.

To improve performance, domain knowledge can be used to assign different weights to different features. Weights can also be determined by calculating statistics (e.g. frequencies) of features in the training data. These statistics can be used to determine which features are good predictors of the class labels and which features are less relevant. A useful tool for measuring feature relevance that is especially beneficial for NLP tasks, is information gain (IG) (Daelemans, van den Bosch, 1992).

Information Gain looks at each feature and measures how much information it contributes to the knowledge needed to predict the correct class label. The most common way of measuring the IG of a feature i is to compute the difference in uncertainty (i.e. entropy) between situations without and with knowledge of the value of that feature:

$$w_i = H(C) - \sum_{v \in V_i} P(v) \times H(C|v)$$

Where C is the set of class labels, V_i is the set of values for feature i, and $H(C)$ is the entropy of the class labels. Entropy measures the amount of uncertainty of a variable, and is defined as $H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$. The probabilities are estimated from relative frequencies in the training set. The computation of IG in the definition above is symbol based, for real values an intermediate step must be taken (which I will not discuss here).

Simply storing instances in memory is very expensive in terms of memory usage. Therefore researchers at Tilburg University developed an alternative to IB1 algorithm: IGTREE. IGTREE stores instances in a tree structure instead of a flat array. Arcs on paths from the root to leaves are feature values, leaves contain a distribution of classes. As a consequence, instances with identical feature values are collapsed into one path. This reduces the amount of memory needed to store all the training instances and speeds up classification.

Thus, by storing instances in a tree structure, a reduction in memory use and an increase in classification speed can be achieved. But efficiency of the algorithm can be improved even further by structuring the tree in a clever way. This can be done by using information gain to determine the order in which instance-feature-values should be added as arcs of the tree. The main idea behind this method is to place important features higher in the tree than less important features.

As we saw earlier, information gains indicates the importance of a certain feature. When the computation of information gain points to one feature as clearly being the most important, only instances with the same value as the test sentence for this feature will be considered in the distance calculation. Secondly, there is no need anymore to store all the paths: paths lower in the tree that do not contribute to further disambiguation of instance classification can me omitted.

The main difference between IB1/IGREE and the original k-nearest neighbors algorithm is that k does not refer to the nearest cases (instances), but to the nearest distances. Suppose for example we have a certain instance x, and the shortest distance between x and any neighbor is 8. If we run the IB1/IGTREE algorithm on x with k set to 1, all instances with a distance of 8 from x will be considered during class assignment. Groups of instances that share the same distance to x are called *bins*, if $k = 1$, only one bin needs to be examined. So, an appropriate name for this kind of algorithm would actually be k-nearest bins algorithm. Usually k is set to 1.

The ideas presented above were implemented in a software tool called TiMBL (Tilburg Memory Based Learner). TiMBL is a combination of many different MBL implementations. Over the years TiMBL has been tested on a large range of NLP classifications tasks: from grapheme-to-phoneme conversion to word sense disambiguation and semantic role labeling. I used TiMBL in my work for the latter. I will come back to this in chapter 5.

2.6.4 Discussion of classification methods for SRL

The previous sections provided a brief introduction to some of the classification methods that play a prominent role in the remainder of this thesis. However, these methods are not necessarily the best methods for the SRL task. This raises the question what the best classification method actually is.

Proceedings from the CoNLL shared task (see chapter 5) may help to find an answer to this question. The goal in the CoNLL task is to develop a machine learning system that is able to label PropBank argument structures automatically. In CoNLL-2005 19 systems participated and 8 different learning algorithms were used (Carreras and Màrquez, 2005): (1) Maximum Entropy (ME) statistical framework, (2) Support Vector Machines (SVMs), (3) SNoW, (4) decision trees, (5) memory based learning (MBL), (6) Relevant Vector Machines (RVMs), (7) consensus in pattern matching (CPM) and (8) Tree Conditional Random Fields (T-CRF). However, the CoNLL systems not only differ in learning method used, but also in features used in classification and pre-/post-processing methods, so it is difficult to draw conclusions regarding the best machine learning method by comparing the separate systems. However, some researchers have evaluated the influence of the classification method within their system.

Tjong Kim Sang et al (Tjong Kim Sang et al., 2005) for example, experimented with MBL, SVMs and Maximum Entropy models in the system with which they participated in the CoNLL-2005 shared task. They found that a system that combined all three methods performed slightly better (F_1 = 73,24) than the best individual system (F_1 = 73,17). Of the individual systems, the SVM-based system scored best (F_1 = 73,17), followed by Maximum Entropy Models (F_1 = 71,89) and MBL (F_1 = 71,80).

Although not participants in the CoNLL task, Pradhan et al. (2005) provide another interesting comparison of machine learning approaches. They compared their SVM based system to four other shallow semantic parsers for FrameNet. Two of these other systems use the same set of features and the same data, therefore the influence of the learning algorithm could be assessed directly. They found that their SVM system scored best on the argument classification task (87% accuracy), followed by a decision tree system (79%) and the lattice backoff system by Gildea and Palmer (2002) which scored 77%.

In both these research projects the best results were reported with SVMs, but this still does not provide a definite answer to the question what the best machine learning method is. Actually, I think this might not even be the most relevant question in automatic SRL research. Although machine learning methods play an important role in automatic SRL, I think that they are only a small part of the equation. Most systems achieve the largest increase in performance mainly by applying pre- and post-processing methods, optimizing algorithmic parameters and optimizing the feature set, not by changing the machine learning algorithm. Moreover, in the CoNLL-2005 task it was quite clear that combined systems performed better than individual systems (Carreras and Màrquez, 2005).

Besides, I think that in the quest for the best machine learning method, we must not concentrate on classification performance alone, but also consider other criteria. For example, in general SVMs perform well on the SRL task, but their training training time can be considerable, especially on large data sets. Furthermore, the theory behind SVMs is relatively complex, which makes them difficult to use. I see SVMs as the most promising method for SRL in terms of performance. However, since it requires a fair amount of skill to get satisfactory results with SVMs, less complex methods are probably more appropriate in many research environments (such as my work).

I think that especially MBL and decision tree classification can provide transparent and convenient classification methods for SRL research. If a classification method is used that does not require extensive tuning or expert knowledge, there is more room to look at the other factors that can make classification successful, such as pre-/post-processing steps and feature selection.

In other words, the answer to the original question *What is the best machine learning method for SRL?* depends on how "best" is defined, performance should not be the only criterion.

Chapter 3

From CGN dependency structures to PropBank roles

3.1 Introduction

In this chapter I describe how semantic roles can be inferred from dependency structures. My goal is to annotate a small dependency treebank with semantic roles semi-automatically. The corpus I will use for my experiments is a small portion (± 3000 sentences) of the D-Coi corpus, the semantic annotation scheme I will use is based on PropBank (Babko-Malaya, 2005).

The ideas presented in this chapter form the basis of a rule-based semantic tagger. The details of the implementation of this rule-based tagger can be found in chapter 4. The tagger is able to tag a corpus with semantic roles automatically. After manual correction the annotated corpus can be used to train learning SRL systems. In chapter 5 I will discuss the learning procedure of such a system.

3.2 PropBank vs. FrameNet for rule-based annotation

I decided to use the PropBank method of annotation in my project. The main motivation for this choice is the abstract nature of PropBank argument labeling. Although PropBank roles are not abstract in the sense that different verbs have different role sets, roles are labeled with generic labels: $\text{ARG}_0 \ldots \text{ARG}_n$ and a fixed set of ARGMs.

The agent argument of a predicate is generally labeled as ARG0, regardless of the predicate's frame. So, if we are able to identify agent arguments automatically, we can assign them the ARG0 label without looking at the corresponding PropBank frame file. The same principle applies to other arguments. Since there are no PropBank frame files available for Dutch, this is a very convenient approach.

The way FrameNet roles are labeled differs from the PropBank method in that labels reflect the semantics of the arguments structure directly; the labels of the arguments a predicate

can take are frame specific, i.e. every frame file contains a unique set of role labels. For example, the agent role in the communication frame is called COMMUNICATOR, whereas the agent role in reading frame is called READER. Therefore, assigning a label to an argument is only possible after having looked at the relevant frame file. Of course, this can only be done if such a frame file exists. Since there are no frame files for Dutch available at all, rule-based FrameNet annotation for my corpus was not feasible.

A second motivation for choosing PropBank annotation for automatic role labeling is that there is much literature available on this subject. This is mainly due to the CoNLL shared tasks (see chapter 5), in which many automatic SRL systems participated and different approaches were compared.

3.3 The D-Coi corpus

My role-labeling approach is based on dependency trees from the Dutch Language Initiative (D-Coi) corpus. D-Coi[1] is a project funded by the Flemish/Dutch STEVIN program, a program stimulating the advancement of speech and language technology in the Netherlands and Flanders. The D-Coi project is a preparatory project which will deliver a blueprint and the tools needed for the construction of a 500-million-word reference corpus of contemporary written Dutch. The corpus will be annotated with several layers of annotation. Annotation schemes from previous projects - such as the Spoken Dutch Corpus (CGN) - will be used, as well as novel schemes. Traditional schemes include POS tagging, lemmatization and syntactic annotation. Semantic annotation is an example of a novel type of annotation which is applied to a large Dutch corpus for the first time. Goal of semantic annotation of the D-Coi corpus is to investigate the feasibility of (semi-) automatic semantic annotation and explore possible formalisms (Monachesi and Trapman, 2006).

3.4 CGN dependency structures

3.4.1 Background

A resource containing sentences annotated with rich syntactic information, such as D-Coi, is often called a *treebank*. Existing treebanks can be divided into two types, according to their annotation schemata (Xia and Palmer, 2001):

- Phrase-structure treebanks[2], such as the English Penn Treebank (Marcus et al., 1994).
- Dependency structure treebanks, such as the Dutch Alpino Treebank (van der Beek et al., 2002). URL: `http://www.let.rug.nl/~vannoord/trees/`.

In contrast to the majority of previously developed SRL systems, my system will be based on a treebank of the second type: a dependency treebank.

[1] URL: http://lands.let.ru.nl/projects/d-coi/
[2] Also called *constituent structure* treebanks

When compared to constituent trees, dependency structures are more abstract. On the other hand, they are more explicit about the relation between constituents (van der Beek et al., 2002). The type of relations captured and the way this information is represented, depends on the annotation scheme chosen.

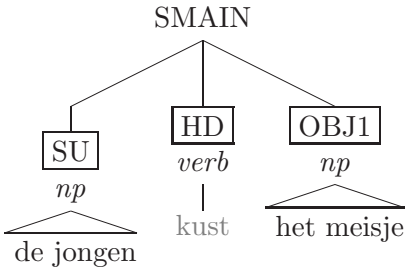
The schema upon which my work is based and that is used in the D-Coi project, has been derived from the syntactic annotation of the Corpus Gesproken Nederlands (CGN, 'spoken Dutch corpus'). For this corpus a new syntactic annotation schema for Dutch sentences was developed (Moortgat et al., 2000). This schema was later used in the Dutch Alpino dependency parser (Bouma et al., 2000) as well. Dependency trees used in the CGN are similar to those used for the German Negra project[3].

3.4.2 Dependency graphs

Dependency relations can generally be regarded as triples consisting of a head word, relation and a dependent, e.g. in example 3.1 *De Jongen* has a subject relation with the predicate *kust*. This relation can be written as the triple $(de\ jongen, subject, kust)$. Dependency relations in a sentence can be visualized in a *dependency graph*[4].

A CGN dependency graph is a directed acyclic graph in which nodes and edges are labeled with respectively c-labels (category-labels) and d-labels (dependency labels) [5]. C-labels of nodes denote phrasal categories, such as NP (noun phrase) and PP (prepositional phrase), c-labels of leafs denote POS-tags. D-labels describe the grammatical relation between the node and its head. An overview of all c- and d-labels used in the CGN can be found in appendix A.

(3.1) De jongen kust het meisje
"The boy kisses the girl"



The building blocks of dependency structures are *local dependency domains*. Local dependency domains are substructures with a non-terminal as root, i.e. the root node does not have a POS tag. The root node of a local dependency domain has one or more children. Every

[3]URL: `http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/`.

[4]Throughout this thesis and in the literature, the terms *dependency tree* and *dependency graph* are used interchangeably. Strictly spoken, in the CGN case *graph* is the correct term, since nodes can be connected to a secondary mother via a secondary edge.

[5]In most examples however, I will assume that d-labels are attached to nodes instead of edges, because that makes explanation of certain concepts easier and is more in line with the Alpino format (see chapter 4).

child is connected to it's parent by a d-labeled edge. In general, these children can be divided into three groups:

- **Heads** - the head of the dependency domain projects the c-label of the mother node: it is the phrasal head of the encapsulating syntactic constituent. For example, the head of a NP node is a noun, the head of a PP is a preposition. Every node has at most one head daughter.
- **Complements** - The way the thematic structure of the head must be interpreted is determined by its complements. Examples of complements are subject, direct object, indirect object, as well as arguments with a less obvious thematic roles such as verbal complements. A node can only have one complement daughter of every category, e.g. one subject, one direct object, etc.
- **Modifiers** - Modifiers mark such notions as time, place and quantity. Modifiers can be omitted without affecting the thematic structure. A node can have several modifier daughters.

Local dependency structures can be classified by the type of their head. For example, structures headed by a preposition are in the prepositional domain, structures with a verbal head are in the verbal domain. It is the verbal domain that is relevant in our SRL task, because we're only interested in verbal argument structures.

3.5 Mapping CGN dependency relations to semantic arguments

Intuitively, dependency structures are a great resource for a rule-based tagger. This is mainly because dependency trees directly encode the argument structure of lexical units (Hacioglu, 2004).

In the next sections, I will try to make the relation between dependency structures and PropBank roles concrete by comparing the CGN dependency annotation scheme (Moortgat et al., 2000) to the PropBank annotation scheme (Babko-Malaya, 2005). My approach is not based on a specific linguistic theory, but is an attempt to make intuitive notions explicit. To my knowledge, my research on the relation between CGN dependency structures and PropBank roles is novel work, at least in a Dutch context.

3.5.1 A generic mapping

The head node of local dependency structure in the verbal domain is a verb. In general, verbs correspond to the predicates in PropBank propositions. The argument structure of a verb is encoded by its *complements* and *modifiers*. Complements are verbal arguments, and therefore correspond to numbered arguments in PropBank. Modifiers in CGN structures have the same function as modifiers in PropBank propositions and therefore correspond to ARGMs.

