Controlled English for Knowledge Representation

Doctoral Thesis

for the
Degree of a Doctor of Informatics

at the
Faculty of Economics, Business Administration
and Information Technology of the
University of Zurich

by
Tobias Kuhn

from
Zurich

Accepted on the recommendation of
Prof. Dr. Michael Hess
Dr. Norbert E. Fuchs

2010

The Faculty of Economics, Business Administration and Information Technology of the University of Zurich herewith permits the publication of the aforementioned dissertation without expressing any opinion on the views contained therein.

Zurich, 14 April 2010

The Vice Dean of the Academic Program in Informatics: Prof. Dr. H. C. Gall

Abstract

Knowledge representation is a long-standing research area of computer science that aims at representing human knowledge in a form that computers can interpret. Most knowledge representation approaches, however, have suffered from poor user interfaces. It turns out to be difficult for users to learn and use the logic-based languages in which the knowledge has to be encoded. A new approach to design more intuitive but still reliable user interfaces for knowledge representation systems is the use of controlled natural language (CNL). CNLs are subsets of natural languages that are restricted in a way that allows their automatic translation into formal logic. A number of CNLs have been developed but the resulting tools are mostly just prototypes so far. Furthermore, nobody has yet been able to provide strong evidence that CNLs are indeed easier to understand than other logic-based languages.

The goal of this thesis is to give the research area of CNLs for knowledge representation a shift in perspective: from the present explorative and proof-of-concept-based approaches to a more engineering focused point of view. For this reason, I introduce theoretical and practical building blocks for the design and application of controlled English for the purpose of knowledge representation. I first show how CNLs can be defined in an adequate and simple way by the introduction of a novel grammar notation and I describe efficient algorithms to process such grammars. I then demonstrate how these theoretical concepts can be implemented and how CNLs can be embedded in knowledge representation tools so that they provide intuitive and powerful user interfaces that are accessible even to untrained users. Finally, I discuss how the understandability of CNLs can be evaluated. I argue that the understandability of CNLs cannot be assessed reliably with existing approaches, and for this reason I introduce a novel testing framework. Experiments based on this framework show that CNLs are not only easier to understand than comparable languages but also need less time to be learned and are preferred by users.

Acknowledgements

The research project described in this thesis was performed at the Department of Informatics and the Institute of Computational Linguistics of the University of Zurich. It was funded by the research grant programs ("Forschungskredit") 2006 and 2008 of the University of Zurich, and by the Swiss State Secretariat for Education and Research within the EU research project REWERSE.

I would like to thank my doctoral advisor Michael Hess for giving me the opportunity to write my doctoral thesis in the computational linguistics group, and for his constant support. I also want to thank my supervisor Norbert E. Fuchs for the advice, discussions, and practical help he has provided me with during the last five years. I am very thankful for his extraordinary commitment.

Many thanks go to my colleagues of the Institute of Computational Linguistics for the pleasant and creative atmosphere and for the countless serious and not-so-serious lunch discussions. Particularly, the discussions with Alexandra Bünzli, Stefan Höfler, Kaarel Kaljurand, Cerstin Mahlow and Michael Piotrowski have influenced this thesis. I also want to thank all the secretaries and technicians for their indispensable help in administrative and technical issues.

I would like to acknowledge the Institute for Empirical Research in Economics of the University of Zurich for taking care of the recruitment of the participants for one of the user experiments I had to perform. I am especially grateful to Alain Cohn for fruitful discussions on the design of the experiment. Furthermore, I would like to thank everyone who participated in one of my experiments.

George Herson, Marc Lutz and Philipp Müller deserve special thanks for proofreading this thesis (or parts thereof) and for their valuable comments and suggestions.

Finally, I would like to thank my girlfriend and my family for their support and encouragement, and generally for everything they have done for me over the years.

Contents

List of Abbreviations

ACE	Attempto Controlled English
APE	Attempto Parsing Engine
CLOnE	Controlled Language for Ontology Editing
CNL	Controlled Natural Language
Codeco	Concrete and Declarative Grammar Notation for Controlled Natural Languages
CPL	Computer Processable Language
DCG	Definite Clause Grammar
DRS	Discourse Representation Structure
HPSG	Head-Driven Phrase Structure Grammar
NLP	Natural Language Processing
MLL	Manchester-like Language
OWL	Web Ontology Language
PCR	Prediction, Completion, Resolution
PENG	Processable English
REWERSE	Reasoning on the Web with Rules and Semantics
SBVR	Semantics of Business Vocabulary and Business Rules

CHAPTER 1

Introduction

> “ Knowledge is power. ”
>
> Francis Bacon [9]

> “ I am convinced that the unwritten knowledge scattered among men of different callings surpasses in quantity and in importance anything we find in books, and that the greater part of our wealth has yet to be recorded. ”
>
> Gottfried Leibniz [97]

As the quotations of these famous men show, knowledge is considered one of the greatest goods of mankind, but also something fragile and hard to record. Recording and representing human knowledge with the help of computers is the topic of this thesis. Many attempts on this problem failed in the past, maybe because too little attention was paid to how human knowledge has always been represented: in natural language.

Controlled natural languages (CNLs) have been proposed to allow humans to express their knowledge in a natural way that can also be understood by computers. CNLs are artificially defined subsets of natural language — e.g. English — with the goal to improve the communication between humans and computers.

The use of CNL for formal knowledge representation has been investigated by many different research groups and a variety of systems have emerged. The majority of them, however, never left their prototype status and — to my knowledge — none of them could yet find broad industrial usage. In order to bring this development forward, this thesis introduces reusable and reliable methodologies that facilitate the definition, deployment, and testing of such systems. My objective is to give this research area a shift in perspective: from the present explorative and proof-of-concept-based approaches to a more engineering focused point of view.

The remainder of this chapter is organized as follows. First, the approach of using CNL for knowledge representation is motivated (Section 1.1). Afterwards, a hypothesis is established and the structure of the overall approach is introduced (Section 1.2). Finally, an outline of the content of the remaining chapters is given (Section 1.3).

1.1 Motivation

We live in a world where computers become more and more prevalent. The number of people who have to deal with computers in their everyday life has dramatically increased over the past decades. While 24% of the employed population, for example, used a computer at work in the USA back in 1984, this number has more than doubled in only 13 years reaching 51% in 1997 [49]. I would claim that we are not very far from the point where virtually everybody needs to directly interact with computers on a daily basis.

People's education in computer science, however, could not keep up with the rapid raise of computers. While twice as many people used a computer at work in the USA in 1997 compared to 1984, the number of people passing their bachelor's degree in computer science has shrank from more than 32'000 to less than 26'000 students during the same period of time [47]. Luckily, this figure increased again in later years so that about 43'000 students earned their bachelor's degree in computer science in 2006. Nevertheless, this number is still very low, corresponding to less than 0.1% of the employed population.

Thus, while more and more people need to deal with computers in their everyday life, the percentage of people with a higher education in computer science remains very low. The presented data covers only the USA but the situation of professional computer use and higher computer education can be assumed to be similar in other western countries. As a consequence, more and more people with no particular background in computer science need to interact with computers.

This situation raises the need to communicate with computers in an easy and intuitive way that does not require special education of the human user. However, the fact that humans and computers utilize completely different kinds of languages is a major hindrance for the efficient communication between the two. Computers use formal languages (like programming languages and logic-based ontology languages) while humans express themselves in natural languages (like English).

The most straightforward solution to this problem is to write computer programs that are able to process natural language in a sensible way. Despite some early success stories in this research area (e.g. SHRDLU by Terry Winograd [178]), the general task of natural language processing (NLP) turned out to be an extremely difficult problem. A large amount of research has been directed to this problem over decades. Despite the fact that notable progress has been made on particular subproblems (see [35] or [81] for an overview), computers still fail to process natural language in a general and reliable way.

The second BioCreative contest [90] that took place in 2008 nicely illustrates the difficulty of NLP with the current state-of-the-art technologies. This contest contained a seemingly simple task called "interaction pair subtask" that was about finding protein interactions mentioned in biomedical literature. 16 teams of leading scientists participated with their NLP systems. The best performing systems reached precision and recall scores between 30% and 40%. Thus, the systems failed to find 60% to 70% of the interaction pairs in the text, and 60% to 70% of the found interaction pairs were incorrect. This shows that NLP is very far from being a solved problem. Today, computers are simply unable to interpret natural language reliably.

While computers fail to understand natural languages, humans are known to have a hard time learning formal languages. For example, many web users fail to properly use even very simple boolean operators in search engines [76]. Also people using logic languages professionally often have difficulties. For example, users of the ontology language OWL often mix up fundamental concepts of logic like existential and universal quantification [136].

Altogether, this results in the obvious problem that humans and computers have to communicate but neither speaks the language of the other.

The use of graphical diagrams can solve this problem up to a certain degree. Properly designed diagrams can be understood without training. At some point however, when it comes to representation of complex situations, this does not work anymore. Graphical diagrams lose their intuitive understandability if they become too complex. For example, Kaufmann and Bernstein [88] describe a study where four user interfaces (query interfaces for the Semantic Web) were compared, three that are natural language based and one based on graphical diagrams, with the result that the users who used the graphical interface had the lowest performance by all means.

The approach that will be pursued within this thesis is to solve this problem by designing intermediary languages that are based on formal logic — and thus are understandable for computers — but look like natural language. The underlying idea is that statements in formal logic can always be expressed in a natural way. Consider John Sowa's statement [158]:

> ❝ Although mathematical logic may look very different from ordinary language, every formula in any notation for logic can always be translated to a sentence that has the same meaning. ❞

Natural language is a superset of logic in terms of expressiveness, and thus everything that can be stated in logic can also be stated in a sentence of natural language. As a consequence, it is possible to define subsets of natural language in a way that the valid sentences of the language can be mapped deterministically to formulas of a given logic formalism. Such languages are called *controlled natural languages* (CNLs).

On the one hand, a CNL looks like a natural language and is intuitively understood by the human speakers of the respective natural language. On the other hand, a CNL is restricted in such a way that computers can process and interpret it. The idea is to define languages that share the intuitive human understandability of natural languages with the processability of languages based on formal logic.

This thesis focuses on English as the underlying natural language of CNLs. The reasons for this are that English has a relatively simple structure (especially its morphology), is widely spoken around the globe, and is the common language in the academic world.

1.2 Hypothesis and Approach

The purpose of this thesis is to provide building blocks for the proper definition and application of controlled English to be used for the representation of knowledge. This approach is based on the following hypothesis:

> **Controlled English efficiently enables clear and intelligible user interfaces for knowledge representation systems.**

This general hypothesis will be tested on the basis of the following questions covering different aspects:

1. **How should controlled English grammars be represented?**
2. **How should tools for controlled English be designed?**
3. **How can the understandability of controlled English be evaluated?**
4. **Is controlled English indeed easier to understand than other formal languages?**

The last question is crucial. The complete approach loses its legitimation if this question has to be answered with “no”.

Figure 1.1 shows a graphical representation reflecting the layout of the overall approach. It shows that the contribution of this thesis can be subdivided into the three aspects of definition, deployment and testing of controlled English.

The definition aspect is covered by the study of how the syntax of controlled English languages can be defined in an appropriate way, so they can be used conveniently within knowledge representation tools. The deployment aspect is discussed on the basis of several tools that have been implemented and that demonstrate how

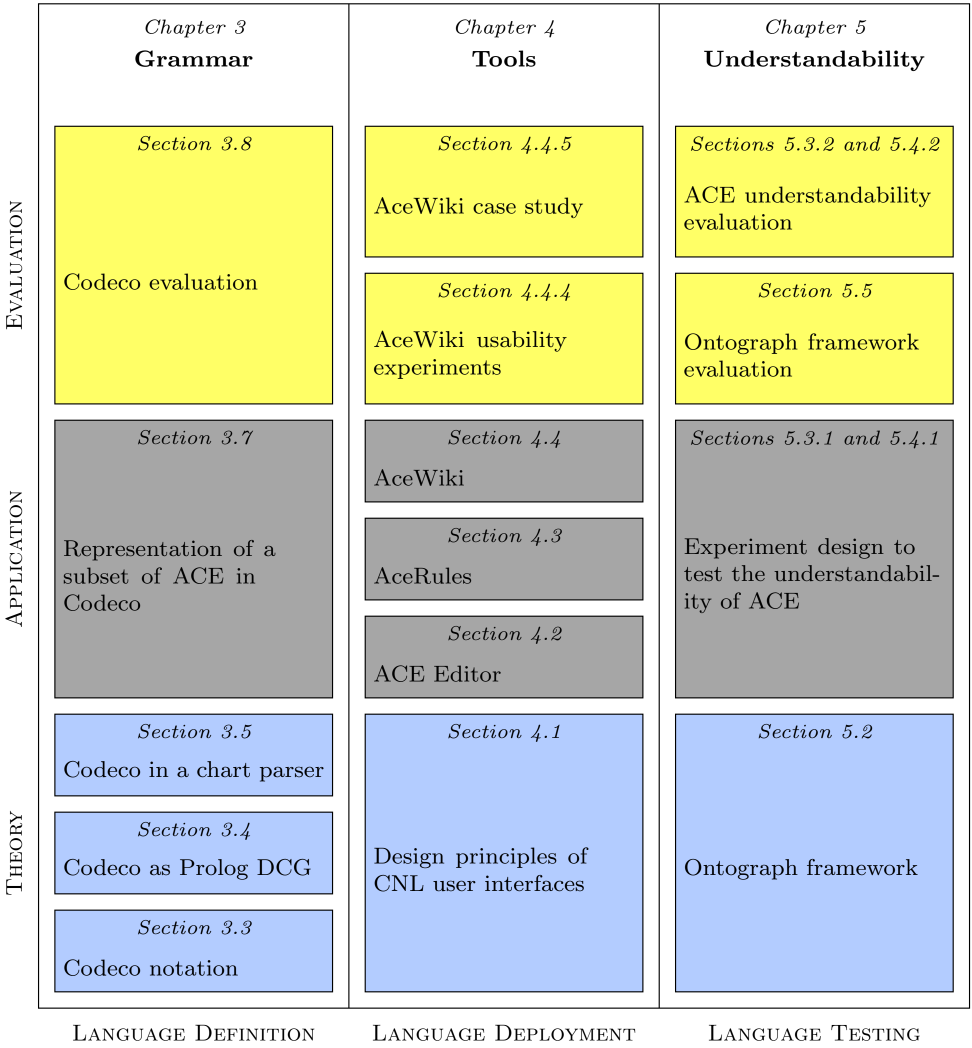


Figure 1.1: This figure show the layout of the overall approach of this thesis. The covered aspects are definition, deployment and testing of controlled English. Within these aspects, contributions can be categorized on an orthogonal dimension into theory, application and evaluation. Each of the three aspects is covered by a chapter of this thesis. The language definition aspect is covered by a chapter about grammars for controlled English. The chapter about tools targets the language deployment aspect. The language testing aspect, finally, is covered by a chapter about how the understandability of CNLs can be evaluated.

user interfaces using controlled English can be designed in a proper way. In terms of testing, a framework is introduced that enables the tool-independent and reliable evaluation of human understandability of controlled English.

For each of the three aspects (i.e. definition, deployment and testing), theoretical foundations, practical applications, and evaluation results are described. The theoretical parts introduce new notations and frameworks in cases where existing approaches are not suitable for the domain of CNLs. The practical application of these theoretical foundations is demonstrated on the basis of the language Attempto Controlled English (ACE) and on the basis of prototypical tools that make use of ACE. Evaluations in the form of technical experiments, human subject experiments, and case studies have been performed and the results will be presented and discussed.

Even though this thesis concentrates on English and on knowledge representation, the discussed approaches should be general enough to be applied to controlled subsets of other natural languages and to other problem areas, e.g. specifications and machine translation.

1.3 Outline

This thesis consists of six chapters. Following this introductory chapter, Chapter 2 entitled “Background” describes the background of the fields of controlled natural language and knowledge representation. A special focus is given to the language ACE.

Chapters 3 to 5 contain my own scientific contribution. Chapter 3 “Grammar” introduces a grammar notation called *Codeco*, in which controlled natural languages like ACE can be defined in a concrete and declarative way. Chapter 4 “Tools” demonstrates how tools can be designed that use controlled English. This is shown on the basis of three concrete tools that I developed — ACE Editor, AceRules and AceWiki — all of which make use of ACE. In the case of the AceWiki system, the results of several small usability studies are described. Chapter 5 “Understandability” introduces a novel testing framework for CNLs called *ontographs*. Within this framework, the understandability of languages like ACE can be evaluated and compared in human subject experiments. The results of two such experiments are described.

Chapter 6 “Conclusions and Outlook” draws the conclusions from the results described in the chapters 3 to 5 and contains a brief look into the future of controlled English and knowledge representation.

The appendix of this thesis, finally, consists of two parts. Appendix A contains the ACE Codeco grammar that defines a large subset of ACE in a formal and declarative way. Appendix B contains the resources that have been used for two experiments based on the ontograph framework.

CHAPTER 2

Background

In this chapter, the background and the state of the art in the fields of controlled natural language and knowledge representation are described. CNLs in general are discussed first: how they emerged and what kind of CNLs exist (Section 2.1). Then, a closer look is taken at the language Attempto Controlled English that is the language to be used throughout this thesis (Section 2.2). Finally, the field of knowledge representation is introduced with a special focus on expert systems and the Semantic Web (Section 2.3).

2.1 Controlled Natural Languages

Roughly speaking, controlled natural languages (sometimes simply called "controlled languages") are artificially defined languages that coincide with (or are at least close to) a subset of a particular natural language [180]. Many different languages of this kind exist today, at different stages of maturity. Pool [128] counts 41 projects that define controlled subsets of English, Esperanto, French, German, Greek, Japanese, Mandarin, Spanish and Swedish.

Below, the different types of CNLs and their historical background are described. Then, an overview of the state of the art is given and an important problem of CNLs — the writability problem — is discussed. Finally, existing CNL editors are introduced and described, and general design principles for CNLs are discussed.

2.1.1 Types of CNLs

In general, two directions of CNLs can be identified. On the one end of the scale, there are CNLs that are intended to improve the communication among people, especially among non-native speakers. Such languages are not mainly designed towards good processability by computer but towards good understandability by human readers. They have no formal semantics and are usually defined by informal guidelines.

On the other end of the scale, there are CNLs that are completely unambiguous and have a direct mapping to formal logic. These languages are designed to improve the communication between humans and computers, for example for querying or editing knowledge bases. Such languages can be defined by formal grammars.

These two forms of CNLs have different goals and can look very different. O'Brian [120] compares CNLs of different types and comes to the conclusion that no common core language can be identified. However, many intermediate stages between these two general directions of CNLs exist — from informal over semi-formal to completely formal semantics — which makes it hard to draw a strict line between them.

Furthermore, every CNL that is designed to improve human–computer communication can also be used for communication between humans; and controlling a language for enabling a better communication between humans does potentially also improve the computer processability. For these reasons, it makes sense to use the term "controlled natural language" in a broad sense for all such languages and not to introduce different names for them. However, this thesis clearly focuses on the second type of CNLs, those that are unambiguous and can be translated into logic.

Another direction of CNLs are the ones that are defined to improve machine translation (see e.g. [109, 5]). Such languages are designed to be computer-processable but they do not need to have an explicit formal semantics. It is sufficient to define connections between the controlled subsets of different natural languages that enable the automatic translation from one language into the other.

The term "controlled natural language" is related to the term of "fragments of natural language" or simply "fragments of language" (see e.g. [131]). Whereas CNLs are designed to improve communication (either computer–human or human–human), fragments of language are defined to study the language, e.g. to study the computational properties of certain linguistic elements. While the goal is completely different, the resulting sublanguages can look very similar.

2.1.2 History of CNLs

Controlled natural languages can be considered as old as logic itself. Aristotle's syllogisms [7] are arguably the first controlled natural language being defined around 350 B.C. Aristotle found that there are certain patterns of pairs of statements like "all A are B" and "all B are C" for which a third statement "all A are C" must necessarily follow no matter which words are assigned to A, B and C.

It took more than 2000 years before syllogisms were superseded by first-order logic based on the work of Frege [48] in 1879. From that point on, logic was no longer mainly written in natural language but in "unnatural" formulas that make use of logical variables. While this progress was seminal for the study of logic, it also had the downside that logical formulas could no longer be read and verified by anybody, but were readable only by people educated in logic.

In the 20th century, several controlled subsets of natural languages — mostly English — have been defined with the goal to improve communication among people around the globe. One of the first of them was *Basic English* [121], which was presented in 1932. It restricts the grammar and makes use of only 850 English words. It was designed to be a common basis for communication in politics, economy and science. Other examples include *Special English*[1], which is a simplified English used since 1959 for radio and television news, and *SEASPEAK* [161], which is an international language defined in 1983 for the communication among ships and harbors.

Starting from the 1970s, controlled subsets of English have been defined for more technical subjects. Companies like Caterpillar [172], Boeing [72], IBM [14], and others [4] defined their own subsets of English that they used for their technical documentation. The goal was to reduce the ambiguity of their documents and to prevent misunderstandings, especially for non-native readers. This effort was driven by the increasing complexity of the technical documentation and the increased activity over language borders. Such kinds of languages are still used today, e.g. in the form of ASD Simplified Technical English[2]. However, these languages are designed to improve only human–human communication, have no focus on computer processability, and have no formal semantics.

Also in the 1970s, natural language interfaces to databases had become popular [70, 174], which also led to the study of controlled natural languages in this context [113]. Arguably because of insurmountable NLP problems, the research drifted away again from this topic and together with the approaches using full natural language also the CNL-based approaches disappeared.

The programming language COBOL is worth a special mention here even though it cannot be considered a CNL in the strict sense. Emerging in the late 1950s, it is one of the oldest programming languages but still one of the most widely used today [110]. COBOL is oriented towards business and administration systems and uses a large set of keywords that should make COBOL code resemble natural language and thus easier to understand. Nevertheless, programs in COBOL still look very artificial and cannot be understood without training.

Even though the general approach was proposed already in the 1970s [154], it was only in the mid 1990s when languages like Attempto Controlled English were actually implemented trying to control natural languages in a way, so they can be mapped directly to some sort of formal logic. This can be seen as an act of revitalizing

[1] `http://www.voanews.com/specialenglish/`

[2] `http://www.asd-ste100.org/`

the spirit of the early times of logic where logical statements were expressed in natural language. The goal is to make logic again understandable for people with no particular education in logic.

2.1.3 CNL: State of the Art

The following survey of the state of the art of CNLs focuses on the formal and unambiguous languages excluding the ones that are solely designed for human–human communication and are thus not essential for this thesis. The existing CNLs of the first kind can be roughly subdivided into general-purpose languages, business rule languages, and languages developed specifically for the Semantic Web.

2.1.3.1 General-Purpose CNLs

General-purpose CNLs are developed without a specific application scenario in mind. They are designed to be usable by third parties in their own application areas.

Attempto Controlled English (ACE) is a mature general-purpose controlled English that comes with a parser that translates ACE text into a logic-based representation. It is the language that is used throughout this thesis and for this reason, it will be described in more detail in the next section.

PENG (Processable English)[3] is a language that is similar to ACE but adopts a more light-weight approach in the sense that it covers a smaller subset of natural English. It is designed for an incremental parsing approach and was one of the first languages used within a special editor with lookahead features to tell the user how a partial sentence can be continued. Such editors will be discussed in more detail later on.

Common Logic Controlled English [157] is a further subset of English — similar to ACE — that has been designed as a human interface language for the ISO standard Common Logic[4].

Formalized-English [104] is another proposal of a CNL to be used as a general knowledge representation language. It has a relatively simple structure and is derived from a conventional knowledge representation language. Formalized-English still contains a number of formal-looking language elements and is thus not a strict subset of natural English.

Computer Processable Language (CPL) [32] is a controlled English developed at Boeing. Instead of applying a small set of strict interpretation rules, the CPL interpreter resolves various types of ambiguities in a "smart" way that should lead to acceptable results in most cases. Thus, CPL can be considered more liberal than the other four languages.

[3] see [148] and `http://web.science.mq.edu.au/~rolfs/peng/`

[4] `http://cl.tamu.edu/`

2.1.3.2 CNLs for Business Rules

Business rule systems allow companies to explicitly and formally define the rules of their business and are an interesting application area of CNLs. Business rules typically need to be approved and followed by people with no particular background in knowledge representation or logic and for this reason it is important to have an intuitive representation as CNLs can offer. There are some CNLs that have been defined for this particular problem area.

RuleSpeak and SBVR Structured English [149, 159] are two such CNLs defined by the SBVR standard ("Semantics of Business Vocabulary and Business Rules"). These languages are defined informally by sets of guidelines based on experiences of best practice in rule systems. They are not strictly formal languages, but they are connected to the semantics definition of the SBVR standard.

2.1.3.3 CNLs for the Semantic Web

Recently, several CNLs have been proposed that are designed specifically for the Semantic Web and that can be translated into the Web Ontology Language (OWL). The vision of the Semantic Web will be introduced in more detail in Section 2.3.2.

Rabbit [69] is one of these languages. It has been developed and used by Ordnance Survey, i.e. Great Britain's national mapping agency. Rabbit is designed for a specific scenario, in which it is used for the communication between domain experts and ontology engineers in order to create ontologies for specific domains.

The Sydney OWL Syntax [38] is a second example. It is based on PENG and provides a bidirectional mapping to OWL. Thus, statements in the Sydney OWL Syntax can be translated into OWL expressions, and vice versa.

CLOnE (Controlled Language for Ontology Editing) [166, 57] is a third example of a CNL that can be translated into OWL. It is a simple language defined by only eleven sentence patterns which roughly correspond to eleven OWL axiom patterns. Due to its simple design, only a small subset of OWL is covered.

Lite Natural Language [10] can be mentioned as a fourth example. It maps to DL-Lite, which is a logical formalism optimized for good computational properties and which is equivalent to a subset of OWL.

Furthermore, ACE has also been applied to the area of the Semantic Web and can be mapped to OWL [82], and Schwitter et al. [145] show a comparison of the three languages Sydney OWL Syntax, Rabbit, and ACE.

2.1.4 The Writability Problem of CNLs

One of the biggest problems — if not *the* biggest problem — of CNLs is the difficulty to write statements that comply with the restrictions of the language. Many different researchers working on CNLs encountered this problem. For example, Power et al. [129] write

> ❝ The domain expert can define a knowledge base only after training in the controlled language; and even after training, the author may have to try several formulations before finding one that the system will accept. ❞

and Schwitter et al. [146] state:

> ❝ It is well known that writing documents in a controlled natural language can be a slow and painful process, since it is hard to write documents that have to comply with the rules of a controlled language [...]. ❞

Thus, the problem of writing texts in a CNL is a well-known problem, even though it does not yet have a generally accepted name. I will call it the *writability problem* of CNLs.

CNLs are designed to be intuitively understood by the speakers of the respective natural language. However, readability and writability are two completely different problems. In order to be able to read a language, one does not need to know the coverage with respect to the natural language. Writing syntactically correct statements without tool support is much more complicated because the user needs to learn the syntax restrictions, which are in many cases not trivial to explain. In some sense, this writability problem can also be encountered when learning foreign natural languages. Reading and understanding a foreign language is much simpler than writing texts of the same quality.

Three approaches have been proposed so far to solve the writability problem of CNLs: error messages, predictive editors, and language generation. These three approaches will now be introduced, and they will be illustrated by examples from the literature.

2.1.4.1 Error Messages

The most straightforward way to deal with the writability problem is simply to have a parser that provides good error messages. In this case, users are supposed to learn the restrictions of the language and should then write statements freely. In the case parsing fails (because the statement is not a correct statement of the respective CNL), the CNL system tries to determine why the statement is not parsed. This cause is then reported to the user together with one or more suggestions how the problem can be resolved in the form of canned text. The user then has to reformulate the statement and submit it again to the CNL system. This goes on until the statement follows the syntactic restrictions of the CNL.

The hardest part of this approach is to track down the exact cause of the error and to give sensible suggestions. The input can potentially be any sentence of the underlying natural language. This entails all kinds of problems that come with the processing of natural language, i.e. problems that CNLs actually should remedy.

ACE and its parser were designed according to the error message approach until the development of the ACE Editor (see Section 4.2) began. It simply turned out to be very difficult to provide good error messages. Nevertheless, many CNLs adopt this error message approach including CPL, Rabbit and CLOnE.

CPL probably has the most sophisticated implementation of the error messages approach. Clark et al. show how users of CPL are supported by an advice system that can trigger more than 100 different advice messages [31]. They give a concrete example that illustrates the error messages approach: If the user enters the incorrect CPL sentence

> The initial velocity of the object is 17.

where the measurement unit is missing then — according to Clark et al. — the following error message (they call it "advice message") is returned by the CPL system:

> Always specify a unit for numbers (e.g., "10 m", not "10"). Use the word "units" for dimensionless units. e.g., Instead of writing: "The velocity is 0." write "The velocity is 0 m/s."

In this example, the error messages approach works out nicely. However, more complex cases are much harder to cover.

2.1.4.2 Predictive Editors

A different approach to solve the writability problem of CNLs are editors that are aware of the grammar of the respective CNL and that can look-ahead within this grammar. Such editors are able to show what kind of words or phrases can follow a partial sentence. In this way, users can create a CNL statement in small steps, and are guided by the editor at every point on how the statement can be continued until it is complete.

The basic idea of predictive editors is not very new. It was proposed already in the 1980s by Tennant et al. [169]. They describe a system that enables users to write natural language queries to databases in a controlled way.

The language PENG is an example of a CNL that has been designed from the beginning to be used in predictive editors. Schwitter et al. [146] introduce the predictive editor ECOLE that is used to write PENG statements and they give the following example that illustrates how predictive editors work:

> A [adjective | common noun]
>
> A person [verb | negation | relative sentence]
>
> A person lives [’.’ | prepositional phrase | adverb]

The user starts writing a sentence by choosing the article "a" and the system tells the user that the text can be continued by either an adjective or a common noun. Choosing the common noun "person" in a next step, the system returns that a verb, a negation, or a relative sentence can be added. The interaction between user and predictive editor goes on in this way until the sentence is finished.

Using the ACE Editor (to be presented in Section 4.2), ACE can be used according to the predictive editor approach too.

The hardest problem with this approach is to define the grammar of the CNL in a form that is processable by the editor, so the editor can look-ahead and inform the user how the statement can be continued.

2.1.4.3 Language Generation

Finally, the language generation approach tries to solve the writability problem by fully relying on CNL generation in a way that makes CNL analysis unnecessary [130]. The basic idea is that a (possibly incomplete) statement in a CNL is shown to the user together with possible actions to modify the model that underlies the statement. If a certain modification action is performed by the user, a new CNL statement is generated that reflects the updated model. In this way, CNL statements can be built incrementally without allowing the user to directly modify the text.

This approach is also known as *conceptual authoring* or as *WYSIWYM* that stands for "What You See Is What You Meant" [129]. Analogous to the "What You See Is What You Get" approach of word processors that allow users to edit a text on the character level and immediately show a graphical representation, a WYSIWYM editor allows users to edit a formal knowledge base on the semantic level and immediately shows a representation in a (controlled) natural language.

Again, for illustration purposes a concrete example is very helpful. Power et al. give the following example [129]: A user may start with the general and incomplete statement

> Do **this action** by using *these methods*.

where the user can click on "**this action**" or on "*these methods*". By clicking on "**this action**", the user can choose from several actions, for example "schedule", which changes the underlying model. On the basis of the updated model, a new text is generated:

> Schedule **this event** by using *these methods*.

By clicking on "**this event**", the user can choose from different events, for example "appointment", which leads to the following situation:

> Schedule the appointment by using *these methods*.

This process goes on until the statement is completed. Important is that the user does not have direct control of the text but can change it only by manipulating the underlying model by given modification actions.

The problem with this approach is that it does not produce an independent language but one that highly depends on very specific tools. Due to the missing parser, the resulting CNL is basically a visualization language and not a real knowledge representation language.

2.1.4.4 Writability Problem Summary

In summary, each approach has its own assets and drawbacks. In my view, however, the predictive editor approach is the one with the most promise to solve the writability problem of CNLs. The error messages approach has the serious downside that ultimately full natural language has to be processed, which is known to be very difficult. The language generation approach, on the other hand, is problematic because the resulting CNL highly depends on specific tools and actually is just a visualization language. While the predictive editor approach is challenging too, the problems are rather technical than fundamental. Later chapters will show how the problems of predictive editors can be solved, and once they are solved this approach turns out to be very elegant and reliable.

For these reasons, this thesis will focus on the predictive editor approach and the other two approaches will not be further investigated.

2.1.5 CNL Editors

In any case, proper tool support is needed to solve the writability problem of CNLs and indeed many different CNL editors exist.

One example — that has already been mentioned before — is ECOLE [146], an editor for the language PENG. It is a predictive editor in the sense that it looks ahead within the grammar and shows what word category can come next. ECOLE can show the partial syntax tree and a paraphrase of the partial sentence at each point of the sentence creation process. Additionally, the semantic representation is dynamically created and can be shown to the user. Furthermore, ECOLE is able to inform the user after every new sentence regarding its consistency and informativity with respect to the previous sentences.

Another example is the GINO system [12], an ontology interface tool. Ontologies can be queried by formulating questions in a dedicated CNL. GINO can also be used to extend an existing ontology. However, such additions are only triggered by the CNL and the actual definition is done in a form-based way without using the CNL.

Namgoong and Kim [115] present another CNL tool that includes a predictive editor and is designed specifically for knowledge acquisition. Their application area is the biological and medical domain. In contrast to most other predictive editing systems, their tool can also handle anaphoric references.

Other predictive editors are presented by Bringert et al. [21], a simple one called “Fridge Poetry” and a more sophisticated one that can translate the created text directly into controlled subsets of other natural languages.

ROO (Rabbit to OWL Ontology construction) [43] is an editor that builds upon the Protégé ontology editor and that uses the language Rabbit. ROO allows entering Rabbit sentences, helps to resolve possible syntax errors, and translates the sentences into OWL. The ROO tool is a part of a whole methodology allowing domain experts and knowledge engineers to work together in an efficient way.

Several tools have been developed in the context of the CLOnE language. The ROA editor [41], for example, is a tool that focuses on round-tripping, i.e. on parsing CNL into a logical form and on verbalizing this logical form again in CNL. Inglesant et al. [75] present another CNL editor inspired by CLOnE.

ACE View [83] is a plugin for the Protégé ontology editor based on ACE. ACE View enriches Protégé with alternative interfaces based on controlled natural language to create, browse and modify the ontology. Furthermore, questions in ACE can be used to query the asserted as well as the automatically entailed content of the ontology.

Interestingly, the approach of CNL and predictive editors has also been applied to adventure games [101].

Chapter 4 will introduce more tools that use ACE and that have been implemented within the scope of this thesis, a general editor (ACE Editor) and two more specific tools (AceRules and AceWiki).

2.1.6 Design Principles for CNLs

Many different CNL approaches and languages exist, but there seem to be some generally accepted design principles. The four general principles of *clearness*, *naturalness*, *simplicity* and *expressivity* can be identified, even though there are no agreed names for them so far. Clark et al. [33], for example, describe the two opposing concepts of *naturalness* and *predictability*. Their term *predictability* covers the two principles of clearness and simplicity as described here.

Clearness means that all statements of a certain CNL should have a clear meaning. Thus, the meaning of the statements should be describable in a systematic and coherent way, for example by a strictly defined mapping to some kind of formal logic. Clearness also means that a CNL should contain as little ambiguity as possible or no ambiguity at all in the best case.

Naturalness means that the statements of a CNL should look like a certain natural language. Thus, the statements of the CNL should be acceptable and intuitively understandable for the speakers of the respective natural language and this understanding should fit the defined meaning of the CNL (or should fit the possible meanings in the case of ambiguity).

Simplicity means that a CNL should be easy to describe. Thus, it should be easy to detect — for both humans and machines — whether a certain statement is part of the given CNL or not. Simplicity also implies that the respective CNL is easy to teach and learn.

Expressivity means that a CNL should semantically cover as much as possible of the respective problem domain: the more situations or problems describable in the CNL the better. For general-purpose CNLs that do not target a specific problem domain, the semantic coverage should be as broad as possible with respect to all possible problem domains.

These four principles are often in conflict with each other. This means that no CNL can completely satisfy all four principles. Examples of such conflicts that should also clarify the four principles are given below.

Clearness can conflict with naturalness when one has to decide whether a CNL should use a linguistic structure that is natural but ambiguous or an alternative structure that is less natural but also less ambiguous. For example, ACE defines that prepositional verbs have to attach their preposition using a hyphen (e.g. "John waits-for Mary") in order to distinguish it from the reading where the preposition introduces an independent prepositional phrase (e.g. "John waits for no reason"). Thus, ACE in this case sacrifices some naturalness for achieving a clear and unambiguous language.

In the same way, clearness can be in conflict with simplicity in cases where an ambiguous structure could be replaced by an unambiguous but more complicated one. For example, instead of using just "or" that can be meant inclusively or exclusively, one could define that every occurrence of "or" must be followed by either "or both" or "but not both" to disambiguate the two possible readings. This would be a sacrifice of simplicity for the sake of clearness.

Clearness obviously also conflicts with expressivity. For simple statements like for example "every mammal is an animal" it is relatively simple to come up with a systematic and clear definition of its meaning, e.g. by a mapping to common first-order logic. If more complex matters should be expressible, however, like for example "a study has proven that at least 50% of the population who was in jail more than once during their childhood consider suicide at some later point of time in their lives" then the clear and systematic description of the meaning becomes obviously very difficult and can probably not be expressed in standard first-order logic.

Naturalness can conflict with simplicity. Natural linguistic phenomena that are complicated to describe can be represented in a CNL in a simplified way leading to a less natural but simpler language. For example, in the context of a negation one would usually use "anything" and not "something", e.g. "John does not drink anything". However, for the sake of simplicity a CNL can be defined in a way that "something" has to be used in all cases, which is not perfectly natural but simpler. ACE, for example, makes use of this simplification and does not support "anything".

Naturalness conflicts with expressivity in cases where no fully natural solution can be found to achieve a certain degree of expressivity. Variables — as used by ACE for example — can be used to support arbitrary references, e.g. "a person X teaches a person Y and the person X is not a teacher". Variables are not fully natural but they improve the expressivity of the language.

Finally, the conflict between simplicity and expressivity is quite obvious. The more expressive a language is, the more syntactic structures it must provide to express things, and consequently the more complex the language description gets. For example, modal statements can be expressed by modal verbs like "can" and "must".

However, this requires that one has to define how these modal verbs have to be used and how they interact with other structures like negation or questions.

All these trade-offs have to be considered when defining a new controlled natural language and thus every CNL has to be positioned somewhere by means of clearness, naturalness, simplicity and expressivity.

2.2 Attempto Controlled English (ACE)

After all these general discussions, let us turn to a concrete example of a CNL: Attempto Controlled English (ACE) [50, 42]. ACE is the language that is used throughout this thesis, and for this reason it will be given special attention.

ACE is one of the most mature controlled natural languages and has been under active development for more than 14 years since its development began in 1995. ACE was first introduced by Fuchs and Schwitter [54] and since then, more than 40 scientific papers have been published by the Attempto group[5]. Today, *Google Scholar* lists almost 500 articles containing the term "Attempto Controlled English"[6], which makes it probably the most widespread CNL in academia.

ACE is a general-purpose CNL that is not restricted to a certain domain. The vocabulary is exchangeable and can thus be adapted to specific problem areas. For certain syntactic structures (e.g. plurals), the logical representation is deliberately underspecified, which gives some freedom to applications in the sense that they can use the structures in a manner most useful for the respective application area.

The Attempto Parsing Engine (APE) is the reference implementation of ACE and translates ACE texts into logic. The source code of APE is freely available under an open source license.

I had the privilege to be a member of the Attempto group for the last five years and to make contributions towards the further development of the language ACE and its applications.

In what follows, I will introduce the basic features of the syntax and semantics of ACE and I will outline some of its application areas. Please note that all current and past members of the Attempto group have to be credited for what I will describe here.

2.2.1 Syntax of ACE

The syntax of ACE is defined in a general and informal way by a number of construction rules [1] and in a more detailed, semi-formal way by the ACE syntax report [3]. The only fully formalized syntax definition of the full language of ACE is the grammar of the parser APE. A formal and declarative grammar of a large subset of ACE

[5] `http://attempto.ifi.uzh.ch/site/pubs/`

[6] `http://scholar.google.com/scholar?q=%22Attempto+Controlled+English%22` (retrieved in November 2009)

has been developed within the scope of this thesis. This grammar will be introduced in the next chapter and is shown in Appendix A.

Below, I give a quick and informal overview of ACE, covering most but not all of its language features.

Words

ACE basically contains two types of words: function words and content words. While function words are built-in and cannot be changed, content words can be defined and modified by the user with a simple lexicon format.

"If", "every", "or" and "is" are some examples of function words. Content words can be defined in six categories: nouns, proper names, verbs, adjectives, adverbs and prepositions. They can be defined freely but they are not allowed to contain blank spaces. For example, instead of "credit card" one has to write "credit-card".

Noun Phrases

In the simplest case, a noun phrase consists of a countable noun (e.g. "country"), a mass noun (e.g. "water"), or a proper name:

a country
some water
the city
Switzerland
the Earth

Note that nouns always need a determiner like "a", "some" or "the". ACE also supports plural noun phrases:

some countries
the cities
5 customers

Furthermore, indefinite pronouns are function words that act as noun phrases:

something
somebody

Numbers and string are also considered noun phrases:

1200
"swordfish"

Measurement units like "m" or "kg" can be used to define certain quantities:

200 m
3 kg of salt

Nouns can be directly modified by adjectives, which can be in positive, comparative or superlative form:

some landlocked countries
a richer customer
some most important persons
some fresh water

Nouns can be followed by an *of*-phrase, i.e. a prepositional phrase using the preposition “of”:

a customer of John

Furthermore, nouns can be preceded by a possessive noun phrase that ends with the Saxon genitive marker “’s” or just “’”:

John’s customer
Paris’ suburbs

Verb Phrases

Verb phrases can use intransitive, transitive and ditransitive verbs in simple present tense:

Mary waits
some men see Mary
Bill gives a present to John

In the case of transitive and ditransitive verbs, passive voice can be used:

Mary is seen by some men
a present is given to John by Bill
John is given a present by Bill

Note that the subject must be explicitly defined using “by”. Transitive verbs can take subordinated sentences as their complement:

Mary knows that a customer waits

Furthermore, the copula verb “be” can be used to construct verb phrases:

Switzerland is a country

The copula can also be used together with adjectives in positive, comparative or superlative forms:

John is rich
some customers are richer
Switzerland is located-in Europe
Mary is most fond-of Bill

Adjectives with prepositions like “located-in” and “fond-of” are called *transitive adjectives*, because they require a complementing noun phrase, as transitive verbs do. Furthermore, such verb phrases using adjectives can use the constructs “as .... as” and “more ... than”:

> John is as rich as Mary
> John is more fond-of Mary than Bill
> John is more fond-of Mary than of Sue

All kinds of verb phrases can be modified by adverbs, which can stand in front of the verb or at the end of the verb phrase:

> John manually connects some cables
> a customer waits patiently

In the same way as adverbs, prepositional phrases can be used to modify verb phrases:

> a customer waits in Zurich

Relative Clauses

Nouns, proper names, and indefinite pronouns can have a relative clause containing an arbitrary verb phrase:

> something that contains some water
> John who is rich
> some products which are bought by Mary

Relative clauses start with the relative pronoun “that”, “which” or “who”.

Coordination

Noun phrases, adjectives, and adverbs can be coordinated by “and”:

> some customers and some managers and Mary
> a rich and important customer
> John works carefully and silently

Verb phrases, relative clauses, and (subordinated) sentences can be coordinated by “and” or “or”:

> a car crashes or breaks-down
> a person who works or who travels
> a customer waits and Mary works
> John knows that a customer waits and that Mary works

Note that the relative pronoun has to be repeated for the coordination of subordinated sentences. Prepositional phrases are coordinated by concatenation, i.e. no conjunction is used:

> John flies from Tokyo to Zurich

Quantifiers

The universal quantifiers “every” and “all” can be used instead of their existential counterparts “a” and “some”:

every country
all water

The indefinite pronouns can also have the universal form:

everything
everybody

The existential quantifiers “there is” and “there are” take a noun phrase and can optionally be followed by the phrase “such that” that introduces a subordinated sentence:

there is a customer
there are some products such that every customer likes the products

The universal quantifiers “for every” and “for all” also have to be followed by a subordinated sentence:

for every product a customer likes the product
for all water a filter cleans the water

The quantifiers “at least”, “at most”, “less than”, “more than”, and “exactly” can be used for numerical quantification:

at least 3 customers
more than 4 kg of salt
exactly 365 days

The distributive quantifier “each of”, finally, quantifies over the members of a plural noun phrase (i.e. it enforces the distributive reading):

each of some customers
each of at most 5 products

Negation

Verb phrases can be negated by “does not”, “do not”, “is not” and “are not”:

John does not work
some customers are not rich

The quantifier “no” is the negated version of “every” and “all”:

no country
no water

Indefinite pronouns have negative forms too:

nothing
nobody

Noun phrases using "no", "nothing" or "nobody" can optionally be followed by "but" and a mass or plural noun or a proper name:

nothing but meat
nobody but customers
no man but John

Furthermore, sentences can be negated by preceding them with "it is false that":

it is false that a customer waits

Negation as failure can be represented by "not provably" and "it is not provable that":

a customer is not provably a criminal
it is not provable that John is trustworthy

Modality

ACE has support for the modal verbs "can", "must", "should" and "may":

John cannot fail
Bill must buy a car
Mary should receive a letter from John
Sue may leave

The same kind of modality can also be expressed by prefixing a sentence by one of several fixed phrases using the special adjectives "possible", "necessary", "recommended" and "admissible":

it is not possible that John fails
it is necessary that Bill buys a car
it is recommended that Mary receives a letter from John
it is admissible that Sue leaves

Conditional Sentences

Conditional sentences consist of the two function words "if" and "then" each of which is followed by a sentence:

if John sleeps then Mary does not work

Complete Sentences

Complete sentences are either declarative, interrogative (i.e. questions), or imperative (i.e. commands). Declarative sentences consist of a sentence or a sentence coordination and end with a full stop:

> John sleeps and Mary works.
> It is false that every customer is important.

Questions end with a question mark and can be subdivided into *yes/no*-questions and *wh*-questions. *Yes/no*-questions are like simple sentences but start with an auxiliary verb:

> Is every product expensive?
> Does John wait?

Wh-questions are simple sentences that contain one or more of the query words "who", "what", "which", "whose", "where", "when" and "how":

> Who is a customer?
> Which products contain what?
> Whose father is rich?
> Where does John work?

Commands consist of a noun phrase followed by a comma and a verb phrase and end with an exclamation mark:

> John, give a card to every customer!

Anaphoric Pronouns

Reflexive pronouns like "herself" refer back to the subject of the respective verb phrase:

> A woman helps herself.
> Mary sees a man who hurts himself.

Pronouns like "she" and "him", to be called *irreflexive*, are used to establish references to a previous element of the text that is not the subject of the respective verb phrase:

> Mary waits and she sees John.
> A man sees a woman who likes him.

Such pronouns are called *anaphoric* pronouns and they are one of different ways to represent what is called *anaphoric references* or just *anaphors*. In ACE, anaphoric pronouns are only allowed if they can be resolved. Every anaphoric pronoun must have a matching element in the preceding text, i.e. "woman", "man", "Mary", and again “man” in the four examples shown above. Such elements that are referred to by anaphoric references are called *antecedents*. Antecedents can be inaccessible if they are, for example, under the scope of negation.

The following sentences are syntactically incorrect because the anaphoric pronouns cannot be resolved (sentences that are not well-formed are marked with a star):

> * John sees a woman who hurts himself.
> * Mary does not love a woman and sees her.

Both sentences look incorrect or at least strange if read as natural English sentences. In the first sentence, the reflexive pronoun “himself” does not match in gender with the subject of its verb phrase “woman”. The second sentence is incorrect because “her” can neither refer to “Mary” (because irreflexive pronouns cannot refer to the subject of the verb phrase) nor to “woman” (because it is under the scope of negation).

In fact, reflexive pronouns like “himself” can in some cases of natural English not only refer to the respective subject but to any preceding argument of the same verb [140], e.g. “nobody tells Mary about herself”. ACE, however, makes the simplified assumption that such reflexive pronouns can only refer to the subject.

ACE also supports possessive variants of reflexive and irreflexive pronouns, in the form of “her”, “his”, etc. (irreflexive) and “her own”, “his own”, etc. (reflexive), as shown by the following examples:

> Mary feeds her own dog.
> A man sees a woman who likes his car.

The sections 2.2.2.5 and 2.2.2.6 will show in more detail how ACE handles scopes and resolves anaphoric references.

Variables

In order to be able to make arbitrary anaphoric references, ACE has support for variables. Variables start with an upper case letter and may be followed by one or more digits. Variables can occur on their own or as apposition to a noun or an indefinite pronoun:

> X
> a man A
> the product P1
> something T

Variables can be seen as a borderline case in terms of naturalness. They can be found in natural language (e.g. in mathematical texts) but they are rather rare in everyday language.

Structures like “a man A” or “something T” introduce a new variable. This is only allowed if the respective variable is not already defined in the previous accessible text. For example, the sentence

> * A man X sees a man X.

is syntactically incorrect because the variable “X” is introduced twice.

2.2.2 Semantics of ACE

The semantics of ACE is defined by an unambiguous mapping to a logic-based representation. The syntax restrictions of ACE — as described in the previous section — eliminate many but not all potential ambiguities of natural English. A number of interpretation rules [2] are applied that define in a deterministic way how the linguistic structures that could be ambiguous in full English are interpreted in ACE.

ACE texts are represented by an extended version of discourse representation structures (DRSs). DRSs build upon discourse representation theory [85], which is a theory for the formal representation of natural discourse. Such DRSs can be mapped in a direct and simple way to first-order logic. The DRSs produced by APE use an extended syntax to represent negation as failure and the different kinds of modality.

Below, the DRS language that is used to represent ACE texts is introduced. After that, some interesting topics are explained concerning the semantic representation of ACE: prepositional phrases, plurals, scopes, and anaphoric references.

2.2.2.1 Discourse Representation Structures for ACE

In this section, I briefly introduce the DRS notation and sketch how ACE texts are mapped to DRSs. For a comprehensive description of this mapping consult the Attempto DRS report [51].

DRSs consist of a domain and of a list of conditions, and are usually displayed in a graphical box notation:

Domain
Condition1 ... ConditionN

The domain is a set of discourse referents (i.e. logical variables) and the conditions are a set of first-order logic predicates or nested DRSs. A reified (i.e. “flat”) notation is used for the predicates. For example, the noun phrase “a country” that normally would be represented in first-order logic as

```
country(A)
```

is represented as

```
object(A,country,countable,na,eq,1)
```

relegating the predicate “country” to the constant “country” used as an argument in the predefined predicate “object”. In that way, the potentially large number of predicates is reduced to a small number of predefined predicates. This makes the processing of the DRS easier and allows us to include some linguistic information, e.g. whether a unary relation comes from a noun, from an adjective, or from an intransitive verb.

Nouns are represented by the object-predicate:

John drives a car and buys 2 kg of rice.

```
A B C D
-------------------------------------
object(A,car,countable,na,eq,1)
predicate(B,drive,named('John'),A)
object(C,rice,mass,kg,eq,2)
predicate(D,buy,named('John'),C)
```

Adjectives introduce property-predicates:

A young man is richer than Bill.

```
A B C
------------------------------------------
object(A,man,countable,na,eq,1)
property(A,young,pos)
property(B,rich,comp_than,named('Bill'))
predicate(C,be,A,B)
```

As shown in the examples above, verbs are represented by predicate-predicates. Each verb occurrence gets its own discourse referent, which is used to attach modifiers like adverbs (using modifier_adv) or prepositional phrases (using modifier_pp):

John carefully works in an office.

```
A B
-------------------------------------
object(A,office,countable,na,eq,1)
predicate(B,work,named('John'))
modifier_adv(B,carefully,pos)
modifier_pp(B,in,A)
```

The relation-predicate is used for *of*-constructs, Saxon genitive, and possessive pronouns:

A brother of Mary's mother feeds his own dog.

```
A B C D
--------------------------------------
object(A,brother,countable,na,eq,1)
relation(B,of,named('Mary'))
object(B,mother,countable,na,eq,1)
relation(A,of,B)
relation(C,of,A)
object(C,dog,countable,na,eq,1)
predicate(D,feed,A,C)
```

The examples so far have been simple in the sense that they contained no universally quantified variables and there was no negation, disjunction or implication. For such more complicated statements, nested DRSs become necessary. In the case of negation, a nested DRS is introduced that is prefixed by a negation sign:

A woman does not leave a country.

```
A
object(A,woman,countable,na,eq,1)

     B C
  ¬  object(B,country,countable,na,eq,1)
     predicate(C,leave,A,B)
```

The ACE structures "every", "no", "if ... then" and "each of" introduce implications that are denoted by arrows between two nested DRSs.

Every man owns a car.

```

                                          B C
  A                                   ⇒   object(B,car,countable,na,eq,1)
  object(A,man,countable,na,eq,1)         predicate(C,own,A,B)
```

Disjunctions — which are represented in ACE by "or" — are represented in the DRS by the logical sign for disjunction:

A man works or travels.

```
A
object(A,man,countable,na,eq,1)

  B                        C
  predicate(B,work,A)  ∨   predicate(C,travel,A)
```

The DRS elements introduced so far are standard elements of discourse representation theory. However, ACE uses some non-standard extensions. Negation as failure is one of them and it is represented by the tilde operator:

John is not provably rich.

```

     A B
  ~  property(A,rich,pos)
     predicate(B,be,named('John'),A)
```

The modal constructs for possibility ("can", "it is possible that") and necessity ("must", "it is necessary that") are represented by the standard modal operators "◇" and "□":

Sue can drive a car and must work.

```
 
-----------------------------------------------
      A B
      -----------------------------------
  ◇   object(A,car,countable,na,eq,1)
      predicate(B,drive,named('Sue'),A)

      C
      ------------------------------
  □   predicate(C,work,named('Sue'))
```

The modal constructs for recommendation ("should", "it is recommended that") and admissibility ("may", "it is admissible that") are represented by the operators "SHOULD" and "MAY":

A machine should be safe and Mary may use the machine.

```
A
-----------------------------------------
object(A,machine,countable,na,eq,1)

            B C
            ----------------------
SHOULD      property(B,safe,pos)
            predicate(C,be,A,B)

         D
         -----------------------------
MAY      predicate(D,use,named(Mary),A)
```

Sentences occurring as the complement of verb phrases lead to DRSs where a discourse referent stands for a whole sub-DRS:

John knows that his customer waits.

```
A B
---------------------------------------------
predicate(A,know,named('John'),B)

        C D
        -------------------------------------
        relation(C,of,named('John'))
B  :    object(C,customer,countable,na,eq,1)
        predicate(D,wait,C)
```

Finally, questions and commands also introduce nested boxes. The used operators are in this case "QUESTION" and "COMMAND":

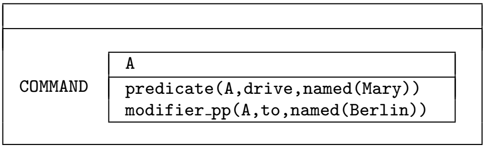
Does a customer wait?

```
 
-------------------------------------------------
            A B
            -------------------------------------
QUESTION    object(A,customer,countable,na,eq,1)
            predicate(B,wait,A)
```

What borders Switzerland?

```
 
-------------------------------------------------
            A B
            -------------------------------------
QUESTION    query(A,what)
            predicate(B,border,A,named(Switzerland))
```

Mary, drive to Berlin!



```
            A
COMMAND     predicate(A,drive,named(Mary))
            modifier_pp(A,to,named(Berlin))
```

In the case of *wh*-questions, `query`-predicates are used to denote the things that are asked for.

2.2.2.2 ACE in other Logic Notations

The DRS is the main output produced by the parser APE. However, the DRS representation can be translated into various other logic notations.

DRSs using only the standard operators (i.e. negation, implication and disjunction) can be mapped to first-order logic in a very simple and straightforward way as shown by Kamp and Reyle [85]. APE is able to produce the first-order representation of such ACE texts.

APE also has support for the TPTP format. TPTP stands for "Thousands of Problems for Theorem Provers" and is a library of logical problems that is used to test reasoners [164]. This library uses a special Prolog-based notation to represent the problems.

The non-standard extensions for possibility and necessity and for discourse referents that represent sub-DRSs can be represented in first-order logic using possible worlds semantics [18]. However, this transformation is not implemented in APE so far.

ACE questions can be mapped to first-order logic in the same way as declarative sentences. The only difference is that questions cannot describe axioms but only, for example, theorems or conjectures to be checked by a reasoner. For recommendation, admissibility, and commands, sensible representations in first-order logic can be defined but there is no standard way how to do this. Negation as failure, in contrast, cannot be expressed in first-order logic.

Furthermore, a subset of ACE can be translated via the DRS representation into the Semantic Web languages OWL and SWRL (Semantic Web Rule Language). This translation is described by Kaljurand [82] and is implemented in APE, which can produce OWL and SWRL statements in different syntactical variants. Since ACE is more expressive than OWL and SWRL, this translation does not succeed for all ACE texts.

2.2.2.3 Prepositional Phrases

The correct attachment of prepositional phrases is a notorious problem of natural language processing (see e.g. [177, 40]). The problem can be best explained by looking at the famous example

A man sees a girl with a telescope.

where in natural English "with a telescope" can attach to "girl" or to "sees". In the first case, the girl has a telescope; in the second case the telescope is the instrument used for seeing the girl.

ACE resolves this ambiguity by the definition that all prepositional phrases in ACE attach to the closest preceding verb and cannot attach to nouns. The only exception are prepositional phrases using the preposition "of", which always attach to the immediately preceding noun and cannot attach to verbs. This exception is justified by the fact that "of"-phrases in natural English hardly ever attach to verbs.

This simple rule ensures that sentences containing prepositional phrases only have one reading in ACE. In the example above, "with a telescope" would refer to "sees" under the ACE semantics and give the following DRS:

A man sees a girl with a telescope.

```
A B C D
--------------------------------------
object(A,man,countable,na,eq,1)
object(B,girl,countable,na,eq,1)
object(C,telescope,countable,na,eq,1)
predicate(D,see,A,B)
modifier_pp(D,with,C)
```

The last line of this DRS shows that the preposition "with" connects the *see*-relation with the telescope.

In order to get the other meaning in ACE, one would have to rephrase the sentence, for example by using a relative clause instead of the prepositional phrase:

A man sees a girl who has a telescope.

```
A B C D E
--------------------------------------
object(A,man,countable,na,eq,1)
object(B,girl,countable,na,eq,1)
object(C,telescope,countable,na,eq,1)
predicate(D,have,B,C)
predicate(E,see,A,B)
```

In this DRS, the telescope is connected by the *have*-relation to the girl.

2.2.2.4 Plurals

Plurals in natural language are known to be interpretable in numerous ways. For example, the sentence

Four men lift three tables.

can be given at least eight readings [144]. Apart from ambiguous scopes (see the next section), the most important issue concerning plurals is the fact that they can be meant collectively or distributively. If the sentence

John visits four customers.

is interpreted in a collective way then the four customers are visited together, i.e. there is just one *visit*-relation between John and a group of four customers. If the same sentence is interpreted distributively, however, then each of the four customers is visited separately by John, i.e. there are four *visit*-relations in this case, one to each of the four customers. In fact, even more possible interpretations exist, e.g. John can have two *visit*-relations to two customers each.

Collective plurals are conceptually more complex than distributive ones because collective plurals have to be represented semantically as some kind of groups that can participate in relations. In the distributive reading, relations that syntactically attach to plurals refer semantically to the members of the plural group so that the plural group itself does not participate in relations.

The default semantics of ACE defines that plurals are always interpreted collectively unless they are preceded by the phrase "each of". Thus, the example above would be interpreted collectively and is represented as follows:

John visits four customers.

```
A B
-------------------------------------
object(A,customer,countable,na,eq,4)
predicate(B,visit,named('John'),A)
```

This representation is underspecified in the sense that the information about the plural phrase is present but no specific plural semantics is assigned to it. For example, one would not be able to detect an inconsistency with the sentence "John does not visit a customer" without additional axioms representing background knowledge about the relation between "four" and "a".

The distributive reading can be expressed in ACE by putting "each of" in front of the plural phrase:

John visits each of four customers.

```
A
--------------------------------------------------------------------
object(A,customer,countable,na,eq,4)

  B                      C
  -------------    =>    ----------------------------------
  has_part(A,B)          predicate(C,visit,named('John'),B)
```

Distributive plurals are represented as implications having only the predicate "has_part" on the left hand side. Again, this representation is underspecified and leads only to sensible reasoning results if an appropriate set of background axioms is used.

This underspecification also has the practical advantage that the plain plural form of ACE (without "each of") can be interpreted distributively in applications that do not need the collective reading of plurals, sidestepping the standard ACE semantics. This is done for example by the ACE to OWL translation [82] used by the AceWiki system that will be introduced in Section 4.4. In this way, the excessive use of "each of" can be avoided.

2.2.2.5 Scopes

Quantifiers like “every” or “it is false that” and other constructs like “if ... then” and “or” have a certain textual range of influence that is denoted as their *scope*. Scopes are another difficult problem in NLP and are often ambiguous (see e.g. [74]).

Scopes are traditionally a matter of semantics, which can be seen by the fact that scope ambiguity is usually considered semantic ambiguity [126]. However, as we have seen in Section 2.2.1 when discussing the syntax of anaphoric pronouns and variables, scopes in ACE also have an influence on the syntax of the language. To be precise, scopes should actually be considered a part of the ACE syntax and not just of the ACE semantics. This will be taken into account in Chapter 3 where a grammar notation for CNLs will be introduced.

While scopes only have a relatively small side-effect on the syntax of ACE, they have a big effect on the semantics. The scopes in languages like ACE correspond to the scopes of quantified variables in first-order logic. In discourse representation theory, they are represented as nested boxes. Scopes are relevant on the one hand for the proper semantic representation and on the other hand for the sensible resolution of anaphoric references.

Since scopes determine the range of influence of scope-triggering structures, resulting representations heavily depend on the interpretation of scopes. For example, the sentence

> It is false that John fails and Mary succeeds.

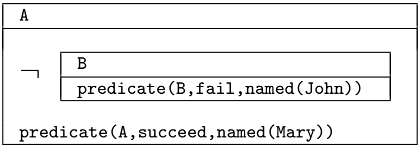
is in natural English ambiguous with respect to the scope of “it is false that”:

> $_($It is false that John fails$_)$ and Mary succeeds.
> $_($It is false that John fails and Mary succeeds$_)$.

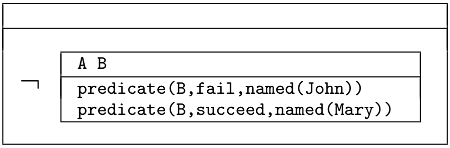
The gray subscript parentheses are not part of the language but are only used to indicate where the scopes open and close. In the first case “Mary succeeds” is not in the scope of “it is false that” and is not affected by the negation. In the second case however, “Mary succeeds” is in the scope of the negation. As one of the consequences, “Mary succeeds” can be concluded from the first sentence but not from the second one.

In ACE, structures like “it is false that” require a subordinated sentence that can only be coordinated by repeating the pronoun “that”. In this way, both readings can be represented in a distinct and clear way:

> $_($It is false that John fails$_)$ and Mary succeeds.


A
¬
B
predicate(B,fail,named(John))
predicate(A,succeed,named(Mary))

(It is false that John fails and that Mary succeeds).


¬
A B
predicate(B,fail,named(John))
predicate(B,succeed,named(Mary))

ACE defines in a relatively simple way where scopes open and where they close. Scopes in ACE always open at the position where the scope-triggering structure begins, with the exception of "or" where the scope opens before the conjunct on the left hand side of the "or". In the case of "if ... then", the scope opens at the position of the "if". Furthermore, questions and commands open scopes at their beginning. Scopes in ACE always close at the first position where a verb phrase, relative clause, *of*-phrase, possessive noun phrase, sentence, or a coordination thereof ends that contains the scope-triggering structure. For example, in the sentence

> Somebody is an ancestor of (every human) and is (not an ancestor of an animal).

the first scope opens at the position of the scope-triggering structure "every" and closes at the end of the respective *of*-phrase. The second scope is opened by "not" and closed after the respective verb phrase. Another example is

> (If a customer (who owns a house or who is rich) opens an account then the customer is assigned-to John).

where a scope opens at the position of "if" and closes at the end of the sentence and another scope opens before the relative clause that is the left-hand-side conjunct of "or" and closes at the end of the respective relative clause coordination.

2.2.2.6 Anaphoric References

In ACE, anaphoric references can be established by using pronouns like "him" or "herself", definite noun phrases like "the country" or "the customer who waits", or variables like "X" possibly combined with a definite noun phrase like "the object T1".

While anaphoric pronouns in ACE are required to be resolvable, definite noun phrases and variables are also allowed if they cannot be resolved to an antecedent. APE returns a warning message in these cases but generates the other outputs as normal. If a variable "X" cannot be resolved then it is simply interpreted as if it was "something X". If a definite noun phrase cannot be resolved then it is treated as if it was indefinite, i.e. using "a" instead of "the".

Pronouns need to be resolvable to an antecedent that agrees in number and gender. Furthermore, reflexive pronouns like "itself" or "herself" can be resolved only to the subject of the respective verb phrase and irreflexive pronouns like "it" or "him" can be resolved only to something that is not the subject:

Mary$_1$ helps herself$_1$.
Every employee$_1$ has a card$_2$ and uses it$_2$.
* Mary helps himself.
* Every employee has a card and uses him.

Every noun phrase is marked by an identifier shown as a gray subscript number in order to clarify the resolution of the anaphoric references. The gender of content words like “Mary” and “card” is defined by the lexicon. The examples above assume that “Mary” is defined as feminine and “card” as neuter.

Definite noun phrases are resolved to noun phrases that are structurally identical or a superset of the anaphoric noun phrase. For example, “card” can be resolved to “card that is valid” but not vice versa. Some examples are

John$_1$ has a card$_2$ and uses the card$_2$.
John$_1$ has a card$_2$ that is valid and uses the card$_2$.
John$_1$ has a card$_2$ that is valid and uses the card$_2$ that is valid.
John$_1$ has a card$_2$ and uses the card$_3$ that is valid.

where the last sentence is a syntactically valid ACE sentence but “the card that is valid” cannot be interpreted as an anaphoric reference because there is no antecedent that is identical or a superset.

All kinds of anaphoric references have to follow the principle of accessibility. Antecedents are only accessible for anaphoric references if they are not within a scope that has been closed up to the position of the anaphoric reference. Proper names are an exception in the sense that they are always accessible. For example, the anaphoric references “himself” and “the house” of the two sentences

$_($Every man$_1$ protects a house$_2$ from $_($every enemy$_{3)}$ and $_($does not destroy himself$_{1))}$.
$_($Every man$_1$ protects a house$_2$ from $_($every enemy$_{3)}$ and $_($does not destroy the house$_{2))}$.

can be resolved to “man” and “house”, respectively, because they are not contained in a scope that is closed up to the position of the anaphoric reference. However, “the enemy” in the case of

$_($Every man$_1$ protects a house$_2$ from $_($every enemy$_{3)}$ and $_($does not destroy the enemy$_{4))}$.

cannot be resolved because the only matching antecedent is in a scope that has been closed before the position of the anaphoric reference. The accessibility constraint does not apply if the antecedent is a proper name. The following example is a valid ACE text where “it” is resolved to the proper name “Switzerland”:

$_($No ocean$_1$ borders Switzerland$_{2)}$. It$_2$ is a landlocked country$_3$.

If more than one possible antecedent can be identified for a particular anaphoric reference then the principle of proximity applies. Anaphoric references are deterministically resolved to the textually closest possible antecedent. Below, two examples are shown where the anaphoric reference could in principle be resolved to two antecedents but is actually resolved to the closer of the two:

> John$_1$ is a son of Bill$_2$. He$_2$ is rich.
> A country$_1$ attacks a small country$_2$ and $_($does not attack a large country$_{3)}$.
> The country$_2$ wins.

The principle of proximity is needed in order to make ACE an unambiguous language.

2.2.3 Applications of ACE

Finally, a brief overview is given of the application areas in which ACE has been applied so far.

In the beginning, ACE was designed by the Attempto group as a specification language [54] and has in this context, among other things, been applied to model generation [53]. Later, the focus of ACE shifted from specifications towards knowledge representation and especially the Semantic Web, which was mainly due to the fact that the Attempto group joined the network "Reasoning on the Web with Rules and Semantics" (REWERSE)[7] that lasted from 2004 until 2008. During this time, ACE has been applied by the members of the Attempto group as an interface language of a first-order reasoner [52], as a query language for the Semantic Web [13], as an annotation language for web pages [55], and as a general Semantic Web language [82]. Furthermore, as a part of my master's thesis I investigated how ACE can be used to summarize scientific results in the biomedical domain [92]. Recently, we have explored the use of ACE for clinical practice guidelines [151].

Also outside of the Attempto group, ACE has been applied in several areas. It has been used as a natural front-end for different rule languages [71, 100], for importing texts in controlled English to a reasoning system [22], and was the basis for an automatic translation system [5].

2.3 Knowledge Representation

Besides the research area of controlled natural language, knowledge representation is another area that is important for this thesis.

Knowledge representation is a discipline that is usually seen as a subfield of artificial intelligence. It is about representing human knowledge in a form that enables some sort of automatic reasoning, i.e. the inference of new knowledge out of existing one. According to John Sowa [156], knowledge representation builds upon three

[7] `http://rewerse.net`

existing fields: *logic* for the formal structure and for the laws of inference, *ontology* for naming and defining the things we want to talk about, and *computation* for the ability to create computer applications out of it.

Different formalisms have been proposed for structuring knowledge, e.g. frames [108] and semantic networks [179]. Often, these languages initially did not have a clearly defined meaning and thus could not be considered proper logic languages. Many of them have been revised in this respect, however, and had a significant influence on subsequent languages like KL-ONE [20] which comes with carefully defined formal semantics.

In terms of ontologies, a number of projects like Cyc [98] have emerged that aim to encode human common sense knowledge in a formal way. The outcomes of other initiatives like WordNet [46] cannot be considered ontologies in the strict sense but are still a very valuable resource for building ontologies for the purpose of knowledge representation.

Also on the computation aspect, a lot of work has been done. Automated reasoning is one of the most important and most complex computation tasks in the area of knowledge representation. Various reasoners exist today that can compute inferences in full first-order logic [163] or in more tractable subsets thereof like Description Logics (e.g. [153, 170]). Beside that, many rule engines exist that can reason with non-standard extensions of classical logic like negation as failure (e.g. [118]).

Thus, the basic ingredients for knowledge representation are available today. Nevertheless, the big break-through of knowledge representation systems did not yet happen. One could ask: *Why?*

There are certainly many possible answers to this question. In my view, however, usability is the most important reason for the missing practical success of knowledge representation approaches. Too little effort has been spent on how human users should interact with such systems, even though usability is known to be a critical issue [106]. Another problem is that user interfaces were mostly put on top of systems that were already finished otherwise. This is not the right approach, as I will argue in the introduction of Chapter 4. Good user interfaces emerge if they are under consideration from the very beginning of the design of the complete system.

Many different branches in the field of knowledge representation exist. Two popular ones — expert systems and the Semantic Web — will be discussed in more detail below. Furthermore, two general usability problems of knowledge representation will be highlighted, namely (1) how to put knowledge into the system and (2) how to get it out again.

2.3.1 Expert Systems

Expert systems are software tools that store expert knowledge and can provide practical advice on the basis of such knowledge. The basic idea is to create programs that can replace human experts to some degree in order to make their knowledge more available.

Research on expert systems began in the 1960s. Dendral [24] was one of the first expert systems, initiated in 1965. It was designed to help chemists to analyze organic molecules and was very successful. Encouraged by this and other success stories, expert systems became very popular in the 1970s and 1980s. R1 (later called XCON) [105] is another example of a successful expert system. It has been developed by Digital Equipment Corporation (today Hewlett-Packard) and could automatically select appropriate configurations of computer system components on the basis of a customer's purchase order. However, many other systems failed to achieve the same success and only a minority of them found widespread usage [62].

The main reason why such systems often failed in practice — apart from organizational factors — was the poor adaptation by its users and only to a lesser degree technical factors [62]. Arguably because of the bad reputation caused by systems that failed to follow up on their promises, the field of expert systems gradually disappeared again from the scientific landscape in the 1990s.

Also in the 1990s, it was proposed to use controlled English in expert systems in order to make knowledge acquisition and maintenance more user friendly [134]. However, this approach was not further pursued and — to my knowledge — no expert system exists to date that is based on CNL.

2.3.2 Semantic Web

The vision of the Semantic Web has been presented by Berners-Lee (the inventor of the World Wide Web) and others in 2001 [11]. The basic idea is to do knowledge representation on the scale of the web. The current World Wide Web should be transformed and extended so that its content can also be understood by computers, at least to some degree. This should be achieved by publishing not only documents on the web but knowledge descriptions with a precisely defined formal meaning. Ideally, the Semantic Web should become as pervasive as the traditional World Wide Web today.

The Semantic Web has received broad attention in the academic world during the past ten years and large amounts of work have been invested to make it a reality. However, as Berners-Lee and his colleagues had to admit in 2008, the idea of the Semantic Web "remains largely unrealized" [150].

Most technical problems could be solved and many Semantic Web languages have been defined and standardized, like RDF [102], the Web Ontology Language (OWL) [16], and SPARQL [132]. Nevertheless, the Semantic Web did not yet happen. The lack of usability is again a plausible reason for this. The logic-based languages and the theories behind them are too complicated for casual web users and the current user interfaces fail to hide this complexity.

In order to solve this problem, natural language interfaces like PANTO [175], GINO [12], and others [88] have been developed. Furthermore, several proposals have been made recently to use CNLs as interface languages for the Semantic Web (see [147, 82, 145] and Section 2.1.3.3). However, the vast majority of those CNL-based tools for the Semantic Web are still early prototypes and the definite proof that they can make better Semantic Web interfaces has yet to be provided.

2.3.3 Knowledge Acquisition and Maintenance

A major problem that is manifest in all types of knowledge representation systems is to reliably acquire and maintain knowledge about a certain domain. The knowledge acquisition process has initially been seen as a *transfer process* that is about transferring the knowledge from the head of domain experts into formal representations to be stored in knowledge representation systems. As it turned out, however, this view of knowledge acquisition was inadequate. Human knowledge does not seem to be stored in human brains in a way that can be mapped easily to formal representations. Instead, the knowledge acquisition process must be regarded as a *modeling process* [30, 162].

Apparently, this modeling process requires some effort on the side of the domain expert whose knowledge should be represented. The main difficulty is that the domain experts are usually not familiar with the methodologies and languages of knowledge representation. On the other hand, the knowledge representation experts are unlikely to be experts in the particular domain. Traditionally, this problem is solved by a team-based approach where a domain expert and a knowledge engineer work together to develop an appropriate formal model of the given area.

This approach is relatively expensive since it requires the availability of both, skilled knowledge engineers and competent domain experts. Furthermore, this approach bears the danger of different kinds of misunderstanding between the domain expert and the knowledge engineer.

For those reasons, it would be much more convenient and reliable if the domain experts could formalize their own knowledge in a more or less autonomous way. However, existing knowledge acquisition approaches that get along without the presence of a knowledge engineer are known to have limitations [173].

Again, CNLs could be the solution for this. CNLs are claimed to make knowledge representations easier to understand and verify for domain experts, and thus could hopefully reduce the need for knowledge engineers in the future [69].

2.3.4 Accessing Knowledge Bases

Once knowledge is represented in a formal way and stored in a knowledge base, the next problem is how to retrieve it. The people who want to access the knowledge base can again in most cases not be expected to be familiar with knowledge representation formalisms. In contrast to the knowledge acquisition task, it cannot practically be assumed that a knowledge engineer is present every time the knowledge base should be accessed. Therefore, it is crucial to provide user interfaces that do not presuppose a knowledge engineering background.

One possible approach to this problem are interfaces that accept full natural language, as, for example, the ORAKEL system [29] and various proposed natural language database interfaces [70, 174]. While such interfaces indeed simplify knowledge access, they also make it less reliable. Arguably, this is the reason why such systems could never find widespread usage.

The use of CNL instead of full natural language could solve this problem. This approach was proposed in the 1980s [113] but the first working prototypes emerged much later (e.g. [13]). It can be assumed that this CNL approach works particularly well for knowledge bases that already used a CNL approach for acquiring the knowledge and store not only logical but also linguistic properties of the knowledge elements.

In summary, one can say that usability aspects have not received the same attention as technical issues in the research area of knowledge representation. While the technical formats and methods seem to have reached a certain degree of maturity, the problem of user interfaces is still unsolved. In my view, CNLs have a great potential to bring us forward in this respect and this potential will be explored in the remainder of this thesis.

CHAPTER 3

Grammar

Languages, both natural and formal, are usually defined by grammars. I will argue that controlled natural languages have requirements concerning their grammars that differ from those of natural languages and also from those of other formal languages. This chapter thus targets the first research question defined in the introduction of this thesis:

1. How should controlled English grammars be represented?

The attention will be restricted on CNLs to be used within predictive editors, as motivated in Section 2.1.4. As we will see, predictive editors pose some specific requirements on the grammar notation.

I will first outline the requirements for a grammar notation to describe the syntax of languages like ACE to be used in predictive editors (Section 3.1). Then, I will show that these requirements cannot be satisfied by existing grammar frameworks (Section 3.2). On the basis of the requirements for CNL grammars, a grammar notation is defined that I call *Codeco* (Section 3.3). I will show how the Codeco notation can be implemented in Prolog as a definite clause grammar (Section 3.4) and how it can be interpreted in a chart parser (Section 3.5). The introduced Codeco notation only covers the syntax of a language and does not include any semantics. I will briefly sketch how semantics could be added and discuss other possible extensions (Section 3.6).

I will then introduce a grammar written in Codeco that describes a large subset of ACE (Section 3.7). A number of tests have been performed on the basis of this grammar. Their results will be described (Section 3.8).

3.1 Controlled English Grammar Requirements

I devise a list of requirements that grammars of controlled English have to meet if they are to be defined, implemented, used, and reused efficiently, under the assumption that the predictive editor approach is taken. These requirements originate from the experiences with defining subsets of ACE used in a predictive editor and are partly justified by the evaluation results of the tools to be presented in Chapter 4.

The most important requirement is that such grammars should be defined in a concrete and declarative way. Furthermore, in order to be usable within predictive editors, lookahead features should be implementable efficiently, i.e. it should be possible to find out with reasonable effort which words can follow a partial text. Additionally, referential elements like anaphoric pronouns should be supported but restricted to cases where an antecedent exists that they can point to. This also requires that scopes can be described. Finally, for the sake of practicality, it is necessary that such grammars are relatively easy to implement.

These requirements will now be explained in more detail.

Concreteness

Concreteness is an obvious requirement. Due to their practical and computer-oriented nature, CNL grammars should be concrete. Concrete grammars are fully formalized and can be read and interpreted by programs, as opposed to abstract grammars that are informal and cannot be processed automatically without additional work.

Declarativeness

As a second requirement, CNL grammars should be declarative in order to facilitate their usage by different kinds of tools. Grammars are declarative if they are defined in a way that does not depend on a concrete algorithm or implementation. Declarative grammars have the advantage that they can be completely separated from the parser that processes them. This makes it easy for such grammars to be used by other programs, for the parser to be changed or replaced, or to have different parsers for the same language.

Another advantage of declarative grammars is that they can be shared easily between different parties using the same CNL, thus ensuring compatibility between systems.

Furthermore, declarative grammars are easy to change and reuse. The removal of some rules from a grammar leads to a language that is a subset of the language described by the initial grammar. In the same way, the addition of rules leads to a superset of the language. With non-declarative grammars, these properties do usually not hold.

Lookahead Features

Predictive editors require the availability of lookahead features, i.e. the possibility to find out how a partial text can be continued. For this reason, CNLs must be defined in a form that enables the efficient implementation of such lookahead features.

Concretely, this means that a partial text, for instance "a brother of Sue likes ...", can be given to the parser and that the parser is able return the complete set of words that can be used to continue the partial sentence according to the grammar. For the given example, the parser might say that "a", "every", "no", "somebody", "John", "Sue", "himself" and "her" are the possibilities to continue the partial sentence.

The Figures 4.2 and 4.3 of Chapter 4 clarify the lookahead features requirement. Efficient lookahead support is crucial for implementing predictive editors.

Anaphoric References and Scopes

Anaphoric references require special handling in CNL grammars. It should be possible to describe the circumstances under which anaphoric references are allowed in an exact, declarative, and simple way that — in order to have a clear separation of syntax and semantics — does not depend on the semantic representation. Ideally, anaphoric references should be possible not only to sentence-internal antecedents but also to those in preceding sentences.

Concretely, a CNL should allow the use of a referential expression like "it" only if a matching antecedent (e.g. "a country") can be identified in the preceding text, as shown for ACE in Section 2.2.2.6. Every sentence that contains an expression that can only be interpreted in a referential way but cannot be resolved must be considered syntactically incorrect. Thus, CNL grammars have to be able to represent the fact that anaphoric references should be allowed only if they can be resolved.

As shown in Section 2.2.2.6, the resolvability of anaphoric references depends on the scopes of the preceding text. Scopes are raised by certain structures like negation, and they cover certain areas of the text that denote the range of influence of the respective expression. Section 2.2.2.5 showed that scopes can be ambiguous in natural language and explained how they are defined in ACE. While, in natural languages, scopes can be considered a semantic phenomenon, they have to be treated as a syntactic issue in CNLs if the restrictions on anaphoric references are to be described appropriately.

Thus, a grammar that defines the syntax of a CNL needs to specify anaphoric references, their antecedents, and the positions at which scopes are opened and closed.

Implementability

Finally, a CNL grammar notation should be easy to implement in different programming languages. As a consequence, a CNL grammar notation should be neutral with respect to the programming paradigm of its parser.

The implementability requirement is motivated by the fact that the usability of CNLs heavily depends on good integration into user interfaces like predictive editors. For this reason, it is desirable that the CNL parser is implemented in the same programming language as the user interface component of a tool based on CNL.

Another reason why implementability is important is that the parser is often not the only tool that needs to know the CNL grammar. There can be many other tools that need to read and process the grammar, e.g. general editors (see Section 2.1.5), paraphrasers [84] and verbalizers[1]. Furthermore, more than one parser might be necessary for practical reasons. For example, a simple top-down parser is probably the best for parsing large texts in batch mode and for doing regression tests (e.g. through language generation). On the other hand, a chart parser is better suited for providing lookahead capabilities.

3.2 Existing Grammar Frameworks

Many different grammar frameworks exist. Some are aimed at describing natural languages whereas others focus on defining formal ones. In the case of formal languages, such grammar frameworks are usually called *parser generators*. Furthermore, definite clause grammars (DCG) are a simple but powerful formalism that is used for both natural and formal languages.

As I will show, none of these frameworks fully satisfies the requirements for controlled English grammars defined above.

3.2.1 Natural Language Grammar Frameworks

Since CNLs are based on natural languages, CNL grammars share many properties with natural language grammars. For example, CNL grammars describe the same words and word classes and use the same syntactic structures like sentences, noun phrases, verb phrases, and relative clauses.

A large number of different grammar frameworks exist to process natural languages. Some of the most popular ones are *Head-Driven Phrase Structure Grammars* (HPSG) [127], *Lexical-Functional Grammars* [86], *Tree-Adjoining Grammars* [80], *Combinatory Categorial Grammars* [160], and *Dependency Grammars* [107]. More of them are discussed by Cole et al. [35]. Most of these frameworks are defined in an abstract and declarative way. Concrete grammar definitions based on such frameworks, however, are often not fully declarative.

[1] see e.g. `http://attempto.ifi.uzh.ch/site/docs/owl_to_ace.html`

Despite many similarities, a number of important differences between natural language grammars and grammars for CNLs can be identified that have the consequence that the grammar frameworks for natural languages do not work out very well for CNLs. Most of the differences originate from the fact that the two kinds of grammars are the results of opposing goals. Natural language grammars are descriptive in the sense that they try to describe existing phenomena. CNL grammars, in contrast, are prescriptive, meaning that they define something new. Thus, one could say that natural language grammars are *language descriptions* whereas CNL grammars are *language definitions*. A range of further differences originate from this most fundamental distinction.

For instance, grammars for natural languages and those for CNLs differ in complexity. Natural languages are very complex and this has the consequence that the grammars describing such languages can also be very complex. For this reason, grammatical frameworks for natural languages — like the ones mentioned above — have to be very general in order to be able to describe natural languages in an appropriate way. For CNLs — that are typically much simpler and abandon natural structures that are difficult to process — these frameworks would be an overkill.

Partly because of the high degree of complexity, providing lookahead features on the basis of those frameworks is very hard. Another reason is that lookahead features are simply not relevant for natural language applications, and thus no special attention has been given to this problem. The difficulty of implementing lookahead features with natural language grammar frameworks can be seen by the fact that no predictive editors exist for CNLs that emerged from an NLP background like CPL or CLOnE.

A further problem is caused by the fact that many implementations of the grammar frameworks for natural languages depend on logic-based programming languages like Prolog. Such grammars are usually very hard to process in procedural or object-oriented languages like C or Java. This is because Prolog-based grammars depend on unification and backtracking that come for free in this language. In procedural languages, however, unification and backtracking are not built-in and grammars relying on them are hard to interpret.

The handling of ambiguity is another important difference. Natural language grammars have to deal with the inherent ambiguity of natural languages. Context information and background knowledge can help resolving ambiguities (e.g. structural ambiguities) but there is always a remaining degree of uncertainty. Natural language grammar frameworks are designed to be able to cope with such situations, can represent structural ambiguity by using underspecified representations, and require the parser to disambiguate by applying heuristic methods. In contrast, CNLs (the formal ones on which this thesis focuses) remove structural ambiguity by their design, which makes underspecification and heuristics unnecessary in most cases.

Finally, the resolution of anaphoric references to appropriate antecedents is another particularly difficult problem for the correct representation of natural language.

In computational linguistics, this problem is usually solved by applying complex algorithms to find the most likely antecedents (see e.g. [95, 59, 78]). The following example will clarify why this is such a difficult problem: An anaphoric pronoun like "it" can refer to a noun phrase that has been introduced in the preceding text but it can also refer to a broader structure like a complete sentence or paragraph. It is also possible that "it" refers to something that has been introduced only in an implicit way or to something that will be identified only in the text that follows later. Furthermore, "it" can refer to something outside of the text, meaning that background knowledge is needed to resolve it. Altogether, this has the consequence that sentences like "an object contains it" have to be considered syntactically correct even if no matching antecedent for "it" can be found in the text.

In order to address the problem of anaphoric references, natural language grammar frameworks like HPSG establish "binding theories" [28, 127] that consist of principles that describe under which circumstances two components of the text can refer to the same thing. Applying these binding theories, however, just gives a set of possible antecedents for each anaphor but does not allow for deterministic resolution of them. This stands in contrast to the requirements for CNL grammars where anaphoric references should always be resolvable in a deterministic way.

3.2.2 Parser Generators

Apart from the approaches introduced above to define grammars for natural languages, a number of systems exist that are aimed at the definition and parsing of formal languages (e.g. programming languages). In the simplest case, grammars for formal languages are written in Backus-Naur Form [117, 89]. Examples of more sophisticated grammar formalisms for formal languages — called *parser generators* — include Yacc [79], GNU bison[2] and ANTLR [122]. The general problem of these formalisms is that context-sensitive constraints cannot be defined in a declarative way.

Simple context-free languages can be described in a declarative and simple way, e.g. by using plain Backus-Naur style grammars. However, such grammars are very limited and even very simple CNLs cannot be defined appropriately. The description of number and gender agreement restrictions, for example, could theoretically be described in such grammars but not in a practical way.

It is possible to describe more complex grammars containing context-sensitive elements with such parser generators. However, this has to be done in the form of procedural extensions that depend on a particular programming language to be interpreted. Thus, the property of declarativeness gets lost when more complex languages are described.

Furthermore, since such parser generators are designed to describe formal languages they have no special support for natural language related things like anaphoric

[2] http://www.gnu.org/software/bison/

references.

Before discussing the lookahead capabilities of parser generators, it has to be noted that the term *lookahead* is somewhat overloaded and is sometimes used with a different meaning. In the context of parsers for formal languages, *lookahead* denotes how far the parsing algorithm looks ahead in the fixed token list before deciding which rule to apply. Lookahead in our sense of the word — i.e. predicting possible next tokens — is not directly supported by existing parser generators. However, as long as no procedural extensions are used, this is not hard to implement. Actually, lookahead features for formal languages are available in many source code editors in the form of *code completion*. The "code assist" feature of the Eclipse IDE, for example, provides code completion and is one of the most frequently used features by Java developers [114].

Formalized English is an example of a CNL that is defined in a parser generator language [103]. It is a quite simple language only covering a very small part of natural English and having many formal looking elements in it. For such simple and not fully natural-looking languages, the parser generator approach to CNL grammars can be viable.

3.2.3 Definite Clause Grammars

Definite clause grammars (DCGs) [125], finally, are a simple but powerful way to define grammars for natural and formal languages and are mostly written in logic-based programming languages like Prolog. In fact, many of the grammar frameworks for natural languages introduced above are usually implemented on the basis of Prolog DCGs.

In their core, DCGs are fully declarative and can thus in principle be processed by any programming language. Since they build upon the logical concept of definite clauses, they are easy to process for logic-based programming languages. In other programming languages, however, a significant overhead is necessary to simulate backtracking and unification. Thus, DCGs are a good solution when using logic-based programming languages but interpreting them in other languages is not practical in many cases.

DCGs are good in terms of expressivity because they are not necessarily context-free but can contain context-sensitive elements. Anaphoric references, however, are again a problem. Defining them in an appropriate way is difficult in plain DCGs. The following two exemplary DCG rules show how antecedents and anaphors could be defined in a Prolog DCG grammar:

```
np(Agr, Ante-[Agr|Ante]) -->
    determiner(Agr),
    noun(Agr).

np(Agr, Ante-Ante) -->
```

```
        anaphoric_pronoun(Agr),
        { once(member(Agr,Ante)) }.
```

The code inside the curly brackets defines that the agreement structure of the pronoun is unified with the first possible element of the antecedent list.

This approach has some problems. First of all, the curly brackets contain code that is not fully declarative. A more serious problem, however, is the way how connections between anaphors and antecedents are established. In the example above, the accessible antecedents are passed through the grammar by using input and output lists of the form "`In-Out`" so that new elements can be added to the list whenever an antecedent occurs in the text. The problem that follows from this approach is that the definition of anaphoric references cannot be done locally in the grammar rules that actually deal with anaphoric structures but they affect almost the complete grammar. In fact, it affects all rules that are potentially involved on the path from an anaphor to its antecedent. For example, a grammar rule dealing with the general structure of a sentence would look as follows:

```
    s(AnteIn-AnteOut) -->
        np(Agr, AnteIn-AnteTemp),
        vp(Agr, AnteTemp-AnteOut).
```

As this example shows, anaphoric references also have to be considered when writing grammar rules that have otherwise nothing to do with anaphors or antecedents. This is not very convenient and it would be desirable to be able to define anaphoric references in a simpler way only at the level of the grammar rules that actually deal with antecedents and anaphors.

Different extensions of DCGs have been defined in order to describe phenomena of natural language in a more appropriate way. Extraposition Grammars [124], for example, extend DCGs for a better description of natural language constructions called *left extrapositions*, i.e. phenomena of sentence elements appearing at earlier positions than normal.

Assumption Grammars [39] are another variant of DCGs motivated by natural language phenomena that are hard to express otherwise, like free word order and complex cases of coordination. Anaphoric references can be represented in a very simple and clean way with Assumption Grammars. The following example (taken from [39]) shows how this can be done:

```
    np(X,VP,VP) :- proper_name(X), +specifier(X).
    np(X,VP,R) :- det(X,NP,VP,R), noun(X-F,NP), +specifier(X-F).
    np(X,VP,VP) :- anaphora(X), -specifier(X).
```

The "`+`"-operator is used to establish "hypotheses" that can be consumed later by the use of the "`-`"-operator. In this way, the restrictions for anaphoric references can be defined locally in the rules that define the structure of antecedents and anaphors. This solution comes very close to what we need for CNLs. However, there are still some problems. It is unclear how anaphors can be resolved deterministically if more than one matching antecedent is available, and how irreflexive pronouns like “it” can be prevented from referring to the subject of the respective verb phrase.

A further problem with the DCG approach concerns lookahead features. In principle, it is possible to provide lookahead features with standard Prolog DCGs as Rolf Schwitter and I could show [93]. However, this approach is not very efficient and can become impractical for complex grammars and long sentences. For DCG variants like Assumption Grammars, it is unknown how lookahead could be implemented.

3.2.4 Concluding Remarks on Existing Grammar Frameworks

In summary, none of the introduced existing grammar frameworks fully satisfies all requirements for CNL grammars.