Table 3.1 summarizes these basic relations. There a many complement types in the CGN

Table 3.1: Basic relation between dependency categories in the verbal domain and PropBank roles

PropBank label	**Dependency category**
$\text{Arg}_0 \ldots \text{Arg}_n$	Complement
ArgM-xxx	Modifier
Predicate	Head

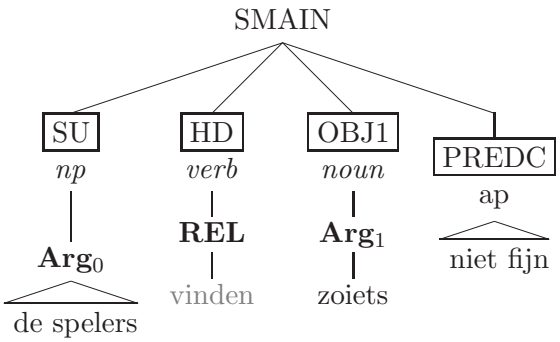
annotation, e.g. subject, object and verbal complements. To which PropBank argument a complement is related depends on the type of the complement. Subject and object complements for example are useful in the assignment of lower numbered PropBank arguments.

3.5.2 Subject and object complements

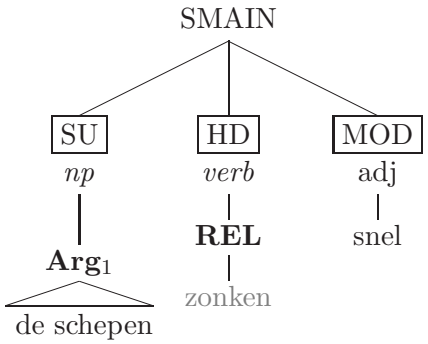
The dependency labels that are the most helpful in identifying verbal argument types are subject (SU), direct object (OBJ1) and indirect object (OBJ2).

The ARG0 label is assigned in general to agent arguments according to the PropBank guidelines, i.e. causers of an action. These correspond to subjects in CGN annotation. The ARG1 label is usually assigned to the patient argument, which are direct objects of transitive verbs, and subjects of ergatives. Finally, ARG2 is assigned to the indirect object of ditransitives. The relation between subject/object complements and PropBank arguments is illustrated by the following examples:

(3.2) De spelers vinden zoiets niet fijn. (*unergative*)
"The players don't like such a thing."



(3.3) De schepen zonken snel. (*ergative*)
"The ships sank fast."



In all example trees in this thesis, edges to nodes that fill an argument role are labeled with a PropBank label in **bold** face. The predicate of the corresponding proposition (*zonken* in the example above) has a grey color and is labeled with REL.

The examples above show the difference in labeling of ergative and unergative verbs. In the first example, the subject node is labeled with ARG0, because *vinden* (find) is unergative. In the second example, the subject node receives a ARG1 label because *zinken* (sink) is ergative.

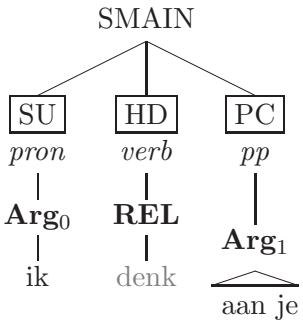
Thus, knowledge of the ergativity of the verb can improve accuracy of role assignment. However, in the treebank I used, verb ergativity information was not available. Therefore, in the remainder of this chapter I disregard the ergativity phenomenon, and assume all verbs to be unergative.

3.5.3 Other complements

So far, we have linked ARG0, ARG1 and ARG2 to subject and object complements. According to Palmer et al. (2005), "no consistent generalizations can be made across verbs for the higher numbered arguments". Therefore it is not possible to label higher numbered arguments automatically with 100% accuracy without looking at the PropBank frame files.

Although not with 100% accuracy, I think it is possible to label higher numbered arguments heuristically. Typical examples of complements that can be labeled this way are prepositional complements with a fixed preposition as head (PCs in CGN annotation). "Fixed" means that the preposition can't be replaced by another preposition without altering its semantics. For example:

(3.4) Ik denk aan je.
"I think of you"



Intuitively, it is clear that most PCs correspond to numbered arguments. But to which numbered argument a certain PC can be linked depends on the frame file of its predicate. Since I could not fall back on information from frame files, I tried to find some sort of heuristic to assign the correct role.

I found that most of the time PCs rank after numbered arguments assigned to the subject, direct object and indirect object. If we take a look at the role set for the English verb *think* (table 3.2) for example, we see that the ARG0 argument corresponds to the grammatical subject and since there is no direct or indirect object, ARG1 is filled by a PC.

A general rule that I derived by comparing CGN structures with PropBank frames is that PC complements fill the role of the first numbered argument after arguments assigned by subject and object complements. If there is more than one complement that is a numbered argument candidate, they will be numbered in order of appearance (from left to right).

Table 3.2: Roleset think.01 "think" with corresponding dependency labels

PropBank label	**Dependency category**
Arg_0: thinker	SU
Arg_1: thought	PC
Arg_2: attributative	—

I found that this heuristic can be applied to other complement types as well:

- **PREDCs with c-label NP, AP or PP** - PREDCs (predictive complements) can either be arguments of the verb or play an additional complement role. The first case holds for subject en object oriented predicative complements, as in:

  (3.5) Ze leek tevreden
  "She seemed content"

  In the above example *tevreden* fills the ARG1 role. The second case holds for adjective phrases as in:

  (3.6) Ze schilderde het huis rood
  "She painted the house red"

  In this case, ROOD is not a numbered argument, because it is not a NP, AP or PP (actually, it is an adjective).

- **ME**s - Phrases labeled with the d-label ME are complements indicating a quantity (weight, price, length). For example:

  (3.7) De fiets kost 20 euro
  "The bike costs 20 euro".

- **VC**s - VCs are verbal complements of modal verbs and causative/permissive verbs like *laten* (to let). For example:

  (3.8) Ze lijkt terughoudend te zijn
  "She seems to be reluctant".

I want to stress that the argument numbering heuristic used here certainly not always results in a correct labeling of roles. To take full advantage of the information captured in dependency structures, additional research is needed.

3.5.4 Modifiers

In the previous sections we saw how complements can be mapped to numbered arguments. I will now turn to a mapping for ArgMs. It is more difficult to find relations between the set of ARGMs and dependency labels, because the labeling ArgMs often depends on sentence interpretation. For example:

(3.9)

(a) Ze loopt [op de straat $_{ArgM-LOC}$]
"She walks on the street"

(b) Ze loopt [op hoge hakken $_{ArgM-MNR}$]
"She walks in high heels"

Both sentences have the same syntactic structure: pronoun + verb + prepositional phrase headed by *op*, but the interpretation of the prepositional phrases is different. In (a), the prepositional phrase should be interpreted as a location (ARGM-LOC). In (b), the PP acts as an indication of manner (ARGM-MNR). This example illustrates that the choice for an ArgM label can not always be made on the basis of syntactic structure alone.

Fortunately there are some ArgMs types that can be assigned automatically without such problems. I will use these in my tagger:

- **ArgM-NEG** - Is assigned to negation markers: *niet* (not) , *nooit* (never), *geen* (none), *nergens* (nowhere).

- **ArgM-REC** - Is assigned to reflexives and reciprocals such as *himself* and *itself*. Examples of Dutch words that fill the ARGM-REC role are: *mezelf* (myself), *zichzelf* (onezelf), *hemzelf* (himself), etc.

- **ArgM-PRD** - In PropBank, ARGM-PRD (markers of secondary predication) mark phrases that add extra meaning to a predicate. For example:

  (3.10) Hij spreekt [als directeur $_{ArgM-PRD,PREDM}$] van dit bedrijf.
  "He speaks as director of this company".

  PREDM (predicative modifier) nodes in CGN dependency structures correspond to ARGM-PRDs.

- **ArgM-PNC** - The label ARGM-PNC (purpose clauses) is assigned to clauses that show the motivation for some action. A typical example would be:

  (3.11) Ik ging liggen [om te relaxen $_{ArgM-PNC,MOD,OTI}$].
  "I laid down to relax."

  ARGM-PNC arguments correspond to dependency nodes with d-label MOD (modifier) and c-label OTI ("om te" clause).

- **ArgM-LOC** - ARGM-LOCs (locative modifiers) indicate where some action takes place. They can mark physical location as well as abstract ones. In CGN dependency structures the d-label LD is used to mark complements that indicate location and direction, so it is the most likely candidate to be mapped to ARGM-LOC. For example:

  (3.12) Ik ga mijn vakantie [in Spanje $_{ArgM-LOC,LD}$] doorbrengen.
  "I will spend my holiday in Spain."

3.6 Special cases

Up till now I introduced a general mapping from nodes in dependency structures with a verbal head to PropBank arguments. I only considered dependency relations with non-auxiliary verbs and verbs with active voice. In this section I will discuss special cases, such as auxiliaries.

3.6.1 Auxiliaries

The first group of "special" verbs that I would like to mention are auxiliaries used to form passive voice or a perfect aspect. In Dutch, these auxiliaries are *hebben* (to have), *zijn* (to be) and *worden* (to become). Auxiliaries of this kind were not annotated in the PropBank corpus, neither will these verbs be annotated within my project. After all, I will try to follow the PropBank annotation practice as closely as possible.

A second kind of auxiliary in Dutch (but also in other languages such as English) are copula. Copula are ignored in almost all automatic SRL implementations. Bearing this in mind, I will not consider copula in my project. Copula can be recognized at word level in combination with syntactic structure (they take a verbal complement). Prime Dutch copula are: *worden*, *zijn*, *blijven*, *schijnen*, *lijken*, *blijken*, *dunken*, *heten*, *voorkomen*.

Modal verbs form a third category of Dutch auxiliaries. In PropBank, they are not annotated as verbs in their own right, but receive the modifier label ARGM-MOD and take no arguments.

English phrasal modals such as *going* (to) and *have* (to) form an exception in PropBank, they take their own negation and adverbial marks but not any numbered arguments. In Dutch, prime modal verbs are *hoeven*, *kunnen*, *moeten*, *mogen*, and *willen*.

The last category of auxiliaries are *causative auxiliaries: doen* (literally *to do*, comparable to the causative use of *to make*) and *laten* (let).

An example from the D-Coi corpus:

(3.13) Vanaf begin jaren negentig wordt geprobeerd de SSN marktgerichter te laten werken.
"Since the early nineties, attempts were made to let the SSN operate more market-based."

Dutch causative auxiliaries correspond to English causatives that are followed by the base form of a verb ("to" + infinitive) like: *have*, *make* and *let*. In PropBank these verbs are treated like any other predicate and so will they by my rule-based tagger.

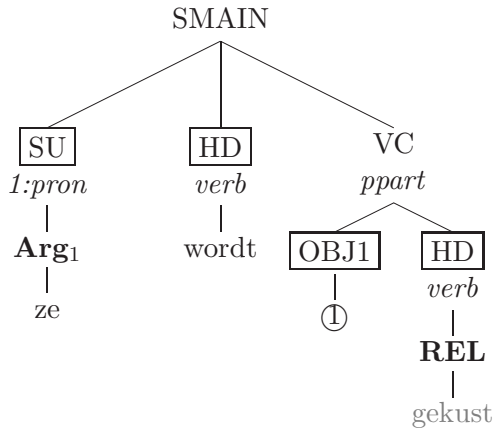
3.6.2 Passive participles

Another phenomenon requiring special attention in argument labeling is passive voice. In Dutch, passive participles in combination with the auxiliary verbs *worden* (literally "to become") or *zijn* (to be) can be used to form passive voice.

Passive voice differs from active voice in the way semantic arguments are realized. In active sentences the subject is the do-er of the action, and gets ARG0. In passives sentences the subject of the sentence is acted upon by some other other agent or by something unnamed, and is labeled with ARG1.

Passive sentences are derived from their active counterparts by movement of the object to the subject position. The movement leaves a trace, which is labeled as a direct object in CGN dependency structures. For example:

(3.14) Ze wordt gekust
"She is kissed"



In active sentences, the OBJ1 would receive the ARG1 label. In passive constructions like this one however, OBJ1 is a trace node linked to the pronoun *ze* which is the subject. As a consequence, the SU node receives the ARG1 label. So, the correct labeling of passive participles in CGN can be achieved by labeling referents, instead of co-indexed traces. In chapter 4 I will discuss this co-indexing mechanism in more detail.

3.6.3 Relative clauses

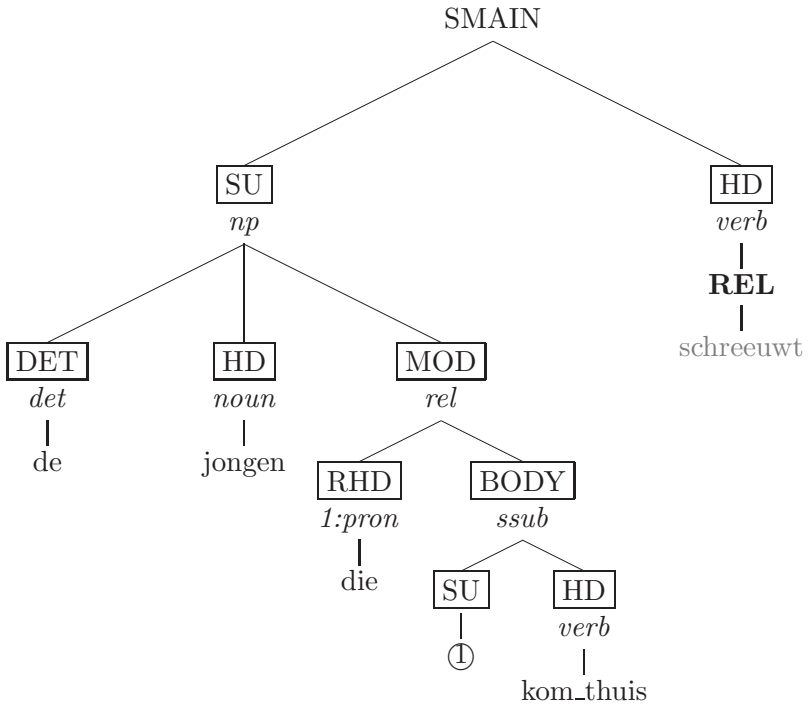
Relative clauses in CGN dependency structures contain a trace, which is co-indexed with a relative pronoun, e.g. *die* (which) or *waar* (where). In PropBank annotation a link is provided in the argument label to the antecedent NP. For example, in the sentence

(3.15) John found the place where his parents had met

the ARG1 argument of *met* in PropBank would be "`where → the place`".

CGN dependency structures differ from the syntactic tree structure used in the PropBank corpus in that the NP antecedent is not a single NP. This is clarified by the example below: the nodes *de* and *jongen* do not have a common NP parent node.

(3.16) De jongen die thuiskomt schreeuwt.
"The boy who comes home screams"



The problem is which node(s) should take the ARG1 label. A possible solution would be to label the relative pronoun and the noun of the antecedent NP (*jongen*). Another possibility is to only label the relative pronoun *die*. For this project I chose the first method: both the pronoun and the target NP are labeled with the same argument label.

3.7 Concluding remarks

In this chapter I explored the relation between CGN dependency structures and PropBank roles. I did not intent to provide a new linking theory or to extend existing linking theories, instead the methods presented here are based on intuitive notions which are guided by example sentences from the CGN annotation description and the PropBank guidelines. My goal was to develop a basic mapping that can be used to label dependency structures with PropBank labels automatically.