The grammar frameworks for natural languages do not work out well for CNLs: the high complexity of these frameworks makes it very hard to use such grammars in different programming languages and to provide lookahead features. Furthermore, anaphoric references cannot be restricted to be allowed only if they can be resolved.

Parser generators — as used to define formal languages — have the problem that context-sensitive structures cannot be defined in a declarative way. They cannot combine context-sensitivity and declarativeness. Since they are targeted towards formal languages, they also have no special support for natural language structures like anaphoric references.

DCGs fulfill most of the requirements set out but unfortunately not all of them: apart from the fact they are hard to interpret efficiently in programming languages not based on logic, they also have no fully satisfying support for anaphoric references, even though Assumption Grammars come very close in this respect. (Nevertheless, DCGs have many good properties for defining CNL grammars and, in fact, the grammar notation to be introduced in the remainder of this chapter can be translated into a Prolog DCG notation.)

Especially the proper definition of anaphoric references in CNL grammars seems to be a real problem. This problem is illustrated by the work of Namgoong and Kim [115], which does not really fit into one of the categories introduced above. The core of the CNL they introduce is a simple context-free grammar that is defined declaratively but cannot cover things like anaphoric references. For this reason, they had to extend their notation for representing anaphoric references. The extended grammar notation, however, is not declarative anymore and thus the advantages of declarative grammars are lost.

In order to overcome the discussed problems, I will introduce a new grammar notation designed specifically for CNLs.

The Grammatical Framework (GF) [5] is a grammar framework that is similar to the approach to be presented here in the sense that it is designed specifically for CNLs. It does not focus on languages with a deterministic mapping to logic, however, but on machine translation between controlled subsets of different natural languages. It allows for declarative definition of grammar rules and can be used in predictive editors, but it lacks support for anaphoric references.

3.3 The Codeco Notation

On the basis of the aforementioned requirements, I created a grammar notation called *Codeco*, which stands for "*co*ncrete and *de*clarative grammar notation for *co*ntrolled natural languages". This notation has been used to describe a large subset of ACE that will be introduced later in this chapter and is shown in its full extent in Appendix A.

The Codeco notation has been developed with ACE in mind, and the elements of Codeco will be motivated by ACE examples. Nevertheless, this notation should be general enough for other controlled subsets of English, and for controlled subsets of other languages. Codeco can be conceived as a proposal for a general CNL grammar notation. It has been shown to work well for a large subset of ACE, but it cannot be excluded that extensions or modifications would become necessary to be able to express the syntax of other CNLs. Some possible extensions will be discussed in Section 3.6.

The elements of the Codeco notation are shown here in a pretty-printed form. Additionally, a Prolog-based representation is introduced that uses Prolog as a representation language (i.e. not as a programming language). Thus, Codeco grammars can be represented in Prolog, but are not Prolog programs.

In the following sections, the different elements of the Codeco notation are introduced, i.e. grammar rules, grammatical categories, and certain special elements. After that, the issue of reference resolution is discussed in detail.

Later in this chapter, it will be shown how Codeco grammars can be run in Prolog as DCG grammars, how they can be processed in chart parsers, and how lookahead features can be implemented.

3.3.1 Simple Categories and Grammar Rules

Basically, a Codeco grammar is a set of grammar rules that are composed of grammatical categories, e.g. "*np*" standing for noun phrases and "*vp*" standing for verb phrases. Additionally, there must be a designated start category that represents all possible well-formed statements of the language, e.g. "*s*" standing for sentences of the given language.

Grammar rules in Codeco use the operator "$\overset{:}{\rightarrow}$" (where the colon on the arrow is needed to distinguish normal rules from scope-closing rules as they will be introduced later). Grammar rules make use of grammatical categories, for instance:

$$vp \quad \overset{:}{\longrightarrow} \quad v \quad np$$

On the left hand side, the arrow operator takes exactly one category that is called the *head* of the rule (i.e. "*vp*" in the example above). On the right hand side, there must be a sequence of categories which is called the *body* of the rule (i.e. the sequence of the two categories "*v*" and "*np*" in the example above). Such grammar rules state that the category of the head can be derived from (we can also say *expanded to*) the sequence of categories of the body. The example shown above thus means that a "*vp*" consists of a "*v*" followed by an "*np*".

The right hand side of a rule can be empty, i.e. a sequence of zero categories. An example is shown here:

$$adv \;\; \overset{:}{\longrightarrow}$$

Such rules state that the category of the head can be derived from an empty sequence of categories, i.e. they define optional parts of the grammar. The example above means that an "*adv*" is always optional when it appears in the body of another rule.

Terminal categories are categories that cannot be expanded further. Such categories correspond to the tokens of the input text. In Codeco, terminal categories are represented in square brackets:

$$v \;\; \overset{:}{\longrightarrow} \;\; [\,\text{does not}\,] \;\; verb$$

This example means that if the input text contains the token "does not" followed by something that can be considered a "*verb*", then they can together be considered a "*v*".

In their Prolog representation, Codeco grammar rules are represented by using the infix operator "`=>`". The predefined Prolog operator "`,`" is used to connect the different categories of the body. Terminal categories are represented as Prolog lists, and the empty list is used to represent an empty category sequence. Therefore, the above examples would be represented as follows in Prolog:

```
vp => v, np.
adv => [].
v => ['does not'], verb.
```

In these simple cases, the Codeco Prolog notation is very similar to the Prolog DCG notation, the only difference being that "`=>`" is used instead of "`-->`".

The Codeco notation introduced so far allows us to define simple context-free grammars. Looking at our requirements for a CNL grammar notation, we still need means for the representation of context-sensitive structures and for anaphoric references. First of all, however, we need to be able to define lexicon entries.

3.3.2 Pre-terminal Categories

In order to provide a clean interface between grammar and lexicon, Codeco has a special notation for pre-terminal categories. Pre-terminal categories are conceptually somewhere between non-terminal and terminal categories, in the sense that they can be expanded but only in a very restricted way. In order to distinguish them from the other types of categories, they are marked with an underline:

$$np \quad \overset{:}{\longrightarrow} \quad [\,\text{a}\,] \quad \underline{noun}$$

Pre-terminal categories can expand only to terminal categories. This means that pre-terminal categories can occur on the left hand side of a rule only if the right hand side consists of exactly one terminal category. Such rules are called *lexical rules* and are displayed with a plain arrow, for instance:

$$\underline{noun} \quad \rightarrow \quad [\,\text{person}\,]$$

In the Prolog notation, lexical rules are represented in the same way as grammar rules, and pre-terminal categories use the prefix operator "$":

```
    np => [a], $noun.
    $noun => [person].
```

Lexical rules can be stored in a dynamic lexicon but they can also be part of the static grammar.

3.3.3 Feature Structures

In order to support context-sensitivity, non-terminal and pre-terminal categories can be augmented by flat feature structures. Feature structures [87] are sets of name/value pairs; they are shown here using the colon operator ":" where the name of the feature stands on the left hand side and the value on the right hand side. Values can be variables, which are displayed as boxes:

$$vp\begin{pmatrix}\text{num: }\boxed{\text{Num}}\\ \text{neg: }\boxed{\text{Neg}}\end{pmatrix} \quad \overset{:}{\longrightarrow} \quad v\begin{pmatrix}\text{num: }\boxed{\text{Num}}\\ \text{neg: }\boxed{\text{Neg}}\\ \text{type: tr}\end{pmatrix} \quad np\begin{pmatrix}\text{case: acc}\end{pmatrix}$$

$$v\begin{pmatrix}\text{neg: }+\\ \text{type: }\boxed{\text{Type}}\end{pmatrix} \quad \overset{:}{\longrightarrow} \quad [\,\text{does not}\,] \quad verb\begin{pmatrix}\text{type: }\boxed{\text{Type}}\end{pmatrix}$$

$$np\begin{pmatrix}\text{noun: }\boxed{\text{Noun}}\end{pmatrix} \quad \overset{:}{\longrightarrow} \quad [\,\text{a}\,] \quad \underline{noun}\begin{pmatrix}\text{text: }\boxed{\text{Noun}}\end{pmatrix}$$

The names of the features are atoms (i.e. atomic symbols) and their values can be atoms or variables. The feature values "plus" and "minus" are pretty-printed as "+" and "–", respectively. The order in which the features are listed has no semantic relevance.

An important restriction is that feature values *cannot* be feature structures themselves. This means that feature structures in Codeco are always flat. The restriction to flat feature structures should keep Codeco simple and easy to implement. In theory, however, this restriction can easily be dropped as Section 3.6.2 will show.

In the Codeco Prolog notation, features are represented by predicate arguments in the form of name/value pairs separated by the infix operator ":". The given examples would be represented as follows:

```
vp(num:Num,neg:Neg) => v(num:Num,neg:Neg,type:tr), np(case:acc).
v(neg:plus,type:Type) => ['does not'], verb(type:Type).
np(noun:Noun) => [a], $noun(text:Noun).
```

Names starting with uppercase letters are considered variables in Prolog, e.g. "Num" and "Type".

Due to the support for feature structures, not only context-free languages can be defined in Codeco but also context-sensitive aspects can be modeled, e.g. number agreement restrictions.

3.3.4 Normal Forward and Backward References

So far, the introduced elements of Codeco are quite straightforward and not very specific to CNLs or predictive editors. The support for anaphoric references, however, requires some novel extensions.

In principle, it is easy to support sentences like

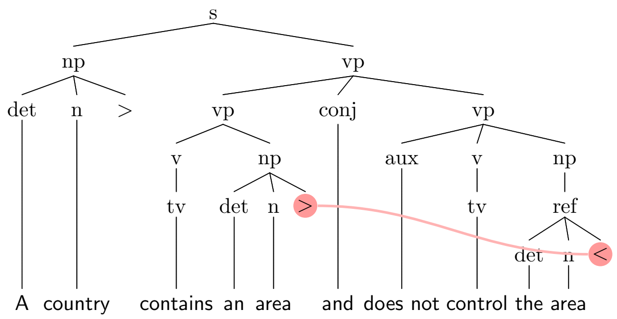
> A country contains an area that is not controlled by the country.
> If a person X is a relative of a person Y then the person Y is a relative of the person X.

where "the country", "the person X" and "the person Y" are resolvable anaphoric references. However, given that we only have the Codeco elements introduced so far, it is not possible to suppress sentences like

> Every area is controlled by the country.
> The person X is a relative of the person Y.

where "the country", "the person X" and "the person Y" are not resolvable. This can be acceptable (e.g. ACE supports such sentences by simply interpreting the definite noun phrases in a non-anaphoric way) but in many situations it is better to disallow such non-resolvable references. For example, anaphoric references might be harder to understand for the users if a predictive editor also suggests "the country" at positions where it cannot be meant anaphorically.

In Codeco, anaphoric references can be defined in a way, so they can be used only at positions where they can be resolved. This is done by using the special categories ">" and "<". These special categories allow us to establish special kinds of nonlocal dependencies across the syntax tree as the following illustration shows:



“>” represents a *forward reference* and marks a position in the text to which anaphoric references can refer to, i.e. “>” stands for antecedents. “<” represents a *backward reference* and refers back to the closed possible antecedent, i.e. “<” stands for anaphors. These special categories can have feature structures and they can occur only in the body of rules, for example:

$$np \overset{:}{\longrightarrow} [\,\mathrm{a}\,]\quad \underline{noun}\left(\text{text: }\boxed{\text{Noun}}\right)\ >\!\left(\begin{array}{l}\text{type: noun}\\ \text{noun: }\boxed{\text{Noun}}\end{array}\right)$$

$$ref \overset{:}{\longrightarrow} [\,\mathrm{the}\,]\quad \underline{noun}\left(\text{text: }\boxed{\text{Noun}}\right)\ <\!\left(\begin{array}{l}\text{type: noun}\\ \text{noun: }\boxed{\text{Noun}}\end{array}\right)$$

The forward reference of the first rule establishes an antecedent to which later backward references can refer to. The second rule contains such a backward reference that refers back to an antecedent with a matching feature structure. In this example, forward and backward references have to agree in their type and their noun (represented by the features “type” and “noun”). This has the effect that “the country”, for example, can refer to “a country” but “the area” cannot.

Forward references always succeed, whereas backward references succeed only if a matching antecedent in the form of a forward reference can be found somewhere earlier in the syntax tree. Every backward reference must connect to exactly one forward reference. Forward references, however, can connect to any number of backward references, including zero.

In order to distinguish these simple types of forward and backward references from other reference types that will be introduced later, they are called *normal forward references* and *normal backward references*, respectively.

In the Prolog notation, they are represented by the symbols “>” and “<”:

```
np => [a], $noun(text:Noun), >(type:noun,noun:Noun).
ref => [the], $noun(text:Noun), <(type:noun,noun:Noun).
```

Syntactically, such references act like normal grammatical categories with the only exception that they cannot occur in the head of a rule.

Altogether, these special categories provide a very simple way to establish non-local dependencies in the grammar for describing anaphoric references. However, as we will discover, these simple kinds of references are not general enough for all types of references we would like to represent. For this reason, more reference types have to be defined, but first accessibility constraints need to be discussed.

3.3.5 Scopes and Accessibility

As already pointed out several times in this thesis, anaphoric references are affected by scopes. Anaphoric references are resolvable only to positions in the previous text that are accessible, i.e. that are not inside closed scopes. An example introduced earlier is

> Every man protects a house from every enemy and does not destroy ...

where one can refer to "man" and to "house" but not to "enemy" (because "every" opens a scope that is closed after "enemy"). The Codeco elements introduced so far do not allow for such restrictions. Additional elements are needed to define where scopes open and where they close.

The position where a scope opens is represented in Codeco by the special category "//" called *scope opener*, for example:

$$\mathit{quant}\Big(\text{exist: }-\Big) \quad \overset{:}{\longrightarrow} \quad /\!/ \quad [\,\text{every}\,]$$

Scope openers always succeed and all they do is opening scopes. Like forward and backward references, they can only occur in the body of rules. Furthermore, they do not have feature structures because there would be no benefit attaching feature structures to scope openers. Otherwise they behave syntactically like normal categories.

Scopes that are open have to be closed somewhere. In contrast to the opening positions of scopes, their closing positions can be far away from the scope-triggering structure. For this reason, the closing positions of scopes cannot be defined in the same way as their opening positions. Instead, the positions where scopes close are defined in Codeco by the use of scope-closing rules "$\overset{\sim}{\longrightarrow}$", for instance:

$$\mathit{vp}\Big(\text{num: }\boxed{\text{Num}}\Big) \quad \overset{\sim}{\longrightarrow} \quad v\left(\begin{array}{l}\text{num: }\boxed{\text{Num}}\\ \text{type: tr}\end{array}\right) \quad \mathit{np}\Big(\text{case: acc}\Big)$$

This rule states that a "*vp*" can be expanded to a "*v*" followed by an "*np*" (under the restrictions defined by the feature structures) and additionally defines that any scope that is opened by the direct or indirect children of "*v*" and "*np*" is closed at the end of "*np*".

Scope-closing rules are thus handled in the same way as normal rules with the only difference that they additionally define that all scopes that are opened in the body (including direct and indirect children of the body categories) are closed at the end position of the last body category. If no scopes have been opened, scope-closing rules simply behave like normal rules.

Every antecedent that is not in a scope that has been closed up to a certain position in the text is accessible from that position. The principle of accessibility will be explained in more detail in Section 3.3.10.1.

In the Prolog notation, scope openers are represented in a straightforward way by the special category "//". For the example above, one would write:

```
quant(exists:minus) => //, [every].
```

Scope closing rules are represented in the same way as normal rules with the only difference that they use the operator "~>" instead of "=>":

```
vp(num:Num) ~> v(num:Num,type:tr), np(case:acc).
```

Scope openers and scope-closing rules allow us to define where scopes open and where they close. In this way, anaphoric references can be restricted so that they can be used only if they can be resolved to an accessible antecedent. In contrast to most other approaches, scopes are defined in a way that is completely independent from the semantic representation, which gives us a clear separation of syntax and semantics.

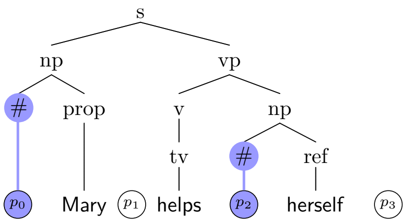
3.3.6 Position Operators

The introduced Codeco elements allow us to use definite noun phrases as anaphoric references in a way, so they can be used only if they are resolvable. However, with only the elements introduced so far at hand, it is not possible to use forward and backward references to define, for example, that a reflexive pronoun like "herself" is allowed only if it refers to the subject of the respective verb phrase. Concretely, the introduced Codeco elements do not allow for a distinction of the following two cases:

A woman helps herself.
* A woman knows a man who helps herself.

The problem is that there is no way to check whether a potential antecedent is the subject of a given anaphoric reference or not. What is needed is a way of assigning an identifier to each antecedent.

To this aim, Codeco employs the position operator "#", which can occur in the body of rules. This operator takes a variable and assigns it an identifier that represents the respective position in the text. In the sentence "Mary helps herself", for example, four different positions exist: p_0, p_1, p_2 and p_3 in "p_0 Mary p_1 helps p_2 herself p_3". A variable of a position operator is unified with the position identifier that corresponds to the location of the position operator in the syntax tree. The following picture visualizes how position operators work:



The first position operator unifies its variable with the position identifier p_0 because, in the syntax tree, the position operator occurs on the left of the first token of the input text. The second position operator unifies its variable with the position identifier p_2 because of its location between the second and the third token.

With the use of position operators, reflexive pronouns can be defined in a way, so they can be used only if a matching antecedent exists that is the subject of the given verb phrase:

$$np\left(\text{id: }\boxed{\text{Id}}\right) \;\xrightarrow{:}\; \#\boxed{\text{Id}} \quad prop\left(\text{human: }\boxed{\text{H}}\right) \;>\! \begin{pmatrix}\text{id: }\boxed{\text{Id}}\\ \text{human: }\boxed{\text{H}}\\ \text{type: prop}\end{pmatrix}$$

$$ref\left(\text{subj: }\boxed{\text{Subj}}\right) \;\xrightarrow{:}\; [\,\text{itself}\,] \quad <\! \begin{pmatrix}\text{id: }\boxed{\text{Subj}}\\ \text{human: }-\end{pmatrix}$$

Note that a position operator does not behave like a normal category in the sense that it does not take a whole feature structure but just a single variable.

The actual form of the position identifiers is not relevant as long as every position in the text is assigned exactly one unique identifier, and as long as position identifiers are different from any feature value that does not come from a position operator.

For the Prolog notation, "#" is defined as a prefix operator:

```
np(id:Id) =>
  #Id,
  prop(human:H),
  >(id:Id,human:H,type:prop).

ref(subj:Subj) =>
  [itself],
  <(id:subj,human:minus).
```

Position operators allow us to use identifiers — e.g. for identifying the subject of a verb phrase — in a very simple and declarative way. With position operators, the use of reflexive pronouns can be constrained appropriately. As we will see later, however, a further extension is needed for the appropriate definition of irreflexive pronouns.

3.3.7 Negative Backward References

A further problem that has to be solved concerns variables as they are supported, for instance, by ACE. Phrases like "a person X" can be used to introduce a variable "X". A problem arises if the same variable is introduced twice, as in the following example:

* A person X knows a person X.

One solution is to allow such sentences and to define that the second introduction of "X" overrides the first one so that subsequent occurrences of "X" can only refer to the second one. In first-order logic, for example, variables are treated this way. In CNLs, however, the overriding of variables can be confusing for the readers. ACE, for example, does not allow variables to be overridden and returns an error message if one tries to do so.

Such restrictions cannot be defined by the Codeco elements introduced so far. Another extension is needed: the special category "$\not<$" that can be used to ensure that there is no matching antecedent. This special category establishes *negative backward references*, which can be used — among other things — to ensure that no variable is introduced twice:

$$\textit{newvar} \xrightarrow{:} \underline{\textit{var}}\left(\text{text: } \boxed{\text{V}}\right) \not< \left(\begin{array}{l}\text{type: var}\\ \text{var: } \boxed{\text{V}}\end{array}\right) > \left(\begin{array}{l}\text{type: var}\\ \text{var: } \boxed{\text{V}}\end{array}\right)$$

The special category "$\not<$" succeeds only if there is no accessible forward reference that unifies with the given feature structure. In the above example, the negative backward reference ensures that a variable can be introduced only if there is no antecedent that introduces a variable with the same name.

In the Prolog notation, negative backwards references are represented by the symbol "/<". The example above would thus look as follows:

```
np =>
  $var(text:V),
  /<(type:var,var:V),
  >(type:var,var:V).
```

Negative backward references give us the possibility to define that certain structures are allowed unless a certain kind of antecedent is available. Note that the principle of accessibility — to be explained in detail in Section 3.3.10.1 — does also apply for negative backward references. Thus, only *accessible* matching antecedents are relevant.

3.3.8 Complex Backward References

The introduced Codeco elements are still not sufficient for expressing all the things we would like to express. As already mentioned, there is still a problem with irreflexive pronouns like “him”. While reflexive pronouns like “himself” can be restricted to refer only to the respective subject, the introduced Codeco elements do not allow for preventing irreflexive pronouns from referring to the subject as well:

John knows Bill and helps him.
* John helps him.

These two cases cannot be distinguished so far. It thus becomes necessary to introduce *complex backward references*, which use the special structure “$<^{+}\ldots{}^{-}\ldots$”. Complex backward references can have several feature structures: one or more positive ones (after the symbol “$^{+}$”), which define how a matching antecedent must look like, and zero or more negative ones (after “$^{-}$”), which define how the antecedent must *not* look like. The symbol “$^{-}$” can be omitted if no negative feature structures are present.

In this way, irreflexive pronouns can be correctly represented, for instance, as shown below:

$$ref\left(\text{subj: }\boxed{\text{Subj}}\right) \;\overset{:}{\longrightarrow}\; [\,\text{he}\,] \quad <^{+}\begin{pmatrix}\text{human: } + \\ \text{gender: masc}\end{pmatrix}^{-}\left(\text{id: }\boxed{\text{Subj}}\right)$$

Complex backward references refer to the closest accessible forward reference that unifies with one of the positive feature structures but is not unifiable with any of the negative ones. The complex backward reference of the example shown above ensures that “he” can be used only if an accessible antecedent exists that has masculine gender but is not the subject of the respective verb phrase.

In the Prolog notation, the symbol “<” is used for complex backward references, i.e. the same symbol as for normal backward references. The difference to normal backward references is that the feature structures are inside of terms using the symbols “+” and “-” representing the positive and the negative feature structures, respectively:

```
ref(subj:Subj) =>
    [he],
    <( +(human:plus,gender:masc),
       -(id:Subj) ).
```

Complex backward references are a powerful construct, with which anaphoric references can be restricted in a very general way. The following two examples — which are rather artificial and would probably not be very useful in practice — illustrate the general nature of complex backward references. As a first example, one could define the word “it/him” so that it can only refer to the closest antecedent that is either neuter (i.e. not human) or masculine but that is not the subject:

$$ref\left(\text{subj: }\boxed{\text{Subj}}\right) \;\overset{:}{\longrightarrow}\; [\,\text{it/him}\,] \quad <^{+}\left(\text{human: }-\right)\begin{pmatrix}\text{human: } + \\ \text{gender: masc}\end{pmatrix}^{-}\left(\text{id: }\boxed{\text{Subj}}\right)$$

As another example, one might want to define that the word "this" can be used to refer to something which is neuter and has no variable attached or which is a relation (whatever that means), but which is not the subject and is not a proper name. This complex behavior could be achieved by the following rule:

$$\mathit{ref}\left(\text{subj: }\boxed{\text{Subj}}\right) \quad \overset{:}{\longrightarrow} \quad [\,\text{this}\,] \quad <^{+}\begin{pmatrix}\text{hasvar: }-\\ \text{human: }-\end{pmatrix}\left(\text{type: relation}\right)^{-}\left(\text{id: }\boxed{\text{Subj}}\right)\left(\text{type: prop}\right)$$

In the Prolog notation, these examples would look as follows:

```
ref(subj:Subj) =>
    ['it/him'],
    <( +(human:minus),
       +(human:plus,gender:masc),
       -(id:Subj) ).

ref(subj:Subj) =>
    [this],
    <( +(hasvar:minus,human:minus),
       +(type:relation),
       -(id:Subj),
       -(type:prop) ).
```

Complex backward references that have exactly one positive feature structure and no negative ones are equivalent to normal backward references. For this reason, algorithms defined for complex backward references implicitly also cover normal ones. Normal backward references are subsumed by complex backward references and can thus be considered syntactic sugar of the language.

3.3.9 Strong Forward References

Finally, one further extension is needed in order to handle antecedents that are not affected by the accessibility constraints. For example, proper names are usually considered accessible even if under negation:

> Mary does not love Bill. Mary hates him.

In such situations, the special category "≫" can be used, which introduces a *strong forward reference*:

$$\mathit{np}\left(\text{id: }\boxed{\text{Id}}\right) \quad \overset{:}{\longrightarrow} \quad \mathit{prop}\left(\text{human: }\boxed{\text{H}}\right) \quad \gg\begin{pmatrix}\text{id: }\boxed{\text{Id}}\\ \text{human: }\boxed{\text{H}}\\ \text{type: prop}\end{pmatrix}$$

Strong forward references are always accessible even if they are within closed scopes. Apart from that, they behave exactly the same way as normal forward references.

In the Prolog notation, strong forward references are represented by the symbol “>>”. The introduced example would thus look as follows:

```
np(id:Id) =>
  prop(human:H),
  >>(id:Id,human:H,type:prop).
```

All elements of the Codeco language have now been introduced.

3.3.10 Principles of Reference Resolution

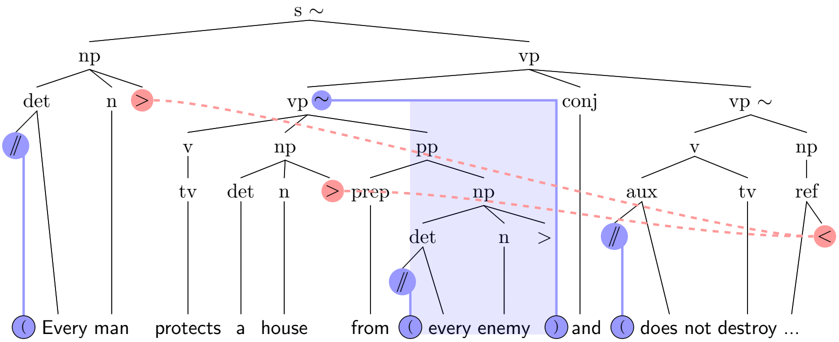
The resolution of references in Codeco — i.e. the matching of backward references to the appropriate forward references — requires some more explanation.

All three types of backward references (normal, negative and complex ones) are resolved according to the three principles of accessibility, proximity and left-dependence. The principle of accessibility has already been mentioned in the preceding sections. It implies that an antecedent within a scope cannot be referred to from outside the scope. The principle of proximity defines that textually closer antecedents have precedence over those that are more distant. The principle of left-dependence, finally, defines which variable bindings have to be considered when resolving a reference. These three principles will now be explained in detail.

3.3.10.1 Accessibility

The principle of accessibility states that one can refer to forward references only if they are accessible from the position of the backward reference. A forward reference is accessible only if it is not within a scope that has already been closed before the position of the backward reference, or if it is a strong forward reference.

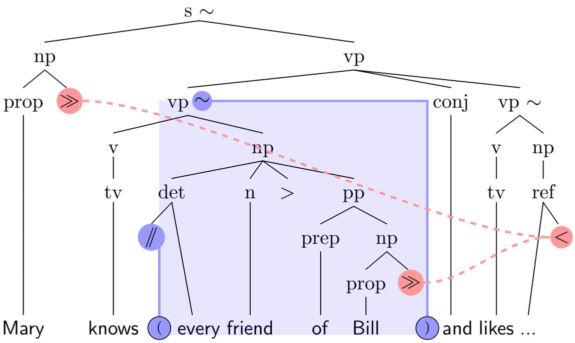
This accessibility constraint can be visualized in the syntax tree. The syntax tree for the partial sentence shown in Section 3.3.5 could look as follows:



All nodes that represent the head of a scope-closing grammar rule are marked with "∼". The positions in the text where scopes open and close are marked by parentheses. In this example, three scopes have been opened but only the second one (the one in front of "every enemy") has been closed (after "enemy"). The blue area marks the part of the syntax tree that is covered by this closed scope.

As a consequence of the accessibility constraint, the forward references for "man" and "house" are accessible from the position of the backward reference at the very end of the shown partial sentence. In contrast, the forward reference for "enemy" is not accessible because it is inside a closed scope. The possible references are illustrated by dashed red connection lines. Thus, the shown partial sentence could be continued by the anaphoric references "the man" or "the house" (or equivalently "himself" or "it", respectively) but not by the reference "the enemy".

Strong forward references are not affected by the accessibility constraint: they are always accessible. The following example shows a partial sentence with two occurrences of strong forward references:



As shown by the dashed red connection lines, the second of the two strong forward references is also accessible even though it is within a closed scope. The normal forward reference for "friend", however, is not accessible. Thus, the given partial sentence could be continued by a reference like "him" pointing to "Bill" or by a reference like "herself" pointing to "Mary". The reference "the friend", however, would not be allowed.

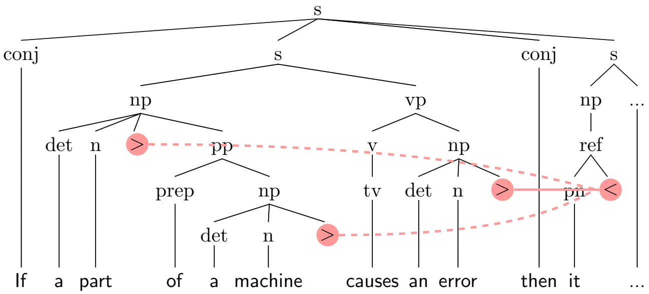
The difference between normal, negative and complex backward references has no effect on the accessibility constraint.

In summary, Codeco defines accessibility constraints by using the scope-opening category "//" and scope-closing grammar rules. Additionally, strong forward references can be used to bypass the accessibility constraints.

3.3.10.2 Proximity

Proximity is the second principle for the resolution of backward references. If a backward reference could potentially point to more than one forward reference then, as a last resort, the principle of proximity defines that the textually closest forward reference is taken. This ensures that every backward reference resolves deterministically to exactly one forward reference.

In the following example, the reference "it" could in principle refer to three antecedents:



The pronoun "it" could refer to "part", "machine", or to "error". According to the principle of proximity, the textually closest antecedent is taken. In this example, the backward reference is resolved to the forward reference of "error" that is only two tokens away whereas the other two forward references have a distance of five and eight tokens, respectively.

In other words, a backward reference can be resolved only to a forward reference if no other matching forward reference exists that is closer to the backward reference.

3.3.10.3 Left-dependence

The principle of left-dependence, finally, means that everything to the left of a backward reference is considered for its resolution but everything to its right is not. The crucial point is that variable bindings entailed by a part of the syntax tree that is to the left of the reference are considered for the resolution of the reference. In contrast, variable bindings that would be entailed by a part of the syntax tree that is on the right are not considered.

Variables as they occur in the grammar can be instantiated to an atom or unified to another variable during the construction of the syntax tree for a given input text. These bindings occur when a category of the body of a rule is matched to the head of another rule. The node in the syntax tree that corresponds to the body category of the one rule and at the same time to the head of the other rule can be considered the location where the binding takes place. In this way, every variable binding can be located in the syntax tree. The principle of left-dependence defines now that all variable bindings that are located in the syntax tree to the left of a reference have to be considered when resolving the reference but the bindings to the right of the reference must not be considered.

A concrete example will illustrate why the principle of left-dependence is important. The following example shows two versions of the same grammar rule:

$$\mathit{ref} \;\overset{:}{\longrightarrow}\; [\,\text{the}\,] \quad <\!\begin{pmatrix}\text{type: noun}\\ \text{noun: }\boxed{\text{N}}\end{pmatrix} \quad \underline{\mathit{noun}}\begin{pmatrix}\text{text: }\boxed{\text{N}}\end{pmatrix}$$

$$\mathit{ref} \;\overset{:}{\longrightarrow}\; [\,\text{the}\,] \quad \underline{\mathit{noun}}\begin{pmatrix}\text{text: }\boxed{\text{N}}\end{pmatrix} \quad <\!\begin{pmatrix}\text{type: noun}\\ \text{noun: }\boxed{\text{N}}\end{pmatrix}$$

The only difference is that the backward reference and the pre-terminal category "*noun*" are switched.

In the first version, the backward reference is resolved without considering how the variable "N" would be bound by the category "*noun*". If, for example, the two forward references

$$>\!\begin{pmatrix}\text{type: noun}\\ \text{noun: country}\end{pmatrix} \quad >\!\begin{pmatrix}\text{type: noun}\\ \text{noun: area}\end{pmatrix}$$

are accessible from the position of the backward reference then the backward reference would match both forward references. According to the principle of proximity, it would then be resolved to the closer of the two, i.e. to the one that represents the noun "area". There is no possibility to refer to "country" in this case.

In the second version of the grammar rule, the backward reference follows the category "*noun*" and the respective binding for the variable "N" is considered for the resolution of the backward reference. Thus, for resolving the backward reference, it is considered that the variable "N" is bound to "area", "person", "country", or whatever occurs in the input text (as long as it is supported by the lexicon). Given the accessible forward references shown above, the backward reference can be resolved to the first forward reference if the variable is bound to "country", or it can be resolved to second one if the variable is bound to "area". Apparently, this version of the grammar rule is more sensible than the first one.

As a rule of thumb, backward references should generally follow the textual representation of the anaphoric reference and not precede it.

3.3.11 Restriction on Backward References

In order to provide a proper and efficient lookahead algorithm that can also handle backward references, their usage must be restricted.

Backward references are restricted in the way that they must immediately follow a terminal or pre-terminal category in the body of grammar rules. Thus, they are not allowed at the initial position of the body of a rule and they are not allowed to follow a non-terminal category.

This restriction ensures that backward references are close to the terminal categories that represent them. Section 3.5.4 will show how lookahead features can be implemented if this restriction is followed.

However, the basic algorithms to be presented that transform Codeco into Prolog DCGs or use it within a chart parser also work for grammars that do not follow this restriction. Only the lookahead feature would not work as expected.

3.4 Codeco as a Prolog DCG

Codeco is a declarative notation that is designed to be processable by different kinds of parsers. This section explains how grammars in Codeco notation can be transformed into definite clause grammars (DCGs) in Prolog. The next section will show how Codeco can be interpreted by a chart parser.

Prolog DCGs have already been introduced in Section 3.2.3. They are a simple and convenient way to define grammars and they can be executed with very little computational overhead. Grammar rules in the Prolog DCG notation use the operator "`-->`" and are internally transformed by Prolog in a very simple way into plain Prolog clauses using the operator "`:-`".

SWI Prolog[3], which is a popular open source implementation of Prolog, is used here. However, the presented code should be executable with no or minimal changes in any other Prolog implementation. Below, it is shown how the different elements of Codeco map to the Prolog DCG representation.

Feature Structures

First of all, the feature structures of Codeco need to be represented appropriately in the DCG. This can be done by using Prolog lists. Since such lists unify only if they have the same length and if their elements are pairwise unifiable, it is required to know how many and what kind of feature names occur in the Codeco grammar. For this reason, the different feature names have to be extracted and put into a distinct order before the actual transformation can start.

Every feature structure in the Codeco grammar is then transformed into a Prolog list of the length of the number of distinct feature names in the grammar. The starting point is a list of unbound variables. For every name/value pair of the feature structure, the list element at the position that corresponds to the feature name is unified with the feature value.

For example, let us assume that a grammar uses exactly seven feature names, which are sorted as follows:

```
id subj type num case gender text
```

A feature structure like

$$\left(\begin{array}{l} \text{id: } \boxed{\text{Id}} \\ \text{subj: } \boxed{\text{Id}} \\ \text{num: sg} \end{array}\right)$$

[3] `http://www.swi-prolog.org/`

is in this case translated into

```
[Id,Id,_,sg,_,_,_]
```

in the Prolog DCG representation. Note that each occurrence of the underscore symbol "_" denotes a different variable in Prolog. Doing this transformation consistently with all feature structures of the whole grammar ensures that two Prolog lists representing two feature structures unify if and only if the two feature structures would be unifiable.

This transformation is just a simplified version of what more elaborate feature systems like ProFIT [45] or GULP [36] do.

Categories and Position Operators

Grammatical categories of Codeco — including the special categories for scope openers ("`//`"), forward references ("`>`" and "`>>`"), and negative backward references ("`/<`") — are transformed into Prolog predicates having two arguments. Normal and complex backward references ("`<`") will be discussed below.

The first argument contains the feature structure, or simply a list of unbound variables if the category has no feature structure. The second argument contains a complex term of the form "`R1/R2`" where `R1` is the incoming list of references and `R2` is the outgoing list. These reference lists contain all forward references and scope openers that are accessible. They are used to correctly resolve backward references. For example, a Codeco category of the form

$$np\begin{pmatrix}\text{id: }\boxed{\text{Id}}\\ \text{subj: }\boxed{\text{Id}}\\ \text{num: sg}\end{pmatrix}$$

is translated into the following Prolog term:

```
np([Id,Id,_,sg,_,_,_],R1/R2)
```

As we will see shortly, reference lists are connected with each other and threaded through the grammar rules.

Position operators like $\#\boxed{\text{P}}$ simply lead to a unary Prolog predicate with the name "`#`" containing the respective variable:

```
#(P)
```

Normal and Complex Backward References

Normal and complex backward references are treated the same way as the other categories with the only difference that the first argument is not a feature structure itself but a list of feature structures, each of which is prefixed by either “+” or “-”. Normal backward references only get one “+”-term. E.g.

$$<\begin{pmatrix}\text{type: noun}\\ \text{text: }\boxed{\text{Text}}\end{pmatrix}$$

is translated into:

```
<([+[_,_,noun,_,_,_,Text]],R1/R2)
```

Complex backward references get a “+”-term for every positive feature structure and a “-”-term for every negative one. E.g.

$$<^{+}\Big(\text{gender: masc}\Big)^{-}\Big(\text{id: }\boxed{\text{Subj}}\Big)$$

is translated into:

```
<([+[_,_,_,_,_,masc,_],-[Subj,_,_,_,_,_,_]],R1/R2)
```

Grammar Rules

Normal grammar rules are converted by transforming all involved categories (the one in the head and the zero or more categories of the body) and by then unifying the reference lists of the categories in the right way. This is done by unifying the input list of the head with the input list of the first body category, whose output list is in turn unified with the input list of the second body category, and so on until the output list of the last body category is unified with the output list of the head.

For example, the grammar rule

$$vp\Big(\text{num: }\boxed{\text{Num}}\Big) \quad \overset{:}{\longrightarrow} \quad adv \quad v\begin{pmatrix}\text{num: }\boxed{\text{Num}}\\ \text{type: tr}\end{pmatrix} \quad np\Big(\text{case: acc}\Big)$$

would be transformed into:

```
vp([_,_,_,Num,_,_,_],RIn/ROut) -->
  adv([_,_,_,_,_],RIn/R1),
  v([_,_,tr,Num,_,_,_],R1/R2),
  np([_,_,_,_,acc,_,_],R2/ROut).
```

The threading of the reference lists (which has already been sketched in Section 3.2.3 and which is very similar to how Prolog interprets the DCG notation as plain Prolog programs) ensures that all accessible references are known at every point of the parsing process. Furthermore, references can be added or removed at any point by returning a different output list than the one that is received as input list.

Scope-closing rules are basically transformed in the same way, but the resulting DCG rules end with an additional special predicate of the form “`~(RIn,RTemp,` ROut)” where RIn is the input reference list of the head, RTemp is the output list of the last body category, and ROut is the output list of the head. This special predicate is used to remove the references that are not accessible from the outside of the rule.

If the above example had been a scope-closing rule (i.e. using “$\overset{\sim}{\longrightarrow}$” instead of “$\overset{\cdot}{\longrightarrow}$”) then the resulting Prolog DCG rule would have looked as follows:

```
vp([_,_,_,Num,_,_,_],RIn/ROut) -->
  adv([_,_,_,_,_,_,_],RIn/R1),
  v([_,_,tr,Num,_,_,_],R1/R2),
  np([_,_,_,_,acc,_,_],R2/R3),
  ~(RIn/R3/ROut).
```

Special Predicates

The last thing to be done is to define the meaning of the special predicates “>”, “>>”, “<”, “/<”, “//”, “~” and “#”. Note that none of these special predicates read any token from the token list to be parsed.

“>” and “>>” simply add a new reference to the reference list. References are represented by terms consisting of the type of the reference and the respective feature structure. This behavior can be implemented in Prolog as follows:

```
>(F, T/[>(F)|T]) --> [].
>>(F, T/[>>(F)|T]) --> [].
```

Note that the new references are added to the beginning of the lists. As a consequence, the lists are reversed in the sense that references that occur earlier in the text appear later in the reference list.

“<”-references succeed if there is a forward reference in the list of references that unifies with a positive feature structure and is not unifiable with any negative one. The closest possible reference should be taken if more than one such forward reference exists. This can be implemented in the form of the two Prolog DCG rules

```
<(L, [R|T]/[R|T]) --> {
    R =.. [_,Q],
    \+ member(-Q, L),
    \+ \+ member(+Q, L),
    !,
    member(+Q, L)
  }.

<(L, [R|T]/[R|T]) --> <(L,T/T).
```

where “member/2” (i.e. the predicate that has the name “member” and takes two arguments) is a built-in Prolog predicate that is used to access the elements of lists. The list is searched for a matching forward reference starting from the beginning of the list which corresponds to the textually closest reference (because the lists are reversed). It is first checked that no negative feature structure matches by using the Prolog negation operator "\+". Then, it is checked whether at least one of the positive feature structures matches without already performing the actual unification. This is achieved by using double negation "\+ \+". In the case this succeeds, the cut "!" is used to prevent from looking for further forward references. The actual unification is performed only after the cut in order to account for the fact that more than one of the positive feature structures may match.

"/<"-references should succeed only if there is no matching forward reference in the reference list. This can be implemented with the following DCG rule:

```
/<(F, T/T) --> {
    \+ ( member(R,T), R =.. [_,F] )
  }, !.
```

The implementation of the "//"-predicate is very simple. It just adds the symbol "//" to the references list:

```
//(_, T/[//|T]) --> [].
```

The "~"-predicate needs to remove all references that are not accessible from the outside of the scope-closing rule. If a scope opener has been added by one of the body categories then everything that has been added later — except strong forward references — has to be removed again. However, the elements of the reference list that were already present in the input list of the head should not be changed. The following code implements this:

```
~(RIn/RTemp/ROut) --> {
    append([X,[//|N],RIn],RTemp),
    \+ member(//,N),
    findall(>>(R),member(>>(R),X),Y),
    append([Y,N,RIn],ROut)
  }, !.
```

The predicates "append/2" and "findall/3" are again built-in. "append/2" takes a list of lists and returns another list that is obtained by consecutively appending the lists contained in the input list. "findall/3" looks for all possible solutions for a given goal and returns them in a list. These predicates are easy to implement by hand if the used Prolog interpreter does not support them.

If no scope has been opened then the temporary reference list coming from the last body category is returned unchanged:

```
~(_/ROut/ROut) --> [].
```

Finally, the "#"-predicate needs to bind its variable to a certain position identifier. The easiest way to do this in Prolog is to take the number of tokens that still have to be processed (taking the number of tokens that already have been processed is more complicated). This cannot be done in the form of a DCG rule but a plain Prolog clause has to be used:

```
#(#(P),L,L) :- length(L,P).
```

The built-in predicate "`length/2`" is used to determine the length of the list of unprocessed tokens. The variable is not bound to the bare position number "`P`" but to the complex term "`#(P)`" in order to prevent from unwanted unifications with features values that do not come from a position operator.

Altogether, these transformations allow us to execute Codeco grammars as Prolog DCGs. Performance measurements will be presented in Section 3.8.4.

3.5 Codeco in a Chart Parser

As already mentioned earlier, DCGs have some shortcomings. Compared to the parsing approach of Prolog DCGs, chart parsers are much better suited for implementations in procedural or object-oriented programming languages. This is because chart parsers do not depend on backtracking. Furthermore, lookahead features can be implemented in a much more efficient way with chart parsers.

The basic idea of chart parsers is to store temporary parse results in a data structure that is called *chart* and that contains small portions of parse results in the form of *edges* (sometimes simply called *items*).

The algorithm for Codeco to be presented here is based on the chart parsing algorithm invented by Jay Earley that is therefore known as the *Earley algorithm* [44]. Grune and Jacobs [64] discuss this algorithm in more detail, and Covington [37] shows how it can be implemented. The specialty of the Earley algorithm is that it combines top-down and bottom-up processing.

The parsing time of the standard Earley algorithm is in the worst case cubic with respect to the number of tokens to be parsed and only quadratic for the case of unambiguous grammars. However, this holds only if the categories have no arguments (e.g. feature structures). Otherwise, parsing is NP-complete in the general case. In practice, however, Earley parsers perform very well for most grammars and never come even close to these worst-case measures.

The processing of Codeco grammars within an Earley parser as described in this section has been implemented in Java and is available as open source software. Evaluation results of this implementation will be shown in Section 3.7. This Java implementation is the basis for the predictive editor that is used in the ACE Editor (see Section 4.2) and in AceWiki (see Section 4.4).

Below, a meta language is introduced. This meta language is then used to describe the elements of a chart parser and to explain the different steps of the parsing algorithm. Finally, it is shown how lookahead information for partial texts can be extracted from the chart.

3.5.1 Meta Language

In order to be able to talk about the elements of chart parsers (i.e. rules and edges) in a general way, it is very useful to define a meta language. In this section, the following meta symbols will be used:

F stands for a feature structure, i.e. a set of name/value pairs.

A stands for any (terminal, pre-terminal or non-terminal) category, i.e. a category name followed by an optional feature structure.

α stands for an arbitrary sequence of zero or more categories.

r stands for a forward reference symbol, i.e. either "$>$" or "$\gg$".

ρ stands for an arbitrary sequence of zero or more forward references "rF" and scope openers "$/\!/$".

s stands for either a colon ":" or a tilde "$\sim$" so that "$\xrightarrow{s}$" can stand for "$\xrightarrow{:}$" or for "$\xrightarrow{\sim}$".

i stands for a position identifier that represents a certain position in the input text.

All meta symbols can have a numerical index to distinguish different instances of the same symbol, e.g. α_1 and α_2. Other meta symbols will be introduced as needed.

3.5.2 Chart Parser Elements

Before the actual parsing steps can be described, the fundamental elements of Earley parsers have to be introduced: the edges and the chart. Furthermore, a graphical notation is introduced that will be used to describe the parsing steps in an intuitive way.

Edges

The edges of a chart parser are derived from the grammar rules and like the grammar rules they consist of a head and a body. Every edge has the following general form (using the meta language introduced above):

$$\langle i_1, i_2 \rangle \quad A \;\rightarrow\; \alpha_1 \bullet \alpha_2$$

A is the head of the edge. The body of the edge is split into a sequence α_1 of categories that have already been recognized and another sequence α_2 of categories that still have to be processed. The dot "$\bullet$" indicates where the first sequence ends and the second one starts. Furthermore, every edge has a start position i_1 and an end position i_2.

The following example should clarify how edges work. If the grammar contains a rule

$$s \xrightarrow{:} np \quad vp$$

then this rule could lead to the edge

$$\langle 0, 0 \rangle \quad s \rightarrow \ \bullet \ np \quad vp$$

where start and end position are both 0 and where the body starts with the dot symbol "$\bullet$". This edge has the meaning that we are looking at the position 0 for the category "*s*" but nothing has been found so far. If the text between position 0 and 2, for example, is recognized as an "*np*" then a new edge

$$\langle 0, 2 \rangle \quad s \rightarrow \quad np \ \bullet \ vp$$

could be added to the chart. This edge means that we started looking for an "*s*" at position 0 and until position 2 we already found an "*np*", but a "*vp*" is still needed to make it a complete "*s*". Assuming that a "*vp*" is later recognized between the positions 2 and 5, a new edge

$$\langle 0, 5 \rangle \quad s \rightarrow \quad np \quad vp \ \bullet$$

could be added. This edge represents the fact that between the positions 0 and 5 a complete "*s*" has been recognized consisting of an "*np*" and a "*vp*".

Edges like the last one where all categories of the body are recognized are called *passive.* All other edges are called *active* and their first category of the sequence of not yet recognized categories is called their *active category.*

Edges with Antecedents

Processing Codeco grammars requires an extended notation for edges. Whenever a backward reference occurs, we need to be able to find out which antecedents are accessible from that position. For this reason, edges coming from a Codeco grammar have to carry information about the accessible antecedents.

First of all, every edge must carry the information whether it originated from a normal rule or a scope-closing one. Edges originating from normal rules are called *normal edges* and edges coming from scope-closing rules are called *scope-closing edges.* Like the rules they originate from, normal edges are represented by an arrow with a colon "$\xrightarrow{:}$" and scope-closing edges use an arrow with a tilde "$\xrightarrow{\sim}$".

Furthermore, every Codeco edge has two sequences which are called *external antecedent list* and *internal antecedent list.* Both lists are displayed above the arrow: the external one on the left of the colon or tilde, the internal one on the right thereof. Both antecedent lists are sequences of forward references and scope openers. Hence, Codeco edges have the following general structure:

$$\langle i_1, i_2 \rangle \quad A \xrightarrow{\rho_1 \ s \ \rho_2} \alpha_1 \bullet \alpha_2$$

ρ_1 is the external antecedent list. It represents the antecedents that come from outside the edge, i.e. from earlier positions than the starting position i_1.

ρ_2 is the internal antecedent list. It contains the antecedents that come from inside the edge, i.e. from the categories of α_1 and their children. Internal antecedents are textually somewhere between the start and the end position of the respective edge.

Scope openers in the antecedent lists show where scopes have been opened that are not yet closed up to the given position.

As a concrete example, the antecedent lists of an edge could look as follows:

$$\dots \xrightarrow{/\!/ \;\; \gg\begin{pmatrix}\text{type: prop}\\ \text{text: 'Sue'}\end{pmatrix} \;:\; >\begin{pmatrix}\text{type: var}\\ \text{text: 'X'}\end{pmatrix} \;\; /\!/ \;\; >\begin{pmatrix}\text{type: noun}\\ \text{text: 'man'}\end{pmatrix}} \dots$$

In this case, the external antecedent list consists of a scope opener and a strong forward reference. The internal antecedent list consists of a normal forward reference followed by a scope opener followed by another normal forward reference. The colon ":" separates these two lists. Concretely, a backward reference could potentially attach to three different antecedents in this situation: the proper name "Sue", the variable "X", and the noun "man".

Chart

A chart in the context of chart parsers is a data structure used to store the partial parse results in the form of edges. Before the parsing process for a certain text starts, the chart is empty.

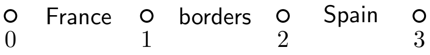
Edges are only added to the chart if they are not yet contained. Thus, it is not possible to have more than one copy of the same edge in the chart. Furthermore, edges are not changed or removed from the chart once they are added (unless the input text changes).

Traditionally, chart parsers perform a *subsumption check* for each new edge to be added to the chart [37]. A potential new edge is added to the chart if and only if no equivalent or more general edge already exists.

For reasons that will become clear later, the algorithm to be presented requires an *equivalence check* and not a subsumption check. New edges are added to the chart except for the case that an edge is already contained that is fully equivalent to the new one.

Graphical Notation

In order to be able to describe the chart parsing steps for the Codeco notation in an intuitive way, a simple graphical notation is used that is inspired by Gazdar and Mellish [58]. The different positions of the input text are represented by small circles that are arranged as a horizontal sequence. For example, the simple sentence "France borders Spain" would lead to four small circles standing for the four positions in the input text:



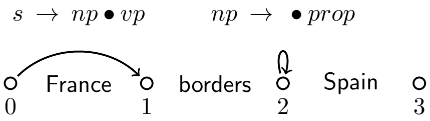
Each position has an identifier: 0, 1, 2 and 3 in the given example.

Edges are represented by arrows that point from their start position to their end position and that have a label with the remaining edge information. For the example shown above, two possible edges could be

$$\langle 0,1\rangle \quad s \;\rightarrow\; np \bullet vp$$

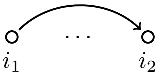
$$\langle 2,2\rangle \quad np \;\rightarrow\; \bullet\; prop$$

that would be represented in the graphical notation as follows:



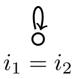
A general edge is represented by

$$A \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \bullet \alpha_2$$



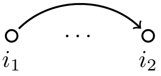
where the three dots "..." mean that i_2 is either the same position as i_1 or directly follows i_1 or indirectly follows i_1. Thus, the case $i_1 = i_2$ that would be represented as

$$A \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \bullet \alpha_2$$

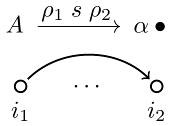


is included by the representation above. In this way, active edges can be generally represented by

$$A_1 \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \bullet A_2\alpha_2$$



where A_2 is the active category of the edge. Passive edges, in contrast, have the general form



where the dot is at the last position of the body.

This graphical notation will later be used to describe the parsing steps in an explicit but intuitive way.

3.5.3 Chart Parsing Steps

In a traditional Earley parser, there are four parsing steps: initialization, scanning, prediction and completion. In the case of Codeco, an additional step — to be called *resolution* — is needed to resolve the references, position operators, and scope openers.

Below, the general algorithm is explained. After that, a graphical notation to describe the parsing steps is introduced that uses the notation for edges introduced above. This notation is then used to define each of the five parsing steps, i.e. initialization, scanning, prediction, completion and resolution. Finally, some brief complexity considerations are shown.

3.5.3.1 General Algorithm

The general algorithm starts with the initialization. This step is performed at the beginning to initialize the empty chart. Then, prediction, completion and resolution are performed several times which together will be called the *PCR* step. This PCR step corresponds to the "Completer/Predictor Loop" as described by Grune and Jacobs [64]. A text is then parsed by consecutively scanning the tokens of the text. After each scanning of a token, again the PCR step is performed. The following piece of pseudocode shows this general algorithm:

```
parse(tokens) {
    new chart
    initialize(chart)
    pcr(chart)
    foreach t in tokens {
        scan(chart,t)
        pcr(chart)
    }
}
```

The PCR step consists of executing the prediction, completion and resolution steps until none of the three is able to generate an edge that is not yet in the chart. This can be implemented as follows:

```
pcr(chart) {
    loop {
        c := chart.size()
        predict(chart)
        complete(chart)
        resolve(chart)
        if c=chart.size() then return
    }
}
```

Prediction, completion and resolution are performed one after the other and starting over again until none of the three can contribute a new edge. The actual order of these three steps can be changed without breaking the algorithm. It can have an effect (positive or negative) on the performance though.

In terms of performance, there is potential for optimization anyway. First of all, the algorithm above checks the chart for new edges only after the resolution step. An optimized algorithm checks after each step whether the last three steps contributed a new edge or not. Furthermore, a progress table can be introduced that allows the different parsing steps to remember which edges of the chart they already checked. In this way, edges can be prevented from being checked by the same parsing step more than once.

Such an optimized algorithm can look as follows (without going into the details of the progress table):

```
pcr(chart) {
    step := 0
    i := 0
    new progressTable
    loop {
        c := chart.size()
        if step=0 then predict(chart,progressTable)
        if step=1 then complete(chart,progressTable)
        if step=2 then resolve(chart,progressTable)
        if c=chart.size() then i := i+1 else i := 0
        if i>2 then return
        step := (step+1) modulo 3
    }
}
```

The variable `i` counts the number of consecutive idle steps, i.e. steps that did not increase the number of edges in the chart. The loop can be exited as soon as this value reaches 3. In this situation, no further edge can be added because each of the three sub-steps has been performed on exactly the same chart without being able to add a new edge.

3.5.3.2 Graphical Notation for Parsing Steps

Building upon the graphical notation for edges introduced above, the parsing steps will be described by the use of a graphical notation that has a large arrow in the middle of the picture and that corresponds to the following scheme:

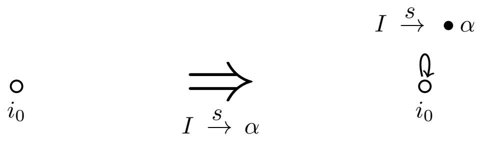
$$\text{(edges in the chart)} \qquad \underset{\text{(rule in the grammar)}}{\Longrightarrow} \qquad \text{(edge to be added)}$$

On the left hand side of the arrow, edges are shown that need to be in the chart in order to execute the described parsing step. If a grammar rule is shown below the arrow then this rule must be present in the grammar for executing the parsing step. On the right hand side of the arrow the new edge is shown that has to be added to the chart when the described parsing step is executed, unless the resulting edge is already there.

If a certain meta symbol occurs more than once on the left hand side of the picture and in the rule representation below the arrow then this means that the respective parts have to be unifiable but not necessarily identical. When generating the new edge, these unifications have to be considered but the existing edges in the chart and the grammar rules remain unchanged.

3.5.3.3 Initialization

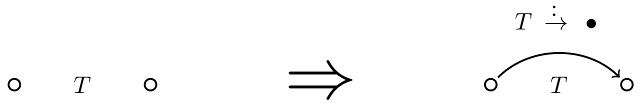
At the very beginning, the chart has to be initialized. For each rule that has the start category (according to which the text should be parsed) on its left hand side, an edge is introduced into the chart at the start position:


i_0
$\Longrightarrow$
$I \xrightarrow{s} \alpha$
$I \xrightarrow{s} \bullet\, \alpha$
i_0

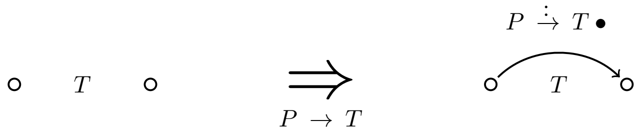
i_0 stands for the start position of the chart, i.e. the position that represents the beginning of the input text, and I stands for the start category of the grammar. The only difference to the standard Earley algorithm is that the information about normal and scope-opening rules is taken over from the grammar to the chart, represented by s.

3.5.3.4 Scanning

During the scanning step, a token is read from the input text. This token can be interpreted as a terminal symbol T for which a passive edge is introduced that has T on its left hand side:



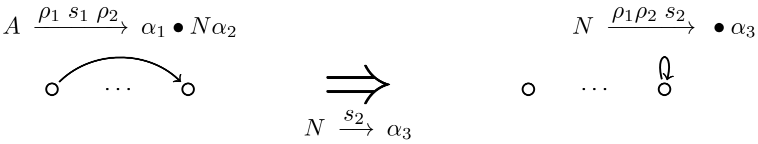
Furthermore, the token can also be a possible extension for one or more pre-terminal symbols P:



The rule $P \rightarrow T$ can come from the static grammar or from a dynamically managed lexicon.

3.5.3.5 Prediction

The prediction step looks out for grammar rules that could be applied at the given position. For every active category in the chart that matches the head of a rule in the grammar, a new edge is introduced (unless it is already in the chart):


$A \xrightarrow{\rho_1\ s_1\ \rho_2} \alpha_1 \bullet N\alpha_2$
$\cdots$
$N \xrightarrow{s_2} \alpha_3$
$N \xrightarrow{\rho_1\rho_2\ s_2} \bullet \alpha_3$
$\cdots$

N denotes a non-terminal category. The external antecedent list of the new edge is a concatenation of the external and the internal antecedent list of the existing edge. The internal antecedent list of the new edge is empty because it has no recognized categories in its body and thus cannot have internal antecedents.

Remember that the three dots "…" mean that an arbitrary number of nodes can exist between the two shown nodes and that the two nodes can also be one and the same. Thus, the new edge that is produced by such a prediction step can itself be used to produce more new edges by again applying the prediction step at the same position.

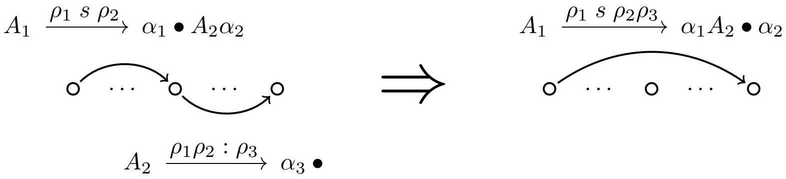
3.5.3.6 Completion

The completion step takes the active categories of the active edges in the chart and looks for passive edges with a corresponding head. If such two edges can be found then a new edge can be created out of them.

Informally speaking, if there is an edge that is not yet finished (i.e. active) and needs a certain category to proceed (i.e. its active category) and this category matches the head of a finished (i.e. passive) edge that starts at the position where the first edge ends then the category is recognized in the input text and the first edge can make one step forward and span to the end of the recognized category.

In the standard Earley algorithm, there is only one kind of completion step. The extensions for references, however, make it necessary to differentiate between the cases where the passive edge is a normal edge and those where it is a scope-closing edge.

In the case of a normal edge, the completion step looks as follows:

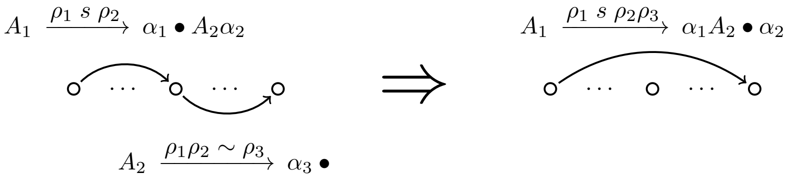


$$A_1 \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \bullet A_2\alpha_2 \qquad \Longrightarrow \qquad A_1 \xrightarrow{\rho_1\ s\ \rho_2\rho_3} \alpha_1 A_2 \bullet \alpha_2$$

$$A_2 \xrightarrow{\rho_1\rho_2\ :\ \rho_3} \alpha_3 \bullet$$

In contrast to the standard Earley algorithm, not only the active category of the active edge has to match the head of the passive edge, but also the references of the active edge have to be present in the same order in the passive edge.

If no scope has been opened then scope-closing edges are completed in exactly the same way as normal edges:

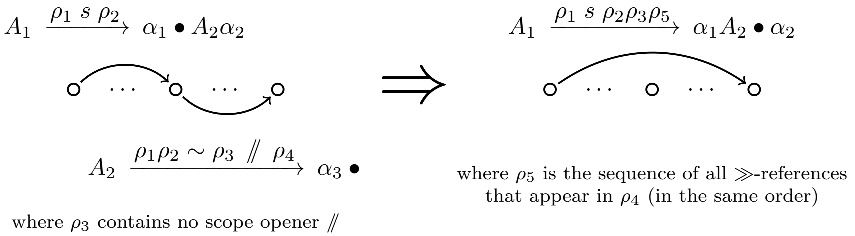


$$A_1 \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \bullet A_2\alpha_2 \qquad \Longrightarrow \qquad A_1 \xrightarrow{\rho_1\ s\ \rho_2\rho_3} \alpha_1 A_2 \bullet \alpha_2$$

$$A_2 \xrightarrow{\rho_1\rho_2\ \sim\ \rho_3} \alpha_3 \bullet$$

where ρ_3 contains no scope opener $/\!\!/$

If one or more scopes have been opened then all scope openers and all normal forward references that come after the first scope opener are removed from the internal antecedent list for the new edge to be added:



$$A_1 \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \bullet A_2\alpha_2 \qquad A_2 \xrightarrow{\rho_1\rho_2 \sim \rho_3\ /\!\!/\ \rho_4} \alpha_3 \bullet \quad \Longrightarrow \quad A_1 \xrightarrow{\rho_1\ s\ \rho_2\rho_3\rho_5} \alpha_1 A_2 \bullet \alpha_2$$

where ρ_3 contains no scope opener $/\!\!/$

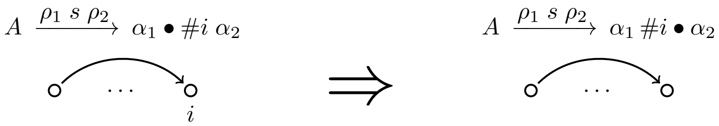
where ρ_5 is the sequence of all $\gg$-references that appear in ρ_4 (in the same order)

It is important to see that these three completion rules are non-overlapping in the sense that for two given edges at most one new edge can be generated.

3.5.3.7 Resolution

In order to handle position operators, scope openers, and references, an additional parsing step is needed which I call *resolution.* It is now shown how and under which circumstances these special elements of Codeco can be resolved. Generally, only elements occurring in the position of an active category are resolvable.

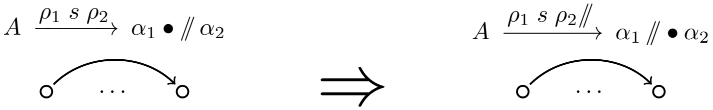
A position operator is resolved by unifying its variable with an identifier that represents the given position in the input text:



$$A \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \bullet \#i\ \alpha_2 \quad \Longrightarrow \quad A \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1\ \#i \bullet \alpha_2$$

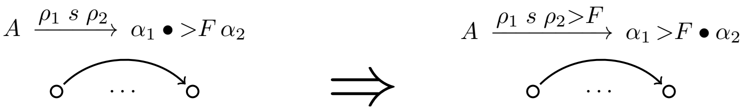
Remember that the two occurrences of i on the left hand side mean that the two parts must be unifiable, and that the i on the right hand side represents the unified version of the two.

Scope openers are resolved by adding the scope opener symbol to the end of the internal antecedent list:



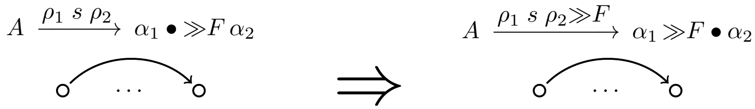
$$A \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \bullet /\!\!/\ \alpha_2 \quad \Longrightarrow \quad A \xrightarrow{\rho_1\ s\ \rho_2 /\!\!/} \alpha_1\ /\!\!/ \bullet \alpha_2$$

Forward references are resolved in a similar way as scope openers. Together with their feature structure, they are added to the end of the internal antecedent list:



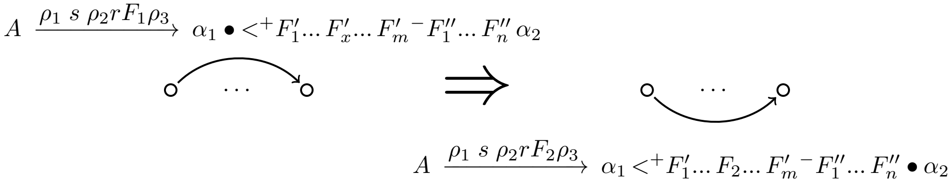
$$A \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \bullet {>}F\ \alpha_2 \quad \Longrightarrow \quad A \xrightarrow{\rho_1\ s\ \rho_2{>}F} \alpha_1\ {>}F \bullet \alpha_2$$

Strong forward references are resolved accordingly:



$$A \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \bullet {\gg}F\,\alpha_2 \quad\Longrightarrow\quad A \xrightarrow{\rho_1\ s\ \rho_2{\gg}F} \alpha_1 {\gg}F \bullet \alpha_2$$

Complex backward references can be resolved to an internal antecedent or — if this is not possible — to an external one. The resolution to an internal antecedent works as follows (with $1 \leq x \leq m$ and $0 \leq n$):

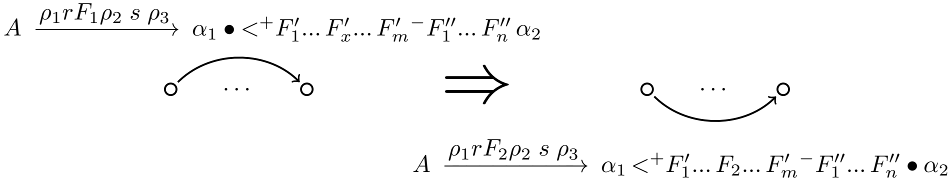


$$A \xrightarrow{\rho_1\ s\ \rho_2 r F_1 \rho_3} \alpha_1 \bullet <^+ F'_1 ... F'_x ... F'_m\,{}^- F''_1 ... F''_n\,\alpha_2 \quad\Longrightarrow\quad A \xrightarrow{\rho_1\ s\ \rho_2 r F_2 \rho_3} \alpha_1 <^+ F'_1 ... F_2 ... F'_m\,{}^- F''_1 ... F''_n \bullet \alpha_2$$

where F_1 is unifiable with F'_x and is not unifiable with any F'', and where ρ_3 contains no rF_2 such that F_2 is unifiable with an F' while being not unifiable with any F''

where F_1 and F'_x are unified and F_2 is the result of this unification

The positive feature structures of the complex backward reference are denoted by F', the negative ones by F''. The resolution of a complex backward reference to an external antecedent is straightforward:



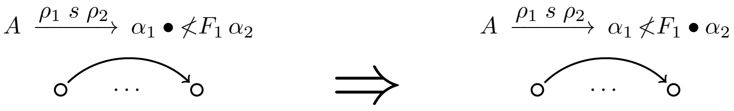
$$A \xrightarrow{\rho_1 r F_1 \rho_2\ s\ \rho_3} \alpha_1 \bullet <^+ F'_1 ... F'_x ... F'_m\,{}^- F''_1 ... F''_n\,\alpha_2 \quad\Longrightarrow\quad A \xrightarrow{\rho_1 r F_2 \rho_2\ s\ \rho_3} \alpha_1 <^+ F'_1 ... F_2 ... F'_m\,{}^- F''_1 ... F''_n \bullet \alpha_2$$

where F_1 is unifiable with F'_x and is not unifiable with any F'', and where ρ_2 and ρ_3 contain no rF_2 such that F_2 is unifiable with an F' while being not unifiable with any F''

where F_1 and F'_x are unified and F_2 is the result of this unification

Note that the same edge can produce more than one new edge when several positive feature structures can unify with the same forward reference. Since normal backward references are equivalent to complex ones for the case $x = m = 1$ and $n = 0$, they do not need to be discussed separately.

Finally, negative backward references have to be resolved. They can be resolved only if no matching antecedent exists, neither internal nor external:


$A \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \bullet \nless F_1\, \alpha_2$
. . .
$\Rightarrow$
$A \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \nless F_1 \bullet \alpha_2$
. . .

where ρ_1 and ρ_2 contain no rF_2 such that F_2 can unify with F_1

Here it becomes clear why an equivalence check — and not just a subsumption check — is needed before adding new edges to the chart. A negative backward reference that is resolvable given certain antecedent lists is not necessarily resolvable in the case of antecedent lists that are more general (e.g. have at certain positions variables instead of atoms but are otherwise the same). More specific edges can behave differently than general ones, and for this reason an edge has to be added to the chart even if a more general edge already exists.

3.5.3.8 Complexity Considerations

Let us have some brief and scruffy complexity considerations for the shown algorithm in terms of both, space and time. This can be done by a comparison to the standard Earley algorithm that has been proven to be efficient in practical applications.

The space requirements of chart parsers can be measured by the size of the chart, i.e. by the number of contained edges. As long as the positive feature structures of complex backward references are pairwise disjoint (i.e. not unifiable), the special elements of Codeco increase the number of edges in the chart — compared to the standard Earley algorithm — only linearly with respect to the number of special elements used in the grammar, and only by a constant factor with respect to the length of the token list. This can be seen by the fact that the scanning, prediction, and completion steps do not produce more edges than in the standard algorithm. Furthermore, for each edge and its descendant edges containing special elements the resolution step can be applied at most once for each special element. Only in the case of complex backward references with positive feature structures that are not pairwise disjoint, this does not necessarily hold. Thus, the chart can be expected to remain reasonably small as long as complex backward references with more than one positive feature structure are used with caution.

In terms of time complexity, it is easy to verify that the additional time needed — compared to the standard algorithm — for processing any edge in the prediction or resolution step or any two edges in the completion step is linear with respect to the number of elements in the external and internal antecedent list. Since the number of elements in the antecedent lists is linearly correlated to the number of parsed tokens and since the number of tokens increases the chart only by a constant factor, it can be concluded that the amount of additional time that is needed grows only in a linear way with respect to the number of tokens. Furthermore, checking for the equivalence of edges does not take more than twice as much time compared to checking for subsumption (because equivalence can be checked by a mutual check for subsumption). Altogether, the presented algorithm can be expected to be reasonably fast.

3.5.4 Lookahead with Codeco

Given the introduced chart parsing algorithm, providing lookahead features — as they are needed for predictive editors — can be implemented efficiently in a relatively simple way.

The basic idea is that the lookahead information is stored in the active categories of the edges in the chart. Active categories denote categories that are predicted to possibly occur after the end position of the edge. Thus, every terminal category that is an active category of an edge that has its end position at the end of a partial text is a possible token to continue the partial text. Pre-terminal categories and backward references, however, make the actual algorithm slightly more complicated.

The possible next tokens will be described as sets of options where at least one of the options must be fulfilled by a token to be a possible continuation of a given partial text. The algorithm to be introduced can describe the possible next tokens in an abstract and in a concrete way by generating a set of abstract options O_a and another set of concrete options O_c. An abstract option would say, for example, that any proper name is a possible next token. A concrete option, in contrast, would say for example that the concrete proper name "Bill" is a possible token.

In order to get this lookahead information, the partial text has to be parsed, i.e. the chart has to be filled with the edges that represent the syntactic representation of the partial text. As a next step, the abstract options can be extracted. After that, the set of concrete options can be created using the set of abstract options and the lexicon entries.

Abstract Options Extraction

First of all, a formal structure for abstract options has to be defined. In the algorithm to be presented, abstract options have the form

$$C/\,\{X_1 \ldots X_n\}$$

with $0 \leq n$ and where C and each X_j are terminal or pre-terminal categories. C denotes a category of possible next tokens with X_j being exceptions in the form of more specific categories describing tokens that are not possible. For instance, the abstract option

$$\underline{var}\ /\left\{\ \underline{var}\Big(\text{varname: X}\Big)\ \ \underline{var}\Big(\text{varname: Z}\Big)\right\}$$

states that all tokens of the pre-terminal category "*var*" are possible next tokens with the exception of those with a "varname" feature value of "X" or "Z". Concretely, this means that any variable is a possible next token except "X" and "Z". Another example is

$$\underline{pron}\begin{pmatrix}\text{refl: –}\\ \text{gender: fem}\end{pmatrix} / \{\}$$

that denotes that any irreflexive feminine pronoun is a possible next token. Terminal categories can also appear in abstract options, e.g.

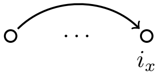
$$[\,\text{that}\,] \;/\, \{\}$$

stating that the word "that" is a possible next token.

The set of abstract options O_a is extracted from the edges of the chart. This is done by iterating over all edges that have their end position at the position where the partial text ends. This position will be denoted by i_x. Furthermore, only edges are relevant that have a terminal or pre-terminal category (denoted by T) as their active category.

First, let us consider edges that have a complex backward reference after their active category. For every edge — and for every possible F'_x therein — of the form

$$A \xrightarrow{\rho_1 \; s \; \rho_2} \alpha_1 \bullet T{<}^{+}F'_1 ... F'_x ... F'_m {}^{-}F''_1 ... F''_n \alpha_2$$



with $1 \leq x \leq m$ and $0 \leq n$, and for every rF_1 that is contained in ρ_1 or in ρ_2 and that has a feature structure F_1 that is unifiable with F'_x, an abstract option

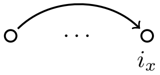
$$T' / \{T''_1 \ldots T''_t\}$$

is added to O_a where T' is the result of category T after unifying F_1 and F'_x and where the exceptions are obtained as follows: For every F'' that is unifiable with F_1, an exception T'' is added that is the result of category T after unifying F_1 and F''. The differentiation between T and T' is necessary because the unification of F_1 and F'_x can entail the binding of variables that also occur in T. Altogether, this has the effect that terminal or pre-terminal categories in front of backward references are reported as possible next tokens with exceptions that describe all cases for which the reference can afterwards not be resolved.

Again, this part of the algorithm described on the basis of complex backward references also applies for normal backward references which will not be discussed separately. Normal backward references are just a special case of complex ones.

Next, we have to handle edges with negative backward references. For every edge of the form

$$A \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \bullet T \not< F \alpha_2$$



an abstract option

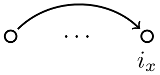
$$T/\,\{T_1' \ldots T_n'\}$$

is added to O_a where the exceptions T_i' are obtained as follows: For every rF' that is contained in ρ_1 or in ρ_2 and that has a feature structure F' that is unifiable with F, an exception T_j' is added that is the result of category T after unifying F and F'. The effect of this is that terminal or pre-terminal categories that are in front of negative backward references are reported as possible next tokens together with exceptions that describe all cases where the negative backward reference can afterwards find a matching antecedent and thus cannot be resolved.

So far, these option descriptions "look" only one step ahead. They do not cover cases where more than one terminal or pre-terminal category exists between the active position and the backward reference. In the case of complex backward references, however, it is possible and useful to look more than one step ahead. The symbol δ is used to represent a sequence of one or more terminal or pre-terminal categories.

For every edge — and for every possible F_x' therein — of the form

$$A \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \bullet T\delta <^{+} F_1' \ldots F_x' \ldots F_m'\, {}^{-}F_1'' \ldots F_n'' \alpha_2$$



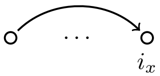
and for every rF that is contained in ρ_1 or in ρ_2 and that has a feature structure F that is unifiable with F_x', an abstract option

$$T'/\,\{\}$$

is added to O_a where T' is the result of category T after unifying F and F_x'.

Finally, edges that have a terminal or pre-terminal category at their active position but are not covered by the patterns introduced so far, an abstract option is created the following way: For every edge of the form

$$A \xrightarrow{\rho_1\ s\ \rho_2} \alpha_1 \bullet T \alpha_2$$



that is not covered by the patterns introduced above, an abstract option

$$T/\,\{\}$$

is added to O_a. This means that when no backward reference is close to the active category then this terminal or pre-terminal category is reported as a category of a possible next token.

In this way, a set of abstract options O_a is obtained that describes the possible next tokens in a general way, i.e. without considering the lexicon. As we will see, such general lookahead information can be important for predictive editors.

Concrete Options Extraction

In contrast to abstract options that can describe possible next tokens (i.e. terminal categories) without explicitly listing them, concrete options show the concrete terminal categories that are possible at the given position in the text.

Concrete options could actually just be terminal categories. For user-friendly predictive editors, however, it can be necessary to know also the pre-terminal categories from which the terminal categories are derived, e.g. for grouping the possible next words into different sub-menus. For this reason, concrete options have the form

$$W \leftarrow C$$

where W is a terminal category denoting a word and C is a pre-terminal category from which W has been derived. The following example of a concrete option represents the possibility to continue the partial text with the noun "country":

$$[\,\text{country}\,] \;\leftarrow\; \underline{noun}\left(\text{human: }-\right)$$

The special symbol "∅" is used at the position of C if the given word does not originate from a lexical rule but directly from the grammar. The concrete option

$$[\,\text{every}\,] \;\leftarrow \varnothing$$

for instance, states that "every" is a possible next token that does not come from the lexicon but is part of the grammar rules.

The set of concrete options O_c can be generated on the basis of the abstract options O_a. For every abstract option

$$W/\,\{\}$$

that is contained in O_a and where W is a terminal category, a concrete option

$$W \leftarrow \varnothing$$

is added to O_c.

Furthermore, for every abstract option

$$C/\,\{X_1 \ldots X_n\}$$

that is contained in O_a and where C is a pre-terminal category, and for each lexical rule

$$C' \rightarrow W$$

that is contained in the grammar and where C' is unifiable with C but is not unifiable with any X_j, the concrete option

$$W \leftarrow C$$

is added to O_c.

In this way, a set of concrete options O_c is obtained. This set contains the concrete word forms that are possible to follow the given partial text.

Lookahead Interface

Parsers that implement these algorithms can provide a simple interface for predictive editors to access the lookahead features. Assuming that the partial text has already been submitted to the parser, the predictive editor module can simply request the set of concrete options O_c and — if needed — also the set of abstract options O_a. In this way, the predictive editor module has all needed information in order to show to the user how the partial text can be continued, e.g. in the form of graphical menus.

The set of concrete options can directly be presented to the user as possible next words. On the basis of the set of abstract options, the predictive editor can, for example, allow users to create new words that are not yet known at the time the lookahead algorithm runs. Thus, the predictive editor does not only know which concrete words can follow a partial text but also which words in general would be allowed if they were in the lexicon.

Completeness and Correctness

Finally, the presented lookahead algorithm can be analyzed with respect to completeness and correctness.

The presented algorithm is complete in the sense that it returns every token that can, together with the tokens of the partial text, be completed to a well-formed statement according to the grammar. The algorithm is also correct in the sense that it only returns the tokens for which, together with the tokens of the partial text, a partial syntax tree exists that is valid with respect to the grammar.

These definitions of completeness and correctness leave some freedom on how to handle certain special cases. The definitions do not say anything about the tokens that lead to a valid partial syntax tree that cannot be completed to a full statement. This can happen, for example, if the edge used for predicting the next token contains at a later position a non-terminal category that does not occur as a head in any of the grammar rules. In this case, the edge can never complete and the predicted token is actually not a possible next token to complete the partial text, even though a valid partial syntax tree can be constructed.

Thus, Codeco grammars should be designed in such a way that invalid statements fail at the earliest possible position, i.e. at the first position for which no continuation to a valid statement exists. I would argue that properly designed grammars should follow this restriction anyway.

3.6 Possible Codeco Extensions

The presented grammar notation called Codeco can be seen as a proposal of a general grammar notation for CNLs. Its design was driven by ACE and — as I will show later in this chapter — it works out nicely for describing subsets of ACE. However, I cannot prove at this point that Codeco is suitable for the definition of CNLs in general (or at least for the definition of English-based CNLs in general). Extensions and modifications of Codeco might become necessary in the future in order to meet the requirements of other CNLs.

Some possible extensions are discussed below, concretely the inclusion of semantics and general feature structures.

3.6.1 Semantics in Codeco

The presented Codeco notation describes only the syntax of a CNL but not its semantics. Codeco grammars have so far mainly been used in predictive editors for which semantics are not essential. For this reason, the focus of Codeco was on syntax and not on semantics.

However, it is easy to extend the Codeco notation so that it is capable of representing semantics. A simple method is to attach λ-terms to the grammar rules, which can afterwards be retrieved from the syntax tree and compiled into a logical formula by β-reduction.

For example, a rule like

$$\mathit{quant}\Big(\text{exist: } -\Big) \ \overset{:}{\longrightarrow} \ /\!/ \ \ [\,\text{every}\,]$$

can be assigned a λDRS representation

$$\lambda P.\lambda Q.\begin{array}{|c|}\hline \\ \hline \begin{array}{|c|}\hline z \\ \hline \\ \hline \end{array} \oplus P@z \Rightarrow Q@z \\ \hline \end{array}$$

as shown by Blackburn and Bos [15] where $\oplus$ is an operator to merge DRS boxes. In this way, the semantics can be represented in a concrete and declarative way too.

An important property of this approach is that syntax and semantics are clearly separated. The semantic representation is completely irrelevant for the definition of the language, i.e. for defining which sentences are part of the language and which are not. This makes it easy, for example, to reuse a Codeco grammar or a part thereof under a different semantic framework.

3.6.2 General Feature Structures

As a second possible extension, the restriction to flat feature structures in Codeco could easily be dropped. This can make complex grammar definitions more elegant because features that are often shared between categories can be grouped. Furthermore, general feature structures would increase the expressivity of Codeco because arbitrarily nested structures could be passed from one grammar rule to another one, which is not possible otherwise.

However, general feature structures also impose new problems. With flat feature structures where the values can only be atoms or variables, only very simple cases of unification can occur: atom with atom, atom with variable, and variable with variable. This is relatively easy to implement in procedural or object-oriented programming languages in an efficient way. With general feature structures, more complex unifications — like feature structure with feature structure — have to be handled that cannot occur in plain Codeco.

Thus, the support for general feature structures would make Codeco more expressive but also more complicated and harder to implement. Furthermore, as the ACE Codeco grammar shows that will be presented in the next section, also complex subsets of English can conveniently be represented with flat feature structures.

3.7 ACE Codeco Grammar

The introduced Codeco notation has been used to describe a large subset of ACE, to be called simply *ACE Codeco*. Appendix A shows the complete grammar consisting of 164 grammar rules.

Both implementations of Codeco — the Prolog DCG representation and the Java Earley parser implementation — can not only be used to parse but also to generate sentences. Thus, all syntactically correct sentences up to a certain sentence length can be generated automatically. This can be used to evaluate the ACE Codeco grammar, the parser implementations, and the Codeco notation itself in various ways.

First of all, the coverage of ACE Codeco with respect to the full language of ACE is described. Then, a subset thereof is introduced that has been used for evaluating the grammar by exhaustive language generation, and the results of this evaluation are shown.

3.7.1 ACE Codeco Coverage

The ACE Codeco grammar covers a large part of ACE including countable nouns, proper names, intransitive and transitive verbs, adjectives, adverbs, prepositions, plurals, negation, comparative and superlative adjectives and adverbs, *of*-phrases, relative clauses, modality, numerical quantifiers, coordination of sentences / verb phrases / relative clauses, conditional sentences, and questions. Anaphoric references are possible by using simple definite noun phrases, variables, and reflexive and irreflexive pronouns.

However, there are some considerable restrictions with respect to the full language of ACE. Mass nouns, measurement nouns, ditransitive verbs, numbers and strings as noun phrases, sentences as verb phrase complements, Saxon genitive, possessive pronouns, noun phrase coordination, and commands are not covered at this point.

Nevertheless, this subset of ACE defined by the Codeco grammar is — to my knowledge — the broadest subset of English that has ever been defined in a concrete and fully declarative way and that includes complex issues like anaphoric references. This grammar is used by the predictive editor of the ACE Editor that will be introduced in Section 4.2.

3.7.2 Evaluation Subset of the ACE Codeco Grammar

When sentences are generated from a grammar in an exhaustive manner then one quickly encounters a combinatorial explosion on the number of generated sentences. In practice, this means that one has to define a sublanguage so that only the sentences of this sublanguage are generated. Otherwise, only very short sentences can be generated within reasonable time. Such a sublanguage has to be restricted on the lexical as well as on the grammatical level.

The following two sections illustrate why it is necessary to restrict lexicon and grammar, and show how a sublanguage of ACE Codeco has been defined for evaluation purposes.

3.7.2.1 Lexical Restrictions

The size of the lexicon is an obvious driver of the combinatorial explosion as the following example shows. Considering a very simple language that only supports sentences of the form "a *noun verb* a *noun*", only one sentence can be generated if there is just one noun "man" and one verb "knows":

> A man knows a man.

However, just by adding one additional noun "woman" and one additional verb "helps", we already get eight possible sentences:

> A man knows a man.
> A man knows a woman.
> A woman knows a man.
> A woman knows a woman.
> A man helps a man.
> A man helps a woman.
> A woman helps a man.

A woman helps a woman.

Apparently, things get even much worse when more lexicon entries are added. Thus, in order to prevent unnecessary combinatorial explosion when evaluating a grammar by language generation, it makes sense to use a minimal lexicon.

For the evaluation subset of the ACE Codeco grammar only the following words are used: the proper name “Mary”, the noun “woman”, the adjective “young”, the transitive adjective “mad-about”, the intransitive verb “wait”, the transitive verb “ask”, the adverb “early”, and the preposition “for”. Furthermore, “X” is the only variable name and “2” the only number. The concrete set of lexical rules of ACE Codeco is shown in Section A.3 of the appendix.

3.7.2.2 Grammatical Restrictions

As we will see, restricting the lexicon is not sufficient to handle the combinatorial explosion. It is also needed to exclude some of the grammar rules for the evaluation subset.

The following example illustrates why grammatical restrictions are necessary: In ACE, a noun can be preceded by a plain adjective like “important” but also by an adjective preceded by “more” or “most” like “more important” or “most important”. Thus, by extending the mini-language introduced above by optional adjectives, we get 16 possible sentences, even if there is only one word per category:

A man knows a man.
An important man knows a man.
A more important man knows a man.
A most important man knows a man.
A man knows an important man.
An important man knows an important man.
...
A most important man knows a most important man.

The sentences containing “more” or “most” always correspond to a valid sentence without “more” and “most”. Thus, these grammar rules do not add much to the conceptual complexity of the grammar. By excluding the grammar rules responsible for “more” and “most”, only four sentences are left:

A man knows a man.
An important man knows a man.
A man knows an important man.
An important man knows an important man.

Because this heavily reduces the number of sentences, it makes sense to evaluate the grammar without the rules for “more” and “most” and to check those rules manually for correctness.

As this example shows, it does not make sense to use all grammar rules for evaluation by exhaustive language generation. Rather, a subset of the grammar rules

sentence length	number of sentences	growth factor
3	6	
4	87	14.50
5	385	4.43
6	1’959	5.09
7	11’803	6.03
8	64’691	5.48
9	342’863	5.30
10	1’829’075	5.33
3–10	2’250’869	

Table 3.1: This table shows the number of sentences that can be generated by the evaluation subset of the ACE Codeco grammar using a minimal lexicon.

should be used by excluding the grammar rules that contribute to the combinatorial explosion but do not contribute much to the conceptual complexity of the language.

Such an evaluation subset has been defined for the ACE Codeco grammar consisting of 97 of the altogether 164 grammar rules. In Section A.2 of the appendix, the grammar rules of ACE Codeco are listed and the ones that belong to the evaluation subset are marked.

3.7.3 Exhaustive Language Generation for ACE Codeco

The evaluation subset of the ACE Codeco grammar has been translated into a Prolog DCG in order to generate all sentences up to the length of ten tokens. This leads to altogether 2’250’869 sentences.

Table 3.1 shows the number of sentences in relation to the sentence length. Every sentence consists of at least three tokens. Only six sentences exist that have the minimal length of three tokens, which look as follows:

```
(1) everybody waits .
(2) Mary waits .
(3) somebody waits .
(4) there is somebody .
(5) who waits ?
(6) X waits .
```

Note that the full stop and the question mark also count as tokens and that “there is” is just one token. 93 distinct sentences exist that have three or four tokens and they are shown in Figure 3.1.

As expected, the growth of the number of sentences is exponential. The growth factor converges to a value somewhere between 5 and 6. Thus, the number of sentences having eleven tokens can be expected somewhere around 10 millions.

```
 (1) a woman waits .
 (2) every woman waits .
 (3) everybody asks everybody .
 (4) "         "    herself .
 (5) "         "    Mary .
 (6) "         "    somebody .
 (7) "         "    who ?
 (8) "         "    X .
 (9) "         does not wait .
(10) "         is everybody .
(11) "         "  herself .
(12) "         "  Mary .
(13) "         "  somebody .
(14) "         "  who ?
(15) "         "  X .
(16) "         "  young .
(17) "         waits .
(18) "         "     early .
(19) "         X waits .
(20) it is false that everybody waits .
(21) "                Mary waits .
(22) "                somebody waits .
(23) "                X waits .
(24) Mary asks everybody .
(25) "    "    herself .
(26) "    "    Mary .
(27) "    "    somebody .
(28) "    "    who ?
(29) "    "    X .
(30) "    does not wait .
(31) "    is everybody .
(32) "    "  herself .
(33) "    "  Mary .
(34) "    "  somebody .
(35) "    "  who ?
(36) "    "  X .
(37) "    "  young .
(38) "    waits .
(39) "    waits early .
(40) somebody asks everybody .
(41) "        "    herself .
(42) "        "    Mary .
(43) "        "    somebody .
(44) "        "    who ?
(45) "        "    X .
(46) "        does not wait .
(47) "        is everybody .
```

```
(48) "        "  herself .
(49) "        "  Mary .
(50) "        "  somebody .
(51) "        "  who ?
(52) "        "  X .
(53) "        "  young .
(54) "        waits .
(55) "        "     early .
(56) "        X waits .
(57) there is a woman .
(58) "        somebody .
(59) "        "        X .
(60) which woman waits ?
(61) "     women wait ?
(62) who asks everybody ?
(63) "   "    herself ?
(64) "   "    Mary ?
(65) "   "    somebody ?
(66) "   "    who ?
(67) "   "    X ?
(68) "   does not wait ?
(69) "   is everybody ?
(70) "   "  herself ?
(71) "   "  Mary ?
(72) "   "  somebody ?
(73) "   "  who ?
(74) "   "  X ?
(75) "   "  young ?
(76) "   waits ?
(77) "   "     early ?
(78) X asks everybody .
(79) " "    herself .
(80) " "    Mary .
(81) " "    somebody .
(82) " "    who ?
(83) " "    X .
(84) " does not wait .
(85) " is everybody .
(86) " "  herself .
(87) " "  Mary .
(88) " "  somebody .
(89) " "  who ?
(90) " "  X .
(91) " "  young .
(92) " waits .
(93) " "     early .
```

Figure 3.1: This figure shows all sentences of the evaluation subset of the ACE Codeco grammar up to the length of four tokens. Note that the tokens "`does not`", "`it is false that`" and "`there is`" consist of more than one word. The sentences are sorted alphabetically and tokens that are unchanged with respect to the previous sentence are represented by quotation marks.

These figures show that it is crucial to define evaluation subsets before grammars can be evaluated by language generation. Otherwise, the number of sentences increases by more than one order of magnitude, and the language generation process described above would probably still be running at the publishing date of this thesis.