I hope this chapter showed how valuable dependency structures (in particular CGN structures) can be for the SRL task. Not only are they rich in syntactic information, they also provide very useful information on the relation between parts of a sentence. This enables the use of relatively straightforward mapping rules to label nodes in the dependency graph with PropBank labels. This property gives dependency trees an important advantage over simple constituent trees.

How accurate the mapping presented here actually is, can be determined by implementing it in an automatic tagger and comparing automatic labeling to hand-corrected labeling. Both experiments were carried out during my project. An evaluation of automatic tagging is given in chapter 5, a description of the implementation of the rule-based tagger in chapter 4. Although I will not discuss the results here in detail, the experiment shows that approximately

65% of the roles assigned by the automatic tagger were assigned correctly. Although I think that this a very encouraging figure, it can probably be improved by taking a wider range of linguistic phenomena into account. I think that this can be a very interesting subject for future research.

Chapter 4

Construction of a Dutch training corpus for automatic SRL

4.1 Introduction

Since the creation of Dutch semantically annotated corpora has lagged dramatically behind (Monachesi and Trapman, 2006), no training data have been available to train learning systems. As a result, almost all research on automatic SRL at Dutch universities focused on English. One of the intentions of the D-Coi project is to fill this gap by annotating a part of the D-Coi corpus with semantic roles. Details on the role labeling approach have yet to be determined, but the approach chosen will be based on current research at Utrecht University.

As a part of this research, a section of the D-Coi corpus is being annotated with PropBank roles manually. For my work, I was able to use 418 sentences from this section which were annotated over a two week period. Although two weeks is too short to annotate a substantial part of the corpus, it is enough to evaluate the used methodology and create a corpus just large enough to experiment with automatic SRL. If automatic role labeling on this small corpus is successful, the same machine learning approaches can be used in the future to annotate a much larger corpus automatically.

This chapter focuses on the development of a training corpus by means of a semi-automatic method. I will discuss some of the earlier work on this subject, show how a rule based tagger based on the ideas presented in the previous chapter can be implemented, discuss some of the problems that occurred during manual correction and describe the composition of the final corpus. The goal is to use this final corpus as training data for the memory based tagger described in chapter 5.

4.2 Bootstrapping approaches

Manually annotated corpora provide a gold standard for training and evaluation of semantic role classifiers. Starting manual annotation from scratch is very time consuming and therefore

expensive. A possible solution is to start from a (partially) automatically annotated corpus. In fact, this reduces the manual *annotation* task to a manual *correction* task. Initial automatic annotation of a corpus is often referred to as *bootstrapping* or *unsupervised SRL*. I will refer to the method of manual correction after bootstrapping as *semi-automatic annotation*.

In recent years relatively little effort has gone into the development of unsupervised SRL systems. This is partly because semantically annotated English corpora such as PropBank and FrameNet currently contain enough data to develop and test supervised SRL systems. Therefore, bootstrapping large collections of English texts has no priority anymore. For languages other than English however, annotated corpora are rare and still very much needed. Therefore, the development of bootstrapping techniques is still very relevant.

Commonly used bootstrapping methods can be divided into two categories: rule-based methods and statistical methods. Both methods generally require an extensive lexicon containing verbs and their argument structures. Most unsupervised taggers start with a corpus of syntactically parsed sentences, but some use their own chunker to recognize potential arguments. I found it quite difficult to find well documented examples of unsupervised taggers in previous work, although I found some. I will give brief overview of these systems below.

4.2.1 Rule-based tagging

The PropBank annotation process was started by running a rule-based argument tagger on a syntactically parsed treebank (the Penn Treebank). The tagger relied on an extensive lexicon, entirely separated from the one used by PropBank. The lexicon encodes class-based mappings between grammatical and semantic roles and is derived from WordNet information and Levin classes of verbs. The tagger achieved 83% accuracy on pilot data.

Apart from creating a starting point for manual annotation, unsupervised SRL can also be used to compute a *baseline rate* for comparison of supervised learning systems. In the CoNLL-05 shared task a rule-based system - developed by Erik Tjong Kim Sang from the University of Amsterdam - was used for this purpose. The system is very basic and finds semantic roles based on only seven simple rules. For example: the NP before a target verb is tagged as ARG0, modal verbs in the target verb chunk are tagged as ARGM-MOD. Overall precision of the baseline system in the CoNLL-05 task was 50%, recall 28,98% and F_1 score 36,70, which is well below the performance of participating learning SRL systems (Tjong Kim Sang et al., 2005).

4.2.2 Statistical bootstrapping

Swier and Stevenson (Swier and Stevenson, 2004) introduced a novel approach to statistical unsupervised role labeling in 2004. They started by making role assignments that are unambiguous according to a verb lexicon. Then, iteratively a probability model is created based on the currently annotated semantic roles and the newly labelled arguments are added to the annotated set. Swier and Stevenson tested their system on a set of 16 abstract semantic roles, such as AGENT, BENEFICIARY and INSTRUMENT. In their work they addressed both the task of argument identification and tagging each argument with a role given a set of chunked

sentences. They achieved 90% correct on identified arguments, and 87% correct on all slots.

It should be noted that this approach to unsupervised role labeling is new and still under development, but I think the first results are very promising. However, in their approach initial role assignment is done using information from VerbNet, a lexical resource for English containing syntactic frames along with the semantic role assigned to each slot of a frame. Because the system relies on VerbNet information, it is difficult to apply the same method to languages for which a lexicon similar to VerbNet is not available, such as Dutch. This rules out the use of statistical approaches like the one developed by Swier and Stevenson in my work.

There is however a accurate dependency parser available for Dutch, which is able to create dependency trees which contain much of the information needed to perform basic semantic tagging. In the previous chapter we saw how information from these dependency trees can be used to assign semantic roles. In this chapter I will discuss how these ideas can be implemented in a rule base tagger. My goal is to create an automatic tagger that is able to tag Alpino XML trees based on rules derived from dependency structures. To my knowledge, using information from dependency trees in a rule-based tagger is a novel approach to unsupervised semantic tagging.

4.3 The Alpino XML-format

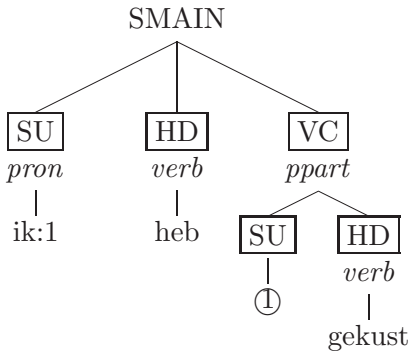
The input for the tagger is a set of sentences annotated by the Dutch dependency parser Alpino (Bouma et al., 2000) [1]. Alpino is based on a hand-crafted Head-driven Phrase Structure Grammar (HPSG). The basic tool for linguistic description in HPSG are *features*. Features describe information such as the category of a word and what other words it must appear with. In order to generate dependency trees, a DT feature was used in the Alpino grammar. It contains information about the relations between a word and other words with which it can form a constituent (van der Beek et al., 2002).

The format of Alpino dependency trees is based on the CGN annotation format. In Alpino trees the same labels are used as in their CGN counterparts and nodes are structured in the same way. The XML (Bray et al., 2004) format used to store dependency trees however differs. In CGN, sentences are stored in the TigerXML format, Alpino uses its own XML format to store parsed sentences (Kakkonen, 2005; Bouma and Kloosterman, 2002).

Although XML is perfectly suitable for storing structured data such as dependency structures, there is one problem. As we have seen in the previous chapter, a dependency structures can be represented in a directed acyclic graph (DAG), the structure of an XML document however corresponds to a tree. The principle difference between a DAG and a tree is that a DAG node can have more than one parent whereas a tree node can only have one parent. To solve this problem, Alpino uses a co-indexing mechanism. Nodes that refer to other other nodes higher in the tree have an index attribute, the value of the index attribute corresponds to the index attribute of its secondary mother node. The corresponding edge is called a secondary edge.

[1] A demonstration of the Alpino parser can be found on the following website: `http://ziu.let.rug.nl/vannoord_bin/alpino`

Figure 4.1: Example of co-indexing



From a linguistic perspective, secondary edges connect traces with their landing site.

By using this co-indexing mechanism, Alpino benefits from XML's power to store tree structured documents, while retaining the properties of DAG's. A second advantage of the Alpino format is the possibility to search dependency structures using the XPath query language. This enables the use of standard programming interfaces and API's for the manipulation Alpino documents, as opposed to, for example, TigerXML, which has it's own query language.

To get an idea of the structure of Alpino XML documents, consider the code fragment below, which is an Alpino XML representation of the dependency tree in figure 4.1:

```
<node begin="0" cat="top" end="3" id="0" rel="top">
  <node begin="0" cat="smain" end="3" id="1" rel="--">
      <node begin="0" end="1" id="2" index="1" pos="pron" rel="su" root="ik" word="ik"/>
      <node begin="1" end="2" id="3" pos="verb" rel="hd" root="heb" word="heb"/>
      <node begin="0" cat="ppart" end="3" id="4" rel="vc">
        <node begin="0" end="1" id="5" index="1" rel="su"/>
        <node begin="2" end="3" id="6" pos="verb" rel="hd" root="kus" word="gekust"/>
      </node>
  </node>
</node>
<sentence>ik heb gekust</sentence>
```

The code contains several `node` elements and one `sentence` element. The sentence element contains the text of the whole sentence, node elements correspond to nodes in the dependency graph. Nodes can have one or more of the following attributes:

begin	The left boundary (word position) of the word sequence the node represents.
end	The right boundary of word sequence the node represents.
id	An unique identification number for the node.
index	An index value for co-referenced nodes
pos	The pos tag of the word represented by the node.
cat	The node's c-label
rel	The node's d-label
word	The word represented by the node
root	If the node represents a verb, the root form of that verb is contained in this attribute.

In order to store semantic role information in Alpino XML files, I added a semantic annotation layer to the original format: nodes that fill a PropBank role get a `pb` attribute with a PropBank label as value.

The fact that a single attribute value is used to store PropBank roles implies that every node in the XML tree can only be assigned one argument, since the `pb` attribute can contain only one value. This can pose a problem if a nodes fills multiple roles (see section 4.5.3). Since the final semantic annotation scheme of the D-Coi corpus has yet to be determined, this problem will probably be solved in a later stage of the D-Coi project. In the remainder of this thesis I will assume that nodes can fill one role only.

4.4 XARA: a rule-based corpus tagger

As part of the semi-automatic annotation strategy I employ to create a training corpus, I build a rule-based semantic annotation tool for Alpino XML files, called XARA (**X**ML-based **A**utomatic **R**ole-labeler for **A**lpino-trees). XARA is written in Java and relies heavily on the XPath language (Clark and DeRose, 1999).

4.4.1 XPath

XPath is a language for addressing parts of an XML document. The primary syntactic construct in XPath is the expression, the most common expression types in XPath are *location paths* (hence XPath's name). Location paths are written as a sequence of *location steps* to get from one node to another node or set of nodes. Location steps are separated by slash ("/") characters and have three components:

- An axes, which specifies the tree relationship (e.g. parent, sibling) between nodes. The axis specifier is the fist component of the location step.
- A node test, which specifies the node type and name. The node test is specified after the optional axis specifier. Node test and axis specifier are separated by a double colon.

- A predicate[2], which can be used to test the node for certain properties, i.e. attribute values. Predicates are specified in square brackets after the node test.

Location paths can be of two types: relative paths and absolute paths. A relative path specifies the path from the current node (*context node*) to the target node. Absolute paths specify the path from the root node to the target node. An absolute location path starts with a slash, a relative location path does not.

Consider the following XPath expression for example:

```
following-sibling:node/node[@id='5']
```

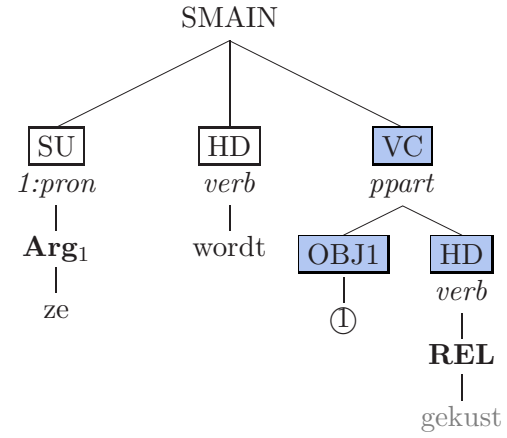
This expression specifies a relative path to `node` elements which are children of siblings that follow the current node element and have an `id` attribute with value `5`.

The Alpino XML format is designed specifically to support a range of linguistic queries in the XPath language directly (Bouma and Kloosterman, 2002). This is the main reason why I chose to use XPath expressions in XARA. Another reason is that Java provides standard class-libraries to work with XPath expressions and XML files. This means that Alpino XML documents can be read and analyzed in Java with little effort.

4.4.2 The annotation process

The input for the tagger is set of directories containing Alpino XML files, called a *treebank*. Each sentence is annotated separately by applying a set of rules. Rules are applied to local dependency domains (subtrees of the complete dependency tree). The local dependency domain to which a rule is applied, is called the rule's *context*. A context is simply defined by an XPath expression which selects a group of nodes.

Suppose for example that we want to apply a certain rule to nodes that are part of a passive participle, i.e the context of our rule are passive participles. Passive participles in Alpino trees are local dependency domains with a root node with c-label PPART:



[2]No to be confused with the term predicate from linguistics were it refers to verbs.

The dark colored nodes are the ones we are interested in. To select these nodes, the following XPath expression can be used:

```
//node[@cat='ppart'][preceding-sibling::node[@rel='hd' and (@root='word')]]
```

which says that we are looking for nodes with the c-label PPART and the auxiliary verb indicating passive tense (*word*) as preceding sibling.

Once we have defined a context we can apply rules to nodes in this context. Rules are implemented by the Rule class. They consist of an XPath expression which specifies a relative path from the context's root node to the target node and an output label. Upon application of the rule, the target node will be labeled with output label.

The output label can have three kinds of values:

- A positive number n, to label a node with Arg_n.
- The value -1, to label the node with the first available numbered argument.
- A string value, to label the node with an arbitrary label, for example an ArgM.

Notice that because the label can be specified as a string value, the set of possible labels is not restricted. In my work I used PropBank labels, but other labels (such as FrameNet labels) can be used just as well.

Formally, a rule in XARA can be defined as a $(path, label)$ pair. Suppose for example that we want to select direct object nodes in the previously defined context and assign them the label ARG1. This can be formulated as:

```
(./node[@rel='obj1'],1)
```

The first element of this pair is an XPath expression that selects direct object daughters, the second element is a number that specifies which label we want to assign to these target nodes. In this case the label is a positive integer 1, which means the target node will receive the label ARG1.