3.8 Codeco Evaluation

On the basis of the ACE Codeco grammar and its evaluation subset, a number of tests can be performed.

On the one hand, it can be evaluated whether the language described by ACE Codeco has the desired properties. We can check whether ACE Codeco contains unwanted ambiguity and whether it is indeed a subset of the full ACE language.

On the other hand, the grammar of ACE Codeco can be taken as a test case to test the Codeco notation and the two implementations thereof. We can evaluate whether the two implementations of Codeco process the ACE Codeco grammar in the same way, as they should. Furthermore, the runtime performances of the two implementations can be tested and compared.

3.8.1 Ambiguity Check of ACE Codeco

Languages like ACE are designed to be unambiguous on the syntactic level. This means that every valid sentence must have exactly one syntax tree according to the given grammar. By exhaustive language generation, the resulting sentences can be checked for duplicates. Sentences generated more than once have more than one possible syntax tree and are thus ambiguous.

Up to the length of ten tokens, no ambiguous sentences are generated by the evaluation subset of ACE Codeco. Thus, at least a large subset of ACE Codeco is unambiguous for at least relatively short sentences.

On the basis of this result, some “soft conclusions” for the full ACE Codeco language can be made. It can be verified that the rules missing in the evaluation subset do not introduce ambiguity. Furthermore, it can be argued that most types of ambiguity would be discoverable in sentences of ten or less tokens. Thus, the ACE Codeco grammar can in good conscience be considered unambiguous.

Actually, several cases of ambiguity could be found in this way in earlier versions of the ACE Codeco grammar. These ambiguities could then be remedied. This shows how important it is to be able to check automatically for ambiguity. Otherwise, it would have been very hard to detect these cases.

3.8.2 Subset Check of ACE Codeco and Full ACE

The ACE Codeco grammar is designed as a proper subset of ACE. It can now be checked automatically whether this is the case, at least for the evaluation subset of

ACE Codeco and up to a certain sentence length.

Every sentence up to the length of ten tokens was submitted to the ACE parser (APE) and parsing succeeded in all cases. Since APE is the reference implementation of ACE, this means that these sentences are syntactically correct ACE sentences.

The number of ten tokens can be considered sufficiently high for detecting most potential classes of sentences that are part of ACE Codeco but not of ACE. Thus, the results of this test indicate that the ACE Codeco grammar indeed describes a subset of ACE.

Interestingly, this test discovered several previously unknown bugs of both, Codeco and APE, which could then be fixed.

3.8.3 Equivalence Check of the Implementations

For the generation of the sentences up to a length of ten tokens, the Prolog DCG parser has been used because it is faster than the Java implementation. However, the Java implementation can also be used for language generation. This enables us to check whether the two implementations accept the same set of sentences, as they should, for the ACE Codeco grammar.

The Java implementation has been used to generate all sentences up to the sentence length of eight tokens. Since the Java implementation is slower than the one based on Prolog DCGs (see the next section), the former cannot generate as long sentence as the latter within reasonable time. The resulting set of sentences generated by the Java implementation was identical to the one generated by the Prolog DCG. This is an indication that the two implementations contain no major bugs and that they interpret Codeco grammars in the same way.

3.8.4 Performance Tests of the Implementations

Finally, the performance of the two implementations of Codeco can be evaluated and compared. Both implementations can be used for parsing and for generation, and thus the runtimes in these two disciplines can be compared. For this test, again the ACE Codeco grammar has been used.

The first task was to generate all sentences of the evaluation subset of ACE Codeco up to the length of seven tokens. The second task was to parse again the sentences that result from the generation task. This parsing task was performed in two ways for both implementations: once using the evaluation subset and once using the full ACE Codeco grammar. The restricted lexicon of the evaluation subset has been used in both cases. These tests were performed on a MacBook Pro laptop computer having a 2.4 GHz Intel Core 2 Duo processor and 2 GB of main memory. SWI Prolog 5.6.61 and Java 1.5.0_19 have been used. Table 3.2 shows the results of these performance tests.

The generation of the 14'240 sentences only requires about 41 seconds in the case of the Prolog DCG implementation. This means that less than 3 milliseconds are

			time in seconds	
task	grammar	implementation	*overall*	*average*
generation	ACE Codeco eval. subset	Prolog DCG	40.8	0.00286
generation	ACE Codeco eval. subset	Java Earley parser	1040.	0.0730
parsing	ACE Codeco eval. subset	Prolog DCG	5.13	0.000360
parsing	ACE Codeco eval. subset	Java Earley parser	392.	0.0276
parsing	full ACE Codeco	Prolog DCG	20.7	0.00146
parsing	full ACE Codeco	Java Earley parser	1900.	0.134
parsing	full ACE	APE	230.	0.0161

Table 3.2: This table shows the results of a performance test of the two implementations of Codeco, i.e. the DCG version and the Java implementation as an Earley parser. The first task was to generate all sentences of the evaluation subset of the ACE Codeco grammar up to the length of seven tokens. This leads to 14'240 sentences. As a second task, the sentences that result from the generation task should be parsed again. This parsing task was performed using the evaluation subset as well as the full grammar of ACE Codeco. The overall time values denote the time needed for all 14'240 sentences. The average values are obtained by dividing the overall time by 14'240. As a comparison, the performance of the existing ACE parser (APE) is shown for the parsing task.

needed on average for generating one sentence. The Java implementation, in contrast, needs about 17 minutes for this complete generation task, which corresponds to 73 milliseconds per sentence. Thus, generation is about 25 times faster when using the Prolog DCG version compared to the Java implementation. These results show that the Prolog DCG implementation is well suited for exhaustive language generation. The Java implementation is much slower but the time values are still within a reasonable range.

The Prolog DCG approach is amazingly fast for parsing the same set of sentences using the evaluation subset of the grammar. Here, parsing just means detecting that the given statements are well-formed according to the grammar. Altogether only slightly more than 5 seconds are needed to parse the complete test set, i.e. less than 0.4 milliseconds per sentence. When using the full ACE Codeco grammar for parsing the same set of sentences, altogether 21 seconds are needed, i.e. about 1.5 milliseconds per sentence. The Java implementation is again much slower and requires almost 30 milliseconds per sentence when using the grammar of the evaluation subset, which leads to an overall time of more than 6 minutes. For the full grammar, 134 milliseconds are required per sentence leading to an overall time of about 32 minutes. Thus, the Java implementation is 76 to 92 times slower than the Prolog DCG for the parsing task. Because all time values are clearly below 1 second per sentence, both parser implementations can be considered fast enough for practical applications. If large amounts of sentences have to be parsed, however, the Prolog DCG version should be preferred.

The fact that the Java implementation requires considerably more time than the Prolog DCG is not surprising. Prolog is known to be a good language for implementing natural language grammars. DCG grammar rules in Prolog are directly translated into Prolog clauses and generate only very little overhead. Java, in contrast, has no special support for executing grammar rules and for this reason the processing of grammar rules has to be implemented on a higher level. Variables that can unify with other variables according to the laws of logic come for free with Prolog but have to be implemented on a higher level in the case of Java. Even though only very simple types of variable unifications can occur with Codeco, it creates a significant overhead in Java.

As a comparison, the existing parser APE — the reference implementation of ACE — needs about 4 minutes for the complete parsing task. Thus, it is faster than the Java implementation but slower than the Prolog DCG version of Codeco. However, it has to be considered that APE does more than just accepting well-formed sentences. It also creates a DRS representation and a syntax tree.

Compared to natural language parsers, the performance results of both implementations can be considered more than satisfying. The parsing times of existing parsers for unrestricted natural language range from about 50 milliseconds up to more than 15 seconds per sentence [111, 34]. Again, these systems do more than just accepting well-formed sentences as they return a representation of the syntax like a syntax tree. Furthermore, they have to handle complex cases of ambiguity, which is not the case for the Codeco implementations. Nevertheless, this comparison shows that the Codeco implementations perform reasonably well.

3.9 Concluding Remarks on Codeco

In summary, the Codeco notation allows us to define controlled subsets of natural languages in a convenient and adequate way. The resulting grammars have a declaratively defined meaning, can be interpreted in different kinds of programming languages in an efficient way, and allow for lookahead features that are important for predictive editors. Furthermore, Codeco enables automatic testing of a given grammar, e.g. by exhaustive language generation, which is very important for the development of reliable practical applications. Altogether, Codeco embodies a more engineering focused approach to CNLs.

The next chapter will show — on the basis of concrete tools that have been developed — that the theoretical considerations of this chapter actually have practical relevance and allow us to build better applications.

CHAPTER 4

Tools

Having introduced the theoretical concepts on how to define CNLs, we can now have a closer look at the practical application thereof. This chapter targets the second research question defined in the introduction of this thesis:

> **2. How should tools for controlled English be designed?**

Controlled natural languages are supposed to improve the communication between humans and computers. Apparently, this requires not only carefully designed CNLs but also appropriate software tools that embed these languages and provide user interfaces that enable the easy usage of the CNL.

CNLs have an apparent impact on the user interface level. However, they should not be seen as an interface issue only. User interfaces in general should be more than just the linking component between the users and the rest of the program, or by quoting Don Norman [137]:

> ❝ What's wrong with interfaces? The question, for one. The interface is the wrong place to begin. It implies you already have done all the rest and now you want to patch it up to make it pretty for the user. That attitude is what is wrong with the interface. ❞

The key point is that good user interfaces originate from system-scale design decisions that take the user interface into account from the very beginning. In my view, existing (controlled) natural language interfaces for knowledge representation systems did not work out so far because the user interfaces have just been put onto the top of existing systems in most cases. Knowledge representation systems should be designed with user interfaces in mind from the beginning.

In this chapter, I will first introduce some design principles for user interfaces based on CNL (Section 4.1). Then, three knowledge representation tools will be described that I have developed and that embed CNL not only as an interface language but as the philosophy of the complete design. The ACE Editor is a general text editor for ACE containing a predictive editor (Section 4.2). AceRules is a rule engine using ACE as its input and output language (Section 4.3). AceWiki, finally, is a relatively mature semantic wiki engine that has been evaluated within several small user studies (Section 4.4).

All three tools are implemented as web applications that run inside a web browser. They are implemented using the Echo Web Framework[1], which is a modern Java-based web application framework. This web-based approach has the advantage that everyone can run the applications without the need to install anything.

4.1 Design Principles for CNL User Interfaces

A lot of work has been done on how user interfaces of computer programs should look like [96] and there are well-known general principles on how such interfaces should be designed. For example, user interfaces should be tailored to the goals and skills of the potential users, they should reuse existing metaphors that the users know from other programs, and the behavior of the interface should be consistent and should not surprise the users [167]. In the case of user interfaces using CNL, some additional and more concrete design principles can be identified.

My own experience from developing the three applications that will be introduced later in this chapter can be summarized by the following three design principles:

CNL user interfaces ...

1. ... should follow the natural spirit of CNLs.

2. ... should solve the writability problem of CNLs.

3. ... should not let users confuse CNL with natural language.

The first principle means that a CNL user interface should assimilate the naturalness of the CNL. Thus, the interface as a whole should be led by the nature of natural language and not, for example, by technical aspects. My claim is that a CNL can unfold its naturalness only if surrounded by a user interface with the same natural flavor. For example, the CNL sentences should not be called “axioms” by the user interface but just “sentences”. Similarly, the interface should rather use

[1] http://echo.nextapp.com/site/

linguistic terms like "verb" and "noun phrase" instead of technical terms like "object property" and "concept description".

The second principle means that the writability problem of CNLs — as described in Section 2.1.4 — has to be solved by the user interface. If the error messages approach is taken, this means that the user interface must be able to show error messages when sentences do not comply with the restrictions of the respective CNL. Furthermore, users must be given practical advice in a reliable and understandable way how to resolve the problem. In the case of the predictive editor approach, such a predictive editor must be tightly integrated so that users are supported by the lookahead capabilities of the editor when writing CNL statements. If the language generation approach is adopted, finally, the user interface must enable users to trigger modifications of the underlying model, which is then immediately verbalized and shown to the users. In any case, writability is a critical aspect to which special attention has to be paid when designing CNL tools.

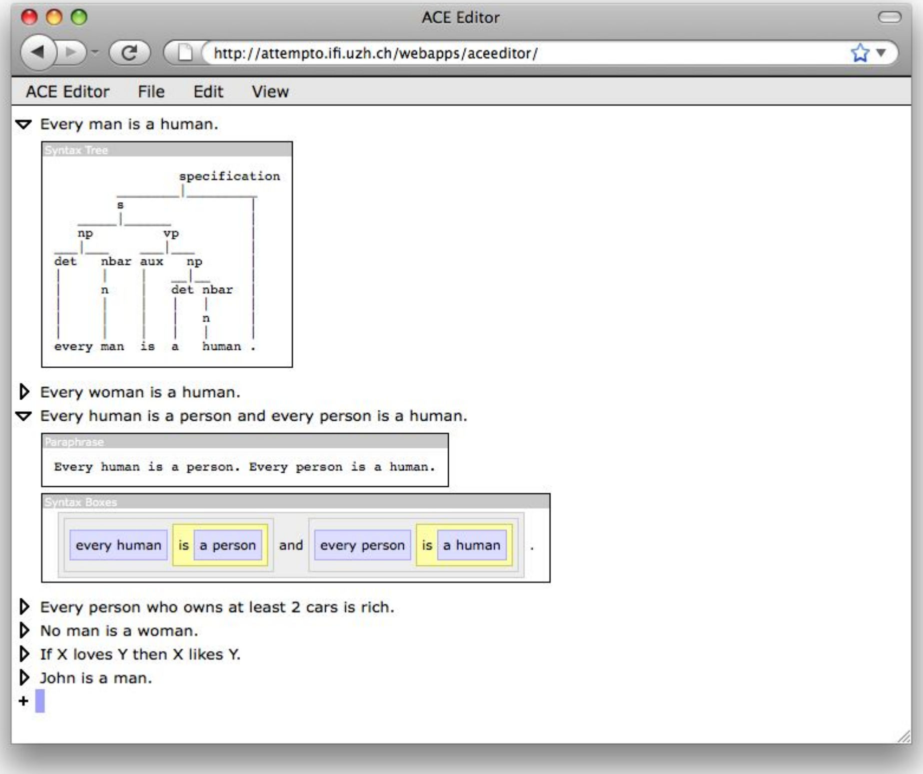
The third principle, finally, means that users should get the opportunity to get familiar with the CNL and to understand the differences to full natural language, even if this understanding only happens on an unconscious level. The main problem is that CNL can easily be confused with full natural language. User interfaces using CNL still need natural language at different places, for example for informal explanations, help pages, and different kinds of labels. This raises the danger of misunderstandings. An informal explanation can be misinterpreted as a formal statement, and vice versa. My claim is that if the text in CNL cannot be clearly distinguished from the text that is in full natural language, it gets much harder for the users to learn how the CNL and the complete system work.

4.2 ACE Editor

The ACE Editor is the first of three tools to be introduced. It is a general editor for writing and modifying ACE texts. Figure 4.1 shows a screenshot. It includes a predictive editor that enables easy creation and modification of ACE sentences. Furthermore, it uses the ACE parser (APE) to show different kinds of parsing results (e.g. syntax trees, paraphrases and DRSs) to the users.

The ACE Editor is not a finished tool but rather a general basis to create domain-specific tools on top of it. The purpose of the ACE Editor is to demonstrate how an editor can be built that enables to write and modify texts in a CNL in a simple and intuitive way. Users should not need to learn the grammar of ACE in advance, but they should be able to learn the language while using the editor.

The most important component of the ACE Editor is the predictive editor that helps users to write syntactically correct ACE statements. This predictive editor makes use of the ACE Codeco grammar introduced in Section 3.7. The grammar is processed by the Java implementation of the Earley parsing algorithm as described in Section 3.5. In this way, the predictive editor can rely on the lookahead features



Every man is a human.

```
                  specification
         ______________|______________
        s                             |
   _____|_____                        |
  np          vp                      |
 _|__       __|__                     |
det  nbar aux    np                   |
 |    |    |   __|__                  |
 |    n    |  det  nbar               |
 |    |    |   |    |                 |
 |    |    |   |    n                 |
 |    |    |   |    |                 |
every man  is  a   human              .
```

Every woman is a human.

Every human is a person and every person is a human.

Every human is a person. Every person is a human.

every human | is | a person | and | every person | is | a human | .

Every person who owns at least 2 cars is rich.

No man is a woman.

If X loves Y then X likes Y.

John is a man.

+

Figure 4.1: This screenshot shows the ACE Editor. This editor can be used to create and modify ACE texts. The ACE sentences are parsed in the background and different kinds of parsing results can be shown, like syntax trees or paraphrases.

of the parser (see Section 3.5.4). The predictive editor is a modular component that can be reused by other applications. For example, it is used by AceWiki that will be introduced later in this chapter.

In the following sections, the general approach behind the predictive editor is explained and the concrete components of the predictive editor interface are described.

4.2.1 Predictive Editing Approach

The biggest problem when designing a predictive editor for a CNL is that there can potentially be many different ways how a partial sentence can be continued. All these possibilities have to be shown to the user in a simple and understandable way. Furthermore, the user should not have to do much more than one mouse click per word to be added. Otherwise, creating a complete sentence gets very cumbersome and slow.

The following concrete example illustrates the difficulty of providing good interfaces for predictive editors: The partial sentence

Every country is ...

can be continued in many different ways. First of all, one can negate the verb phrase by adding "not", e.g. "every country is not a continent". The partial sentence could also be followed by a noun phrase starting with a determiner "a", "every" or "no", e.g. "every country is an area". Noun phrases can also consist of an indefinite pronoun, e.g. "every country is something that controls an area", or of a query pronoun, e.g. "every country is what?". Furthermore, a new variable could be added, e.g. "Every country is X and does not border X", or a proper name, e.g. "every country is Switzerland". The latter is semantically incorrect but syntactically well-formed. Normal or transitive adjectives can also be used, e.g. "every country is important" or "every country is located-in a continent". Adjectives can also have a comparison object, in which case the partial sentence is continued by either "as" or "more", e.g. "every country is more important than John". Another option is to use passive, e.g. "every country is visited by John". Finally, a reference can be used, e.g. "every country is itself", even though it does not make much sense in this particular example.

Looking at all those different possibilities, it becomes clear that it is not easy to come up with an interface that supports all of them but is still easy to use in an efficient way. The fact that words like proper names, adjectives, and verbs are open classes that can potentially contain a high number of individual words makes it even more complicated.

The approach of the ACE Editor to solve this problem is to use menu boxes that occupy most of the space of the predictive editor user interface. Each of these menu boxes contains the menu items for a particular type of word. The menu items are vertically listed and scroll bars are used when they need more space than the menu box provides. The predictive editor window shows only the menu boxes which represent possible word classes for the given partial sentence. For example, the menu box for verbs is only shown when a verb is a possible continuation. This has the consequence that the number of menu boxes depends on the actual position in the sentence. Depending on their number, the menu boxes are arranged horizontally in one or two rows.

In this way, the different possibilities to continue a partial sentence can be shown in the predictive editor window with reasonable space requirements. Still, the selection of a word only takes one mouse click and possibly some scrolling.

Another problem when designing predictive editors is that both, beginners and advanced users, should be able to use the editor efficiently. Beginners have to completely rely on the menus provided. Advanced users, however, do not need the menus all the time and are faster when the input can be done by typing on the keyboard instead of clicking with the mouse.

The predictive editor of the ACE Editor accounts for this issue by providing a text field in which users can write freely. The predictive editor can thus be used in two ways, by clicking on the menu items and by typing in the text field. These two possibilities are available at every point and the user is free to switch from one to the other even in the middle of the sentence creation process. In this way, beginners and advanced users can use the same editor.

The results of the AceWiki experiments to be presented in Section 4.4.4 confirm that the predictive editor is easy to use for untrained users. From my own experience, I can say that it can also be used conveniently by advanced users in an efficient way.

4.2.2 Components of the Predictive Editor

Figure 4.2 shows the interface of the predictive editor. It is contained in an internal window that is displayed within the browser window.

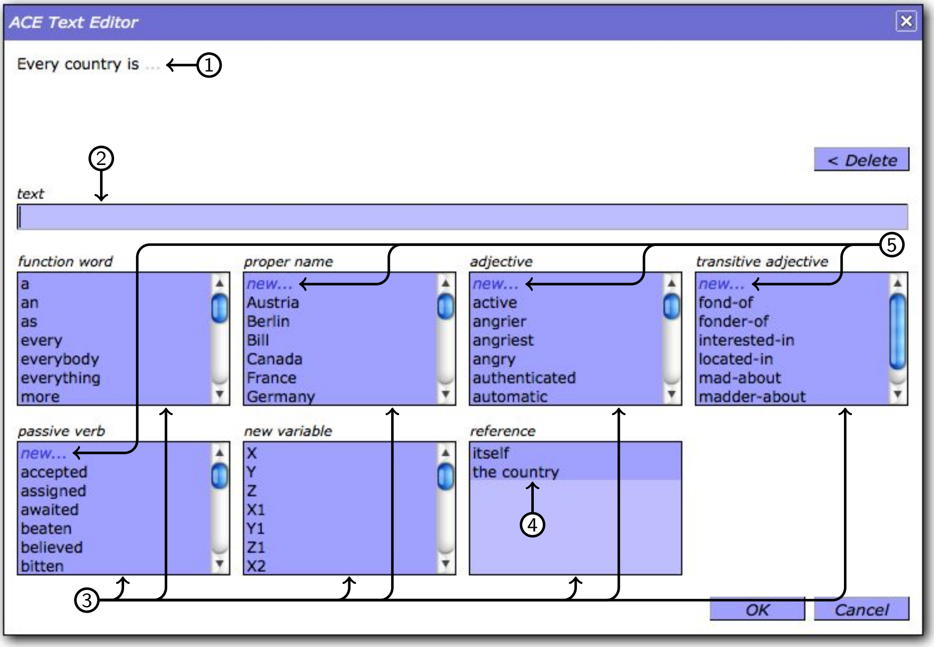
The partial sentence is shown at the very top of the window. This partial sentence has already been entered by the user and it has been accepted by the parser as a correct sentence beginning. The partial sentence is followed by three dots "..." to indicate that further tokens can be added.

Below the partial sentence, there is a text field that can be used to enter one or more words to be added to the end of the partial sentence. When pressing *enter*, the words of this text field are added to the end of the partial sentence if they are accepted by the grammar. The tab key can be used to trigger autocompletion. For example, if "Switz" is typed into the text field and the tab key is pressed then the content of the text field is automatically completed to "Switzerland" if this word is in the lexicon.

Below the text field, there are a number of menu boxes. In this particular case, there are seven of them but the actual number depends on the partial sentence. Each menu box contains a list of menu items, which stand for possible continuations of the partial sentence. These continuations are retrieved by the chart parser applying the algorithm described in Section 3.5.4.

By clicking on one of the menu items, its content is added to the end of the partial sentence. Anaphoric references are handled in exactly the same way, and only references that are possible at the given position are shown. The text field above the menu boxes can also be used to filter the menu items. For example, if "co" is entered into text field, only menu items starting with "co" are shown.

Words of the open word classes (i.e. content words) that are not yet in the lexicon can be added on the fly, i.e. while writing a sentence. Each menu box that stands for a class of content words has a special menu entry "new...". By clicking on such an entry, a small window pops up containing a form that requests the needed linguistical information for the new word to be created, e.g. singular and plural word forms for



Every country is ...

Figure 4.2: This screenshot shows the predictive editor that is a part of the ACE Editor and that is also used by AceWiki. (1) shows the partial ACE sentence. The text field (2) can be used to enter the next words of the sentence and can also be used to filter the entries of the menu boxes (3). Clicking on the entries of the menu boxes (3) is an alternative way to construct a sentence. References can be introduced that point to objects occurring earlier in the sentence (4). Furthermore, new words can be defined on the fly by clicking on one of the menu entries "new..." (5).

the case of nouns.

The button "Delete" can be used to remove the last token from the partial sentence. At the very bottom of the window, there are the two common buttons "OK" and "Cancel". Pressing "OK" succeeds only if the sentence is completed, i.e. it is not possible to submit an unfinished sentence.

The usage of anaphoric references is straightforward. It is checked which anaphoric references are allowed at the given position and then these options are shown to the user in a separate menu box "reference".

When the user clicks on one of the anaphoric references then the predictive editor can — with the help of the chart parser — show how this reference was resolved. This is done by underlining the part of the text that represents the antecedent to which the reference has been resolved. Figure 4.3 illustrates how this works. In this way, anaphoric references can be used in the predictive editor in a simple and intuitive way. They do not introduce further complexity to the user interface.

The predictive editor presented here fully relies on the chart parsing and lookahead algorithms presented in Section 3.5. If software modules that implement such algorithms are available then predictive editors can be implemented with little effort.

4.3 AceRules

The AceRules system is the second tool to be introduced. It is a prototype that exemplifies how ACE can be used to express rules as needed, for example, in business rule approaches [139]. Such kinds of rule systems represent knowledge in the form of sets of rules and facts and enable reasoning over them (i.e. executing the rules). Since such rules have to be created and/or verified by the respective domain experts who are mostly not familiar with formal notations, it is important to provide intuitive interfaces, e.g. by CNLs.

Existing work to use natural language representations for rule systems is based on the idea of verbalizing rules that already exist in a formal representation [66, 77, 99]. The NORMA system [67], for example, is able to verbalize existing formal rules. CNLs like RuleSpeak and SBVR Structured English [149, 159] have been proposed for rule systems, but they are not strictly formal and thus cannot be processed in a fully automatic way.

In contrast to existing approaches, the approach of AceRules is to use CNL as the main rule language that is automatically translated into the internal rule format (parsing) and backwards (verbalizing). Both, input and output of AceRules are represented in ACE.

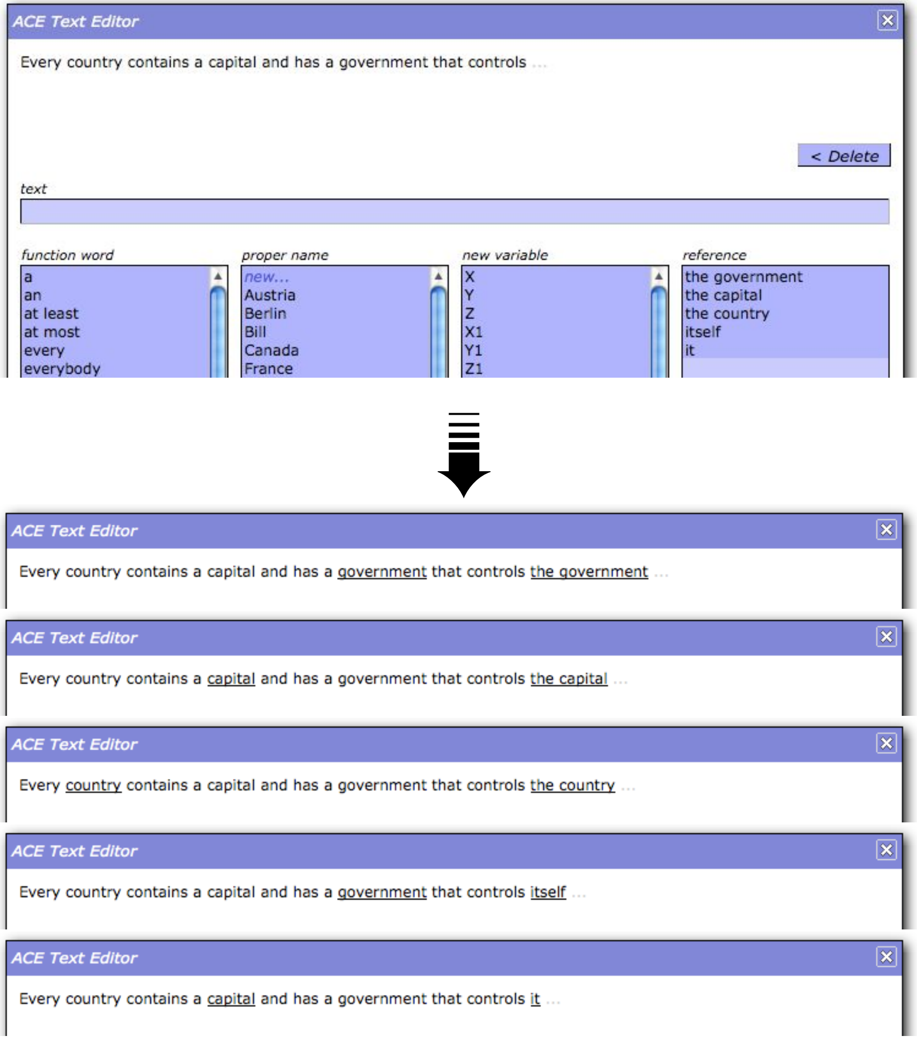
The following sections show how rules are interpreted (i.e. executed) in AceRules, and the multi-semantics architecture of AceRules is explained. Finally, the user interface is shown and described.

4.3.1 Rule Interpretation in AceRules

AceRules is designed for forward-chaining interpreters that calculate the complete answer set for a given set of rules and facts. The general approach of using ACE for rules, however, could easily be adopted for backward-chaining interpreters that take a specific statement and try to find a proof or disproof for it on the basis of the given rules. The forward-chaining approach has been chosen because it is better suited for demonstration purposes.

In order to clarify how AceRules works, let us have a look at the following simple program in the form of a set of rules and facts:

> John is a man and Bill is a man.
> Every man is a person.



Every country contains a capital and has a government that controls ...

Every country contains a capital and has a government that controls the government ...

Every country contains a capital and has a government that controls the capital ...

Every country contains a capital and has a government that controls the country ...

Every country contains a capital and has a government that controls itself ...

Every country contains a capital and has a government that controls it ...

Figure 4.3: This figure shows how anaphoric references are handled in the predictive editor. At the top, the window of the predictive editor is shown with a partial sentence which can be continued by using an anaphoric reference. At the given position five different references are possible, which are shown in the rightmost menu box "reference". When the user clicks on one of these reference entries then the editor shows how the reference has been resolved by underlining the respective part of the text.

```
Mary is a woman and Sue is a woman.
Every woman is a person.
No man is a woman and no woman is a man.
John is a relative of Bill and is a friend of Sue.
Sue is not a relative of John.
If X is a relative of Y then Y is a relative of X.
Everybody who is a relative of a person is a friend of the person.
Every person who is not a relative of John is not a friend of Mary.
```

Submitting this program to AceRules in courteous mode (the different modes will be discussed in the next section), we get the following answer in the form of a set of facts that can be derived from the program:

```
John is a relative of Bill.        Mary is a person.
Bill is a relative of John.        Sue is a person.
John is a friend of Bill.          Bill is a person.
John is a friend of Sue.           It is false that Sue is a friend of Mary.
Bill is a friend of John.          It is false that Sue is a relative of John.
Sue is a woman.                    It is false that Bill is a woman.
John is a man.                     It is false that John is a woman.
Bill is a man.                     It is false that Mary is a man.
Mary is a woman.                   It is false that Sue is a man.
John is a person.
```

As this example shows, input and output of AceRules are both in ACE and no other formal notation is needed for the user interaction. Even inexperienced users should be able to understand input and output and to verify that the output is some kind of conclusion of the input.

AceRules reuses several existing ACE tools. First of all, the program is parsed by the ACE parser (APE) and transformed into its DRS representation. This DRS representation is then translated by AceRules into an internal rule structure. Then different interpreter modules can be applied, which are discussed in the next section. From this step, we get a set of facts in an internal rule format. AceRules transforms this back into a DRS representation. Finally, this representation can be given to the existing ACE verbalizer module (that is used by APE for paraphrasing) returning the final answer in ACE.

4.3.2 Multi-Semantics Architecture of AceRules

A broad variety of different rule semantics have been defined, each exhibiting certain properties. Depending on the application domain, the characteristics of the available information, and on the reasoning tasks to be performed, different properties are desirable.

AceRules as a general demonstration prototype is designed to support various semantics in the way that the rule interpreter module is exchangeable. At the moment, three different semantics are incorporated: courteous logic programs [63], stable models [60], and stable models with strong negation [61].

The original stable model semantics supports only negation as failure, but it has been extended by support for strong negation. Courteous logic programs are based on stable models with strong negation and support therefore both forms of negation. None of the two forms of stable models guarantee a unique answer set. Thus, some programs can have more than one answer. In contrast, courteous logic programs generate always exactly one answer but are restricted to acyclic programs [6]. Depending on the concrete application domain, one or the other semantics might be the best choice.

The three semantics are implemented in AceRules as two interpreter modules. The first one handles courteous logic programs and is implemented natively. For the stable model semantics with and without strong negation there is a second interpreter module that wraps the existing tools *Smodels* [118] and *Lparse* [165].

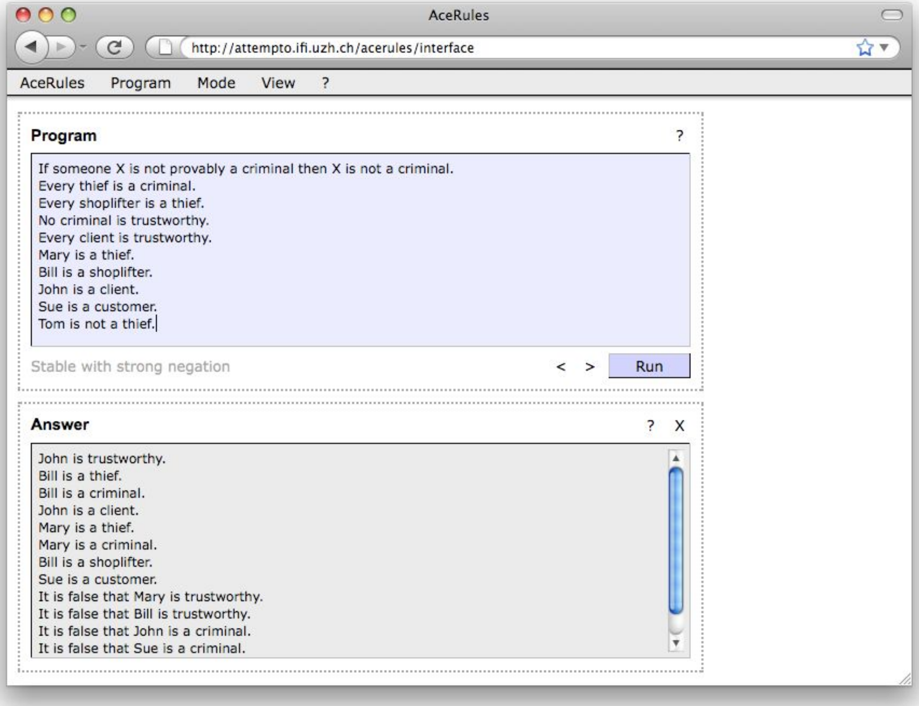
There are various other semantics that could be supported by AceRules, e.g. defeasible logic programs [119] or disjunctive stable models [133]. Only little integration effort would be necessary to incorporate them.

4.3.3 AceRules Interface

AceRules has a simple web interface shown by Figure 4.4. This interface is designed to hide all technical details by relying on CNL. Basically, the interface consists of two text areas. The first text area labeled “Program” can be used to write free text that is interpreted as a rule set in ACE. The second text area is read-only and has the label “Answer”. After pressing “Run”, the calculated answer is shown in the answer text area. If more than one answer is entailed by the program then the different answers are shown in separate tabs.

Users can load and save programs, switch between the different semantics, and change some settings. For example, users can define how many answers should be calculated as a maximum, which is needed for the semantics that do not guarantee a unique answer. Furthermore, there are many help pages that explain the interface and how error messages have to be interpreted.

Something that AceRules does not provide, however, is proper writing support. Because the development of the predictive editor only began after AceRules was developed, it adopts the error messages approach to the writability problem. When the user enters something that is not correct ACE or cannot be transformed into a rule structure then simply an error message is shown to the user. Such error messages try to explain why the given program could not be run, but it might be hard — at least for untrained users — to understand this well enough to be able to fix the problem. Thus, the writability problem is not solved in a satisfying way. However, it would not be hard to include the predictive editor module described in Section 4.2. AceWiki — to be introduced in the next section — demonstrates that the predictive


AceRules
http://attempto.ifi.uzh.ch/acerules/interface
AceRules
Program
Mode
View
?
Program
?
Stable with strong negation
<
>
Run
Answer
?
X

If someone X is not provably a criminal then X is not a criminal.
Every thief is a criminal.
Every shoplifter is a thief.
No criminal is trustworthy.
Every client is trustworthy.
Mary is a thief.
Bill is a shoplifter.
John is a client.
Sue is a customer.
Tom is not a thief.

John is trustworthy.
Bill is a thief.
Bill is a criminal.
John is a client.
Mary is a thief.
Mary is a criminal.
Bill is a shoplifter.
Sue is a customer.
It is false that Mary is trustworthy.
It is false that Bill is trustworthy.
It is false that John is a criminal.
It is false that Sue is a criminal.

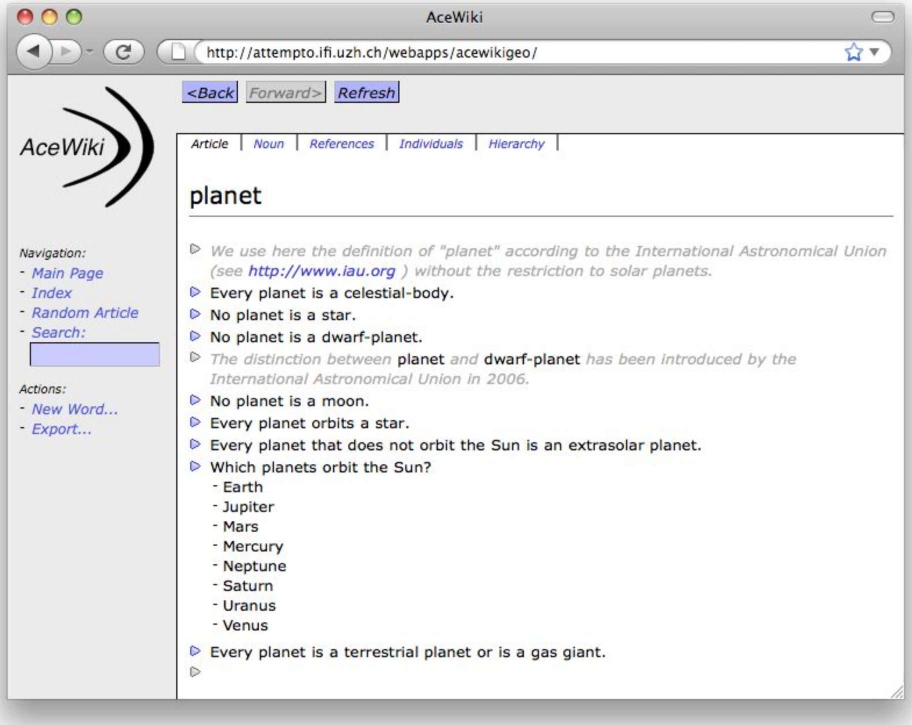
Figure 4.4: This is a screenshot of the AceRules web interface showing an exemplary program (i.e. rule set) at the top and the inferred answer at the bottom.

editor can easily be integrated in a specialized application.

Altogether, the AceRules prototype shows how rules can be defined and executed without requiring the users to learn a complicated formal language. AceRules has not been tested in user experiments and the poor writing support would probably be a hindrance in using AceRules efficiently. However, the experiences with AceWiki show that predictive editors can be embedded in a way that enables untrained user to deal with CNL-based tools in an efficient way.

4.4 AceWiki

AceWiki is the third and last tool to be presented. It is a semantic wiki using ACE to represent the content of its articles. In this way, the content is both, human readable and interpretable for reasoners. The general goal is to show that semantic wikis



planet

▷ We use here the definition of "planet" according to the International Astronomical Union (see http://www.iau.org) without the restriction to solar planets.
▷ Every planet is a celestial-body.
▷ No planet is a star.
▷ No planet is a dwarf-planet.
▷ The distinction between planet and dwarf-planet has been introduced by the International Astronomical Union in 2006.
▷ No planet is a moon.
▷ Every planet orbits a star.
▷ Every planet that does not orbit the Sun is an extrasolar planet.
▷ Which planets orbit the Sun?
- Earth
- Jupiter
- Mars
- Mercury
- Neptune
- Saturn
- Uranus
- Venus
▷ Every planet is a terrestrial planet or is a gas giant.
▷

Figure 4.5: This screenshot of the AceWiki interface shows an article about planets. Articles in AceWiki consist of declarative ACE sentences and ACE questions (both in black) and of unrestricted natural language comments (in gray).

can be at the same time more expressive and more usable than existing systems. Figure 4.5 shows a screenshot of the AceWiki web interface.

Wikis are a flexible and dynamic way to share knowledge and have become well-known by the incredible success of Wikipedia. During the eight years since the start of Wikipedia in 2001, altogether more than 14 million articles have been created in the different language versions[2].

Below, other existing semantic wikis are discussed. Then, AceWiki is introduced and it is shown how knowledge is expressed and how reasoning takes place. Furthermore, the results of several tests that have been performed on AceWiki are discussed. Two experiments have been set up to test the usability of AceWiki. Additionally, a

[2] http://meta.wikimedia.org/wiki/List_of_Wikipedias retrieved in November 2009

small case study has been performed in order to test the general usefulness.

4.4.1 Other Semantic Wikis

Semantic wikis are a relatively new field of research that began in 2004 when semantic wikis were introduced by Tazzoli et al. [168] describing the PlatypusWiki system. During the last years, an active community emerged and many new semantic wiki systems were presented.

Semantic wikis combine the philosophy of wikis (i.e. quick and easy editing of textual content in a collaborative way over the web) with the concepts and techniques of the Semantic Web (i.e. enriching the data on the web with well-defined meaning). The idea is to manage formal knowledge representations within a wiki environment.

Generally, two types of semantic wikis can be distinguished: On the one hand, there are text-centered approaches that enrich classical wiki environments with semantic annotations. On the other hand, logic-centered approaches use semantic wikis as a form of online ontology editors. A brief overview of existing semantic wiki engines is given here focusing on those that have been under active development during the last years.

Semantic MediaWiki [91] and IkeWiki [141] are two well-known examples of text-centered semantic wikis. Semantic MediaWiki is probably the most popular and one of the most mature existing semantic wiki engines. It builds upon the MediaWiki engine (which is used e.g. for Wikipedia) and has been used and extended by many companies and research groups. IkeWiki is also a mature and popular semantic wiki engine, which — among other applications — was the basis for the SWiM system [94] that is dedicated specifically to mathematical content. Recently, the developers of IkeWiki started a new project called KiWi [142], for which a new semantic wiki engine has been developed that builds upon the experiences made with IkeWiki.

The SweetWiki system [25] is another example of a text-centered semantic wiki and is special in the sense that it focuses on social tagging and folksonomies. HyperDEWiki [143] is a further example of the text-centered approach, with a focus on views and ontology evolution. Hyena [135], finally, is yet another semantic wiki engine in the text-centered fashion. It allows users to access and modify the wiki data not only through the web-based interface but additionally provides an editor that can be run locally.

OntoWiki [8] and myOntology [152] are two examples of logic-centered semantic wikis. Web-Protégé [171] is the web version of the popular Protégé ontology editor and can be seen as another example of a logic-centered semantic wiki, even though its developers do not call it a "semantic wiki".

In general, there are many new web applications that do not call themselves "semantic wikis" but exhibit many of their characteristic properties. Freebase[3], Knoodl[4],

[3] see [17] and `http://www.freebase.com`

[4] `http://knoodl.com`

and SWIRRL[5] are some examples.

Altogether, semantic wikis seem to be a promising approach to get the domain experts more involved in the creation and maintenance of ontologies. This could increase the number and quality of available ontologies, which is an important step into the direction of making the Semantic Web a reality.

However, two major problems can be identified with existing semantic wikis. First, most of them have a very technical interface that is hard to understand and use for untrained persons, especially for those who have no particular background in formal knowledge representation. Second, existing semantic wikis support only a relatively low degree of expressivity — mostly just "subject predicate object"-structures — and do not allow users to assert complex axioms. These two shortcomings have to be overcome to enable average domain experts to manage complex ontologies through semantic wiki interfaces.

I argue for using controlled natural languages to solve these problems. CNLs are easy to understand but can still support a high degree of expressivity. AceWiki exemplifies how this can be done.

The WikiOnt-CNL[6] system has a similar approach. It supports different CNLs (Rabbit and ACE at the moment) for verbalizing OWL axioms. In contrast to the AceWiki approach, users cannot create or edit the CNL sentences directly but only the underlying OWL axioms in a common formal notation. Furthermore, no reasoning takes place in WikiOnt-CNL.

Moreno and Bringert [112] present another approach of using CNLs in a wiki environment. They use a multi-lingual CNL framework, which allows them to translate the wiki content automatically into different languages. Thus, the CNL is not used to enable reasoning over the content of the wiki but to provide the content in different language versions in an automatic way.

The AceWiki system to be presented here, is a logic-centered semantic wiki and is unique in the sense that the user interaction is fully based on CNL and reasoning is tightly integrated.

4.4.2 Expressing Knowledge in AceWiki

As most wikis do, AceWiki structures its content in the form of articles. In contrast to common wiki systems, articles in AceWiki are written in ACE and not in uncontrolled natural language. AceWiki integrates the predictive editor that has been described in Section 4.2 in order to enable the convenient creation and modification ACE statements.

The following sections describe how articles in AceWiki look like and how the user interface distinguishes between ACE and full natural language. Then, pattern-based suggestions and export features are discussed.

[5] `http://www.swirrl.com`

[6] see [155] and `http://tw.rpi.edu/proj/cnl/`

4.4.2.1 Articles

Articles in AceWiki are closely related to words. Every word gets its own article and every article is assigned to exactly one word. The respective word is used as the title of the article page and denotes the topic of the article. ACE statements contained in the article are supposed to be related somehow to the word that is indicated by the title. However, it has no semantic relevance in which article a certain statement appears. The order of the statements also has no effect on the reasoning results.

AceWiki reuses the predictive editor of the ACE Editor to enable the easy creation and modification of statements in ACE. However, the subset of ACE used by AceWiki is slightly different from the one used by the ACE Editor. For example, modal statements using "can", "should", etc. are not included at the moment for simplicity reasons. Questions are supported, however, and they can be used to query the existing knowledge, which is explained in detail later on.

Since it cannot be expected that any problem domain of the real world can be fully formalized, AceWiki has support for comments in unrestricted natural language. Comments can be used, for example, to write down things that are too complicated to be formalized, are vague or uncertain, or concern the article itself (e.g. "this article should be improved"). If one only uses comments then AceWiki becomes a normal non-semantic wiki.

In AceWiki, words have to be defined before they can be used. At the moment, five types of words are supported: proper names, nouns, transitive verbs, *of*-constructs (i.e. nouns that have to be used with *of*-phrases), and transitive adjectives. Proper names can optionally have a shorter abbreviation with the same meaning, e.g. "EU" can be used synonymously to "European Union". Figure 4.6 shows the lexical editor of AceWiki that helps users in creating and modifying word forms in an appropriate way.

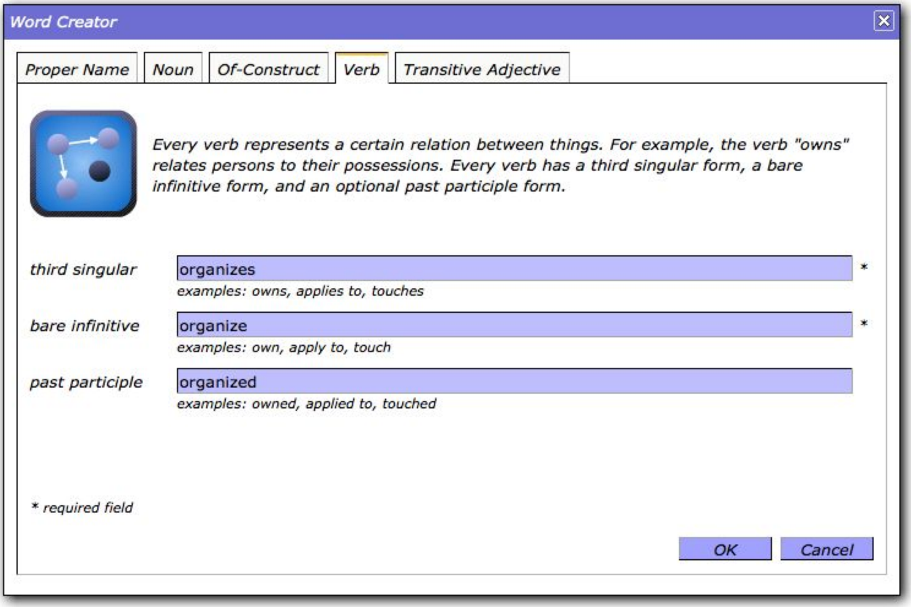
Words can be removed again from the wiki, together with their article and the statements therein. However, in order to ensure that every used word is defined somewhere, only words that are not used by a statement of a different article can be removed.

4.4.2.2 CNL and Full Natural Language

As motivated in Section 4.1, it is important to have a clear separation between formal CNL statements and text in natural language as it occurs in every user interface. Otherwise, there is the danger that the formal CNL statements are confused with the informal text.

AceWiki complies with this design principle by applying a very simple rule: Every text of the AceWiki interface that is not ACE is displayed in *italics* whereas words and sentences in ACE are shown in normal font.

This can be seen, for example, in Figure 4.5 and 4.6. ACE sentences like "no planet is a star" and ACE words like "planet" and "organizes" are displayed in normal



Every verb represents a certain relation between things. For example, the verb "owns" relates persons to their possessions. Every verb has a third singular form, a bare infinitive form, and an optional past participle form.

third singular organizes *
examples: owns, applies to, touches

bare infinitive organize *
examples: own, apply to, touch

past participle organized
examples: owned, applied to, touched

** required field*

Figure 4.6: The lexical editor of AceWiki helps users to create or modify words. This example shows how a new transitive verb "organize" is created.

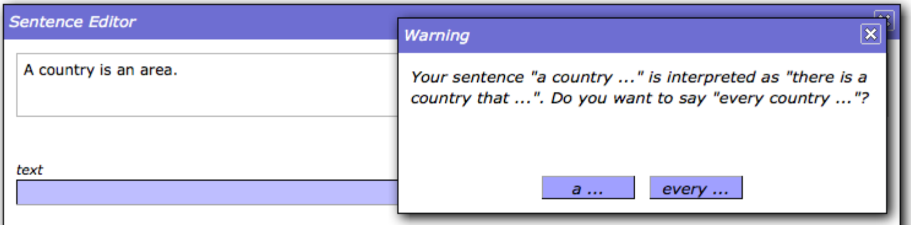
font. Comments, button and tab names, informal explanations, and window titles, however, are not ACE and they are thus typeset in italics.

In this way, users can learn that the text in normal font has to follow certain restrictions that enable the system to interpret it. This can prevent the users from confusing ACE with natural English and gives them the opportunity to learn how ACE and AceWiki work.

4.4.2.3 Pattern-based Suggestions

The two usability experiments that have been performed on AceWiki and will be explained in Section 4.4.4 showed that some common mistakes can be described by very simple patterns.

Concretely, the second experiment showed that users often create ACE sentences like "a student studies at a university", which is interpreted in ACE as having only existential quantification: "there is a student that studies at a university". However, it can be assumed that the user in this case wanted to say "every student studies at a university". The pattern that can be identified is very simple: For ACE sentences starting with "a", using "every" instead of the initial "a" is more sensible in most



Sentence Editor

A country is an area.

text

Warning

Your sentence "a country ..." is interpreted as "there is a country that ...". Do you want to say "every country ..."?

Figure 4.7: This figure shows a solution to the problem that users often state sentences starting with "a ..." when they should say "every ...".

cases.

After the second AceWiki experiment, a new feature has been added to AceWiki that asks the users each time they create a sentence of the form "a ..." whether it should be "every ...". The users can then say whether they really mean "a ..." or whether it should be rather "every ...". In the latter case the sentence is automatically corrected. Figure 4.7 shows a screenshot of the dialog presented to the users. Since this feature was not yet available for the AceWiki versions used for the two experiments, there are no empirical results so far whether this approach works out. Due to the simplicity of this pattern, I am confident that users have no difficulty to understand it and to let the system automatically correct their statements when it is appropriate.

This "a/every"-pattern is the only pattern for which automatic suggestions are implemented in AceWiki so far, because it is the only simple pattern that has been identified. However, other such simple mistake patterns might exist for which this approach could help.

4.4.2.4 Export Features

AceWiki has a client-side export feature, which allows users to export the complete knowledge base of a certain AceWiki instance. The knowledge base can be exported as a single ACE text, together with a lexicon file containing the definition of the words. Together, this can be used to parse the content of an AceWiki instance locally with the ACE parser.

Apart from that, the knowledge base can also be exported in the OWL format. In this way, the content of an AceWiki instance can be loaded in an external reasoner or ontology editor like Protégé.

4.4.3 Reasoning in AceWiki

AceWiki is designed to seamlessly integrate a reasoner that can give feedback about the consistency and the entailments of the knowledge base. For the work to be presented here, I used the OWL reasoner Pellet[7], but other reasoners can be used.

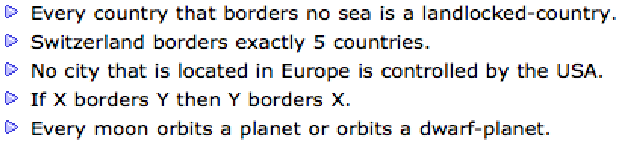
The goal of AceWiki is that users should not have to worry about how and when reasoning is performed. It should just happen. The users should not need to explicitly start the reasoning process but it should happen automatically in the background whenever required.

Below, the coverage of the reasoner is discussed. Then, it is shown how the consistency of the knowledge base is ensured and how questions can be posed. After that, it is explained how AceWiki reflects type hierarchies and assignments of individuals to types. Finally, the special issue of unique name assumption is discussed.

4.4.3.1 Reasoner Coverage

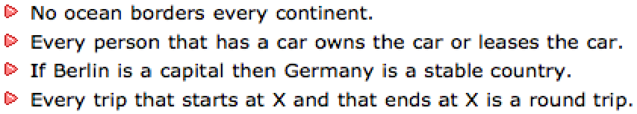
AceWiki is designed to be expressive, and users should be free to add complex statement even if the reasoner cannot handle them. In order to prevent confusion by the users, AceWiki shows for every statement whether it participates in reasoning or not.

Sentences that participate in reasoning are marked by blue triangles. Actually, most sentences that can be expressed in AceWiki can be translated into OWL and can thus be handled by the reasoner. Some examples of such sentences are shown here:



▷ Every country that borders no sea is a landlocked-country.
▷ Switzerland borders exactly 5 countries.
▷ No city that is located in Europe is controlled by the USA.
▷ If X borders Y then Y borders X.
▷ Every moon orbits a planet or orbits a dwarf-planet.

AceWiki relies on the ACE to OWL translation that has been defined and implemented by Kaarel Kaljurand [82]. Sentences that cannot be translated into OWL cannot participate in reasoning with the current reasoner. Such sentences are marked by red triangles, as the following examples show:



▷ No ocean borders every continent.
▷ Every person that has a car owns the car or leases the car.
▷ If Berlin is a capital then Germany is a stable country.
▷ Every trip that starts at X and that ends at X is a round trip.

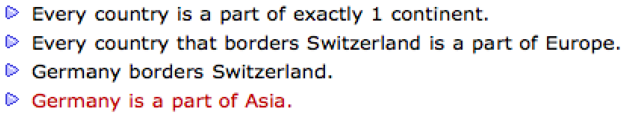
In this way, it is easy to explain to the users that only the statements with a blue triangle are considered for reasoning.

Since the complete content of the wiki can be exported, statements with red triangles can potentially be used outside of AceWiki, e.g. within a different reasoner. Thus, even though such statements cannot be interpreted by the built-in reasoner they can still be useful.

[7] see [153] and `http://clarkparsia.com/pellet/`

4.4.3.2 Consistency

Consistency checking plays a crucial role because any other reasoning task requires a consistent ontology in order to return useful results. To ensure that the ontology is always consistent, AceWiki checks every new sentence — immediately after its creation — whether it is consistent with the current ontology. Otherwise, the sentence is not included in the ontology. In the following example, the last sentence is in conflict with the existing knowledge:



▷ Every country is a part of exactly 1 continent.

▷ Every country that borders Switzerland is a part of Europe.

▷ Germany borders Switzerland.

▷ Germany is a part of Asia.

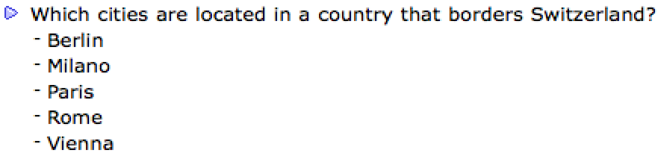
After the user created the last sentence of this example, AceWiki detected that it contradicts the current ontology. The sentence is included in the wiki article but the red font indicates that it is not included in the ontology. This sentence can be removed again by the user, or the user can keep it and try to reassert it later when the rest of the ontology has changed.

Of course, it might be an earlier sentence that is actually incorrect and not necessarily the last one. Thus, a sentence colored red can actually be correct when an incorrect sentence has been added earlier that did not introduce an inconsistency at that point. Only the users can detect and correct such mistakes. Until they do so, it is the best to keep the ontology consistent by not considering sentences that would introduce inconsistency.

As a side remark, the inconsistency of the example above is not triggered by the four sentences alone. The wiki must also contain the knowledge that Germany is a country and that Asia and Europe are continents. Otherwise, there would be no inconsistency. Furthermore, unique name assumption (to be discussed in Section 4.4.3.5) is applied that implies that Europe and Asia are different individuals.

4.4.3.3 Queries

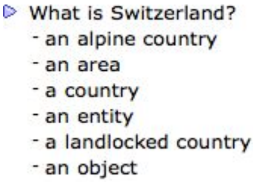
AceWiki supports queries that are formulated as ACE questions and evaluated by the reasoner. At the moment, only simple *wh*-questions are supported that contain exactly one *wh*-word, for instance:



▷ Which cities are located in a country that borders Switzerland?
- Berlin
- Milano
- Paris
- Rome
- Vienna

Such simple queries correspond to type descriptions. The individuals that can be proven to belong to the described type are the answers of such a query. Such simple queries can be directly answered by a reasoner like Pellet and do not need a separate query engine. A possible future improvement of AceWiki could be to support a more expressive query language, for example SPARQL [132].

Questions of the form "what is Switzerland?" would lead to the only answer "Switzerland" if the method described above is applied. This answer is trivial and not very useful. One would rather expect an answer like "a country" describing a type of the respective individual. For this reason, AceWiki returns in such cases all types the respective individual can be proven to belong to:



▷ What is Switzerland?
- an alpine country
- an area
- a country
- an entity
- a landlocked country
- an object

In both cases, the answers are not necessarily complete. This means that adding knowledge to the knowledge base might give additional answers to existing questions.

4.4.3.4 Assignments and Hierarchies

Apart from consistency checks and query answering, the reasoner is also used to infer the type memberships of individuals and the hierarchy of types. These kinds of reasoning results are displayed in special tabs of the AceWiki articles.

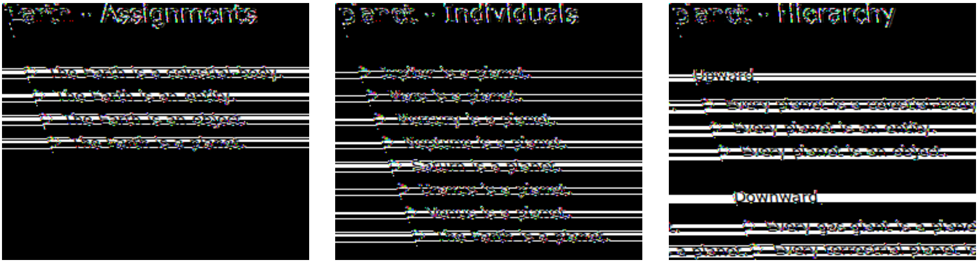
Every article about an individual (i.e. a proper name) has a tab called "Assignments" that shows all types the individual can be proven to belong to. Articles about types (i.e. nouns) have two such tabs. One is called "Individuals" and shows all individuals that can be proven to belong to the given type; the other one is called "Hierarchy" and shows all sub- and super-types of the given type.

In all cases, the results are displayed in ACE, as shown in Figure 4.8. Thus, AceWiki uses ACE not only as a language to assert knowledge and as a query language but also as a language to show inferred knowledge.

4.4.3.5 Unique Name Assumption

The term *unique name assumption* refers to the policy that individuals that are given different names are implicitly considered distinct. In logical theories, this is often not the case, i.e. two different constants a and b can refer to the same actual individual. Also in natural language, this is not generally the case. For example, the names "Bobby", "Bob Dylan" and "Robert Zimmerman" can refer to the same individual. Another example would be the country "Myanmar" that can also be called "Burma" or "Birma".

However, in a closed environment like AceWiki, there is no advantage in using different names for the same thing. Users should be forced to use the same word for



earth - Assignments

- The earth is a celestial body.
- The earth is an entity.
- The earth is an object.
- The earth is a planet.

planet - Individuals

- Jupiter is a planet.
- Mars is a planet.
- Mercury is a planet.
- Neptune is a planet.
- Saturn is a planet.
- Uranus is a planet.
- Venus is a planet.
- The earth is a planet.

planet - Hierarchy

Upward

- Every planet is a celestial body
- Every planet is an entity.
- Every planet is an object.

Downward

- Every gas giant is a plane
- a planet.
- Every terrestrial planet is

Figure 4.8: These are examples of the content of the assignments tab for individuals (left), of the individuals tab for types (middle), and of the hierarchy tab for types (right). The sentences describe knowledge that is inferred by the reasoner on the basis of the complete knowledge base.

referring to the same thing. Furthermore, it would be very annoying to be forced to explicitly state the distinctness of individuals with statements like "Switzerland is not Germany".

For these reasons, AceWiki applies the unique name assumption in the sense that individuals (i.e. proper names) that have different names are always considered distinct even if this is not explicitly stated. The only exception are abbreviations for proper names, which are semantically identical to the longer proper name they are assigned to.

4.4.4 AceWiki Experiments

At different stages of the development of AceWiki, small usability experiments have been set up to test how well normal users are able to cope with the current version of AceWiki. Two such experiments have been performed so far. Table 4.1 shows an overview of these two experiments in comparison to the case study to be presented in the next section.

The first usability experiment of AceWiki took place in November 2007. The second experiment was conducted one year later in November 2008. Both experiments had the nature of cheap ad hoc experiments with the goal to get some feedback about possible weak points of AceWiki. Since the settings of the two experiments were different and since the number of participants was relatively low, it is not possible to draw strong statistical conclusions from the results. Nevertheless, these experiments give valuable feedback about the usability of AceWiki.

In what follows, the design of the two experiments is introduced and the background of the participants is described. After that, the results of both experiments are shown and discussed.

	Experiment 1	Experiment 2	Case Study
time	Nov 2007	Nov 2008	Nov/Dec 2008
AceWiki version	0.2.1	0.2.9	0.2.10
number of participants (n)	20	6	1
participants	mostly students	students	AceWiki developer
level of preexisting knowledge about AceWiki	none	low	highest
domain to be represented	"the real world"	universities	Attempto project

Table 4.1: This table compares the settings of the three tests that have been performed on AceWiki.

4.4.4.1 Design

In both experiments, the participants were told to create a formal knowledge base in a collaborative way using AceWiki. The domain to be represented was the real world in general in the first experiment, and the domain of universities (i.e. students, departments, professors, etc.) in the second experiment. Otherwise, the participants were free in what kind of knowledge to add. The only requirement was that it should be general and verifiable knowledge.

The reason for not giving the participants a more concrete task was the difficulty of defining a knowledge representation task that is specific enough so that it can be clearly evaluated but is still general in the sense that it does not exhibit how to represent the knowledge in ACE. In order to avoid these problems, the chosen task was very general. Due to the requirement that the added knowledge has to be general and verifiable, the results can nevertheless be clearly evaluated.

In the first experiment, the participants received no instructions at all how AceWiki has to be used. In the second case, they attended a 45 minutes lesson about AceWiki.

Furthermore, the two experiments were based on different versions of AceWiki that differ significantly in some respects. In the version that has been used for the first experiment, templates could be used for the creation of certain types of sentences (e.g. class hierarchies). Templates were removed in later versions because of their lack of generality. Another difference is that there was no reasoner included in the AceWiki version used for the first experiment and users thus received no feedback in the form of reasoning results. Another difference was that the word class of transitive adjectives was not yet supported in the early version of AceWiki.

The participants of both experiments were told to spend at least half an hour on adding knowledge to AceWiki. They participated from home by accessing a dedicated AceWiki instance through a normal web browser. Within a time frame of several days, the participants could work on their task whenever they wanted. They could stop

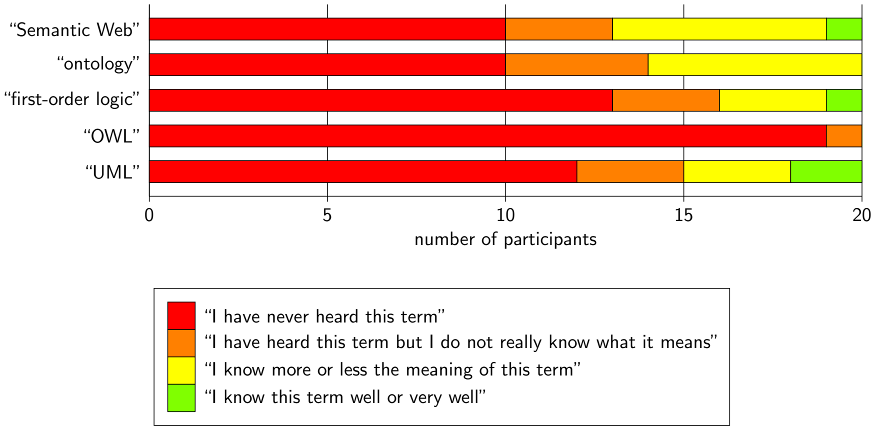


Figure 4.9: This chart shows how familiar the participants of the first AceWiki experiment were with the terms *Semantic Web*, *ontology*, *first-order logic*, *OWL* and *UML*. This data was retrieved from five questions of the questionnaire. For each of the terms, the question was "How familiar are you with this term?" with the four possible answers shown above. This chart shows that the participants had no considerable background in the field of knowledge representation and logic.

working with AceWiki at any point and could continue at a later time. Since all participants accessed the same AceWiki instance, they could see the contributions of others. They were encouraged to remove or correct statements that other participants created if they found them to be incorrect, not sufficiently general, or not verifiable. Thus, the experiments tested not only the ability of the individuals but also the achievements of the community as a whole.

4.4.4.2 Participants

The goal of the two experiments was to test AceWiki on participants with no particular background in knowledge representation.

For the first experiment, the requirements for participation were only basic English skills and access to a computer with broadband internet connection. 20 participants were recruited who met these requirements.

In order to assess the technical background of the participants, they were asked in the questionnaire how familiar they are with a couple of technical terms related to knowledge representation and logic. Figure 4.9 shows the results, which exhibit that the participants had no considerable background in these fields. For each of the terms *Semantic Web*, *ontology*, *first-order logic*, *OWL* and *UML*, at least half of the participants stated that they have never even heard the term before.

		Exp. 1		Exp. 2	
		ind.	comm.	ind.	comm.
total sentences created	S	186	179	113	93
correct sentences	S^+	148	145	76	73
correct sentences that are complex	S_x^+	91	89	54	51
sentences using “a” instead of “every”	S^e	9	9	23	12
sentences using misclassified words	S^w	9	8	0	0
other incorrect sentences	S^-	20	17	14	8
% of correct sentences	S^+/S	80%	81%	67%	78%
% of (almost) correct sentences	$(S^+ + S^e)/S$	84%	86%	88%	91%
% of complex sentences	S_x^+/S^+	61%	61%	71%	70%
total words created	w	170	170	53	50
individuals (i.e. proper names)	w_p	44	44	11	10
classes (i.e. nouns)	w_n	81	81	14	14
relations total	w_r	45	45	28	26
transitive verbs	w_v	39	39	20	18
of-constructs	w_o	6	6	2	2
transitive adjectives	w_a	–	–	6	6
sentences per word	S/w	1.09	1.05	2.13	1.86
correct sentences per word	S^+/w	0.87	0.85	1.43	1.46
total time spent (in minutes)	t	930.9	930.9	360.2	360.2
av. time per participant	t/n	46.5	46.5	60.0	60.0
av. time per correct sentence	t/S^+	6.3	6.4	4.7	4.9
av. time per (almost) correct sentence	$t/(S^+ + S^e)$	5.9	6.0	3.6	4.2

Table 4.2: This table shows the results of the first (Exp. 1) and the second (Exp. 2) experiment. The results can be seen from the individuals perspective (ind.) or the community perspective (comm.)

For the second experiment, six participants have been recruited from the course “semantic annotation of parallel corpora” hold at the Institute of Computational Linguistics of the University of Zurich. Being students in computational linguistics, they had some general background in computer science. Thus, compared to the first experiment, they had a stronger background concerning the underlying knowledge representation theories of AceWiki but they were nevertheless no experts in these fields.

4.4.4.3 Results

Table 4.2 shows the results of the two experiments. Since the participants worked together on the same knowledge base and could change or remove the contributions of others, we can look at the results from two perspectives.

On the one hand, there is the community perspective where we only consider the final result, not counting the sentences that have been removed at some point and only looking at the final versions of the sentences.

On the other hand, from the individuals perspective we also count the sentences that have been changed or removed later by another participant. The different versions of a changed sentence count for each of the respective participants. However, sentences created and then removed by the same participant are not counted, and only the last version counts for sentences that have been changed by the same participant.

Below, these results are discussed with respect to the created sentences and words, the time needed, and the feedback of the participants.

Sentences

The first part of the table shows the number and type of sentences the participants created. In total, the resulting knowledge bases contained 179 and 93 sentences, respectively. These sentences had to be checked manually for correctness. S^+ stands for the number of sentences that are (1) logically correct and (2) sensible to state.

These two criteria require some more explanation. The first criterion is quite simple: In order to be classified as correct, the sentence has to represent a correct statement about the real world using the common interpretations of the words and applying the interpretation rules of ACE.

The second criterion can be best explained on the basis of the sentences of the type S^e. Such sentences have already been discussed in Section 4.4.2.3 in the context of pattern-based suggestions. S^e sentences start with "a ..." like for example "a student studies at a university" that is interpreted in ACE as "there is a student that studies at a university". While this is a logically correct statement about the real world, the writer probably wanted to say "every student studies at a university", which is more precise and more sensible. For this reason, S^e sentences are not considered correct, even though they are correct from a purely logical point of view.

Another frequent type of error — denoted by S^w — are sentences using words in the wrong word category like for example "every London is a city" where "London" has been added as a noun instead of a proper name.

It is interesting that the incorrect sentences of the types S^e and S^w had the same frequency in the first experiment, but evolved in different directions in the second one. There was not a single case of an S^w-mistake anymore in the second experiment. This might be due to the fact that the lexical editor has been enriched with icons and explanations, based on the insights from the results of the first experiment.

On the other hand, the number of S^e-mistakes increased. This might have been caused by the removal of the templates feature from AceWiki. In the first experiment, the participants were encouraged to say "every ..." because there were templates for such sentences. In the second experiment, however, those templates were not available anymore and the participants were tempted to say "a" instead of "every". This is bad news for AceWiki, but the good news is that there are indications that the development of AceWiki is on the right track nevertheless. First, while none of the S^e-sentences has been corrected in the first experiment, almost half of them have been removed or changed by the community during the second experiment. This indicates that some participants of the second experiment recognized the problem and tried to resolve it. Second, the S^e-sentences can be detected and resolved in a very easy way as explained in Section 4.4.2.3.

Another point to consider is that the fact that the statements of S^+ are correct with respect to the real world does not necessarily mean that the participants who wrote the statements gave them the correct ACE interpretation. This understandability issue will be discussed in Chapter 5 and the presented experiment results will show that ACE is understood very well. For this reason, it can be assumed that the participants who wrote the correct statements of S^+ also meant them in the same correct way.

An interesting figure is of course the ratio of correct sentences S^+/S. As it turns out, the first experiment exhibits the better ratio for both perspectives: 80% versus 67% for the individuals and 81% versus 78% for the community. However, since S^e-sentences are easily detectable and correctable (and hopefully a solved problem with the latest version of AceWiki), it makes sense to have a look at the ratio of "(almost) correct" sentences consisting of the correct sentences $S+$ and the S^e sentences. This ratio was better in the second experiment: 84% versus 88% for the individuals perspective; 86% versus 91% for the community perspective. However, the different settings of the experiments do not allow us to draw any statistical conclusions from these numbers. Nevertheless, these results give us the impression that a ratio of correct and sensible statements of 90% and above is achievable with our approach.

Another important aspect is the complexity of the created sentences. Of course, syntactically and semantically complex statements are harder to construct than simple ones. For this reason, the correct sentences have been classified according to their complexity. S_c^+ stands for all correct sentences that are complex in the sense that they contain a negation ("no", "does not", etc.), an implication ("every", "if ... then", etc.), a disjunction ("or"), a cardinality restriction ("at most 3", etc.), or several of these elements. While the ratio of complex sentences was already very high in the first experiment (around 60%), it was was even higher in the second experiment reaching 70%.

Looking at the concrete sentences the participants created, one can see that they managed to create a broad variety of complex sentences. The following sentences are examples created during the first experiment:

Every dog is a mammal.
It is false that every animal is a mammal.
Every country is a part of a continent.
It is false that Winston-Churchill is a prime-minister of Denmark.

These examples have been created without using the templates feature of the used early version of AceWiki. Some examples of the second experiment are shown here:

> Every lecture is attended by at least 3 students.
> Every lecturer is a professor or is an assistant.
> Every professor is employed by a university.
> If X contains Y then X is larger_than Y.
> If somebody X likes Y then X does not hate Y.
> If X is a student and a professor knows X then the professor hates X or likes X or is indifferent_to X.

The last example is even too complex to be represented in the language OWL. Thus, the AceWiki user interface seems to scale very well with respect to the complexity of the ontology.

Words

The second part of Table 4.2 shows the number and types of the words that have been created during the experiment. All types of words have been used by the participants with the exception that transitive adjectives were not supported by the AceWiki version used for the first experiment. While nouns were the predominant word type in the first experiment, transitive verbs were the most frequent type in the second one.

It is interesting to see that the first experiment resulted in an ontology consisting of more words than correct sentences, whereas in the second experiment the number of correct sentences clearly exceeds the number of words. This is an indication that the words in the second experiment have been reused more often and were more interconnected.

Time

The third part of Table 4.2 takes the time dimension into account. On average, each participant of the first experiment spent 47 minutes, and each participant of the second experiment spent 60 minutes. The average time per correct sentence, which was around 6.4 minutes in the first experiment, was much better in the second experiment being only 4.9 minutes.

These time values can be considered very good results, given that the participants had no practical training and had no particular background in formal knowledge representation.

User Feedback

Finally, the feedback that the participants of the first experiment gave in the questionnaire can be evaluated. Unfortunately, there is no such data for the second experiment.

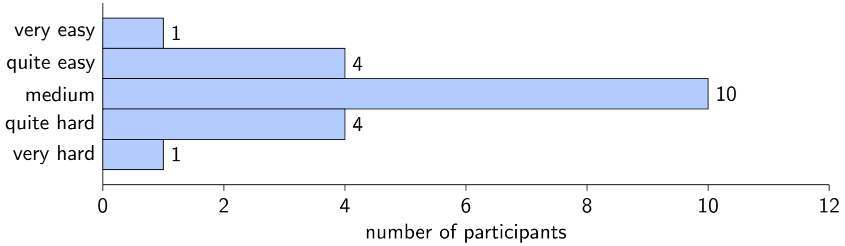


Figure 4.10: This chart shows how the participants answered the question "How easy or how difficult was the handling of AceWiki?" after the first experiment.

The questionnaire asked how easy or how difficult the handling of AceWiki was perceived by the participants. Figure 4.10 shows the result. The responses are distributed symmetrically and have a peak at "medium".

On the one hand, this is a good result since only 25% of the users found it hard to use AceWiki. It has to be considered that the participants only experienced the costs of formal knowledge representation, but not the benefits (since reasoning features were missing in the version of AceWiki that was used for the first experiment). Furthermore, one can argue that knowledge representation is inherently a difficult task that can probably never be made very easy for everybody.

On the other hand, the results show that there is certainly room for improvement. Since the time the first experiment was conducted, the AceWiki interface has undergone various improvements concerning usability. For this reason, it can be assumed that the current version would score better in this respect.

4.4.5 AceWiki Case Study

The two presented usability experiments seem to confirm that AceWiki can be used conveniently by untrained persons. However, usability does not imply the usefulness for a particular purpose. For this reason, I performed a small case study in the form of a self-experiment to exemplify how an experienced user like myself can represent a strictly defined part of real world knowledge in AceWiki in a useful way.

The rest of this section describes the design of the case study and discusses the results.

4.4.5.1 Design

The case study presented here consists of the formalization of the content of the Attempto website[8] in AceWiki. This website contains information about the Attempto

[8] `http://attempto.ifi.uzh.ch`

total sentences created	S	538
complex sentences	S_x	107
% of complex sentences	S_x/S	19.9%
total words created	w	261
individuals (i.e. proper names)	w_p	184
classes (i.e. nouns)	w_n	46
relations total	w_r	31
transitive verbs	w_v	11
of-constructs	w_o	13
transitive adjectives	w_a	7
sentences per word	S/w	2.061
time spent (in minutes)	t	347.8 (= 5.8 h)
av. time per sentence	t/S	0.65 (= 38.8 s)

Table 4.3: This table lists the results of the AceWiki case study that was about formalizing the content of the Attempto website in AceWiki.

project and its members, collaborators, documents, tools, languages and publications, and the relations among these entities. Thus, the information provided by the Attempto website is a piece of relevant real world knowledge.

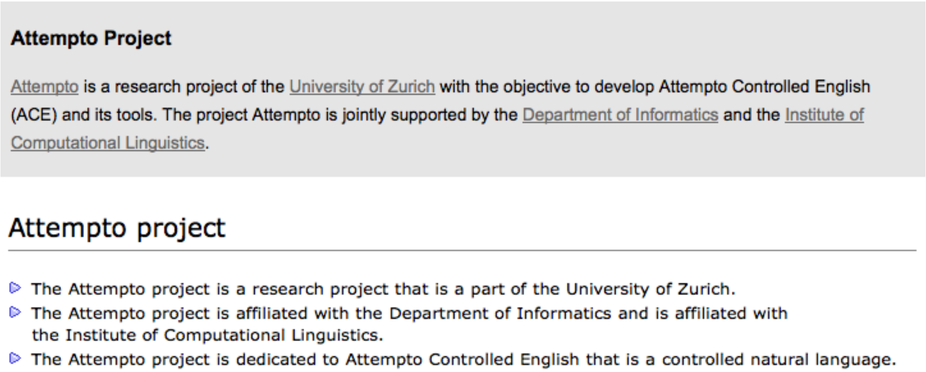
In the case study to be presented, I used a plain AceWiki instance and filled it with the information found on the public Attempto website. The goal was to represent as much as possible of the original information in a natural and adequate way. This was done manually using the predictive editor of AceWiki without any kind of automation.

4.4.5.2 Results

Table 4.3 shows the results of the case study. The formalization of the website content took less than six hours and resulted in 538 sentences. This gives an average time per sentence of less than 40 seconds. These results give us some indication that AceWiki is not only usable for novice users but can also be used in an efficient way by experienced users.

Most of the created words are proper names (i.e. individuals), which is not surprising for the formalization of a project website. The ratio of complex sentences is much lower than the ones encountered in the experiments but with almost 20% still on a considerable level.

Basically, all relevant content of the Attempto website could be represented in AceWiki. Of course, the text could not be taken over verbatim but had to be rephrased. Figure 4.11 exemplarily shows how the content of the website was formalized. The resulting ACE sentences are natural and understandable.



Attempto Project

Attempto is a research project of the University of Zurich with the objective to develop Attempto Controlled English (ACE) and its tools. The project Attempto is jointly supported by the Department of Informatics and the Institute of Computational Linguistics.

Attempto project

▷ The Attempto project is a research project that is a part of the University of Zurich.
▷ The Attempto project is affiliated with the Department of Informatics and is affiliated with the Institute of Computational Linguistics.
▷ The Attempto project is dedicated to Attempto Controlled English that is a controlled natural language.

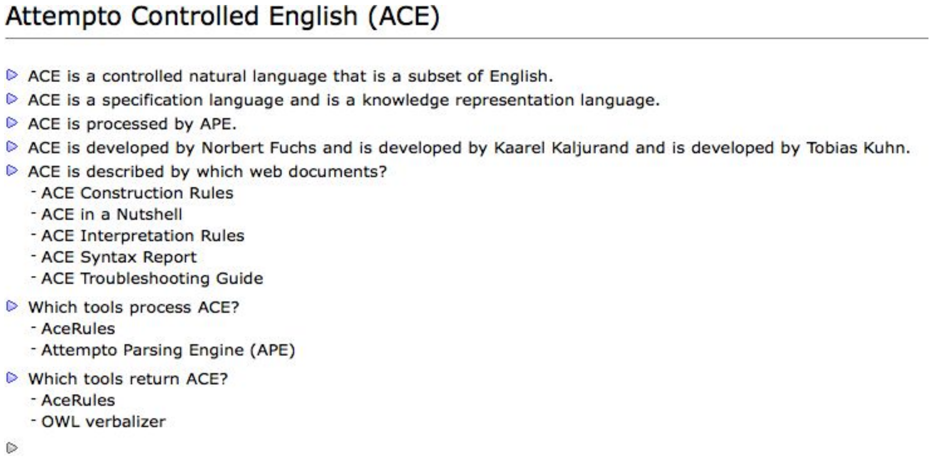
Figure 4.11: This figure shows, as an example, a text that occurs on the Attempto website (top) and how it was represented in AceWiki (bottom).

However, some minor problems were encountered. Data types like strings, numbers and dates would have been helpful but are not supported. ACE itself has support for strings and numbers, but AceWiki does not make use of it so far. Another problem was that the words in AceWiki can consist only of letters, numbers, hyphens, and blank spaces (the latter are internally represented as underscores). Some things like publication titles or package names contain colons or dots which had to be replaced by hyphens in the AceWiki representation. These problems could be solved in the future by adding support for data types to AceWiki and being more flexible with respect to user-defined words.

Figure 4.12 shows an example of a wiki article that resulted from the case study. It illustrates how queries can be used for content that is automatically generated and updated. This is an important advantage of such semantic wiki systems. The knowledge has to be asserted once but can be displayed in different places. In the case of AceWiki, such automatically created content is well separated from asserted content in a natural and simple manner by using ACE questions.

The abbreviation feature for proper names has been used extensively, e.g. to define that "ACE" is the abbreviation of "Attempto Controlled English" as shown on Figure 4.12. This abbreviation feature was even more important to describe publications, which can be identified by their titles. However, sentences containing spelled-out publication titles become very hard to read. In such cases, abbreviations have been defined, which can be used conveniently to refer to the publications.

Altogether, the presented case study seems to confirm that AceWiki is usable and useful. However, the fact that the AceWiki developer is able to use AceWiki in an efficient way for representing real world knowledge does of course *not* imply



Attempto Controlled English (ACE)

▷ ACE is a controlled natural language that is a subset of English.
▷ ACE is a specification language and is a knowledge representation language.
▷ ACE is processed by APE.
▷ ACE is developed by Norbert Fuchs and is developed by Kaarel Kaljurand and is developed by Tobias Kuhn.
▷ ACE is described by which web documents?
- ACE Construction Rules
- ACE in a Nutshell
- ACE Interpretation Rules
- ACE Syntax Report
- ACE Troubleshooting Guide

▷ Which tools process ACE?
- AceRules
- Attempto Parsing Engine (APE)

▷ Which tools return ACE?
- AceRules
- OWL verbalizer

▷

Figure 4.12: This figure shows an exemplary wiki article (about the language ACE) that resulted from the case study. Queries are used to generate content that is automatically updated.

that every experienced user is able to do so. However, the results can be seen as an upper boundary of what is possible to achieve with AceWiki, and the results show that AceWiki can *in principle* be used in an effective way to represent real world knowledge.

4.5 Concluding Remarks on CNL Tools

The three tools ACE Editor, AceRules and AceWiki exemplify how CNLs can make intuitive and powerful knowledge representation systems by applying the identified design principles. The presented studies show that such tools can be used by non-specialists after little or no training.

Some essential aspects, however, remain unexplored with these studies. First of all, it is not certain that the ACE statements are actually always understood in the way defined by the ACE interpretation rules. Furthermore, there is no evidence so far that languages like ACE are indeed easier to understand than other languages. These understandability issues will be tackled in the next chapter that shows how the understandability of CNLs and other formal languages can be evaluated and compared.

CHAPTER 5

Understandability

This chapter takes a closer look at the topic of CNL understandability. The tools and the evaluation results of the experiments presented in the previous chapter show that CNLs are suitable for providing usable interfaces to knowledge representation systems. However, it is not completely clear at this point whether the ACE statements are understood correctly. This leads us to the third and fourth research question of this thesis as defined in the introduction:

> **3. How can the understandability of controlled English be evaluated?**
>
> **4. Is controlled English indeed easier to understand than other formal languages?**

Controlled natural languages are claimed to be easier to learn and understand than other formal languages and user studies are the only way to verify this claim. However, it is not trivial to set up such studies in a proper way. The results of the experiments presented in the previous chapter do not imply that the participants actually understood all statements they wrote.

So far, there is no agreed methodology on how the understandability of CNLs should be evaluated. Furthermore, there is no solid evidence that CNLs are indeed easier to understand than comparable common formal languages.

Below, this chapter presents the state of the art in CNL evaluation and describes some problems with existing approaches (Section 5.1). Then, I will introduce a general framework for human subject experiments to test the understandability of CNLs, which solves the identified problems of existing approaches (Section 5.2). Afterwards, the results of two experiments are described that have been performed to test and compare the understandability ACE (Sections 5.3 and 5.4). These experiments also give us evidence whether the ontograph approach altogether worked out (Section 5.5). Finally, the conclusions will be summarized (Section 5.6), limitations of ontographs are discussed, and some possible other application areas are sketched (Section 5.7).

5.1 Existing CNL Evaluation Approaches

Many different user experiments have been conducted in the past to evaluate different kinds of CNLs. Two general types of approaches can be identified: task-based and paraphrase-based approaches. I will argue that it is difficult to get reliable results concerning the understandability of CNLs with either approach.

5.1.1 Task-based CNL Experiments

Several user experiments have been conducted where a specific task was given to the participants who should accomplish the task by entering CNL statements into a given tool. Examples of such task-based experiments are the AceWiki experiments presented in Section 4.4.4, having a very general task. Other such experiments have been described by Bernstein and Kaufmann [12] and by Funk et al. [57, 56]. In all cases, the participants received tasks to add certain knowledge to the knowledge base using a tool that is based on CNL. An example taken from [57] is the task

> Create a subclass *Journal* of *Periodical*.

for which the participants are expected to write a CNL statement in the form of "Journals are a type of Periodicals". To evaluate whether the task was accomplished, the resulting knowledge base can be checked whether it contains this actual piece of information or not. This approach bears some problems if used to test the understandability of a language.

First of all, such experiments mainly test the ability to write statements in the given CNL and not the ability to understand them. A user succeeding in the task shown above does not necessarily understand what the statement means. In order to add "Journal" as a subclass of "Periodical", the user only has to map "subclass" and "type of", but does not have to understand these terms.

Another problem is that it is hard to determine with such experiments how much the CNL contributes to the general usability and understandability, and how much is due to other aspects of the tool. It is also hard to compare CNLs to other formal languages with such studies, because different languages often require different tools.

For these reasons, it would be desirable to be able to test the understandability of CNLs in a tool-independent way, which is not possible with task-based approaches.

Due to the writability problem of CNLs (see Section 2.1.4), however, tools are needed to test the writability of CNLs, and task-based approaches seem to be a good solution for this.

Task-based evaluation approaches can be complemented by measuring the subjective usability, for example by applying the popular SUS method [23]. However, such subjective usability scores are — as the inventor of SUS admits — a "quick and dirty" method that does not allow for direct conclusions on comprehension of the concepts involved. For this reason, such usability scores are not suitable for reliably measuring the understandability of CNLs either.

Inglesant et al. [75] introduce an approach that is related to what I will present in this chapter by using graphical representations. They performed a task-based evaluation of their CNL tool where the participants had to formalize a scenario that was described as a textual listing and at the same time as a diagram. As it turned out, however, the participants mostly ignored the diagram and only relied on the textual description.

5.1.2 Paraphrase-based CNL Experiments

Paraphrase-based approaches are a way how CNLs can be tested in a tool-independent manner. In contrast to task-based approaches, they aim to evaluate the comprehensibility of a CNL rather than the usability of tools using CNL.

Hart et al. [69] present such an approach to test their CNL (i.e. the Rabbit language). The authors conducted an experiment where the participants were given one Rabbit statement at a time and had to choose from four paraphrases in natural English, only one of which was correct. The authors give the following example of a Rabbit statement and four options:

Statement: Bob is an instance of an acornfly.

Option 1: Bob is a unique thing that is classified as an acornfly.

Option 2: Bob is sometimes an acornfly.

Option 3: All Bobs are types of acornflies.

Option 4: All acornflies are examples of Bob.

They used artificial words like "acornfly" in order to prevent that the participants classify the statement on the basis of their own background knowledge. Option 1 would be the correct solution in this case. Similar experiments are described by Hallett et al. [65] and Chervak et al. [27]. Again, there are some problems with such approaches.

First of all, since natural language is highly ambiguous, it has to be ensured somehow that the participants understand the natural language paraphrases in the way they are intended, which just takes the same problem to the next level. For the example above, one has to make sure that the participants understand the phrases "is classified as", "are types of" and "are examples of" in the correct way. The problem is even more complicated with words like "unique", "sometimes" and "all". If one cannot be sure that the participants understand the paraphrases then the results do not permit any conclusions about the understandability of the tested language.

Furthermore, since the formal statement and the paraphrases look very similar in many cases (both rely on English), it is yet again hard to determine whether understanding is actually necessary to fulfill the task. The participants might do the right thing without understanding the sentences (e.g. just by following some syntactic patterns), or by misunderstanding both — statement and paraphrase — in the same way.

For the example above, a participant might just think that "an instance of" sounds like having the same meaning as "a unique thing that is classified as" without understanding any of the two. Such a person would be able to perform very well on the task above. In this case, the good performance would imply nothing about the understandability of the tested language.

Nevertheless, paraphrase-based approaches also have their advantages. One of them is that they scale very well with respect to the expressivity of the language to be tested. Basically, CNLs built upon any kind of formal logic can in principle be tested within such experiments, once the problems identified above are solved in one way or another.

5.2 The Ontograph Framework

Since task-based and paraphrase-based approaches have serious problems if used for testing the understandability of CNLs, I will introduce a new approach.

I propose an approach that is based on diagrams and solves the discussed problems of existing approaches. My approach relies on a graphical notation that I developed and that I call *ontographs* (as a contraction of "ontology graphs"). This notation is designed to be very simple and intuitive. The basic idea is to describe simple situations in this graphical notation so that these situation descriptions can be used in human subject experiments as a common basis to test the understandability of different formal languages. This approach allows for designing reliable understandability experiments that are completely tool-independent.

In this section, the elements of the ontograph notation are introduced, some characteristic properties of this notation are described, it is argued that these properties make ontographs intuitively understandable, it is shown how experiments can be designed on the basis of ontographs, and, finally, some related approaches are described.

5.2.1 The Ontograph Notation

Every ontograph diagram consists of a legend that introduces types and relations and of a mini world that describes the actual individuals, their types, and their relations.

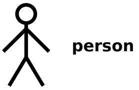
As a side remark, the distinction between legend and mini world roughly corresponds to the distinction between TBox and ABox [116] that is used in Description Logics and in other knowledge representation theories.

The ontograph elements of the legend and of the mini world are now to be described, and it is shown how these elements are compiled into a complete ontograph.

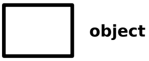
Legend

The legend of an ontograph introduces the graphical representations of types and relations.

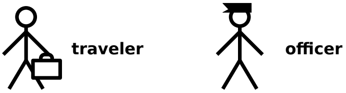
Types are introduced by showing their name beside the symbol that represents the respective type. For example, a type "person" would be introduced as follows:



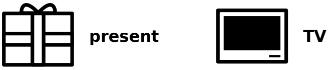
Another type "object" can be introduced as follows:



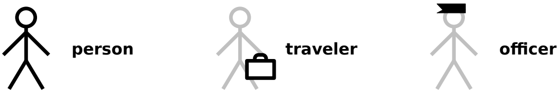
Starting from such general types, more specific ones can be defined. For example, "traveler" and "officer" can be defined as specific types of the general type "person":



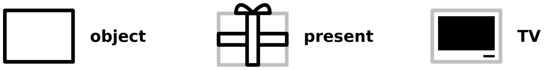
There can also be specific types for the general type "object", for example "present" and "TV":



If a legend contains a type like "person" and, at the same time, a specific type like "traveler" then the part of the symbol of the specific type that is copied from the general type is displayed in gray:

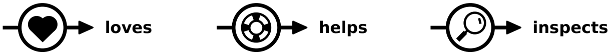


The same is done for objects:



The purpose of this is to specify that only the black part of the symbol represents the respective type. This becomes important for individuals that belong to several types. For example, the suitcase is the deciding criterion in the case of the “traveler” definition (and not e.g. the missing hat).