4.4.3 Specification of a tagger

The combination of a context and a set of rules is called a *tagger* in XARA and is implemented by the Tagger class. In XARA, taggers for normal verbal domains, past/present participles, passive participles and modals are defined. Each Tagger instance tags different parts of the XML tree.

In appendix C definitions of all currently defined taggers can be found. As an example, listing 4.1 shows the definition of a tagger for passive participles. This code might look a bit intimidating at first sight because it contains a lot of XPath expressions. However, they are structured in a very accessible way.

Listing 4.1: The definition of a tagger

```
t = new Tagger();
/* definition of the context for the rule */
t.setContext(doc, "//node[@cat='ppart'][preceding-sibling::node[@rel='hd'
    and (@root='word' or @root='ben')]]");
/* rules for labeling core-elements */
t.addRule(new Rule("./node[@rel='hd']", "PRED"));
t.addRule(new Rule("//node[@rel='mod' and @cat='pp']/node[@rel='obj1'][
   preceding-sibling::node[@rel='hd' and @word='door']]", "0"));
t.addRule(new Rule("./node[@rel='obj1']", 1));
t.addRule(new Rule("./node[@rel='obj2']", 2));
/* complements */
t.addRule(new Rule("./node[@rel='vc']", -1));
t.addRule(new Rule("./node[@rel='pc' and @cat='pp']", -1));
t.addRule(new Rule("./node[@rel='predc'][@cat='np' or @cat='ap' or @cat='
   pp']", -1));
t.addRule(new Rule("./node[@rel='me']", -1));
/* modifiers/adjectives */
t.addRule(new Rule("./node[(@word='niet' or @word='nooit' or @word='geen'
    or @word='nergens') and (@pos='adv')]", "ArgM-NEG"));
t.addRule(new Rule("./node[@rel='mod' and @cat='oti']", "ArgM-PNC"));
t.addRule(new Rule("./node[@rel='predm']", "ArgM-PRD"));
t.addRule(new Rule("./node[@rel='ld' ]", "ArgM-LOC"));
t.addRule(new Rule("./node[@pos='pron' and (@word='mezelf' or @word='
   zichzelf' or @word='hemzelf' or @word='haarzelf')]", "ArgM-REC"));
t.tag();
```

After a new tagger instance (`t`) has been created, first the context for the rule is defined. The `setContext` method takes two parameters: an XML document and an XPath expression. The context expression in the example above selects PPART nodes that are preceding siblings of the auxiliaries *worden* (root=*word*) and *zijn* (root=*ben*).

The context definition is followed by one ore more rules specified in the `addRule` method. `addRule` takes two parameters: a rule instance and an output label specification. A rule is created by providing an XPath expression and a target label. The third rule for example, selects direct objects and labels them as ARG1.

The last line of the listing contains a call to the method that starts the actual tagging: `tag()`. This method starts the tagging process on the XML document specified in the context definition. That is, PropBank labels are added to the original XMl document.

As listing 4.1 shows, definitions of taggers, rules and contexts are an integral part the XARA's Java source code. But because XPath expressions are used in tagger definitions, rules can easily be added, modified or removed without extensive knowledge of Java. Moreover, the use of XARA is not restricted to a specific XML structure, i.e XARA can be used on XML based treebanks other than Alpino with relatively little effort. The only requirement is that an XML structure is used that supports XPath queries.

The facts that the rule base in XARA can easily be modified and that the use of XARA is not restricted to a particular XML format, adds to XARA's reusability. Bearing future research in mind, I considered reusability an important issue during the design of XARA.

4.4.4 Future research

The exact XML format to be used in D-Coi was yet to be determined at the time of writing. Therefore, I had to use a provisional dependency tree format which has a limited set of POS-tags. Only basic tags such as adverb, adjective, adverb, determiner and noun were available. Moreover, very few lexical information on words was available in the dependency trees. This limits the accuracy of the tagger. For example, there is not enough information available in the Alpino trees to distinguish ergatives from unergatives. For this reason, all verbs are treated as unergatives by XARA. As a result, numbered arguments of ergatives are labeled incorrectly: their subjects are labeled ARG0 instead of ARG1.

In future releases of the D-Coi treebank, the dependency trees might be enriched with more lexical information. This would not only improve performance of the rule-based tagger, but can also provide better features for supervised SRL systems.

Another possible improvement concerns the definition of rules. As stated in the previous section, currently rules can only be defined within the source code. To improve the generalization properties of the source code, a functionality to use externally defined rules (for example in a text file) can be added. This would separate the rule base from the source code, which is a design principle that is generally considered good practice in software engineering.

Finally, more and better rules can be added to XARA's rule base. This can be done by hand, which requires some more linguistic research into the relation between Alpino dependency trees and PropBank arguments labeling. Or, maybe rules based on corpus evidence can be derived automatically if more manually annotated semantic data becomes available[3].

4.5 Manual correction

XARA is able to assign core-arguments and some ArgMs to an unannotated corpus automatically. However, additional manual annotation of the corpus afterwards is still needed before it can be used in a learning SRL system. For this purpose, 418 sentences which were initially tagged by XARA were hand corrected for my project.

During manual correction, PropBank labels assigned by XARA were checked and missing labels were added. This was done with the help of a software tool: the graphical tree editor TrEd[1]. TrEd is a general tree editor which uses its own file format by default, but with some adjustments it was able to handle the Alpino XML format[2].

The annotation method followed was based on the PropBank annotation guidelines (Babko-Malaya, 2005). But, because of:

- Differences in syntactic structure between the Penn Treebank and D-Coi annotation.
- Lack of Dutch frame files

[3] Although this would actually transform XARA into a supervised tagger.
[1] URL: `http://ufal.mff.cuni.cz/pdt/Tools/Tree_Editors/Tred/`
[2] Thanks to Geert Kloosterman, Groningen University

- Linguistic differences between Dutch en English
- Errors in the syntactic parses, although the dependency parses we used were hand corrected.

we could not follow these guidelines blindly. Some of the problems we encountered could be resolved during annotation, other will require some more research. Below I will explain some of the major issues we encountered.

4.5.1 Absence of Dutch frame files

In PropBank, frame files provide a verb specific description of all possible semantic roles and illustrate these roles by examples. At the moment, no Dutch frame files exist. This is a problem, because frame files are essential for consistent annotation, especially when a corpus is annotated by many annotators. Moreover, the lack of example sentences makes consistent annotation even more difficult.

Defining a set of frame files is very time consuming. For the original PropBank for example, framing a given lexeme/sense pair required about 10-15 minutes. Since there are more than 4500 framesets in PropBank, the total time required for framing was quite substantial.

One of the initiators of the English PropBank, Martha Palmer, provided me with another example. Her research group recently started a new ProbBank for Arabic[4]. Their frame file creator is creating about 20 frame files a week - about one an hour.

These figures illustrate that creating a set of PropBank frames during my project was not feasible, because it would simply cost to much time. As an alternative we decided to annotate Dutch verbs with the same argument structure as their English counterparts, thus to use English frame files instead of creating Dutch ones. Naturally, this causes some problems. First of all, not all verbs can be translated to a 100% equivalent counterpart. Secondly, argument structures may differ.

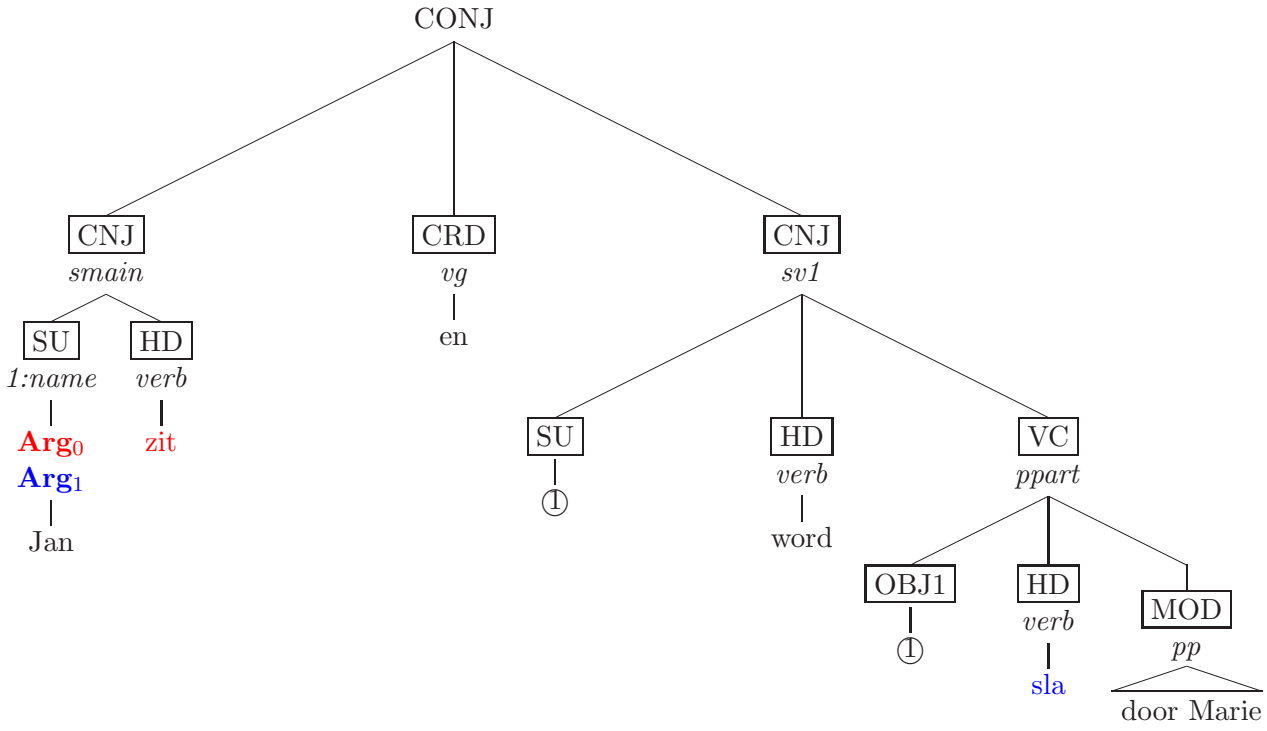
Luckily, during manual annotation these problems proved to be relatively rare. In most cases applying the PropBank argument structure to Dutch verbs was straightforward. If translation was not possible, this observation was reported and an ad hoc decision was made on how to label the verb. If, in the future, the decision is made to develop a Dutch frameset, I hope that these observations prove to be useful.

4.5.2 Co-references

In the CoNLL shared task (see chapter 5), a separate tag (R-) was used to label co-indexed traces. Instead of using the R-tags, I chose to simply not label references but label their referent instead. Since the co-indexing mechanism of Alpino captures the relation between referent and reference, R-labels would only add redundant information.

[4]Since work on this project has just been started, there is no reference available yet

Figure 4.2: Example of coordination



4.5.3 Conjunctions

Conjunctions (or more generally: *coordination*) pose another challenge for annotation. In the example below, *Jan* fills the ARG0 role of the predicate *zit* (sit) as well as the ARG1 role of the predicate *sla* (hit).

(4.1) Jan zit en wordt geslagen door Marie.
"Jan sits and is hit by Marie."

Figure 4.2 shows the dependency tree of this sentence.

In another example:

(4.2) It is dangerous to travel [to and in Angola $_{ArgM-DIR/ArgM-LOC}$].

the chunk *to and in Angola* can receive the ARGM-DIR label as well as the ARGM-LOC label.

Labeling one constituent with multiple roles is a problem for our storage format. After all, roles are stored in an attribute of node elements, this attribute contains a string representation

of a specific role label. Storing multiple argument labels in this attribute instead of one, would require some mechanism to separate the labels. Finding a mechanism that is robust enough to be used for future annotation is not an easy task.

For example, labels could be comma separated, but this makes it very difficult to determine (automatically) to which predicate a specific label belongs. This can be solved by storing, for every label, the predicate to which it is related. Although this approach is fine for automatic labeling, it would slowdown manual annotation.

Another possibility would be to label the co-indexed traces instead of the referents, although this would be an exception to the strategy described in section 4.5.2.

Coordination is not only a practical problem in my work, it is even a problem at linguistic level. In the case of example 4.1, labeling *Jan* with two arguments is not a violation of the PropBank annotation guidelines, since the labels belong to separate predicates. In example 4.2 however, labeling *to and in Angola* twice is a violation PropBank guidelines since they belong to the same predicate. For this problem a linguistic solution has yet to be found.

In my project, sentences which contain "problematic" conjunctions - that is conjunctions that require an argument to be labeled with multiple roles - will simply be left out of the final training corpus.

4.6 Composition of the final training corpus

The D-Coi corpus consists of Dutch sentences which are syntactically parsed by the Dutch dependency parser Alpino. For my project, I had 6.125 sentences from this corpus at my disposal. The parses of these sentences were all hand-corrected. From this corpus I created a smaller corpus with 2.064 sentences to by annotated by my rule based tagger.

Due to the short amount of time available for manual correction, the training corpus was kept small in comparison to existing English training data. Because of the small size of the training corpus, it is important to make sure that there are enough examples available for every verb. This was done by only selecting sentences with high frequency verbs.

From a word frequency list of the CGN corpus[1], I extracted a list of the 100 most frequent non-auxiliary verbs in the CGN corpus. Next, I selected sentences from the D-Coi subcorpus that contained at least one of the high frequency verbs on this list. This yielded 2.064 sentences.

During a two week period, 418 from the initial 2.064 sentences were hand-corrected by one annotator (a linguistics student). These 418 sentences comprise the final corpus that will be used to train an automatic classifier. All PropBank argument types are represented in the corpus, although for some argument types (like ARGM-STR) very few examples are available.

Statistics of the training corpus can be found in tables 4.1 and 4.2.

[1] provided by the *TST centrale* (`http://www.tst.inl.nl/`)

Table 4.1: Training corpus counts

Words	9800
Unique verb lemmas	284
Sentences	418
Arguments annotated	2270
Annotated predicates	698
Annotated verb lemmas	242

Table 4.2: Individual argument counts after manual annotation

Argument	**Count**
Arg0	339
Arg1	675
Arg2	225
Arg3	23
Arg4	11
ArgM-ADV	125
ArgM-CAU	36
ArgM-DIR	14
ArgM-DIS	123
ArgM-EXT	28
ArgM-LOC	149
ArgM-MNR	68
ArgM-MOD	99
ArgM-NEG	28
ArgM-PNC	22
ArgM-PRD	44
ArgM-REC	30
ArgM-STR	6
ArgM-TMP	225

Chapter 5

Training a semantic role classifier

5.1 Introduction

Manual annotation of a large corpus with semantic argument structures is a very labor-intensive thus expensive process. Moreover, manual annotation is not suitable for information extraction on constantly changing text collections, such as news articles on web sites. Therefore, much effort has gone into the development of automatic systems in recent years.