Relations are represented by circles that contain a specific symbol and an arrow going through this circle. As with types, the legend introduces the names of the relations by showing them on the right hand side of the graphical representation. Some examples are the relations “loves”, “helps” and “inspects”:

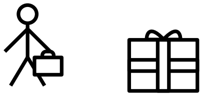


In contrast to types, there is no way to create specific relations based on more general ones.

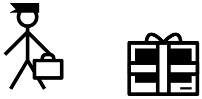
Mini World

In contrast to the legend that only introduces vocabulary and the respective graphical representations, the mini world describes actual situations. Such situations consist of individuals, the types of the individuals, and the actual relations between them.

Every individual is represented by exactly one symbol. These symbols denote the types of the individuals, as introduced by the legend. For example, an individual that is a traveler and another individual that is a present are represented by a traveler symbol and a present symbol:

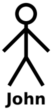


If an individual belongs to several types then it is represented by a combined symbol that is obtained by merging the respective symbols. For example

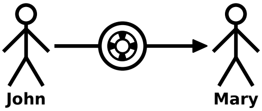


represents an individual that is a traveler and an officer and another individual that is a present and a TV. The ontograph notation requires that every individual belongs to at least one type.

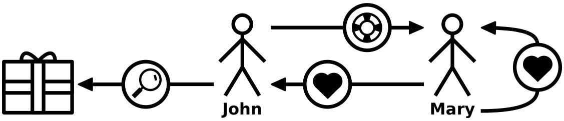
Individuals can optionally have a name that is shown in the mini world below the respective symbol:



Relation instances are represented by arrows that point from one individual to another (or the same) individual and that have a relation symbol (as defined by the legend) somewhere in the middle. "John helps Mary", for example, would be represented as follows:



Not only persons but also objects can participate in relations and, of course, individuals can participate in several relations at the same time:



There is no explicit notation for negation. The fact that something is not the case is represented implicitly by not saying that it is the case. For example, stating that "John does not help Mary" is done by *not* drawing a *help*-relation from John to Mary. Thus, mini worlds are closed in the sense that everything that is true is shown and everything that is not shown is false.

Complete Ontographs

Mini world and legend are compiled into a complete ontograph. The mini world is represented by a large square surrounded by a thick line and labeled with "mini world". This square contains the mini world elements described above. The legend is represented by a smaller upright rectangle to the right of the mini world with the same height. The area of the legend is also surrounded by a thick line, is labeled with "legend", and contains the legend elements as introduced above. These elements are arranged one below the other in a way that type definitions come before relation definitions. Figure 5.1 shows an example. More examples can be found in Appendix B.

5.2.2 Properties of the Ontograph Notation

The ontograph notation has some important characteristic properties, which have to be discussed.

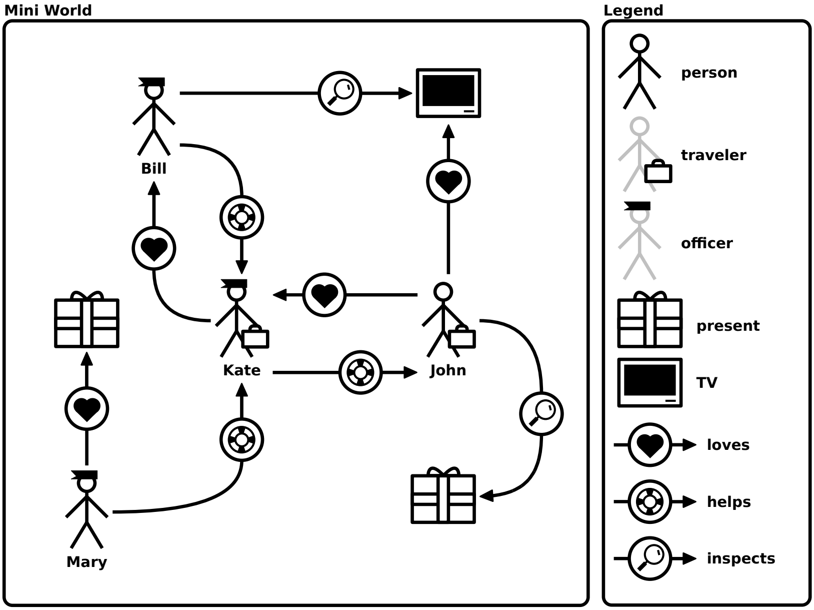


Figure 5.1: This is an example of an ontograph. The legend on the right hand side defines the types and relations. The mini world on the left hand side shows the actual individuals, their types, and the relations between them.

First of all, the ontograph notation does not allow for expressing incomplete knowledge. This means that nothing can be left unspecified and that every statement about the mini world is either necessarily true or necessarily false. For example, one can express "John helps Mary", or one can also state "John does not help Mary", but one cannot express that it is unknown whether one or the other is the case. Most other logic languages (e.g. standard first-order logic) do not behave this way. For the ontograph notation, this has the positive effect that no explicit notation for negation is needed. Everything that is not shown in the ontograph is simply considered false.

Another important property of the ontograph notation is that it has no generalization capabilities. Logically speaking, the ontograph notation has no support for any kind of quantification over the individuals. For example, one cannot express something like "every man loves Mary" in a general way. The only way to express this is to draw a *love*-relation from every individual that is a man to the individual Mary. Thus, every individual and every relation instance has to be represented individually. This has the apparent consequence that the ontograph notation cannot be used to describe situations with an infinite number of individuals and becomes impractical with something around 50 or more individuals and relation instances.

These properties make the ontograph notation a very simple language. They also have the consequence that the ontograph notation is no candidate for becoming a knowledge representation language of its own. A knowledge representation language without support for partial knowledge and generalization would not be very useful.

5.2.3 The Understandability of Ontographs

Ontographs are required to be very intuitive and easy to understand. Several properties of the ontograph notation are indications of its intuitive understandability.

First of all, it can be argued that the use of simple and intuitive graphical icons contributes to the understandability of the ontograph notation. The used icons and the meaning of the arrows are easily identifiable.

Probably more important, however, is the fact that the ontograph notation has no generalization capabilities and does not support partial knowledge. This excludes many potential misunderstandings. An ontograph explicitly shows every existing individual and depicts every single relation instance between them. There is not much to misunderstand with such basic elements.

These arguments are of course not sufficient to prove that the ontograph notation is very easy to understand for the participants of an experiment. The results of two experiments that have been performed will be described in Section 5.5, and they will give us some solid evidence that the ontograph notation is indeed easy to understand.

5.2.4 Ontograph Experiments

Ontographs are designed to be used in experiments testing the understandability of formal languages. They could, in principle, also be used to test the writability of languages by asking the participants to describe given situations. However, the latter approach has not yet been investigated.

In order to test the understandability of a language, an ontograph and several statements (written in the language to be tested) are shown to the participants of an experiment, who have to decide which of the statements are true and which are false with respect to the situation depicted by the ontograph.

An important property of ontographs is that they use a graphical notation that is syntactically very different from textual languages like CNLs. This makes it virtually impossible to distinguish true and false statements of a given textual language with respect to a given ontograph just by looking at the syntax. If participants manage to systematically classify the statements correctly as true or false then it can be concluded with a high degree of certainty that the participants understood the statements and the ontograph.

This point gets even clearer by applying a direct connection to model theory [26]. From a model-theoretic point of view, one can say that ontographs are a language to describe first-order models. The statements that are shown to the participants of an experiment would then be very simple first-order theories. From this point of view, the task of the participants is to decide whether certain theories have the shown ontograph as a model or not. We can say that participants understand a statement if they can correctly and systematically classify different ontographs as being a model of the statement or as not being a model thereof. This conclusion can be drawn because the truth of first-order theories can be defined solely on the basis of their models. Thus, being able to identify the ontographs that are models of a theory represented by a statement means that the respective person understands the statement correctly.

Admittedly, this model-theoretic interpretation of the term "understanding" is relatively narrow and ignores important problems like symbol grounding [68]. Such kinds of problems are not covered by the ontograph approach. Nevertheless, the ontograph framework allows us to draw stronger conclusions about the understandability of a language than other existing approaches.

5.2.5 Related Approaches

Different graphical notations have been defined to represent logical statements, for example Peirce's *existential graphs* [123] and Sowa's *conceptual graphs* [156]. However, such languages are fundamentally different from ontographs in the sense that they aim at representing general logical statement and in the sense that they are not designed to be intuitively understandable but have to be learned.

The combination of intuitive pictures and statements in natural language can also be found in books for language learners, e.g. "English through pictures" [138]. As in the ontograph framework, pictures are used as a language that is understood without explanation.

The idea of "textual model checking" presented by Bos [19] is similar to the ontograph approach in some respects. Like in the ontograph approach, there is the task of classifying given statements as true or false with respect to a given situation. In contrast to the approach presented here, the task is to be performed by computer programs and not by humans, and it is about testing these computer programs rather than the language.

5.3 First Ontograph Experiment

So far, two experiments have been performed using the ontograph framework. The first one was a smaller experiment that only tested ACE without another language for comparison. The goal was to get some first experiences with the ontograph framework and at the same time to get some results about the understandability of ACE and to find possible weak points.

Since the number of participants was considerably lower than in the second experiment and since this first experiment was conducted in a less controlled environment, the results cannot be considered as reliable as the ones from the second experiment. For this reason, this section describes the design and the results of the first ontograph experiment in a relatively brief way.

5.3.1 Design of the first Ontograph Experiment

For the first experiment, 15 participants were recruited who were mostly students and not experts in knowledge representation. They participated in the experiment from home by using a common web browser to connect to our server.

Four ontographs have been used for this experiment. In order to distinguish them from the ontographs used in the second experiment, they will be called 1X, 2X, 3X and 4X. They are shown in Appendix B.1 together with 20 ACE statements each. These statements are subdivided into two series a and b whereas only series a has been used for this experiment and series b is unused so far.

The statements of both series are numbered from 1 to 10. In this way, every statement gets an identifier that consists of its number followed by either "a" or "b" (depending on the series) and postfixed by "+" if the statement is true with respect to its ontograph or "−" if it is false. The statements of the different ontographs can be uniquely referenced with identifiers of the form "ontograph/statement". For example, the identifier 2X/5a+ stands for the fifth statement of series a of ontograph 2X and exhibits that this statement is true with respect to its ontograph.

The ACE sentences are chosen so that they cover a broad variety of semantic structures. All sentences would be expressible in the language OWL and would cover most of the axiom types provided by the OWL standard. The first ontograph 1X only contains individuals and types but no relations. Relations are introduced in ontograph 2X. The statements of the third ontograph 3X use more complicated structures like domain, range, and cardinality restrictions. The statements of ontograph 4X, finally, are only about relations. In this way, a subset of ACE is covered that corresponds to a subset of first-order logic that is similar to the one used by OWL.

These ontograph series contain three special statements to test how the participants perform on logical borderline cases, where human intuition can be misleading. 1X/3a+ is such a borderline statement. It is a very simple statement using "or" where both conjuncts are true. Since ACE defines that "or" is always interpreted inclusively, the statement is true, but it can be expected that this is confusing for the participants. Another borderline statement is 1X/10a+ that is a conditional statement with a false precondition. While such statements are considered logically true no matter what the postcondition is, this can also be confusing for the participants. 2X/7a+, finally, is the third borderline statement. It contains "nothing but ..." in a situation where "nothing" would be correct as well. ACE defines "nothing but ..." to include "nothing at all", which could be another cause for confusion. For these three borderline statements, it can be expected that the participants perform worse than for the other statements.

The experiment was conducted in a way that one ontograph after the other was shown to the participants together with the ten statements of series a. For each ontograph, the participants had at most five minutes to classify each of the statements as true, false, or "don't know". Thus, the participants could spend on average at most 30 seconds per statement.

There was no explanation on how the ACE statements have to be interpreted, with the one exception that "something" can stand for persons and objects. After the experiment, the participants filled out a very short questionnaire about their experiences during the experiment.

5.3.2 Results of the first Ontograph Experiment

The first ontograph experiment can be evaluated on the basis of the general classification scores and the amount of time needed. Additionally, an overview is given of how the individual statements have been classified.

5.3.2.1 General Classification Scores

Figure 5.2 shows how well the participants performed on the classification task. This figure shows the percentages of correct classifications whereas "don't know" answers and cases where the time limit run out count as 0.5 correct answers. Overall 83% of the statements have been classified correctly. This figure has to be compared to 50%, which can be achieved without understanding (by mere guessing, by choosing always "don't know", or by letting the time limit run out).

If the borderline statements described above (i.e. 1X/3a+, 1X/10a+ and 2X/7a+) are not considered then the percentage of correct classifications reaches almost 88%. Considering all statements, ontograph 3X was understood best with almost 90% correct classifications whereas the other three ontographs have scores between 78% and 84%. In the case of the ontographs 1X and 2X, however, these numbers are much higher when the borderline statements are disregarded: 95.8% and 88.6%, respectively.

These results indicate that the ACE statements were generally well understood. As expected, the borderline statements had a negative influence on the classification score.

5.3.2.2 Time

Apart from the classification scores, it is also interesting to see how much time was needed for this classification task. The participants could spend at most 5 minutes per ontograph but they could proceed before this time limit ran out. In this way, the actual time the participants needed could be retrieved from the log files of the server. Figure 5.3 shows these time values.

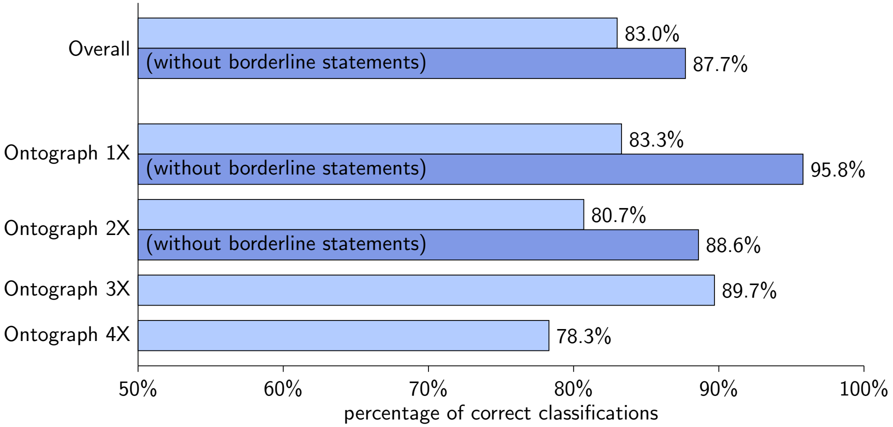


Figure 5.2: This chart shows the percentage of correct classifications of the ACE statements of the first ontograph experiment. The base line is 50% that can be achieved by mere guessing. “don’t know” classifications and cases where the time limit ran out count as 0.5 correct classifications. Series 1 and 2 contain the borderline statements 1X/3a+, 1X/10a+ and 2X/7a+, which were the main reason for incorrect classifications. The scores obtained by ignoring these borderline statements are shown separately.

From the 5 minutes the participants could spend for each ontograph, they needed on average only 3.2 minutes. Thus, less than 20 seconds were needed per statement. Together with the results on the classification scores, this shows that the participants were able to understand the ACE statements very quickly.

The time values for the different ontographs show that 4X required most time followed by 3X. This can be explained by the fact that these ontographs were more complicated than the other two. The fact that the simplest ontograph 1X required more time than 2X can be explained by the fact that 1X was the first ontograph presented to the participants, and thus they were not yet familiar with this notation and the procedure.

5.3.2.3 Individual Statements

Finally, it is interesting to look at how the individual statements have been classified. Figure 5.4 shows the results for the individual statements. They confirm that the borderline statements were often incorrectly classified.

The borderline statement 1X/3a+ (the simple statement using “or” where both conjuncts are true) is true according to the ACE semantics but was often classified as false. One might assume that many participants misinterpreted the “or” as being exclusive (instead of inclusive as ACE defines it). However, if that were the case then

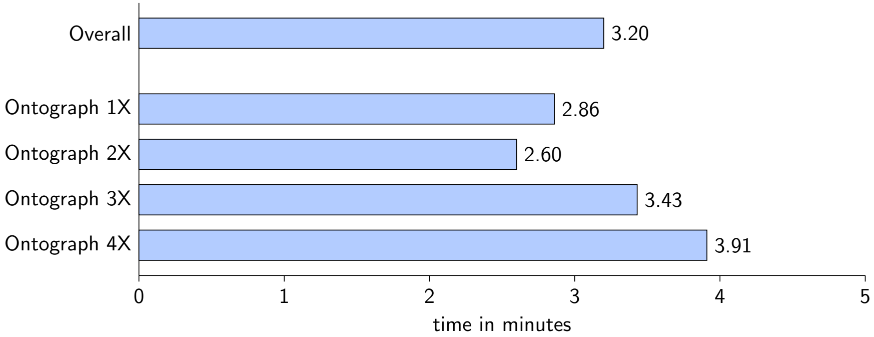


Figure 5.3: This chart shows the average time needed for the classification of the statements of the different ontographs by the participants of the first ontograph experiment. 5 minutes was the time limit.

the participants should also interpret the "or" of statement 1X/9a+ as exclusive, but they did not. A more plausible explanation is that the participants recognized that the statement 1X/3a+ is imprecise in the sense that using "and" instead of "or" would be more accurate. This prevented some of the participants from realizing that the statement is nevertheless true in a logical sense. The situation is different with statement 1X/9a+ where the replacement of "or" by "and" would not be more accurate but would make the true statement false. As a result, 1X/9a+ was classified correctly by almost all participants. It seems that people in such cases often fail to distinguish accuracy from logical truth.

1X/10a+ is the second borderline statement (the conditional statement with a false precondition). It is not surprising that people with no background in logics fail to classify this statement in a correct way.

The third borderline statement is 2X/7a+ (the statement containing "nothing but ..." that would also be correct if "nothing" was used). This is a common mistake that can also be encountered in other languages like OWL: people tend to think that "nothing but ..." implies "at least 1 ..." [136]. It is interesting to see that in this case the more general statement 2X/9a+ (containing "nothing but ..." in a general context) was also classified incorrectly by most of the participants, in contrast to the statements 1X/3a+ and 1X/9a+ where only the specific one was misunderstood. Thus, it seems that the participants really misunderstood "nothing but" and that it was not just because they were mixing up accuracy and logical truth.

I assume that the misunderstanding of such borderline statements is not specific to ACE, but can be encountered in any other language.

The statement 3X/6a+ also exhibits a predominance of the wrong classifications over correct ones. In this case, however, no clear reason can be identified.

Apart from the discussed cases, the correct decision was chosen at least twice

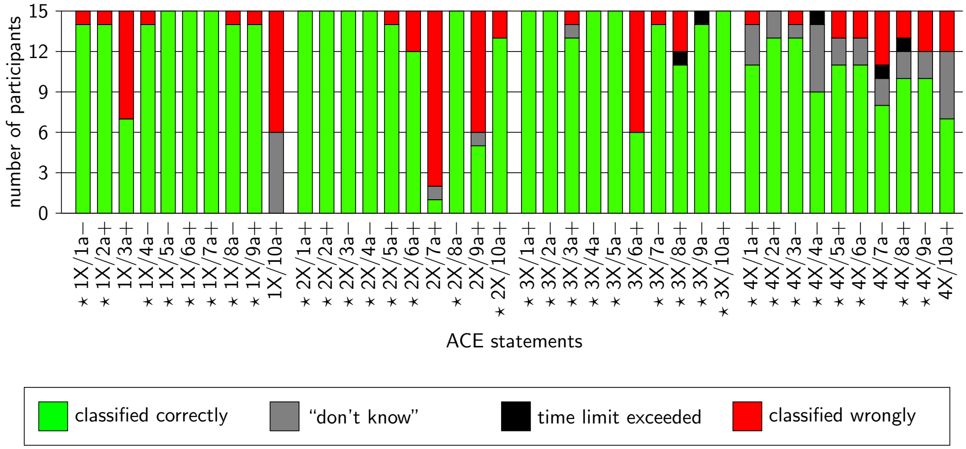


Figure 5.4: This chart shows for each statement how many participants classified it correctly, classified it wrongly, said that they don't know, or exceeded the time limit. The predominance of the correct classification over the wrong one is statistically significant on a 95% confidence level for all statements that are marked with "⋆". Appendix B.1 shows the concrete ontographs and statements.

as often than the wrong one. This preference is statistically significant on a 95% confidence level for all of them except for 4X/7a− and 4X/10a+, applying a simple binomial test with the null hypothesis being a random 50% decision.

As can be seen on Figure 5.4, "don't know" answers were quite frequent for the statements of ontograph 4X. This can be explained by the fact that the statements of this ontograph make use of variables. Some of the participants seem to feel uncomfortable with them. Still, the statements of ontograph 4X were also understood reasonably well.

Another interesting point is that the participants were not told that the ontograph notation reflects the complete information about the mini world and that everything that is not shown in the ontograph can be considered false. The result for statement 2X/2a+, for example, shows that the participants understood this very well without any explanation. They understood that the fact that the ontograph does not show a *see*-relation from Mary to Tom means that "Mary does not see Tom" is a true statement.

5.4 Second Ontograph Experiment

Building upon the experiences of the first experiment, a second experiment was performed that compares ACE to a comparable common formal language. This experiment was more thorough than the first one in the sense that the number of participants was much higher (64 participants) and they acted in a controlled environment, instead of participating from home as in the first experiment. This experiment should give us statistical evidence on whether ACE is indeed easier to understand than a common formal language.

In what follows, the design and the results of the second ontograph experiment are described.

5.4.1 Design of the second Ontograph Experiment

This second ontograph experiment has been designed very carefully in order to get strong statistical results on the question whether ACE is easier to understand than other formal languages.

In the following sections, important aspects of the design are described, namely the choice of the language to compare ACE against, the amount of time the participants should get for learning the languages, the recruitment criteria, the grouping of the participants according to the different tasks, the overall experiment procedure, the design of the language description sheets, and the payout. Furthermore, the results of a test run are described that was performed before the main experiment.

5.4.1.1 Comparable Language

The most important design decision is the choice of the language to which ACE is compared. For this experiment, the Manchester OWL Syntax, a usability-oriented syntax of the ontology language OWL, has been chosen. The inventors of the Manchester OWL Syntax introduce it as follows [73]:

> ❝ The syntax, which is known as the Manchester OWL Syntax, was developed in response to a demand from a wide range of users, who do not have a Description Logic background, for a “less logician like” syntax. The Manchester OWL Syntax is derived from the OWL Abstract Syntax, but is less verbose and minimizes the use of brackets. This means that it is quick and easy to read and write. ❞

As this quotation shows, the Manchester OWL Syntax is designed for good usability and good understandability and thus seems to be an appropriate choice for this experiment. However, the Manchester OWL Syntax requires the statements to be grouped by their main ontological entity (the one in subject position so to speak). This is a reasonable approach for the definition of complete ontologies, but it makes it impossible to state short and independent statements that could be used for a direct comparison to ACE in an experimental setting.

For this reason, a modified version of the Manchester OWL Syntax has been defined specifically for this experiment. The resulting language which I will call “Manchester-like language” or “MLL” uses the same or very similar keywords but

		Series 1	Series 2	Series 3	Series 4
S ::=	I **HasType** T	×	×	×	
	I R I		×		
	I **not** R I		×		
	T **SubTypeOf** T	×	×	×	
	R **SubRelationOf** R				×
	T **EquivalentTo** T	×			
	R **EquivalentTo** R				×
	T **DisjointWith** T	×			
	R **DisjointWith** R				×
	R **HasDomain** T			×	
	R **HasRange** T			×	
	R **IsSymmetric**				×
	R **IsAsymmetric**				×
	R **IsTransitive**				×
T ::=	**not** T	×	×		
	T **or** T	×		×	
	T **and** T	×			
	R **some** T		×		
	R **only** T		×		
	R **min** N T			×	
	R **max** N T			×	
R ::=	**inverse** R				×

Table 5.1: This table shows the grammar rules of MLL (in Backus-Naur style) and the subsets thereof used for the individual series. Each series uses exactly seven out of the 22 grammar rules.

allows us to state short and independent statements, which do not depend on their ordering.

Table 5.1 shows the simple grammar of MLL to be used in this experiment. Furthermore, the table shows how four different subsets have been defined. Each of them has been tested independently in a separate series. The details about the different series will be explained later on.

MLL adopts the color codes of the Manchester OWL Syntax for improved readability. Turquoise is used for the boolean operators on types and for the inverse operator on relations. More complex operators used for type restrictions involving relations are displayed in magenta.

5.4.1.2 Learning Time

Obviously, the understanding of a language highly depends on the amount of time spent for learning the language. This means that one has to define a certain time frame when evaluating the understandability of languages. Some languages might be the best choice if there is only little learning time; other languages might be less understandable in this situation but are more suitable in the long run.

So far, little is known about how the understandability of CNLs compares to the understandability of common formal languages. CNLs are designed to be understandable with no learning and the results of the first ontograph experiment show that this is indeed the case. Since other formal languages like the Manchester OWL Syntax are not designed to be understandable with no learning at all, it would not be appropriate to compare ACE to such a language in a zero learning time scenario.

For this reason, I chose a learning time for this second experiment of about 20 minutes. This seems to be a reasonable first step away from the zero learning time scenario. The effect of longer learning times remains open to be studied in the future.

5.4.1.3 Participants

Another important design decision is the choice of the participants. Such studies are mostly performed with students because they are flexible and usually close to the research facilities of the universities. In my case, there are even more reasons why students are a good choice. Students are used to think systematically and logically but they are usually not familiar with formal logical notations (unless this lies in their field of study). In this way, they resemble domain experts who have to formalize their knowledge and who should profit from languages like ACE.

The requirements for the participants have been defined as follows: They had to be students or graduates with no higher education in computer science or logic. Furthermore, at least intermediate level skills in written German and English were required, because the experiment itself was explained and performed in German, and English was needed to understand the ACE texts.

64 students have been recruited who fulfill these requirements and exhibit a broad variety of fields of study. The students were on average 22 years old and 42% of them were female and 58% were male. Furthermore, it is important to mention that none of them participated in the first ontograph experiment or in any of the AceWiki experiments.

5.4.1.4 Ontographs and Statements

The second experiment was designed in a more focused way than the first one. There are many aspects that could be tested, but not everything can be done within one experiment. This experiment is designed to evaluate the general understandability and leaves more specific aspects to possible future experiments. This means that the second experiment did not include borderline statements anymore, such as implications with false preconditions and statements using “or” where “and” would be more accurate (cf. the results of the first experiment). On the other hand, the semantic coverage should still be kept as broad as possible. However, some things like reflexive relation instances (i.e. relation instances that have the same individual on the left and right hand side) that were part of the first experiment were not present anymore in the second one.

As in the first experiment, the ontographs were divided into four series, which roughly corresponded to the ones of the first experiment: The first series only contains individuals and types without relations; the statements of the second series contain relations with different kinds of simple universal quantifications; the third series contains domain, range, and number restrictions; and, finally, the fourth series consists basically only of relations.

For each of the four series, three ontographs have been created (denoted by A, B and C, respectively) and 10 statement patterns have been defined. For each statement pattern of the respective ontograph two statements are defined: one that is true with respect to the situation depicted by the ontograph (denoted by +) and another one that is false (denoted by –). Every statement is expressed in both languages, ACE and MLL. Thus, every ontograph has 20 statements in ACE and 20 statements in MLL that are pairwise semantically equivalent.

The 20 statement pairs for each ontograph are again subdivided into two statement series of 10 statements each (denoted by a and b). This is done in a way, that each statement series has exactly one statement pair of each pattern and that each statement series contains some true statements and some false ones (but not necessarily in a balanced way). The statements of the different ontographs will again be referenced by identifiers of the form "ontograph/statement", for example 3C/5b–. Additionally, more general references of the form "ontograph-series/statement-pattern" will be used that stand for all statements of the given statement pattern of all ontographs of the given series. Thus, the concrete example 3/5 would stand for all statements of pattern 5 of the ontographs 3A, 3B and 3C.

The ontographs B and C of each series and their statements are structurally equivalent in the sense that they can be transformed into each other by applying one-to-one transformations to the names of individuals, types and relations. As we will see, this is important for ensuring that the task of classifying ACE statements is equally hard as the task of classifying MLL statements, while minimizing the learning effect when the two tasks are to be performed in succession.

5.4.1.5 Groups

In order to enable a good comparison between the two languages ACE and MLL, a "within subject" design is used, which means that each participant is tested on ACE and on MLL. This and the design decision to have a learning phase of about 20 minutes increases the time needed to perform this experiment, compared to the first one. Since the participants of an experiment cannot be expected to concentrate for much longer than one hour, it is impossible to test a participant on all ontograph series in the same experiment session. For this reason, each participant is randomly assigned to one series and is only tested on that one.

The participants are equally (and randomly) distributed into 16 groups according to the cross product of the ontograph series (1, 2, 3 and 4), the order in which the participant is tested on the two languages (AM for ACE first and then MLL; MA for MLL first and then ACE), and the statement series (a and b):

$$groups := (\mathsf{1}, \mathsf{2}, \mathsf{3}, \mathsf{4}) \times (\mathsf{AM}, \mathsf{MA}) \times (\mathsf{a}, \mathsf{b})$$

Having altogether 64 participants gives us 4 participants for each of the 16 groups.

5.4.1.6 Procedure

The experiment was conducted in a computer room with a computer for each participant. The main part of the experiment was performed on the computer screen. Additionally, the participants received different printed sheets during the experiment.

The overall procedure consisted of seven stages:

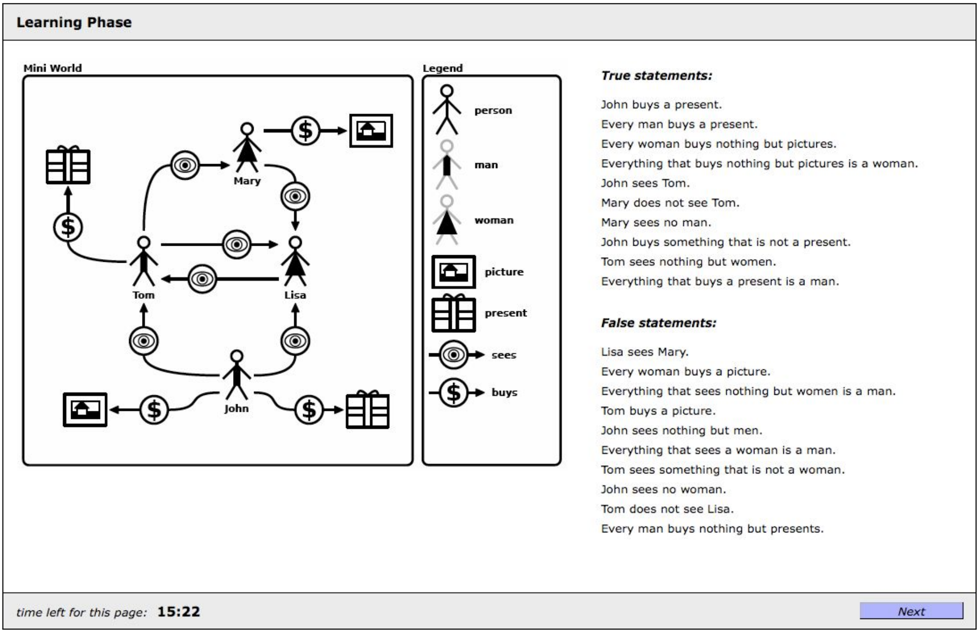
1. Instruction
2. Control Questions
3. Learning Phase 1
4. Testing Phase 1
5. Learning Phase 2
6. Testing Phase 2
7. Questionnaire

For the instruction phase, the participants received a printed instruction sheet that explains the experiment procedure, the payout, and the ontograph notation (showing ontograph 1A, 2A, 3A or 4A, respectively).

The reverse side of the instruction sheet contained control questions for the participants to answer, which allowed us to check whether the participants understood the instructions correctly. The participants had to return the filled-out instruction sheet to the experimenter. The experimenter then checked whether all questions were answered correctly. In the case of false answers, the experimenter explained the respective issue to the participant.

For the learning phase, the participants received a language description sheet of the first language (either ACE or MLL). This language description sheet only explained the subset of the language that is used for the respective series. For this reason, each series had its own instruction sheets for both languages. During the learning phase, the participants had to read the language description sheet. Furthermore, an ontograph (the same as on the instruction sheet) was shown on the screen together with the 10 true statements marked as "true" and the 10 false statements marked as "false" in the respective language. Figure 5.5 shows a screenshot of the experiment screen during the learning phase.

During the testing phase, a different ontograph was shown on the screen (1B, 2B, 3B or 4B for one language; 1C, 2C, 3C or 4C for the other language). Furthermore, the



Learning Phase

True statements:

John buys a present.
Every man buys a present.
Every woman buys nothing but pictures.
Everything that buys nothing but pictures is a woman.
John sees Tom.
Mary does not see Tom.
Mary sees no man.
John buys something that is not a present.
Tom sees nothing but women.
Everything that buys a present is a man.

False statements:

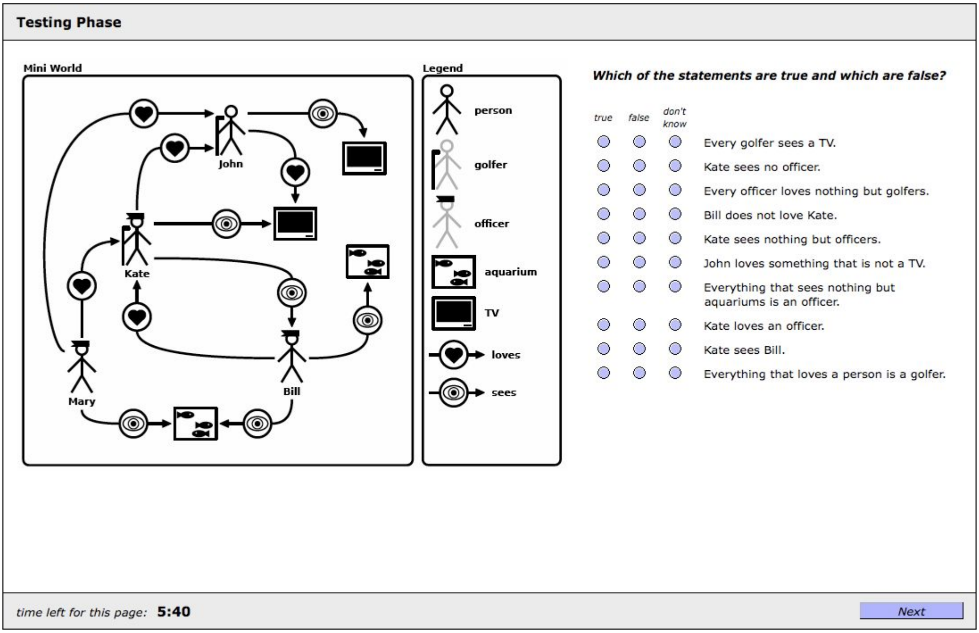
Lisa sees Mary.
Every woman buys a picture.
Everything that sees nothing but women is a man.
Tom buys a picture.
John sees nothing but men.
Everything that sees a woman is a man.
Tom sees something that is not a woman.
John sees no woman.
Tom does not see Lisa.
Every man buys nothing but presents.

Figure 5.5: This figure shows the screen of the second ontograph experiment during the learning phase for the language ACE. An ontograph is shown to the left and to the right several statements are listed as "true statements" and as "false statements". At the bottom left, the remaining time for the learning phase is shown. The button "Next" can be used to proceed before the time limit runs out.

10 statements of the respective statement series in the respective language were shown on the screen together with radio buttons that allowed the participants to choose between "true", "false" or "don't know". Figure 5.6 shows how the experiment screen of the testing phase looked like. During the testing phase, the participants could keep the language description sheet that they got for the learning phase. Thus, they did not need to know the language description by heart but they could read parts of it again during the testing phase if necessary.

For the steps 5 and 6, the procedure of the steps 3 and 4 was repeated in the second language, i.e. ACE if the first language was MLL and vice versa. Because of the fact that the used ontographs B and C and their statements are structurally equivalent, it could be ensured that the ACE task had exactly the same difficulty as the MLL task. Using the same ontograph for both tasks was not an option because this would have entailed a large learning effect between the two tasks.

Finally, the participants received a printed questionnaire form inquiring about



Which of the statements are true and which are false?

true	*false*	*don't know*	
○	○	○	Every golfer sees a TV.
○	○	○	Kate sees no officer.
○	○	○	Every officer loves nothing but golfers.
○	○	○	Bill does not love Kate.
○	○	○	Kate sees nothing but officers.
○	○	○	John loves something that is not a TV.
○	○	○	Everything that sees nothing but aquariums is an officer.
○	○	○	Kate loves an officer.
○	○	○	Kate sees Bill.
○	○	○	Everything that loves a person is a golfer.

Figure 5.6: This figure shows the screen of the second ontograph experiment during the testing phase for the language ACE. An ontograph is shown to the left and to the right ten statements are listed each having three radio buttons with the options “true”, “false” and “don’t know”. At the bottom left, the remaining time for the testing phase is shown. If all statements are classified, the button “Next” can be used to proceed.

their background and their experiences during the experiment. The experiment was finished when the participants turned in the completed questionnaire form.

The learning phases had a time limit of 16 minutes each (20 minutes in the test run) and the time limit for the testing phases was 6 minutes (10 minutes in the test run). The participants were forced to proceed when the time limit ran out but they could proceed earlier. In this way, it can not only be investigated how understandable the languages are but also how much time the participants needed to learn them.

5.4.1.7 Language Description Sheets

The proper design of the language description sheets is crucial for this experiment. If the participants perform better in one language than in the other, it might be that the respective language was merely described better than the other. Thus, the language description sheets have to be written very carefully to be sure that they are not misunderstood and are optimal for learning the respective language under the given time restrictions.

In order to achieve an ideal learning effect given the strict time restrictions, each series has its own set of language description sheets that only describe the part of the language that is needed within the respective series. Appendix B.2 shows all language description sheets used for the experiment. The experiment was performed in German and for this reason German versions of the sheets were used. The appendix shows translations in English. Furthermore, all language description sheets in German and English are available online[1].

The creation of the language description sheets for ACE is not very problematic. Firstly, ACE is designed to be understood without training and thus only little explanation is needed. Secondly, I am experienced in describing and explaining ACE. Thirdly, since I advocate controlled natural languages, nobody would doubt that I will ensure a first-rate description of ACE.

The ACE description sheets only explain a small number of issues, namely that "or" is meant inclusively, that "something", "everything" and "nothing" can also refer to persons, that "nothing but ..." includes "nothing", that "at most" does not exclude zero, and how variables have to be interpreted. In the case of ACE, it might not be necessary to explain all these aspects. However, in order to have an adequate comparison to MLL where such aspects have to be explained anyway, they are also explained for the case of ACE. Because no series makes use of more than three of these aspects, the ACE description sheets of the different series are very short. The most important part of the ACE description sheets is the part that tells the participants to rely on their intuitive understanding when reading the ACE statements.

Things are slightly more complicated with the language description sheets for MLL. MLL is not designed to be understood without training and I had no experience in teaching such languages. Furthermore, people might suspect that I did not do my best to provide a good language description of MLL because using a suboptimal description would make ACE more favorable. Thus, it is crucial to ensure that the description of MLL is as good as it can be.

The quality of the language description sheets for MLL was ensured in three steps. First of all, the four series were designed in a way that at most seven MLL keywords are used per series. Since each series has its own language description sheets, not more than seven keywords have to be described by the same sheet. This should make it easier to understand the needed subset of the language. Furthermore, the fact that the ontographs of the second experiment did not contain reflexive relation instances (i.e. relation instances with the same individual on the left and right hand side) allows further simplifications. In the description of MLL, the word "another" can be used conveniently for saying things like "whenever an individual has the given relation to another individual then ...", which would not be accurate if reflexive relation instances were present. In this case, one would have to say something like

[1] http://attempto.ifi.uzh.ch/site/docs/ontograph/

"whenever an individual has the given relation to an individual that may or may not be the same individual then ...", which makes the complete description more verbose and harder to understand. Thus, the exclusion of reflexive relation instances makes the MLL description sheets much more readable.

In a second step, the different MLL description sheets were given to three participants (who comply with the restrictions of the experiment but who did not participate in it). These three persons read the sheets and gave me feedback about what they did not understand and what could be improved.

As a third step, I used the test run (to be described below) to receive final feedback about the understandability and usefulness of the language description sheets. After the test run, the participants received the sheets again and they were told to highlight everything that was difficult to understand. Only very few things were highlighted (altogether two highlightings in the MLL description, one in the ACE description, and none in the general instructions) and according to this I made a couple of small last changes for the main experiment.

Altogether, the language description sheets were compiled very carefully and it is very unlikely that a different description of MLL would radically increase its understandability. To affirm the quality of the used sheets, Appendix B.2 can be consulted.

5.4.1.8 Payout

In order to be able to recruit a large number of participants for this experiment, it was necessary to financially compensate them. The experiment was designed to last approximately one hour: 22 minutes for each of the two tasks plus some additional time for reading the instructions, for answering the control questions, and for filling out the questionnaire. For their presence during the time of the experiment, the participants received a fixed amount of 20 Swiss francs. At the time of the recruitment, this amount of money was promised to the participants as their minimal reward.

In order to provide incentives for good performance, the participants additionally received a variable amount of money depending on the number of statements they managed to classify correctly. Every correct classification was worth 0.60 Swiss francs. Every "don't know" answer and statements for which the time limit ran out gave 0.30 Swiss francs. Thus, the participants could earn between 20 and 32 Swiss francs, and they could get 26 Swiss francs for sure by always choosing "don't know". The method of the payout calculation was described in the instruction sheets and was checked in the control questions.

While there were monetary incentives for good classification scores, the time needed for the learning and testing phases had no influence on the payout. The reason for this was that quickly choosing "don't know" for all statements (or just guessing quickly and randomly) would become a profitable strategy if short time values gave high rewards. In order to prevent from such behavior, no monetary incentives were given for the amount of time needed. For the same reason, participants that required less time for the tasks were not allowed to leave the experiment earlier, and this was known to the participants from the beginning. It was assumed that no explicit incentives are necessary for the participants to continue with the procedure when they think that they accomplished the task as good as they could (and this was indeed the case, as the results will show).

Thus, the participants had incentives to do the classifications of the statements correctly in order to get a high reward, but there was no concrete incentive for finishing the tasks before the time limit runs out.

5.4.1.9 Test Run

In order to test the design of the experiment and to get some practical experience, I performed a test run before the main experiment. For this test run, ten participants were recruited and nine of them showed up.

The participants could spend up to 20 minutes for each learning phase and up to 10 minutes for each testing phase.

The participants performed very well. On average, they classified 8.92 out of 10 statements correctly. This is an indication that the experiment in general worked out well. ACE was understood better but not much better (0.28 points). Interestingly, four of the nine participants had a perfect score of 10 for both languages. On the one hand, this is good because it shows that the task was feasible and that the instructions were clear. On the other hand, it is bad because it does not return any indication which of the languages was better.

For this reason, I decided to make the task slightly harder for the main experiment. This should reduce the number of perfect scores and thus give us better feedback on which language is better.

It would have been very hard to change the ontographs and the related statements in a way that would leave all the design decisions intact. The easiest and cleanest way to make the task harder was to reduce the time limits. I decided to reduce the maximum learning time from 20 to 16 minutes and the maximum testing time from 10 to 6 minutes. In this way, the maximum learning time was still higher than the average learning time of the test run (13 minutes) and the maximum testing time was about the same as the average testing time of the test run. Thus, the new limits still give a reasonable amount of time for the given task.

5.4.2 Results of the second Ontograph Experiment

Now we can look at the results of the main experiment. These results are retrieved on the one hand from the log files that show how the participants classified the statements and how long it took them, and on the other hand from the questionnaire forms filled out by the participants after the experiment.

As the first one, the second experiment is evaluated on the basis of the general classification score and the amount of time needed. Additionally, the answers in the

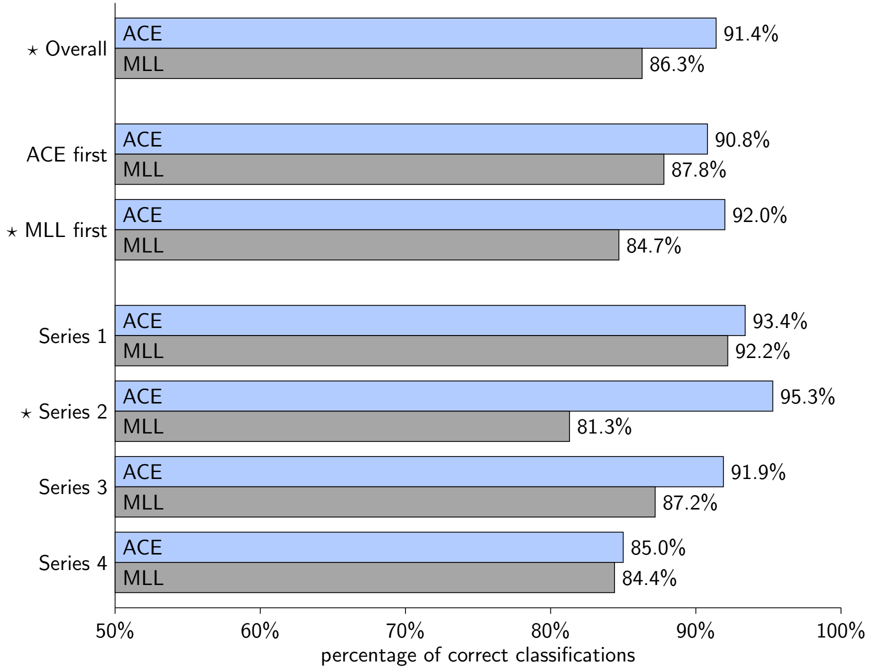


Figure 5.7: This chart shows the average percentages of correct classifications for the second ontograph experiment. The base line is 50% that can be achieved by mere guessing. "Don't know" classifications and cases where the time limit ran out count as 0.5 correct classifications. Significant differences are marked by "⋆" (see Table 5.2 for details).

questionnaire allow us to measure the perceived understandability. The differences that these measurements exhibit between ACE and MLL are then checked for statistical significance. Furthermore, a regression analysis is described that investigates the influence of different factors on the performance of the participants. Finally, we can again have a closer look at how the individual statements have been classified.