Automatic systems can be divided into supervised (learning) systems and unsupervised systems. In the previous chapter I described the implementation of an unsupervised (rule-based) system: XARA. Unsupervised systems are especially useful when no training data is available. On the other hand, supervised systems offer a more flexible approach to role labeling. By training them on different sets of data they can be easily adapted to different text domains and even to different languages.

Although most previous research has been focused on English texts, this earlier research paved the way for the development of non-English systems as well. However, of equal importance to the development of non-English systems is the advancement in NLP tools such as parsers and named entity taggers, as well as the development of syntactically annotated text corpora and lexical resources. Thanks to research programs such as STEVIN, several of such resources are now available for Dutch, e.g. the Alpino parser, the CGN and (in the future) the D-Coi corpus. The success of English systems combined with the availability of Dutch corpus resources (especially dependency treebanks) inspired me to experiment with supervised semantic role labeling on Dutch texts. Although most previous research has been focused on English texts, this earlier research paved the way for the development of non-English systems as well. However, of equal importance to the development of non-English systems is the advancement in NLP tools such as parsers and named entity taggers, as well as the development of syntactically annotated text corpora and lexical resources. Thanks to research programs such as STEVIN, several of such resources are now available for Dutch, e.g. the Alpino parser, the CGN and (in the future) the D-Coi corpus. The success of English systems combined with the availability of Dutch corpus resources (especially dependency treebanks) inspired me to experiment with supervised semantic role labeling on Dutch texts.

In this chapter I will describe the experiments I did with automatic SRL on a small Dutch corpus of sentences parsed by the Dutch Alpino parser and labeled with PropBank argument structures. The corpus I used is the manually annotated corpus described in the previous chapter, which contains 418 sentences. I used XARA to extract feature values from the corpus and TiMBL for the training of the classifier.

In section 5.2, I will describe the features I used in my system and give a short overview of the features considered in previous systems. Next, I will give an overview of common evaluation metrics for learning systems and introduce the metrics used in my evaluation. In sections 5.5 and 5.6 the performance of XARA and the learning system is evaluated.

5.2 Feature selection

Supervised learning algorithms need to be trained on a set of training data before they can be applied to unseen data, as opposed to the unsupervised methods discussed in chapter 4. The most common type of machine learning algorithms applied to the SRL problem are classification algorithms. In chapter 2 I discussed several common classification algorithms in detail.

The goal of classification is to assign class labels to a set of instances automatically. Instances represent the items to be classified and their classes. For example, in the SRL task, instances can represent (*predicate*, *candidate argument*) pairs and their PropBank label.

Instances are usually encoded as a set of features. Each feature describes a different aspect of the item to be classified. For example, in semantic role classification candidate argument constituents can be described by, amongst others, their phrasal tag and head word. Finding a feature set that yields an accurate classification is an important and certainly not trivial task.

In this chapter I will concentrate on memory based learning classification with TiMBL. Choosing the right set features is an important step in the training of classifiers in general, but requires extra care in memory based learning. Particular, irrelevant or redundant features may lead to reduced performance of a memory based classifier (Tjong Kim Sang et al., 2005).

5.2.1 Features used in previous systems

Judging from previous research, it seems that finding the best feature set is often a process of trail-and-error. It is common practice in SRL research to start with a set of features derived from earlier research and evaluate performance while adding and removing new features. The proceedings of the Conference on Computational Natural Language Learning (CoNLL) shared task[1] make an excellent source of information on previously developed PropBank based systems[2].

[1] URL: `http://www.lsi.upc.edu/~srlconll/home.html`

[2] A comparable task for systems labeling FrameNet roles is the Senseval-3 Task. URL: `http://www.senseval.org/`

There have been two editions of the task: in 2004 and 2005. In both editions, the goal of the task was to develop a machine learning system able to recognize participants of the propositions governed by the target verbs. Training and development data were provided to build the learning system. The data sets contain predicted input annotations and the correct outputs. The training data could be used for training of the systems, the development data to tune parameters. Data consisted of sections of the Penn TreeBank with information on predicate-argument structures extracted from the PropBank corpus. In the 2005 edition, 19 systems participated.

Features used in the CoNLL systems can be divided into four categories (Carreras and Màrquez, 2005):

1. Features characterizing the candidate argument.
2. Features describing properties of the target predicate.
3. Features capturing the relation between predicate and candidate argument.
4. Global features describing the complete argument labeling of the predicate.

Features describing the candidate argument usually include: head word of the phrase, argument boundries (first and last word), POS tags of argument boundaries and phrase type.

The target predicate is generally described by its voice (passive/active), lemma and POS tag.

The third type of feature provides information on the kind of relation between predicate and candidate argument. One of the most salient features of this kind is the path feature. Many variants of this feature have been tested in recent systems, but in its most general form the path consists of all nonterminals on the path from predicate to argument separated by symbols indicating upward or downward movement through the tree. For example, the path from the REL node to the ARG1 node in the tree depicted in figure 5.2 is HD ↑ VC ↑ SMAIN ↓ SU.

Only two systems in the CoNLL task used global features. Global features describe the structure of the target proposition. For example, the arguments the target predicate takes and the syntactic categories of the arguments. This kind of feature is called *global* because it does not describe the relationship between predicate and argument but tells something about the entire target proposition.

5.2.2 Previous work on dependency based features

What all CoNLL systems have in common, is that they use the same sources of information to extract syntactic features from: part-of-speech tags, chunks, clauses and full syntactic parses[3]. None of the CoNLL systems used features extracted from dependency structures. Outside the CoNLL task only one system, to my knowledge, used dependency tree features for classification: the system by Kadri Hacioglu (Hacioglu, 2004). Hacioglu's system was trained and tested on data of the 2004 CoNLL shared task that was converted into dependency trees.

[3]Although different pre-processors (parsers etc.) were used to obtain this information.

Hacioglu divides previous approaches into three broad classes based on the type of tokens classified: constituent-by-constituent systems, phrase-by-phrase systems and word-by-word systems. He classifies his own approach as relation-by-relation (R-by-R) semantic role labeling. The basis of his new approach is formed by a new treebank of dependency structures called DepBank. To create the DepBank corpus, first constituency trees from the Penn treebank were converted into dependency trees; furthermore, nodes in the dependency trees that cover a semantic argument were augmented with a PropBank label. For sentences with more than one predicate, the same tree was instantiated with different argument labels. After conversion, every tree was linearized into a set of dependency triples: $(predicate, relation, dependent)$. Not all dependency relations were considered though, only nodes that are children, grandchildren, sibling or siblings' children of the predicate were selected as candidate arguments.

The next step was to extract features from the dependency relations, such as type of the relation and head word of the relation, from the candidate argument relations. Finally, the feature bank was used as input for SVM classifiers. A one-versus-all classifier was trained for each semantic role using the same data (although converted to dependency trees) as the systems in the CoNLL-04 shared task. The system achieved 85,5% precision, 83,6% recall and a F-score of 84,6. These are encouraging results, especially when compared to the results of the highest scoring system in the CoNLL-04 task: 74.18% precision, 69.43% recall and 71.72 F-score. However Hacioglu notes that approximately 8% of semantic roles in the DepBank development set is lost during the conversion process. When predicted roles are compared to the PropBank development set, F-score drops to (a still acceptable) 79.

In a sense, Hacioglu's approach is comparable to mine, because we both use features extracted from dependency trees. However, there are, apart from the machine learning technique used, also some differences:

- Hacioglu does not use a dependency parser to create the dependency trees, instead he converts existing constituent trees. Tree conversion has one big disadvantage: not all phrase nodes in constituent trees may find an equivalent relation node in the dependency tree, which results in loss of semantic roles (approximately 8 percent in Hacioglu's case).
- In Hacioglu's system, a dependency tree is created for every proposition in the sentence. In my approach, all labels from propositions in a sentence are stored in one dependency tree, which makes manual annotation less complicated. Drawback to my approach is that nodes can receive only one label, and thus that a relation can only fill one role (see section 4.5.3).
- Hacioglu only uses features that are typical to dependency trees (such as the head word of the relation). He does not use "traditional" features like phrase type, i.e. features derived from the constituent tree.

  I *will* use features based on constituent structure, since constituent structure is available in Alpino trees and proved to be useful in my rule-based system. Constituent structure in Alpino is derived from the dependency structure. This is the opposite of Hacioglu's approach, in which constituent trees are converted into dependency trees.

I think it would be very interesting to see dependency tree based systems in future CoNLL

shared tasks. Especially if Hacioglu's results are a good indication of the potential of such systems.

5.2.3 Features used in my system

For my classification task, I compiled a set of baseline features based on earlier work and some initial experimentation with TiMBL. I experimented with features of all types except global features, since I had no lexicon to derive global features from.

I chose to use two features that describe the predicate verb:

(1) Predicate's root form - The verb root as given by Alpino. This feature is analogous to the verb lemma feature used in many existing systems. I used the root form instead because Alpino (at least in the treebank format I used) does not provide the lemma.

(2) Predicate's voice - A binary feature indicating the voice of the predicate (passive/active). A predicate is considered passive if it has the auxiliary verb *worden* or *zijn* and is a child of a node with c-label PPART (passive particle).

Note that the predicate's POS tag is not used as a feature in my system, unlike in many existing systems. This is because all verbs in Alpino trees have the same POS tag: VERB. Maybe in a later stadium of the D-Coi project a more elaborate set of POS-tags becomes available. It would be particularly useful if POS tags can indicate whether a verb is ergative or unergative. Another option for future work is to use more sophisticated features (from a linguistic point of view), so that ergativity can be inferred. According to the annotator of the training corpus, ergativity was the main source of errors in the labeling of lower numbered arguments by XARA.

Features used to describe the candidate argument in my system are:

(3) Argument c-label - The category label (phrasal tag) of the node, e.g. NP or PP .

(4) Argument d-label - The dependency label of the node, e.g. MOD.

(5) Argument POS-tag - POS tag of the node if the node is a leaf node, null otherwise.

(6) Argument position - A binary feature which indicates whether the argument is positioned *before* of *after* the predicate.

(7) Argument head-word - The head word of the relation if the node is an internal node or the lexical item (word) if it is a leaf.

(8) Head-word POS tag - The POS tag of the head word.

(9) c-label pattern of argument - The left to right chain of c-labels of the argument and its siblings.

(10) d-label pattern - The left to right chain of d-labels of the argument and its siblings.

(11) c-label & d-label of argument combined - The c-label of the argument concatenated with its d-label.

This feature set represents in general the same information as is used by XARA. The rules I defined in XARA mainly look at the d-label en c-label of the candidate argument. In fact, numbered arguments are assigned only by looking at the d-label. For modifiers additional lexical information is sometimes used, e.g. ARGM-REC is assigned to a specific set of pronouns: *haarzelf*, *mezelf*, etc. This kind of information is represented by the *head-word feature* in my feature set.

Features from this set that were not used in XARA are: predicate's root, label pattern of candidate argument and argument position. I added the position feature, because it is was used in all CoNLL-05 systems (except one) and in the Hacioglu system. The same applies to the the predicate's root (or lemma). The label pattern feature was used in several CoNLL systems and turned out to have a positive effect on the performance of my system.

A feature that was used in all CoNLL-05 systems, but does not appear in my feature set is the path from argument to verb. I did use this feature in my initial feature set, but noticed that it caused a drop in performance. I have no explanation as to why this happened.

5.3 From Alpino trees to TiMBL instances

The data format used by TiMBL is completely different from the XML based format used by Alpino. TiMBL input consists of text files with training and test instances. An instance is a tuple containing feature values and a target class. TiMBL supports several input formats, but they all have in common that every line in the input data file contains one instance and the last feature is considered the target class. For my project I used the C4.5 (comma separated) format, which is the default format in TiMBL. In this section I will discuss a method to encode annotated Alpino trees into a C4.5 instance base.

5.3.1 Instance base encoding

The way instances are encoded in NLP classification tasks depends on the task at hand. For example, in grapheme-to-phoneme conversion an instance might contain a grapheme and its surrounding graphemes as features and the target phoneme as target class. In semantic role labeling, instances generally consists of syntactic features and PropBank labels as target class. We have looked at some of the most prominent of these features in the previous section.

I chose to encode instances analogous to earlier work by van den Bosch et al. (van den Bosch et al., 2004; Tjong Kim Sang et al., 2005). They used a pair-wise approach: for every pair $(predicate, phrase)$ a classification is made. Instances can either receive an argument label as class, or receive a NULL class if the phrase does not fill an argument role of the predicate.

Using every possible predicate/argument pair would result in a very large instance base containing many irrelevant instances. Consequently, training would be very slow. Besides, irrelevant instances may lead to reduced performance of the classifier. In order to keep the number

of instances acceptable, different strategies can be used. A way of reducing the number of instances is to ignore nodes that can never fill an argument role because of their grammatical function. For example nodes with the d-label SVP (verbal particles) are ignored in my system.

Another method is to only consider phrases that are likely to be arguments. For example, van den Bosch et al. only build instances for verb/phrase pairs when the phrase parent is an ancestor of the verb. A strategy more directed towards dependency trees is the one employed by Hacioglu. He considers a dependency relation a potential argument only if it is part of the tree-structured *family* of a predicate. A family is defined as the predicate's parent, children, grandchildren, siblings, siblings' children and siblings' grandchildren with respect to the dependency tree.

5.3.2 Creating predicate/argument pairs

Previous systems were based on the annotation scheme of the PropBank corpus. In PropBank, argument boundaries are defined simply by marking the position of the first and last word of the argument, making argument labeling independent of the syntactic structure. In the CoNLL shared tasks, a similar format was used:

```
WORDS---->  NE--->  POS   PARTIAL_SYNT   FULL_SYNT------>   VS   TARGETS  PROPS------->

The             *   DT    (NP*   (S*        (S(NP*          -    -        (A0*    (A0*
$               *   $        *     *     (ADJP(QP*          -    -           *       *
1.4             *   CD       *     *             *          -    -           *       *
billion         *   CD       *     *             *))        -    -           *       *
robot           *   NN       *     *             *          -    -           *       *
spacecraft      *   NN       *)    *             *)         -    -           *)      *)
faces           *   VBZ   (VP*)    *           (VP*         01   face      (V*)      *
a               *   DT    (NP*     *           (NP*         -    -        (A1*       *
six-year        *   JJ       *     *             *          -    -           *       *
journey         *   NN       *)    *             *          -    -           *       *
to              *   TO    (VP*   (S*          (S(VP*         -    -           *       *
explore         *   VB       *)    *           (VP*         01   explore     *     (V*)
Jupiter     (ORG*)  NNP   (NP*)    *         (NP(NP*)       -    -           *    (A1*
and             *   CC       *     *             *          -    -           *       *
its             *   PRP$  (NP*     *           (NP*         -    -           *       *
16              *   CD       *     *             *          -    -           *       *
known           *   JJ       *     *             *          -    -           *       *
moons           *   NNS      *)    *)            *)))))))   -    -           *)      *)
.               *   .        *     *)            *)         -    -           *       *
```

The first column contains the words of the sentence in their left-to righter order. The "targets" column contains the target predicates. The last columns contain the labeling of the propositions formed by these predicates (*explore* and *face*).

Creating predicate/argument pairs using this format is relatively easy: it is clear from the above example that *a six-year journey to explore Jupiter and its 16 known moons* is a ARG1 argument of the predicate *face* and that *Jupiter and its 16 known moons* is the ARG1 argument the predicate *explore*, although they partially overlap.

Since propositions in the CoNLL format are stored in separate columns, it is always clear to which predicate a particular argument belongs. In my corpus however, it is a bigger challenge to link predicates to their arguments.

In the following Alpino code - in which the sentence *Ik zie je al lopen* is encoded - for example:

```
<node begin="0" cat="top" end="5" rel="top">
  <node begin="0" cat="smain" end="5" rel="--">
    <node pb="ARG0" begin="0" end="1" pos="pron" rel="su" word="ik"/>
    <node pb="PRED" begin="1" end="2" pos="verb" rel="hd" root="zie" word="zie"/>
    <node pb="ARG1" begin="2" end="3" index="1" pos="pron" rel="obj1" word="je"/>
    <node begin="2" cat="inf" end="5" rel="vc">
      <node begin="2" end="3" index="1" rel="su"/>
      <node begin="3" end="4" pos="adv" rel="mod" word="al"/>
      <node pb="PRED" begin="4" end="5" pos="verb" rel="hd" root="loop" word="lopen"/>
    </node>
  </node>
</node>
```

there is one node labeled with the ARG0 tag and there are two predicates in the sentence: *zie* (see) and *lopen* (walk). To which of these predicates the ARG0 node belongs is not stored explicitly. However, it can be derived from the tree structure: since it is a sibling of the verb *zie*, it belongs to this predicate[4].

At first, I only considered predicates paired with their siblings as possible instances. But several exceptions to the "sibling rule" became apparent only during manual annotation. One of which is the position of arguments with d-label SAT (satellite). These exceptions make the task of linking a predicate to its candidate argument challenging.

The instance base I created consists of predicates paired with:

- their siblings
- modal verb siblings of their mother node
- satellite siblings of their mother

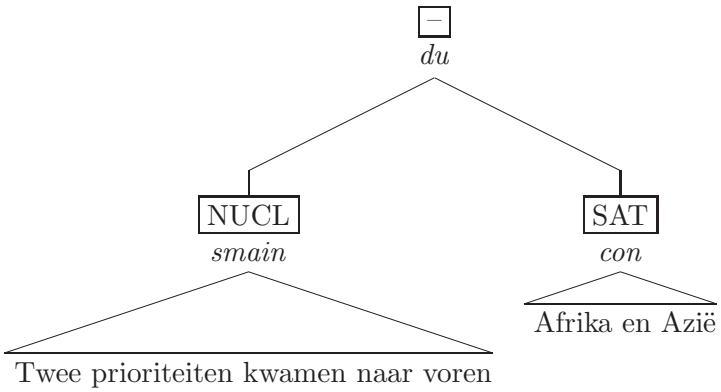
Satellites in Alpino structures are daughters of DU (discourse unit) nodes. Enumerations are a typical example of the satellite phenomenon. Example 5.1 and figure 5.1 illustrate this. Although the node containing *Afrika en Azië* in tree 5.1 is not a sibling of the predicate *naar voren komen* (to surface), it can be considered a candidate argument[5].

(5.1) Twee prioriteiten kwamen naar voren: Afrika en Azië
"Two priorities surfaced: Africa and Asia."

[4]Notice that although the subject *ik* is not a sibling of the predicate *loop* (walk), it linked to that predicate as well by means of co-indexing

[5]The correct PropBank labeling of this type of argument is still subject of debate.

Figure 5.1: Example of a satellite node



5.3.3 The automatic feature extraction process

Once it is clear how instances will be encoded, an instance based can be extracted from the annotated corpus. For example, the following instances can be extracted from the tree in figure 5.2:

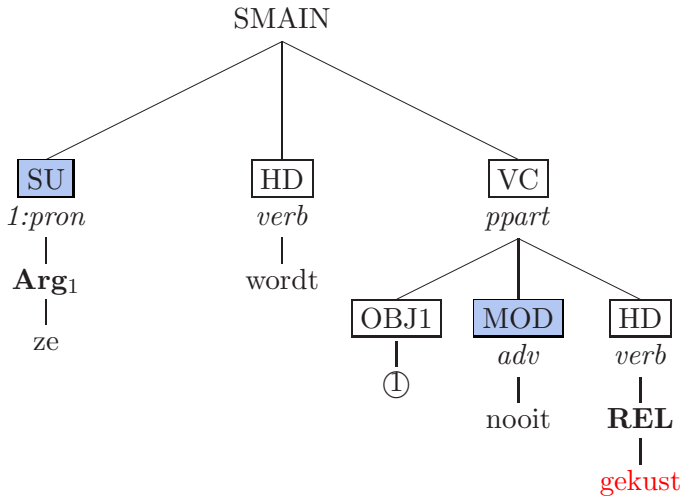
```
kus,passive,#,mod,adv,before,nooit,#,*adv*verb,obj1*mod*hd,mod*,#
kus,passive,#,su,pron,before,ze,verb,pron*verb*ppart,su*hd*vc,su*,ARG1
```

These two example instances consist of 11 features and a target class each. For null values the hash symbol (#) is used. The first instance represents the predicate/argument pair $(kus, nooit)$, the second instance represents the pair (kus, ze). All other possible pairs of verbs and nodes are not considered, since they do not satisfy the requirements for candidate instances formulated in the previous section.

Let's have a closer look at the values of second instance (from left to right):

`kus`	Root form of predicate *gekust*
`passive`	The predicate has a passive voice since it has *wordt* as auxiliary.
`#`	Null, because the argument node (*ze*) does not have a c-label.
`su`	The argument is the subject in the dependency relation.
`pron`	*ze* is a pronoun.
`before`	The argument is positioned before the predicate.
`ze`	The word represented by the argument
`verb`	POS tag of head word in the argument's relation
`pron*verb*ppart`	The left to right chain of c-labels of the argument and its siblings.
`su*hd*vc`	The d-label structure of daughter of this argument's mother
`su*`	d-label (SU) and c-label (empty string) separated by `*`
`ARG1`	The target class

Figure 5.2: "Ze wordt nooit gekust"



The extraction of features from the annotated corpus can be done fully automatically. In order to do this, I developed a FeatureExtractor class in XARA, which is able to convert an Alpino treebank into a C4.5 instance base.

5.4 Classification with TiMBL

I chose to use MBL as a classification method for my automatic SRL task. MBL is based on the k-Nearest Neighbor (k-NN) algorithm. However, k-NN is generally used for numerical data and therefore not suited for typical NLP tasks. Therefore, several variants to k-NN classification algorithm that are suited for NLP classification tasks have been developed and implemented at Tilburg University in collaboration with the University of Antwerp. Different ideas and implementations were combined in a software tool called TiMBL. For a detailed description of the algorithms used in TiMBL, see section 2.6.3 and Daelmans et. al. (2003).

I decided to use TiMBL for my work for a number of reasons:

- Although TiMBL can be applied to any classification task with symbolic or numeric feature values and discrete classes, it was developed specifically with NLP applications in mind.
- The TiMBL software is well documented and easy to use.
- Researchers at the ILK research group of Tilburg university were very helpful in answering my questions. Whats more, they were able to give me expert advise since they participated in the previous CoNLL tasks (van den Bosch et al., 2004; Tjong Kim Sang et al., 2005)

All of the experiments in my project were carried out using TiMBL version 5.1.0.

5.4.1 Training procedure

Typically, classifiers are trained on a train data set and tested on a separate data set. A general rule of thumb is to use 10% of the complete data set as test data and the remaining data as train data. This approach is fine when the train and test sets contain enough examples to get reliable results. My data set of annotated sentences however is quite small (418 sentences, 2501 instances). Splitting this set into separate train and test data would only yield 42 test sentences. This means that infrequent argument types, like ARGM-STR - which occurs only six times in the entire data set - may not occur in the test set at all. Moreover, a training set with a small number of examples can cause *overfitting*. Overfitting occurs when a model does not fit future states (Dunham, 2003). That is, the classifier is able to classify the train instances accurately but does not perform well on new data.

To overcome this data sparsity problem, I trained the classifier using the *leave one out*[6] (LOO) method (`-t leave_one_out` option in TiMBL). With this option set, every data item in turn is selected once as a test item, and the classifier is trained on all remaining items. Accuracy of the classifier is then the number of data items correctly predicted. Using LOO guaranties that the classifier will be tested for every argument type and maximizes the amount of training instances available. This makes it an ideal method for training and testing classifiers with a limited amount of data.

Except for the LOO option, I only used the default TiMBL settings during training. That is, I did not attempt to optimize parameter settings. Because my training database is small, optimizing settings would increase the performance on my data set, but would probably have a negative effect on the performance on new data. In other words: parameter optimization might cause overfitting given the small data set I use.

5.4.2 Evaluation metrics

In this section I will introduce some of the metrics generally used in classification and define some variants to these measures that are relevant for semantic role classification.

The most common performance measure for classification systems is *accuracy*. Classification accuracy is determined by the percentage of instances placed in the correct class. However, the accuracy metric is not very informative about the kind of classification errors made. If certain classification errors are considered more important than others, other metrics are more appropriate, e.g. precision and recall.

Given a class C and an instance i, classification of i can be described in the four ways (Dunham, 2003):

True positive (FP): i is predicted to be in C by the classifier and is actually in it.

[6] LOO is equivalent to n-fold cross validation were n is number of instances.

False positive (FP): i is predicted to be in C by the classifier but is actually not in it.

True negative (TN): i is not predicted to be in C by the classifier and is not actually in it.

False negative (FN): i is not predicted to be in C by the classifier but is actually in it.

To compute class-based performance measures we need two additional numbers: the total number of positive examples P ($P = TP + FN$) and negative examples N ($N = FP + TN$). With these numbers we can compute:

Precision $= \frac{TP}{TP+FP}$, the number of times the classifier has correctly made the decision that some instance has class C, proportional to the total number of times the classifier has assigned class C.

Recall $= \frac{TP}{TP+FN}$, also called *true positive rate* (TPR). The proportional number of times an instance i with class C in the test data has indeed been classified as class C by the classifier.

These measures are class-based; they tell something about the performance of the classifier per class. To measure the performance on the full set, sometimes averaging methods for F-scores are used. TiMBL can compute two different F-score averages: micro-average and macro-average. Micro-average is the weighted average, i.e. each class' F-score is weighted proportionally to the frequency of the class in the test set. Macro-average is the unweighted average.

In the CoNLL shared task, precision, recall and F-score are defined a bit differently, so that they are more suitable to evaluate a classifier's performance on the SRL task.

Given a argument label L and a proposition p, the following metrics were used in the CoNLL evaluation:

OK: Number of correctly predicted non null arguments in p.

OP: Number of over-predicted arguments in p: candidate arguments which are predicted to be an argument but are actually are not.

MS: Number of missed arguments in p.

Precision $= \frac{OK}{OK+OP}$ Proportion of the predicted arguments in p that were predicted correctly.

Recall $= \frac{OK}{OK+MS}$ Proportion of arguments in p that were predicted correctly.

F-Score (Rijsebergen, 1979) $= \frac{2 \times precision \times recall}{precision \times recall}$, harmonic mean of precision and recall. Also known as the F_1 measure, because recall and precision are evenly weighted.

Global precision and recall measures are computed by adding up OK, OP and MS measures of all propositions. The main reason for not using accuracy as a global measure, is that instances can have a "null"-class label. Accuracy measures performance on all class labels, including the null label. This means that if the instance base contains many "null" instances, the classifier can achieve a high accuracy rate by, for example, randomly assigning null to the majority of instances.

For systems participating in the CoNLL shared task, there is an official script available which can compute precision, recall and F-Score automatically. Unfortunately I could not use this script because of the differences in annotation format between my corpus and the CoNLL corpus.

To still be able to avoid the use of the standard TiMBL measures (such as accuracy), I wrote my own script. The source code of this script (in Perl) can be found in appendix B. The script produces precision, recall and F-scores measures by calculating OK, OP and MS in the following way:

OK: Number of instances with predicted label = actual label, and actual label $\neq$ null (correctly predicted arguments)

OP: Number of instances with predicted label $\neq$ actual label and predicted label $\neq$ null (over-predicated arguments).

MS: Number of instances with predicted label $\neq$ actual and actual label $\neq$ null (missed arguments).

so that although it is not entirely fair to compare the results of my system to the CoNLL systems because conditions are quite different, at least the same evaluation measures are used.

5.5 Evaluation

5.5.1 Evaluation of XARA

Before I will turn to the evaluation of the learning system, I will provide a short evaluation of XARA. I performed the evaluation by comparing XARA labeling with the manual labeling.

To produce the evaluation scores I used the same method as with the learning system: I extracted an instance base from the trees labeled by XARA and an instance base from the hand-corrected trees. Both instance bases were then compared by a Perl script that produced precision, recall and F_1 scores.

Table 5.1 shows the performance of XARA on the 418 sentences (2501 instances) in the training set.

The precision score gives an indication of how well ("precise") the rules defined in XARA can label arguments. XARA's score (64,85%) is substantially higher than that of the baseline system of the CoNLL-2005 task, which has a precision of 50%.

Table 5.1: Initial XARA labeling compared to manual corrected labeling

Precision	Recall	$F_{\beta=1}$
64,85%	41,74%	50,79

Recall indicates the "completeness" of the labeling. The low recall score (in comparison to the precision score) can be contributed to the fact that XARA only labels a subset of PropBank arguments (ARGM-DIRs are not labeled for example), which causes a relatively high missed argument count and hence a low recall score.

The F-score gives an indication of the overall performance of the system. XARA's F-score is notably lower than that of the learning system, which has a F_1 of 65,29. The low F-score can be attributed to XARA's low recall score alone, since there is little difference in precision between both systems.

I think that in order to asses the importance of the precision and recall measures, it is a good idea to look at the experiences of human annotators.

According the annotator who performed the corrections on the automatic annotated training corpus, lower numbered arguments were labeled accurately by XARA, thereby providing a convenient starting point for labeling/correcting the other arguments. On the other hand, correcting arguments labeled incorrectly by XARA (predominately ARGMs) requires more time than adding labels to unannotated data. This implies that for a bootstrapping system, precision is more important than recall.

The overall impression of the annotator was that although XARA's automatic labeling is far from perfect, annotation can be performed faster on a corpus initially labeled by XARA, than on an unannotated corpus.

5.5.2 Evaluation of TiMBL classification

Table 5.2 shows TiMBL classification performance on the train data. Training was performed on the the baseline feature set with the LOO method.

Table 5.2: TiMBL performance on training data using the LOO method

Precision	Recall	$F_{\beta=1}$
65,05%	65,53%	65,29

Since all argument types occur in the training data, the learning systems is able to achieve a higher recall than XARA. Precision scores of both systems on the other hand are quite

similar.

Table 5.3 shows the per-class performance of the classifier (as computed by TiMBL).

Table 5.3: Class based results on the baseline feature set, provided by TiMBL

Argument label	Precision	Recall	F-score
null	67,36%	65,11%	66,21
Arg0	74,87%	80,79%	77,72
Arg1	79,10%	79,22%	79,16
Arg2	56,52%	56,25%	56,39
Arg3	28,57%	17,39%	21,62
Arg4	50,00%	11,11%	18,18
ArgM-ADV	50,00%	57,14%	53,33
ArgM-CAU	48,00%	34,29%	40,00
ArgM-DIR	33,33%	50,00%	40,00
ArgM-DIS	68,42%	73,86%	71,04
ArgM-EXT	41,18%	29,17%	34,15
ArgM-LOC	43,33%	39,69%	41,43
ArgM-MNR	40,38%	33,33%	36,52
ArgM-NEG	67,74%	77,78%	72,41
ArgM-PNC	47,83%	57,89%	52,38
ArgM-PRD	45,00%	26,47%	33,33
ArgM-REC	75,76%	83,33%	79,37
ArgM-STR	0,00%	0,00%	0,00
ArgM-TMP	50,43%	54,42%	52,35

Some observations can be made regarding the class-based results:

- A sharp drop in recall for the higher numbered arguments can be observed: recall for ARG0 is 80,79%, recall for ARG4 is only 11,11%. This can be contributed in part to the low number of training examples with these labels in the corpus (see table 4.2).
- The argument label with the highest F-score is ARGM-REC. This is probably due to the fact that the only information needed to assign this label is the head word feature + POS of the head word, which makes classification of ARGM-RECs relatively easy.
- One would expect a better performance on the lower numbered arguments (assuming that the SU and OBJ1 labels are assigned accurately by the Alpino parser). Bearing the annotators comment that ergativity is the main source of errors in XARA in mind, I expect that performance on lower numbered arguments can be increased if a feature is added that indicates whether the predicate is ergative or unergative.

5.6 Discussion and conclusions

The results reported here provide a first insight into the possibilities and problems of semantic role classification based on Alpino dependency structures.

The learning system performed reasonably well on the baseline feature set and manually annotated data. It showed a higher recall than XARA, which indicates that its labeling is more complete than XARA's, i.e. a larger set of argument types is covered. This was in line with my expectations, although I did not expect the precision scores of both systems to be as similar as they turned out to be. I contribute the similar precision scores of both systems to the fact that they use - to a large extent - the same information from the dependency tree to assign roles.

To put things in perspective, I think it is a good idea to look at the performance of previously developed systems, such as the CoNLL systems.

But first, I want to point out that my system and the CoNLL systems are not completely comparable. The data format, data size and evaluation methods (separate test/train/develop sets in CoNLL versus LOO in my approach) are all different. Furthermore, none of the CoNLL systems was based on features extracted from dependency structures. To get results that are better comparable to the CoNLL systems, some improvements can be made in the future:

- The candidate argument selection method I used was not perfect. In the conversion process from Alpino trees to the instance base, 10% of the labeled arguments was lost: of the 2.270 labeled arguments in the training corpus, 2051 appeared in the instance base. A more accurate method of building the instance base would probably solve this problem. However, I think this can be quite a challenging task. Another solution would be to use a different annotation format altogether in which predicate/argument structures are store explicitly, such as the CoNLL shared task format.
- Because of its small size, the class distribution in my corpus was not well balanced. This can be solved obviously by using a larger training set. This requires a larger set of manually annotated data, which hopefully becomes available in a later stage of the D-Coi project.

Nevertheless, to give an idea of CoNLL-05 system performances: the best performing system reached a F_1 of 80. There were seven systems with a F_1 performance in the 75-78 range, seven more with performances in the 70-75 range and five with a performance between 65 and 70.

A system that did not participate in the CoNLL task, but still provides interesting material for comparison since it is based on dependency structures, is the Hacioglu (2004) system. This system scored 85,6% precision, 83,6% recall and 84,6 F_1, which is even higher than the best results published so far on the PropBank data sets (Pradhan et al., 2005): 84% precision, 75% recall and 79 F_1.

My system would rank in the lower regions of the CoNLL-05 results. I think that this certainly is an encouraging result, considering that I

- used a very basic set of features

- did not attempt to optimize TiMBL parameters
- did not perform any pre-/post-processing steps[7].

These are exactly the issues that were addressed in previous work concerning possible improvements of SRL classifiers.

- Previous research (Pradhan et al., 2005) has shown that performance can be improved by using a two-step process for classification, consisting of first identifying arguments, e.g. with a binary NULL versus NON-NULL classifier and then classification of non-null arguments.
- The optimal set of features can be found by employing bi-directional hill climbing (van den Bosch et al., 2004). There is a wrapper script (Paramsearch) available to be used with TiMBL and several other learning systems that implements this approach[8].
- Iterative deepening (ID) can be used as a heuristic way of searching for optimal algorithmic parameters (van den Bosch et al., 2004). ID starts with a large pool of experiments, each with unique combination of algorithmic parameter settings. Each set of settings is applied to a small portion of the training data and held-out data. This is done iteratively, in every iteration the best-performing settings are retained, with an exponentially growing amount of training and held-out data. The process stops when all training data is used or one best setting is left. Selection of the best setting is based on classification accuracy on the held-out data.
- In general, system performance can be improved by applying one or more pre- and post-processing steps. In most systems this is done by applying simple ad-hoc rules (Carreras and Màrquez, 2005). A more elaborate approach was introduced by Tjong Kim Sang et al. (2005). His approach is based on a lexicon of semantic role labeling patterns of Arg0-Arg5 arguments of verbs on the basis of the entire training corpus of PropBank. Labeling of new instances is compared to the role labeling pattern of the same verb at the smallest Levenstein distance. For example, if the test sentence contains the pattern *emphasize Arg0 PRED Arg1 Arg0* and the closest lexicon item is *emphasize Arg0 PRED Arg1*, this (probable) error is corrected by deleting the last Arg0.

Besides these improvements, I think there are a lot of other issues left to make interesting future research. For example, it would be interesting to see how the classifier would perform on larger collections of data and on new genres of data. Pradhan et al. (2005) for example, found a significant drop in performance when their system was tested on texts from different sources than the system was trained on.

To conclude, I hope that research on automatic SRL of Alpino dependency trees will continue, so that some of the issues mentioned in the section can be solved. Furthermore, I hope that findings presented in this chapter will prove to be useful in future research.

[7] Although my candidate argument selection procedure can be regarded as a post-processing step.
[8] URL: `http://ilk.uvt.nl/software.html#paramsearch`.

Chapter 6

Conclusion

A wide variety of issues concerning automatic SRL have been discussed in this thesis. After an introductory chapter on semantic role labeling (chapter 2), I concentrated on the specifics of labeling Dutch texts with semantic roles automatically. First, a novel rule-based bootstrapping approach was introduced (chapters 3 and 4). Thereafter I turned to a machine learning approach for automatic SRL (chapter 5) based on memory based learning techniques. Both methods were successfully applied to a small dependency treebank: XARA achieved an F-score of 50,79, the TiMBL classifier an F-score of 65,29. The fact that XARA's F-score is lower than that of the learning system, can be contributed to the relatively high number of arguments that XARA missed (expressed in the recall measure).

The research I performed can be characterized as a feasibility study; there are many improvements possible in the current version of XARA and the learning system. Some suggestions for improvement were given in chapter 5. Apart from these improvements, some of the following issues may require attention in future research:

- During manual annotation, several issues concerning the applicability of PropBank guidelines to Dutch texts became apparent. These issues are in general of a linguistic nature. For example, the issue of how to label satellite constituents has yet to be resolved.
- Manual annotation during my project was performed by one annotator only. This probably yields a less precisely annotated corpus than in the PropBank project, where differences in annotation between two annotators were resolved in a so called *adjudication phase* (see chapter 2).
- The Alpino XML format was designed to store dependency trees only. To store semantic role labeling information as well, some additions to the data format specification are needed. The extension proposed in this thesis has one major drawback: it is not always easy to determine to which predicate a certain argument belongs (see section 5.3.2). Maybe this issue can be resolved in future research, otherwise another storage structure needs to be devised.
- XARA's performance can probably be improved by altering the current rule set and

by adding new rules. Current rules can be improved as more data becomes available in the dependency trees (for example a more elaborate set of POS tags). New rules can be created based on corpus evidence, either manually of automatically. If new roles cover a larger portion of the PropBank argument types, the recall of XARA is likely to increase, bringing its performance closer to that of the TiMBL classifier.

- Performance of the learning system can probably be improved by altering the feature set, adjusting algorithmic parameters and applying pre-/post-processing steps. But in order to this, more data needs to be annotated manually, since extensive experimentation on the current (small) data set would cause overfitting. Hopefully, more manually annotated data becomes available in a later stage of the D-Coi project.

- I concentrated on the PropBank approach to role labeling. As mentioned earlier, this does not make PropBank the best method for semantic annotation. Maybe FrameNet annotation, a merging approach or abstract roles are more suitable to be used in the final version of the D-Coi corpus.

  I think that the automatic methods presented here are, at least to a certain extend, applicable to annotation methods other than PropBank. Further research is needed to determine whether this assumption is true.

- The proposed mapping from dependency structures to semantic roles (discussed in chapter 3) can be improved with the help of findings in related research, such as currently running research on QA for dutch dependency trees. Other interesting research, related to my rule-based approach, was presented by Andreas Wagner in his Ph.D. thesis *Learning thematic role relations for lexical semantic nets* (Wagner, 2005). The goal of his work was to develop a language independent approach for learning thematic role relations from text by means of statistical corpus analysis. Although he concentrates on thematic roles, some aspects of his work can be relevant for PropBank labeling as well. For example, the heuristics he developed to automatically assign semantic roles to syntactic argument groups can be used to create new rules for XARA. Unfortunately, I was not able to examine Wagner's work in detail within the time-frame of my project.

Although quite a few challenges remain for further research, I think the results presented in this thesis are at least promising. Apparently, automatic labeling of Dutch texts is possible with the help of existing Dutch linguistic resources. Performance scores are even within range of some of the (English) systems participating in the CoNLL-05 shared task. In the light of these encouraging results, I hope that research on this subject will continue in the future.

Bibliography

Olga Babko-Malaya. 2005. PropBank Annotation Guidelines. URL: `http://verbs.colorado.edu/mpalmer/palmer/projects/ace.html`.

C. F. Baker, C.J. Fillmore and J.B. Lowe. 1998. The Berkeley FrameNet project. *Proceedings of COLING-ACL 1998*, Montéal, Canada.

L. van der Beek, G. Bouma, R. Malouf, and G. van Noord. 2002. The alpino dependency treebank. In *Computational Linguistics in the Netherlands (CLIN) 2001*, Twente University.

Antal van den Bosch, Sander Canisius, Walter Daelemans, Iris Hendrickx and Erik Tjong Kim Sang. 2004. Memory-based semantic role labeling: Optimizing features, algorithm, and output. In H.T. Ng and E. Riloff (Eds.), *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004)*, Boston, MA, USA.

G. Bouma, G. van Noord and R. Malouf. 2000. Alpino: wide-coverage computational analysis of Dutch. *Computational Linguistics in the Netherlands 2000*. Selected Papers from the 11th CLIN Meeting.

G. Bouma and G. Kloosterman. 2002. Querying dependency treebanks in XML. In *Proceedings of the Third international conference on Language Resources and Evaluation (LREC)*, Gran Canaria, 2002.

T. Braj, J. Paoli, C. M. Sperberg-McQueen, E. Maler and, F. Yergeau. 2006. Extensible Markup Language (XML) 1.0 (Fourth Edition). *W3C Recommendation 4 February 2004*. URL: `http://www.w3.org/TR/2006/REC-xml-20060816/`.

A. Burchardt, K. Erk, A. Frank, A. Kowalski, S. Pado and M. Pinkal. 2006. The SALSA Corpus: a German Corpus Resource for Lexical Semantics. To appear in *Proceedings of LREC 2006*, Genoa, Italy.

Xavier Carreras and Lluís Màrquez. 2005. Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling. In *proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2005)*, Boston, MA, USA.

J. Clark, S. DeRose. 1999. XML Path language (XPath). *W3C Recommendation 16 November 1999*. URL: `http://www.w3.org/TR/xpath`

W. Daelemans, A. Van den Bosch. 1992. Generalization performance of backpropagation learning on a syllabification task. In M. F. J. Drossaers and A. Nijholt, editors, *Proceedings of TWLT3: Connectionism and Natural Language Processing*, pages 27-37, Enschede, Twente University.

W. Daelemans, J Zavrel, K. van der Sloot, and A. van den Bosch. 2004. *TiMBL: Tilburg Memory Based Learner, version 5.1, Reference Guide*. ILK Technical Report Series 04-02.

Anthony R. Davis. 2001. *Linking by Types in the Hierarchical Lexicon*. University of Chicago Press.

David Dowty. 1991. Thematic proto-roles and argument selection. *Language* 67(3), pages 547-619.

Margaret H. Dunham. 2003. *Data mining, introductory and advanced topics*. Pearson Education Inc., Upper Saddle River, New Jersey, USA.

Michael Ellsworth, Katrin Erk, Paul Kingsbury, and Sebastian Pado. 2004. PropBank, SALSA, and FrameNet: How design determines product. In *Proceedings of the LREC 2004 Workshop on Building Lexical Resources from Semantically Annotated Corpora*, Lisbon, Portugal.

Charles J. Fillmore. 1968. The case for case. In E. Bach & R.Harms, eds, *Universals in Linguistic Theory*, Holt, Rinehart & Winston, New York, pages 1-90.

Charles J. Fillmore. 1971. Some problems for case grammar. *Universals in Linguistic Theory*, Holt, Rinehart and Winston, New York, pages 1-88.

Daniel Gildea and Daniel Jurafsky. 2000. Automatic Labeling of Semantic Roles. In *Proceedings of the 38th Annual Conference of the Association for Computational Linguistics (ACL-00)*, pages 512−520, Hong Kong, October 2000.

Daniel Gildea and Daniel Jurafsky. 2002. Automatic Labeling of Semantic roles. *Computational Linguistics*, 28:3, pages 245−288.

Daniel Gildea and Martha Palmer. 2002. The Necessity of Syntactic Parsing for Predicate Argument Recognition. In *Proceedings of the 40th Annual Conference of the Association for Computational Linguistics (ACL-02)*, Philadelphia, PA, USA.

Ana-Maria Giuglea and Alessandro Moschitti. 2004. Knowledge Discovery using FrameNet, VerbNet and PropBank. In *Proceedings of the Workshop on Ontology and Knowledge Discovering at ECML 2004*, Pisa, Italy.

Jeffrey Gruber. 1965. *Studies in Lexical Relations*. PhD thesis, Indiana University Linguistics Club.

Kadri Hacioglu. 2004. Semantic Role Labeling Using Dependency Trees. In *Proceedings of COLING-04*, August 2004.

H. Hoekstra, M. Moortgat, I. Schuurman T. van der Wouden. Syntactic Annotation for the Spoken Dutch Corpus Project (CGN). In W. Daelemans, K. Sima'an, J. Veenstra J. Zavrel (Eds.), *Computational Linguistics in the Netherlands 2000*. pages 73-87.

Ray Jackendoff. 1990. *Semantic Structures*. MIT Press, Cambridge, MA, USA.

T. Kakkonen. 2005. Dependency Treebanks: Methods, Annotation Schemes and Tools. In *Proceedings of the 15th Nordic Conference of Computational Linguistics (NODALIDA 2005)*, pages 94-104. Joensuu, Finland.

K. Kipper, H. T. Dang and M. Palmer. 2000. Class-based construction of a verb lexicon. In *Proceedings of the seventeenth national conference on artificial intelligence (AAAI) 2000*. Austin, Texas, July 2000.

Taku Kudo. 2003. *Machine Learning and Data Mining Approaches to Practical Natural Language Processing*. Doctor's Thesis submitted to Graduate School of Information Science, Nara Institute of Science and Technology.

Beth Levin. 1993. *English Verb Classes and Alternations: A Preliminary Investigation*. University of Chicago Press, Chicago, USA.

M.P. Marcus, B. Santorini, M.A. Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics* 19(2), pages 313-330.

G. Miller. 1995. Wordnet: A lexical database. *Communication of the ACM 38*, 11 (1995), pages 39-41.

Paola Monachesi and Jantine Trapman. 2006. Merging FrameNet and PropBank in a corpus of written Dutch. In *Proceedings of the workshop Merging and layering linguistic information*. Workshop held in conjunction with LREC 2006, Genoa, Italy, 23 May 2006. pages 32-39.

M. Moortgat, I. Schuurman, T. van der Wouden. 2000. CGN Syntactische Annotatie. Internal report Corpus Gesproken Nederlands.

K. Ohara, S. Fujii, T. Ohori, R. Suzuki, H. Saito, and S. Ishizaki. 2004. The Japanese FrameNet project: An introduction. In *Proceedings of the Workshop on Building Lexical Resources from Semantically Annotated Corpora* at LREC 2004.

Martha Palmer, Daniel Gildea, Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1).

Carl Pollard and Ivan A. Sag. 1987. *Information-based syntax and semantics. Volume 1. Fundamentals*. CLSI Lecture Notes 13. Stanford, CA: Center for the Study of Language and Information.

Sameer Pradhan, Kadri Hacioglu, Valeri Krugler, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2005. Support vector learning for semantic argument classification. *Machine Learning Journal*, 60:1-3, pages 11-39.

C. v. Rijsbergen. 1979. *Information retrieval*. Buttensworth, London. URL: `http://www.dcs.gla.ac.uk/Keith/Preface.html`.

Josef Ruppenhofer, Michael Ellsworth, Miriam R. L. Petruck, Christopher R. Johnson. 2005. *FrameNet: Theory and Practice*. URL: `http://framenet.icsi.berkeley.edu`.

C. Subirats and M. Petruck. 2003. Surprise! Spanish FrameNet!. In *Proceedings of the Workshop on Frame Semantics at the XVII. International Congress of Linguists*, Prague, Czechoslovakia.

Robert Swier and Suzanne Stevenson. 2004. Unsupervised semantic role labelling. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 95102, Barcelona, Spain.

Erik Tjong Kim Sang, Sander Canisius, Antal van den Bosch, and Toine Bogers. 2005. Applying spelling error correction techniques for improving semantic role labeling. In *Proceedings of the Ninth Conference on Natural Language Learning (CoNLL-2005)*, June 29-30, 2005, Ann Arbor, MI, USA.

Vladamir Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, USA.

Andreas Wagner. 2005. Learning Thematic Role Relations for Lexical Semantic Nets. Dissertation, University of Tübingen.

F. Xia and M. Palmer. 2001. Converting dependency structures to phrase structures. In *Proceedings of the First international Conference on Human Language Technology Research*, San Diego, March 18 - 21, 2001.

Appendix A

CGN syntactic labels

The following tables contain the category (c-) and dependency (d-) labels used in the CGN and D-Coi corpora. A detailed description of the CGN annotation can be found in *Syntactic Annotation for the Spoken Dutch Corpus Project* (Hoekstra et al., 2001).

Table A.1: c-labels in CGN syntactic annotation

SMAIN	main clause
SSUB	subordinate clause (verb-final)
SV1	any sentence with a sentence-initial inflected verb
INF	short infinitive group
PPART	past/passive participle group
PPRES	present participle group
CP	clause headed by any kind of complementizer
MWU	merged-word-unit (used for complex numbers and names)
TI	long infinitive group
OTI	long infinitive group headed by *om* (~ for to)
AHI	long infinitive group headed by *aan het* (a Dutch progressive form)
ADVP	adverbial phrases
DETP	determiner group
AP	adjectival group
PP	prepositional group
NP	nominal group
SVAN	subordinate clause headed by *van*
REL	relative clause
WHREL	headless relative
WHQ	WH-question
WHSUB	embedded WH-question
CONJ	conjunction
DU	discourse-unit
LIST	asyndetic conjunction
COMPP	various comparative constructions

Table A.2: d-labels in CGN syntactic annotation

HD	head
HDF	second part of a circumposition
DET	determiner
PART	partitive
SU	subject
SUP	provisional subject
OBJ1	direct or first object
POBJ1	provisional direct or first object
OBJ2	secondary object
SE	obligatory reflexive object
SVP	verbal particle
PREDC	predicative complement
PC	prepositional complement
VC	verbal complement
LD	locational or directional complement
ME	measure complement
CMP	grammatical complementizer
RHD	complementizer heading (headless) relative
WHD	complementizer heading WH question
BODY	body of subordinate clause
PREDM	secondary predicate
MOD	general label for modifiers
CRD	coordinator
CNJ	member of conjunction
NUCL	nuclear clause
SAT	satelite
TAG	tag
DP	any part of a DU
PRT	any part of a particle group
OBCOMP	comparative complement
APPOS	apposition
LP	any part of a LIST
DLINK	discourse particles joining discourse fragments
MWP	any part of a MWU

Appendix B

Perl code of the evaluation script

Listing B.1: Perl code of the evaluation script

```
#!/usr/bin/perl

##############################################################################
#  srl-eval.perl : evaluation program semantic role classification
#
#  Author  : Gerwert Stevens
#  Contact : gerwert@gmail.com
#
#  Version : September 1th, 2006
#
##############################################################################

my $arg1 = $ARGV[0];
my $arg2 = $ARGV[1];

if ((!$arg1) || (!$arg2)) {
    print "Computes precision, recall and f-score from two C4.5 data
        files\n";
    print "Usage: srl-eval <gold_instances> <predicted_instances>\n";
    exit(0);
}

open(GP, "$arg1");
open(PP, "$arg2");

chomp(@lines_gp = <GP>);
chomp(@lines_pp = <PP>);

if (@lines_gp!=@lines_pp) {
        print "Error: input files do not have same number of lines!";
        exit(0);
}

#   ok  :  number of correctly predicted args
#   ms  :  number of missed args
```

```
#   op   :  number of over-predicted
#   cc   :  number of correctly classified args (ok including null args)

my $ok = 0;
my $ms = 0;
my $op = 0;
my $cc = 0;

for ( $i=0; $i<@lines_gp; $i++) {
        $gp_line = @lines_gp[$i];
        $pp_line = @lines_pp[$i];
        @gp_fields = split(/,/, $gp_line);
        @pp_fields = split(/,/, $pp_line);
        $gp_label = pop(@gp_fields);
        $pp_label = pop(@pp_fields);
        if ( ($gp_label eq $pp_label) && ($gp_label ne "#")) {
                $ok++;
        }

        if ( ($pp_label ne $gp_label) && ($pp_label ne "#")) {
                $op++;

        }

        if ( ($pp_label ne $gp_label) && ($gp_label ne "#")) {
                $ms++;
        }

        if ($pp_label eq $gp_label) {
                $cc++;
        }
}
printf("%6.2f%% precision\n%6.2f%% recall\n%6.2f%% F1\n",precrecf1($ok,
    $op, $ms));
printf("%6.2f%% classification accuracy \n", 100*($cc/@lines_gp));

# computes precision, recall and F1 measures
# (this function was orginally written by Xavier Carreras and Lluis
    Marquez for the CoNLL-2005 Shared Task)
sub precrecf1 {
    my ($ok, $op, $ms) = @_;

    my $p = ($ok + $op > 0) ? 100*$ok/($ok+$op) : 0;
    my $r = ($ok + $ms > 0) ? 100*$ok/($ok+$ms) : 0;

    my $f1 = ($p+$r>0) ? (2*$p*$r)/($p+$r) : 0;

    return ($p,$r,$f1);
}
```

Appendix C

XARA's rule definitions

Below, the part of the XARA's Java source code that contains the rule definitions is shown. Each rule consists of an XPath expression and a target label. For a thorough explanation of the rule definitions in this code, see chapter 4.

Listing C.1: Rule definitions in XARA

```
/* ------------------------------------
 * Active voice + infinitives
 * (no auxiliaries )
 * ------------------------------------
*/

Tagger t = new Tagger();
t.setContext(doc, "//node[@cat='smain' or @cat='ssub' or @cat='sv1'" +
              " or @cat='inf']" +
              "[not(./node[@rel='vc' and (@cat='predc' or @cat='ppart' or
                  @cat='conj' or @cat='inf')])]");
// numbered arguments
t.addRule(new Rule("./node[@rel='hd']", "PRED"));
t.addRule(new Rule("./node[@rel='su']", 0));
t.addRule(new Rule("./node[@rel='obj1']", 1));
t.addRule(new Rule("./node[@rel='obj2']", 2));
// complements
t.addRule(new Rule("./node[@rel='vc']", -1));
t.addRule(new Rule("./node[@rel='pc' and @cat='pp']", -1));
t.addRule(new Rule("./node[@rel='predc'][@cat='np' or @cat='ap' or @cat='
    pp']", -1));
t.addRule(new Rule("./node[@rel='me']", -1));
// modifiers
t.addRule(new Rule("./node[(@word='niet' or @word='nooit' or @word='geen'
     or @word='nergens')" +
                  " and (@pos='adv')]", "ArgM-NEG"));
t.addRule(new Rule("./node[@rel='mod' and @cat='oti']", "ArgM-PNC"));
t.addRule(new Rule("./node[@rel='predm']", "ArgM-PRD"));
t.addRule(new Rule("./node[@rel='ld' ]", "ArgM-LOC"));
t.addRule(new Rule("./node[@pos='pron' and " +
```

```
                    "(@word='mezelf' or @word='zichzelf' or @word='
                        hemzelf' or " +
                    "@word='haarzelf')]", "ArgM-REC"));
t.tag();


/* ---------------------
 * Perfect tense
 * ---------------------
 */
t = new Tagger();
t.setContext(doc, "//node[@cat='ppart']" +
                  "[preceding-sibling::node[@rel='hd'][@root='ben' or
                     @root='heb']]");
// numbered arguments
t.addRule(new Rule("./node[@rel='hd']", "PRED"));
t.addRule(new Rule("./node[@rel='su']", 0));
t.addRule(new Rule("./node[@rel='obj1']", 1));
t.addRule(new Rule("./node[@rel='obj2']", 2));
// complements
t.addRule(new Rule("./node[@rel='vc']", -1));
t.addRule(new Rule("./node[@rel='pc' and @cat='pp']", -1));
t.addRule(new Rule("./node[@rel='predc'][@cat='np' or @cat='ap' or @cat='
    pp']", -1));
t.addRule(new Rule("./node[@rel='me']", -1));
// modifiers
t.addRule(new Rule("./node[(@word='niet' or @word='nooit' or @word='geen'
     or @word='nergens')" +
                    " and (@pos='adv')]", "ArgM-NEG"));
t.addRule(new Rule("./node[@rel='mod' and @cat='oti']", "ArgM-PNC"));
t.addRule(new Rule("./node[@rel='predm']", "ArgM-PRD"));
t.addRule(new Rule("./node[@rel='ld' ]", "ArgM-LOC"));
t.addRule(new Rule("./node[@pos='pron' and " +
                    "(@word='mezelf' or @word='zichzelf' or @word='
                        hemzelf' or " +
                    "@word='haarzelf')]", "ArgM-REC"));
t.tag();


/* ---------
*  Passives
* ---------
*/
t = new Tagger();
t.setContext(doc, "//node[@cat='ppart']" +
                      "[preceding-sibling::node[@rel='hd' and (@root='
                          word')]]");
t.addRule(new Rule("./node[@rel='hd']", "PRED"));
t.addRule(new Rule("//node[@rel='mod' and @cat='pp']/node[@rel='obj1'][
    preceding-sibling::node[@rel='hd' and @word='door']]", "Arg0"));
t.addRule(new Rule("./node[@rel='obj1']", 1));
t.addRule(new Rule("./node[@rel='obj2']", 2));
// complements
```

```
t.addRule(new Rule("./node[@rel='vc']", -1));
t.addRule(new Rule("./node[@rel='pc' and @cat='pp']", -1));
t.addRule(new Rule("./node[@rel='predc'][@cat='np' or @cat='ap' or @cat='
    pp']", -1));
t.addRule(new Rule("./node[@rel='me']", -1));
// modifiers
t.addRule(new Rule("./node[(@word='niet' or @word='nooit' or @word='geen'
     or @word='nergens')" +
                    " and (@pos='adv')]", "ArgM-NEG"));
t.addRule(new Rule("./node[@rel='mod' and @cat='oti']", "ArgM-PNC"));
t.addRule(new Rule("./node[@rel='predm']", "ArgM-PRD"));
t.addRule(new Rule("./node[@rel='ld' ]", "ArgM-LOC"));
t.addRule(new Rule("./node[@pos='pron' and " +
                   "(@word='mezelf' or @word='zichzelf' or @word='
                       hemzelf' or " +
                   "@word='haarzelf')]", "ArgM-REC"));

/* ------
 * Modals
 * ------
*/
t = new Tagger();
t.setContext(doc, "//node[@cat='smain' or @cat='ssub' or @cat='sv1' or
    @cat='ppart' or @cat='inf']" +
                      "[child::node[@cat='inf']][child::node[@rel='hd'
                          and (@root='hoef' or" +
                      " @root='kan' or @root='mag' or @root='wil' or" +
                      " @root='heb' or @root='ben' or @root='moet')]]");
t.addRule(new Rule("./node[@rel='hd' and @pos='verb']", "ArgM-MOD"));
t.tag();
```