5.4.2.1 General Classification Scores

Figure 5.7 shows the average percentages of correct classifications per testing phase. As for the first experiment, "don't know" answers and the cases where the time limit ran out are counted as 0.5 correct classifications. 50% is again the baseline because an average of five correct classifications out of ten can be achieved by mere guessing, by choosing always "don't know", or by letting the time limit run out.

91.4% of the statements were classified correctly in the case of ACE and 86.3% in the case of MLL. Thus, out of the ten statements of a testing phase, ACE was on average 0.5 points better than MLL. This is a considerable and statistically significant difference (the details of the used statistical test to compare the two samples are explained later on). One also has to consider that these values are already close to the ceiling in the form of the perfect score of 10, which might have reduced the actual effect.

The results of the participants who received ACE first and then MLL can now be compared with the ones who received MLL first. As expected, both languages were understood better when they were the second language. This can be explained by the fact that the participants were more familiar with the procedure, the task, and the ontograph notation. However, even in the case when ACE was the first language and MLL the second one, ACE was understood better (but in this case not within statistical significance).

Looking at the results from the perspective of the different series, one can see that ACE was better in all cases but only the series 2 and 3 exhibit a clear dominance of ACE (and this dominance is significant only for series 2). According to these results, one could say that languages like MLL are equally easy to understand for very simple statements as the ones in series 1 and for statements about relations as they appear in series 4. In the case of series 1, the reason might be that these statements are so simple that they can be understood even in a rather complicated language. In the case of series 4, the reason is probably that Description Logic based languages like MLL can express these statements without variables whereas ACE needs variables, which are somehow borderline cases in terms of naturalness. (An advantage of ACE that is not manifested in this experiment is that one can express more complicated statements that cannot be expressed by Description Logic based languages.)

Compared to the test run where 44% of the participants had a perfect score for ACE and for MLL, only 23% had a perfect score in the main experiment. Thus, the reduction of the time limits seemed to have had the desired effect.

In summary, the results show that — while both languages are understood reasonably well — ACE is easier to understand than MLL.

5.4.2.2 Time

As a next step, we can look at the time values. For simplicity reasons and since the learning process was presumably not restricted to the learning phase but continued during the testing phase, the time needed for both phases will together be called the *learning time*.

Figure 5.8 shows the learning times of the participants. They could spend at most 22 minutes (16 minutes for the learning phase and 6 minutes for the testing phase). The participants needed much less time for ACE than for MLL. In the case of ACE less than 14 minutes were needed, whereas in the case of MLL the participants needed more than 18 minutes. Thus, MLL required 29% more time to be learned, compared to ACE.

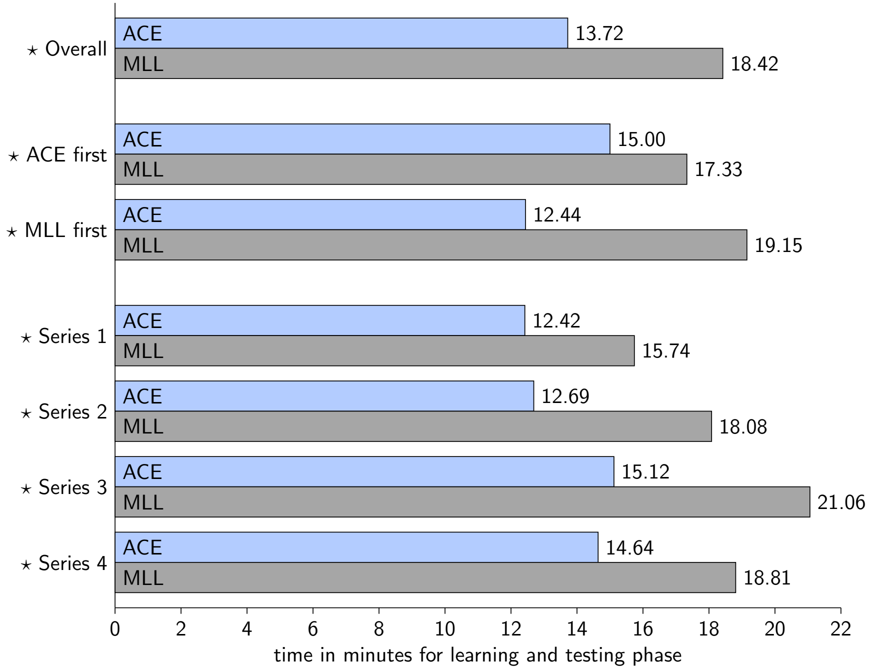


Figure 5.8: This chart shows the average time needed for learning and testing phase. Significant differences are marked by "⋆" (see Table 5.2 for details).

Note that these results can be evaluated only together with the results described above concerning the classification scores. The learning time can only be evaluated together with the degree of understanding it entails. The smaller amount of learning time for ACE could be explained simply by the fact that the language description sheets for ACE contained less text than the ones for MLL. But together with the results described above that show that ACE was understood better and the fact that the language description sheets have been written very carefully, it can be concluded that ACE required less learning time while leading to a higher degree of understanding.

Again, we can split the results according to the participants who received ACE first and those who received MLL first. The results show the expected effect: ACE and MLL required less time as second language. However, ACE required less time than MLL no matter if it was the first language or the second. Thus, even in the cases where ACE was the first language and the participants had no previous experience with the procedure and MLL was the second language and the participants could use the experiences they made before, even in such cases ACE required less time.

Looking at the different series, we can see that this effect spreads over all four series. MLL required on average between 3 and 6 minutes more than ACE.

The better time values of ACE compared to MLL are statistically significant for the whole sample as well as for all presented subsamples.

5.4.2.3 Perceived Understandability

As a third dimension, we can look at the "perceived understandability", i.e. how the participants perceived the understandability of the languages. The questionnaire that the participants filled out after the experiment contained two questions that asked the participants how understandable they found ACE and MLL, respectively. They could choose from four options: "very hard to understand" (value 0), "hard to understand" (1), "easy to understand" (2) and "very easy to understand" (3). The perceived understandability does not necessarily have to coincide with the actual understandability and can be a very valuable measure for the acceptance of a language and the confidence of its users.

Figure 5.9 shows the perceived understandability derived from the questionnaire scores of the languages. Overall, ACE got much better scores than MLL. MLL was close but below "easy to understand" scoring 1.92, whereas ACE was closer to "very easy to understand" than to "easy to understand" scoring 2.59.

By dividing the results into those who received ACE first and those who received MLL first, we see that both languages scored better when ACE was the first language. I do not have a convincing explanation for this and it might just be a statistical artifact.

Looking at the perceived understandability scores from the perspective of the different series, we see that ACE received clearly better scores in all four series. It is interesting that this also holds for the series 1 and 4 where ACE was not much better than MLL in terms of actual understanding, as shown before. Thus, even though the actual understanding of the statements of these series does not show a clear difference, the acceptance and confidence of the participants seems to be higher in the case of ACE.

5.4.2.4 Significance

The charts with the experiment results already indicate in which cases the difference between ACE and MLL is statistically significant. This was done by using the Wilcoxon signed-rank test [176], which is a non-parametric statistical method for testing the difference between measurements of a paired sample. In contrast to Student's t-test, this test does not rely on the assumption that the statistical population corresponds to a standard normal distribution. This relieves us from investigating whether standard normal distribution can be assumed for the given setting.

Table 5.2 shows the obtained p-values for the three dimensions of our comparison (i.e. classification score, time, and questionnaire score). For the complete sample, the

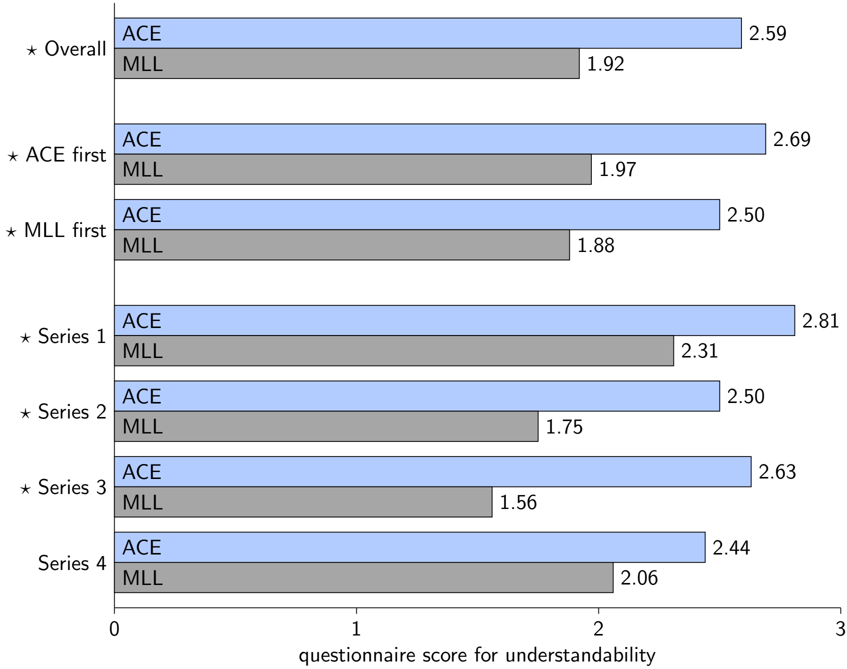


Figure 5.9: This chart shows the average subjective understandability scores derived from the questionnaire. 0 means "very hard to understand", 1 means "hard to understand", 2 means "easy to understand", and 3 means "very easy to understand". Significant differences are marked by "⋆" (see Table 5.2 for details).

values are well within the 95% confidence level for all three dimensions. They are even within the 99% level.

5.4.2.5 Regression

In a next step, a regression analysis can be performed to find out which factors were relevant for a good understandability of the two languages. The data set consists of the classification scores of the 128 test phases (two test phases for each of the 64 participants). Apart from the factors that originate from the experiment design (i.e. the two different languages in different order and the different ontograph series), it can be speculated that the gender of the participants, their age, and their English skills could have an influence on the results.

Table 5.3 shows the result of the regression test. *sc_norm* is the dependent variable and stands for the classification score normalized to 5 (i.e. the normalized score is obtained by subtracting 5 from the number of correctly classified statements). The

		p-value	
classification score:	complete sample	0.003421	⋆
	ACE first	0.2140	
	MLL first	0.005893	⋆
	Series 1	0.5859	
	Series 2	0.003052	⋆
	Series 3	0.1250	
	Series 4	0.6335	
time:	complete sample	1.493×10^{-10}	⋆
	ACE first	0.006640	⋆
	MLL first	3.260×10^{-9}	⋆
	Series 1	0.01309	⋆
	Series 2	0.002624	⋆
	Series 3	9.155×10^{-5}	⋆
	Series 4	0.002686	⋆
questionnaire score:	complete sample	3.240×10^{-7}	⋆
	ACE first	7.343×10^{-5}	⋆
	MLL first	0.001850	⋆
	Series 1	0.02148	⋆
	Series 2	0.02197	⋆
	Series 3	0.0004883	⋆
	Series 4	0.1855	

Table 5.2: This table shows the p-values of Wilcoxon signed-rank tests. The null hypothesis is that the given values are not different for ACE and for MLL. This null hypothesis can be rejected in 16 of the 21 cases on a 95% confidence level, marked by "⋆".

normalization has the effect that 0 stands for what can be achieved, on average, without understanding.

There are eight independent variables: *ace*, *first_lang*, *series_2*, *series_3*, *series_4*, *female*, *age_above_18* and *very_good_engl*. *ace* stands for the language that is tested with 0 meaning MLL and 1 meaning ACE. *first_lang* is 1 if the language was the first language that was tested, and 0 if it was the second one. *series_2* is 1 if the test was performed on series 2, and otherwise it is 0. In the same way, *series_3* and *series_4* represent the series 3 and 4, respectively. Series 1 is encoded by setting all those three variables to 0. *female* determines the gender of the participant: 0 means male, 1 means female. *age_above_18* contains an integer denoting the age in years of the participant after subtracting 18 years. *very_good_engl*, finally, stands for the degree of English understanding of the participants as they stated it in the questionnaire. 0 means that the participant has good English skills but not very good ones. 1 means that the participant has very good English skills.

Thus, the baseline of the regression (i.e. the case where all independent variables

	descriptive		*regression*				
	Avg.	Range	Coef.	Std. Err.	t	$P > \|t\|$	
sc_norm	3.883	−2 to 5					
ace	0.500	0 or 1	0.516	0.180	2.86	0.006	$\star$
first_lang	0.500	0 or 1	−0.219	0.180	−1.22	0.229	
series_2	0.250	0 or 1	−0.480	0.337	−1.42	0.159	
series_3	0.250	0 or 1	−0.278	0.349	−0.80	0.429	
series_4	0.250	0 or 1	−0.880	0.522	−1.69	0.097	
female	0.422	0 or 1	0.141	0.298	0.47	0.637	
age_above_18	4.109	0 to 28	−0.072	0.030	−2.44	0.018	$\star$
very_good_engl	0.391	0 or 1	0.203	0.297	0.68	0.496	
cons			4.302	0.325	13.23	0.000	$\star$

Table 5.3: This table shows the result of the regression analysis of the second ontograph experiment with *sc_norm* being the dependent variable. To the left, a small descriptive analysis of the data is shown. "$\star$" indicates the coefficients of the regression analysis that are statistically significant on a 95% confidence level.

are zero) is about testing MLL as the second language using series 1 where the participant is a 18 year old male person with good (but not very good) English skills. This baseline situation is represented by the constant coefficient *cons*. The value of the constant coefficient can be interpreted as the average normalized score in the baseline situation. Thus, an 18 year old male person with good English skills who was tested on series 1 in MLL as the second language scored on average 9.302 ("denormalized" by adding 5 to 4.302) points out of 10. The values of the other coefficients can be interpreted as the difference from the baseline when changing the respective factor.

If ACE is tested instead of MLL (but everything else is left unchanged) then the score was on average 0.52 points higher (which roughly coincides with the result discovered earlier). This effect is statistically significant on a 95% confidence level.

The results presented in the preceding sections already showed that both languages performed better when they were the second language. The regression test confirms this and we see that being the first language decreased the score on average by 0.22 points. However, this is not significantly different from 0 and thus there is no evidence that second languages systematically perform better.

As expected, series 1 led to the best scores. Using series 2, 3 or 4 decreased the score on average by 0.28 to 0.88 points compared to series 1. However, this is again not statistically significant.

The possible speculation that the gender of the participants has an influence on their performance could not be confirmed. Women on average performed better than men, but not significantly.

A significant effect could be found, however, with respect to the age of the participants. The age of 18 years was chosen as the baseline because this was the minimal age encountered in the set of participants. The older the participants were the worse they performed. Every year of additional age led to a decrease of the score by on average 0.07 points. Thus, a 23 year old participant performed on average about 0.35 points worse than someone who was only 18 years old. This effect is significant on a 95% confidence level.

Finally, we can look at the degree of English skills of the participants. The participants were mostly native German speakers. In order to ensure that they had sufficient knowledge of English to understand the ACE sentences, they were asked in the questionnaire about their English skills. The four choices were "(almost) no skills", "only little skills", "good skills" and "very good skills". The first two choices were only for control reasons because such people do not meet the requirements of the experiment, and they had to be excluded from the data set that was used for the evaluation. Thus, the data set only contains participants with good or very good English skills. As one could expect, people with very good skills performed better than those who had only good skills, but not significantly.

5.4.2.6 Individual Statements

Finally, we can look at the individual statement patterns and how they were classified by the participants. Figure 5.10 visualizes these results. With the exception of one ACE statement pattern and four MLL statement patterns, the predominance of correct classifications over wrong ones is statistically significant on a 95% confidence level using a simple binomial test.

In the case of ACE, the statements of each pattern were classified correctly by at least 69% of the participants. In contrast, the statements of some MLL patterns were classified correctly in only 50% of the cases. While there is no MLL pattern whose statements scored more than 2 points better than the respective ACE statements, the statements of six ACE patterns scored 3 or more points better than the respective MLL statements. It is interesting to have a closer look at those six statement patterns, which are 1/3, 1/5, 2/5, 2/8, 3/8 and 4/9. Table 5.4 shows these six patterns with examples in ACE and MLL to give an impression of the better understandability of ACE.

The statements of 1/3 are simple statements using "or" like for example "Mary is an officer or is a golfer", which seems to be easier to understand than the respective sentence in MLL "Mary **HasType** officer **or** golfer".

1/5 is a very interesting case consisting of plain subtype statements. Such statements are very simple and essential for the creation of ontologies. Many existing ontologies consist mainly of such statements. They are represented in MLL for example by "golfer **SubTypeOf** man". The results show that the ACE version "every golfer is a man" is easier to understand.

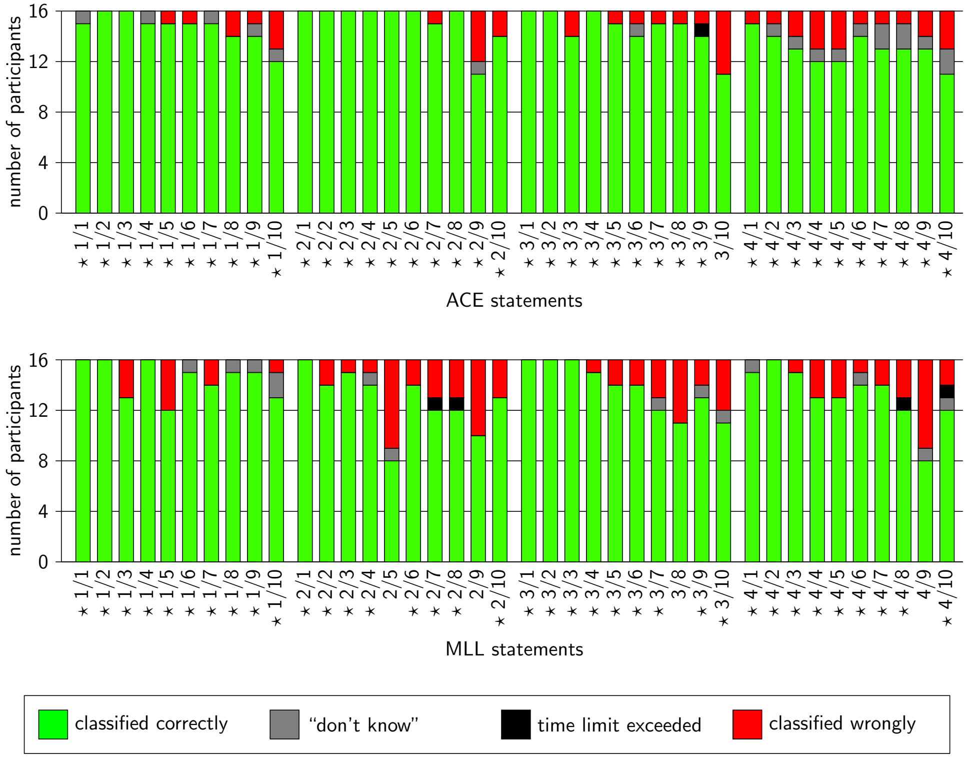


Figure 5.10: These two charts show for each ACE statement pattern and for each MLL statement pattern how many participants classified the statement correctly, classified it wrongly, said that they don’t know, or exceeded the time limit. The predominance of the correct classification over the wrong one is statistically significant on a 95% confidence level for all statement patterns that are marked with “⋆”. Section B.2 of the appendix shows the concrete ontographs and statements.

The largest difference between ACE and MLL is manifested by the statements of 2/5. While everyone scored perfectly in the case of ACE, only half of the participants were able to do so in the case of MLL. These statements are relatively complex and are represented in ACE for example as “John buys something that is not a present” and in MLL as “John **HasType** buys **some** (**not** present)”. Arguably, the reason for this difference is that the combination of the operators “**some**” and “**not**” is hard to understand in the case of MLL whereas it comes completely natural in the case of ACE.

The statements of 2/8 and 3/8 are similar in the sense that both are subtype statements with a complex left hand side. In MLL, they are represented for example

Pattern	ACE Example	MLL Example
1/3	Mary is an officer or is a golfer.	Mary **HasType** officer **or** golfer
1/5	Every golfer is a man.	golfer **SubTypeOf** man
2/5	John buys something that is not a present.	John **HasType** buys **some** (**not** present)
2/8	Everything that buys a present is a man.	buys **some** present **SubTypeOf** man
3/8	Everything that inspects at least 2 letters is an officer.	inspects **min** 2 letter **SubTypeOf** officer
4/9	If X loves Y then X helps Y. If X helps Y then X loves Y.	loves **EquivalentTo** helps

Table 5.4: This table shows the sentence patterns where ACE was clearly better understood than MLL. The shown examples can give an impression why ACE is easier to understand.

by "buys **some** present **SubTypeOf** man" and "inspects **min** 2 letter **SubTypeOf** officer", which seems to be hard to understand. The ACE versions of these examples are "everything that buys a present is a man" and "everything that inspects at least 2 letters is an officer", respectively. Thus, the keyword "**SubTypeOf**" in general seems to be rather hard to understand, whereas the natural quantifier "every" is understood very well.

4/9, finally, was also understood much better in the case of ACE. The respective statements denote the equivalence of two relations. This case is interesting because the respective statements look very different in ACE and in MLL. In MLL this is expressed by short statements like "loves **EquivalentTo** helps" whereas in ACE the same thing is expressed in a much more verbose manner: "If X loves Y then X helps Y. If X helps Y then X loves Y.". Even though the meaning of "**EquivalentTo**" was clearly explained in the instruction sheets for MLL, the more verbose ACE version was understood much better by the participants. Variables as ACE uses them (which are rather rare in natural language) seem to be easier to understand than the more abstract keywords of MLL that do not depend on variables.

5.5 Ontograph Framework Evaluation

Another important question to evaluate is whether the ontograph framework altogether worked out or not. Figure 5.11 shows the results of two questions on the questionnaires of both experiments asking the participants about how understandable they found the ontograph notation and the overall instructions. All values are between "easy to understand" and "very easy to understand". This shows that the ontographs were well accepted by the participants and that it is possible to explain the procedure of such experiments in an understandable way. The fact that these values are considerably higher for the second experiment can be explained by the

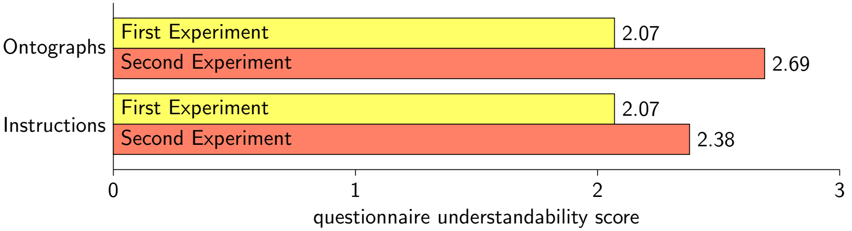


Figure 5.11: This chart shows average values of how understandable the participants found the ontograph notation and the general instructions. These values are based on the answers in the questionnaire. 0 means "very hard to understand", 1 means "hard to understand", 2 means "easy to understand", and 3 means "very easy to understand".

more detailed instructions about the procedure and about the ontograph notation and by the more careful design in general.

Furthermore, the results of the two experiments show that the ontographs were indeed very well understood by the participants. In all discussed cases, the overall percentage of correct classifications exceeded 80%. The ACE statements of the second experiment were even classified correctly in 91.4% of the cases.

Using a simple calculation, we can approximate how often the ontographs are misunderstood. Since the mistakes that are made trying to understand the ontograph notation sum up with the mistakes in understanding the statements (plus other types of mistakes like e.g. oversights), the percentages of incorrect classifications contain different types of errors. In the second experiment, the mistakes made trying to understand the ontographs and the mistakes made trying to understand the ACE sentences lead together to a value of 8.6% incorrect classifications. We have to assume that misunderstandings and other mistakes still lead to the correct classification in 50% of the cases. Thus, we can assume the "rate of mistakes" to be 17.2%. The "rate of ontograph mistakes" must therefore be somewhere between 0% and 17.2%.

In summary, the rate of mistakes due to the ontograph notation cannot be precisely determined but can be shown to be reasonably low.

5.6 Conclusions from the Ontograph Experiments

The results of the two experiments show that the ontograph framework worked out very well and is suitable for testing the understandability of languages.

The first experiment showed that ACE is understandable even without instructions. Some borderline statements were tested and they were understood very poorly. However, there is good reason to believe that this is not specific to CNLs but would be encountered with any other language.

The second experiment showed that ACE is understood significantly better than the language MLL. Furthermore, ACE required less time to be learned and was perceived as easier to understand by the participants. The difference in terms of perceived (i.e. subjective) understandability is bigger than the difference in actual (i.e. objective) understandability. This can lead to the conclusion that common formal languages like MLL do not only have an understandability problem but also an acceptability problem.

MLL is directly derived from the Manchester OWL Syntax in a way that leaves its properties concerning understandability intact. For this reason, the conclusions of the second experiment can be directly applied to the Manchester OWL Syntax, which is the state of the art approach on how to represent ontological knowledge in a user-friendly manner. Thus, I could show that CNLs like ACE can do better in terms of understandability than the current state of the art.

The results show that essential statements like subtype statements are understood better when using ACE. Also variables that are not very frequent in common English are accepted and understood very well. Furthermore, the results show that nested statements can be a problem for languages like MLL whereas they are understood very well when represented in ACE.

Altogether, ACE performed better than MLL in all explored dimensions. These results suggest that CNLs should be used instead of languages like the Manchester OWL Syntax in situations where people have to deal with knowledge representations after little or no training.

5.7 Limitations and Other Applications

Finally, a brief discussion on the limitations of the ontograph approach is given, and some other possible application areas are sketched.

The introduced ontograph approach, of course, also has its limitations. The most important one is probably the fact that only relatively simple forms of logic can be represented. Basically, ontographs cover first-order logic without functions and with the restriction to unary and binary predicates.

I do not see a solution at this point how predicates taking three arguments, for example, could be represented in an intuitive way. It would be even harder to represent more sophisticated forms of logic, e.g. modal or temporal logic. For such cases, it might be necessary to come back to task-based and paraphrase-based approaches to evaluate the understandability of languages. The core of such sophisticated forms of logic, however, could still be tested with the ontograph approach.

Ontographs are designed for the specific purpose of testing the understandability of languages, but they might also be useful in other application areas.

It has already been mentioned that the ontograph framework could be used to evaluate the writability of languages. Such evaluations could be performed by asking the participants of an experiment to describe situations depicted by ontographs.

The teaching and learning of languages could be another possible application area. The task of classifying statements with respect to an ontograph could turn out to be a good exercise for learning the language or for assessing the understanding skills of people. Thus, the same tests could be performed with the goal to evaluate the participants instead of evaluating the respective language.

Another possible application could be to visualize models of a logical theory in the ontograph notation. This approach would be based on the fact that ontographs can be considered a language for describing first-order models, as discussed earlier in this chapter.

In summary, ontographs have been shown to be useful for testing the understandability of languages despite some limitations, and they could possibly also be applied to other application areas.

CHAPTER 6

Conclusions and Outlook

The previous chapters have shown how controlled natural languages in general and subsets of ACE in particular can be defined, deployed and evaluated. Now, the conclusions can be drawn (Section 6.1) and we can have a brief look into the future of the field of controlled natural languages (Section 6.2).

6.1 Conclusions

We can now return to the four questions from the introduction and answer them on the basis of the results presented in the previous chapters.

The first question concerns the definition of languages and has been discussed in Chapter 3. The answer to this question can be summarized as follows:

> **1. How should controlled English grammars be represented?**
>
> Controlled subsets of English should preferably be defined in a concrete and declarative way. Furthermore, they should be defined in a grammar notation that has special support for anaphoric references and that can be used efficiently by different kinds of tools like predictive editors. Existing grammar frameworks do not comply with these requirements. The introduced grammar notation *Codeco*, however, has been shown to be suitable for defining CNLs and has been tested on a large subset of ACE.

The second question that targets the deployment aspect has been examined in Chapter 4. This question can be answered in summary as follows:

> **2. How should tools for controlled English be designed?**
>
> The most important task of tools using controlled English is to solve the writability problem of CNLs. The predictive editor approach could be shown to be a good solution to this problem. This approach makes it possible to create user interfaces that can be used efficiently without training, as the example of AceWiki shows.

The questions number three and four are about the evaluation aspect and have been investigated in Chapter 5. The third question can now be answered as follows:

> **3. How can the understandability of controlled English be evaluated?**
>
> With existing evaluation approaches, the understandability of controlled English cannot be tested reliably. Using the graphical situation depictions of the introduced *ontograph* framework allows us to test and compare the understandability of languages in a tool-independent and reliable way.

The fourth and last question is crucial since it inquires whether controlled English actually has the properties it has been designed for. We are now able to affirm this assumption:

> **4. Is controlled English indeed easier to understand than other formal languages?**
>
> Yes. Using the ontograph framework, I could show that a controlled subset of English like ACE is easier to understand than a comparable formal language. In addition, ACE requires less time to be learned and is preferred by users.

In summary, I could show on the one hand that CNLs like ACE are indeed superior to other languages and that it is possible to embed CNLs in tools in a way that they are easy to use without training. On the other hand, I introduced building blocks that facilitate the practical application of CNL technologies like the Codeco grammar notation and the ontograph framework for understandability evaluations.

Finally, we can return to the hypothesis that is the basis of these four questions:

> **Controlled English efficiently enables clear and intelligible user interfaces for knowledge representation systems.**

Because (1) a simple grammar formalism could be defined that can be used to accurately describe grammars for controlled English, (2) controlled English has in turn proven to be easier to comprehend than an alternative language representing the current state of the art, and (3) prototypical tools show that all this can be combined in a simple way that leads to usable interfaces, the hypothesis is verified.

6.2 Outlook

Ultimately, we can have a brief look into the future of CNLs and on the possible impact of this thesis. The problem of people not familiar with formal languages who have to communicate with computers cannot be expected to disappear in the near future. On the contrary, it can be assumed that more and more people will need to interact with computers and that this communication will become more and more intensive and complex. While a large amount of research has been directed to natural language processing approaches without a real break-through in sight, relatively little work has been done on approaches based on formal CNLs and many aspects thereof are still to be explored. Thus, such CNL approaches are a relatively new and promising direction to bring us further in solving this human–computer interface problem.

While CNLs with a direct connection to formal logic have been quite successful in the academic world, they have not yet been applied extensively on complex real-world scenarios. This is most probably the reason why they could not yet gain ground in industry. My thesis provides building blocks that should facilitate the efficient and reliable use of CNL techniques and should allow us to see the topic of CNLs from a more engineering perspective. This could be the first step towards industry adoption and maybe towards a “CNL killer application” that could boost the research on CNL.

Since user interfaces are crucial for all kinds of knowledge representation systems, the areas of expert systems and the Semantic Web could profit from CNLs too. Both areas have suffered from poor acceptance by their end users. Thus, CNLs have the potential to advance a broad field of existing research.

From my point of view, collaborative approaches like wikis are one of the most promising directions. As the example of Wikipedia shows, such systems can grow without much intervention in a fast and almost magical way once the critical mass is attained. One can only imagine what would become possible if world knowledge the size of the current Wikipedia was available in a logical form that can be interpreted by computers. A new generation of computer applications could emerge that can apply world knowledge in a sensible way.

In any case, we have to be prepared for a future where computers are not only prevalent but pervasive. In such a world, it is important that everyone — and not only a small elite of educated people — can communicate with the computers. Controlled natural languages could help establishing a computer-assisted society where everybody can equally profit from the computers.

APPENDIX A

ACE Codeco Grammar

This appendix shows the complete ACE Codeco grammar, which describes a subset of ACE in the Codeco notation and is introduced in Section 3.7.

First, the used feature names are shown and explained (Section A.1). Then, the actual grammar is shown consisting of altogether 164 grammar rules (Section A.2). Additionally, there are 15 predefined lexical rules that are used for evaluation purposes (Section A.3).

A.1 Features of ACE Codeco

The ACE Codeco grammar makes use of 25 feature names explained below and listed in alphabetical order:

be is used in the case of verb phrases to determine whether the auxiliary verb "be" is used ("+") or not ("–"). The verb "be" is used for the copula (e.g. "John is a customer") and for passive voice (e.g. "John is observed by Bill").

case stands for the syntactic case and is either "nom" for nominative or "acc" for accusative.

copula determines whether a verb phrase is a copula ("+") or not ("–"). A copula in ACE always uses the verb "be", e.g. "Mary is important".

def is "+" if the given structure is definite, e.g. "the country". Otherwise, it is "–".

embv is "+" if the given structure has an embedded verb phrase, e.g. the noun phrase "John who waits". The noun phrase "a friend of Mary", in contrast, contains no verb phrase.

exist is "+" if the given structure is existentially quantified (e.g. "a woman") and "–" otherwise (e.g. "every woman" or "no woman").

gender stands for the gender of proper names, nouns, and pronouns. It is "masc" for masculine elements and "fem" for feminine ones. This feature is only used in cases where the feature "human" is set to "+".

hasvar is used for antecedents and anaphors. It is "+" if a variable is present and "–" otherwise.

human determines whether a proper name, noun, or pronoun is human ("+") or not ("–").

id contains an identifier for noun phrases that is needed to correctly resolve anaphoric pronouns.

noun is used for antecedents and anaphors and contains the used noun.

of is "+" if the respective category contains an *of*-construct like "part of". Otherwise, it is "–".

pl represents the grammatical number. It is "+" if the respective structure is in plural form or "–" if it is singular.

prep is used for transitive adjectives and contains the preposition, e.g. "about" for the transitive adjective "mad-about".

refl represents whether a pronoun is reflexive ("+") or irreflexive ("–"). "Himself", for example, is a reflexive pronoun, whereas "him" is irreflexive.

rel is "+" if the given structure contains a relative clause, or "–" otherwise.

relpron is used for relative clauses and stores the used relative pronoun, i.e. either "that", "who" or "which". This feature is needed to support noun phrases like "a man that is rich and that waits" and "a man who is rich and who waits" but not "a man who is rich and that waits".

subj contains the id of the grammatical subject or "–" if there is no subject. This is needed, for example, to resolve reflexive pronouns like "herself".

text contains the text of the terminal category to be passed up to higher-level categories.

type stores the type of antecedents.

var contains the names of variables.

vcat stands for the verb category and is either "itr" for intransitive verbs or "tr" for transitive ones.

vform contains the verb form which can be either infinitive ("inf") or finite ("fin").

whin has the value "+" if there is a *wh*-word (i.e. "who", "what" or "which") to the left of the beginning of the respective category, or "–" otherwise.

whout has the value "+" if there is a *wh*-word to the left of the end of the respective category, or "–" otherwise. Thus, "whin" and "whout" are the same if no *wh*-word is contained in the given category.

A.2 Grammar Rules of ACE Codeco

Below, the grammar rules of the ACE Codeco grammar are listed. They are identified by consecutive numbers. The grammar rules of the evaluation subset are marked with an underlined number, e.g. $(\underline{3})$.

The grammar rules (87), (88), (89), (91), (92), (93), (115) and (118) are redundant in the sense that they introduce combined terminal categories like "is not" that are already defined in other grammar rules as two independent categories like "is" followed by "not". The only purpose of these grammar rules is to make the usage in predictive editors more convenient. Expressions like "is not" can in this way be added in one step instead of two.

The category *text* is the start category for the full grammar whereas *complete_sentence* is the start category in the case of the evaluation subset.

Texts and Sentences

text stands for a complete text consisting of an arbitrary number of complete sentences (including zero):

(1) $\textit{text} \xrightarrow{:}$

(2) $\textit{text} \xrightarrow{:} \textit{complete\_sentence} \quad \textit{text}$

A complete sentence is represented by the category *complete_sentence* and is either a declarative sentence that ends with a full stop or a question ending with a question mark:

$(\underline{3})$ $\textit{complete\_sentence} \xrightarrow{:} \textit{sentence} \quad [\,.\,]$

$(\underline{4})$ $\textit{complete\_sentence} \xrightarrow{\sim} /\!/ \ \textit{simple\_sentence\_2}\begin{pmatrix}\text{whin: }-\\ \text{whout: }+\end{pmatrix} \quad [\,?\,]$

General sentences are represented by *sentence*:

$$(\underline{5})\quad \textit{sentence} \quad \overset{:}{\longrightarrow} \quad \textit{sentence\_coord\_1}$$

$$(\underline{6})\quad \textit{sentence} \quad \overset{\sim}{\longrightarrow} \quad /\!/ \quad [\,\text{for every}\,] \quad \textit{nc}\begin{pmatrix}\text{subj: }-\end{pmatrix} \quad \textit{sentence\_coord\_1}$$

$$(\underline{7})\quad \textit{sentence} \quad \overset{\sim}{\longrightarrow} \quad /\!/ \quad [\,\text{if}\,] \quad \textit{sentence\_coord\_1} \quad [\,\text{then}\,] \quad \textit{sentence\_coord\_1}$$

Sentences can be coordinated using “or” (*sentence_coord_1*) and “and” (*sentence_coord_2*):

$$(\underline{8})\quad \textit{sentence\_coord\_1} \quad \overset{:}{\longrightarrow} \quad \textit{sentence\_coord\_2}$$

$$(9)\quad \textit{sentence\_coord\_1} \quad \overset{\sim}{\longrightarrow} \quad /\!/ \quad \textit{sentence\_coord\_2} \quad [\,\text{or}\,] \quad \textit{sentence\_coord\_1}$$

$$(\underline{10})\quad \textit{sentence\_coord\_2} \quad \overset{:}{\longrightarrow} \quad \textit{simple\_sentence\_1}$$

$$(\underline{11})\quad \textit{sentence\_coord\_2} \quad \overset{:}{\longrightarrow} \quad \textit{simple\_sentence\_1} \quad [\,\text{and}\,] \quad \textit{sentence\_coord\_2}$$

Uncoordinated sentences are represented in two levels by *simple_sentence_1* and *simple_sentence_2*:

$$(\underline{12})\quad \textit{simple\_sentence\_1} \quad \overset{\sim}{\longrightarrow} \quad /\!/ \quad [\,\text{it is false that}\,] \quad \textit{simple\_sentence\_2}\begin{pmatrix}\text{whin: }-\\ \text{whout: }-\end{pmatrix}$$

$$(\underline{13})\quad \textit{simple\_sentence\_1} \quad \overset{:}{\longrightarrow} \quad [\,\text{there is}\,] \quad \textit{np}\begin{pmatrix}\text{case: nom}\\ \text{def: }-\\ \text{exist: }+\\ \text{pl: }-\\ \text{subj: }-\\ \text{whin: }-\\ \text{whout: }-\end{pmatrix}$$

$$(\underline{14})\quad \textit{simple\_sentence\_1} \quad \overset{:}{\longrightarrow} \quad [\,\text{there is}\,] \quad \textit{np}\begin{pmatrix}\text{case: nom}\\ \text{def: }-\\ \text{exist: }+\\ \text{pl: }-\\ \text{subj: }-\\ \text{whin: }-\\ \text{whout: }-\end{pmatrix} \quad [\,\text{such that}\,] \quad \textit{simple\_sentence\_1}$$

$$(\underline{15})\quad \textit{simple\_sentence\_1} \quad \overset{:}{\longrightarrow} \quad [\,\text{there are}\,] \quad \textit{np}\begin{pmatrix}\text{case: nom}\\ \text{def: }-\\ \text{exist: }+\\ \text{pl: }+\\ \text{subj: }-\\ \text{whin: }-\\ \text{whout: }-\end{pmatrix}$$

$$(\underline{16})\quad \textit{simple\_sentence\_1} \quad \overset{:}{\longrightarrow} \quad \textit{simple\_sentence\_2}\begin{pmatrix}\text{whin: }-\\ \text{whout: }-\end{pmatrix}$$

$$(\underline{17})\quad \textit{simple\_sentence\_2}\begin{pmatrix}\text{whin: }\boxed{1}\\ \text{whout: }\boxed{2}\end{pmatrix} \quad \overset{\sim}{\longrightarrow} \quad \textit{np}\begin{pmatrix}\text{case: nom}\\ \text{id: }\boxed{3}\\ \text{pl: }\boxed{4}\\ \text{subj: }-\\ \text{whin: }\boxed{1}\\ \text{whout: }\boxed{5}\end{pmatrix} \quad \textit{vp\_coord\_1}\begin{pmatrix}\text{pl: }\boxed{4}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{5}\\ \text{whout: }\boxed{2}\end{pmatrix}$$

Verb Phrases

Like sentences, verb phrases can be coordinated using “or” (*vp_coord_1*) and “and” (*vp_coord_2*):

$$(\underline{18})\quad \mathit{vp\_coord\_1}\left(\begin{array}{l}\text{pl: }\boxed{1}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{4}\end{array}\right) \xrightarrow{:} \mathit{vp\_coord\_2}\left(\begin{array}{l}\text{pl: }\boxed{1}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{4}\end{array}\right)$$

$$(19)\quad \mathit{vp\_coord\_1}\left(\begin{array}{l}\text{pl: }\boxed{1}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{4}\end{array}\right) \xrightarrow{\sim} /\!\!/\ \mathit{vp\_coord\_2}\left(\begin{array}{l}\text{pl: }\boxed{1}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{5}\end{array}\right)\ [\,\text{or}\,]\ \mathit{vp\_coord\_1}\left(\begin{array}{l}\text{pl: }\boxed{1}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{5}\\ \text{whout: }\boxed{4}\end{array}\right)$$

$$(\underline{20})\quad \mathit{vp\_coord\_2}\left(\begin{array}{l}\text{pl: }\boxed{1}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{4}\end{array}\right) \xrightarrow{:} \mathit{vp}\left(\begin{array}{l}\text{pl: }\boxed{1}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{4}\end{array}\right)$$

$$(\underline{21})\quad \mathit{vp\_coord\_2}\left(\begin{array}{l}\text{pl: }\boxed{1}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{4}\end{array}\right) \xrightarrow{:} \mathit{vp}\left(\begin{array}{l}\text{pl: }\boxed{1}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{5}\end{array}\right)\ [\,\text{and}\,]\ \mathit{vp\_coord\_2}\left(\begin{array}{l}\text{pl: }\boxed{1}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{5}\\ \text{whout: }\boxed{4}\end{array}\right)$$

Uncoordinated verb phrases represented by *vp* can use an auxiliary verb and can have verb phrase modifiers:

$$(\underline{22})\quad \mathit{vp}\left(\begin{array}{l}\text{exist: }\boxed{1}\\ \text{pl: }\boxed{2}\\ \text{rel: }\boxed{3}\\ \text{subj: }\boxed{4}\\ \text{whin: }\boxed{5}\\ \text{whout: }\boxed{6}\end{array}\right) \xrightarrow{\sim} \mathit{aux}\left(\begin{array}{l}\text{be: }\boxed{7}\\ \text{exist: }\boxed{1}\\ \text{pl: }\boxed{2}\end{array}\right)\ v\left(\begin{array}{l}\text{be: }\boxed{7}\\ \text{copula: }\boxed{8}\\ \text{embv: }\boxed{9}\\ \text{exist: }\boxed{1}\\ \text{pl: }\boxed{2}\\ \text{rel: }\boxed{3}\\ \text{subj: }\boxed{4}\\ \text{vform: inf}\\ \text{whin: }\boxed{5}\\ \text{whout: }\boxed{10}\end{array}\right)\ \mathit{vmod}\left(\begin{array}{l}\text{copula: }\boxed{8}\\ \text{embv: }\boxed{9}\\ \text{subj: }\boxed{4}\\ \text{whin: }\boxed{10}\\ \text{whout: }\boxed{6}\end{array}\right)$$

$$(\underline{23})\quad \mathit{vp}\left(\begin{array}{l}\text{exist: }+\\ \text{pl: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{array}\right) \xrightarrow{\sim} v\left(\begin{array}{l}\text{be: }-\\ \text{copula: }\boxed{6}\\ \text{embv: }\boxed{7}\\ \text{exist: }+\\ \text{pl: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{vform: fin}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{8}\end{array}\right)\ \mathit{vmod}\left(\begin{array}{l}\text{copula: }\boxed{6}\\ \text{embv: }\boxed{7}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{8}\\ \text{whout: }\boxed{5}\end{array}\right)$$

The category *v* represents the main verb or — if "be" is used as a copula verb — the complementing noun phrase or adjective complement:

$$(\underline{24})\quad v\left(\begin{array}{l}\text{be: }-\\ \text{copula: }-\\ \text{pl: }\boxed{1}\\ \text{vform: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{3}\end{array}\right) \xrightarrow{:} \mathit{verb}\left(\begin{array}{l}\text{be: }-\\ \text{pl: }\boxed{1}\\ \text{vcat: itr}\\ \text{vform: }\boxed{2}\end{array}\right)$$

$$(\underline{25})\quad v\left(\begin{array}{l}\text{be: }-\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{pl: }\boxed{2}\\ \text{rel: }\boxed{3}\\ \text{subj: }\boxed{4}\\ \text{vform: }\boxed{5}\\ \text{whin: }\boxed{6}\\ \text{whout: }\boxed{7}\end{array}\right) \xrightarrow{:} \mathit{verb}\left(\begin{array}{l}\text{be: }-\\ \text{pl: }\boxed{2}\\ \text{vcat: tr}\\ \text{vform: }\boxed{5}\end{array}\right)\ \mathit{np}\left(\begin{array}{l}\text{case: acc}\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{3}\\ \text{subj: }\boxed{4}\\ \text{vcat: tr}\\ \text{whin: }\boxed{6}\\ \text{whout: }\boxed{7}\end{array}\right)$$

$$(\underline{26})\quad v\left(\begin{array}{l}\text{be: }+\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{array}\right) \xrightarrow{:} \mathit{verb}\left(\begin{array}{l}\text{be: }+\\ \text{vcat: tr}\end{array}\right)\ [\,\text{by}\,]\ \mathit{np}\left(\begin{array}{l}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{array}\right)$$

$$(\underline{27})\quad v\left(\begin{array}{l}\text{be: +}\\ \text{copula: +}\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{array}\right) \xrightarrow{:} np\left(\begin{array}{l}\text{case: acc}\\ \text{copula: +}\\ \text{embv: }\boxed{1}\\ \text{of: +}\\ \text{pl: –}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{array}\right)$$

$$(\underline{28})\quad v\left(\begin{array}{l}\text{be: +}\\ \text{copula: +}\\ \text{embv: }\boxed{1}\\ \text{pl: –}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{array}\right) \xrightarrow{:} np\left(\begin{array}{l}\text{case: acc}\\ \text{copula: +}\\ \text{embv: }\boxed{1}\\ \text{of: –}\\ \text{pl: –}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{array}\right)$$

$$(\underline{29})\quad v\left(\begin{array}{l}\text{be: +}\\ \text{copula: +}\\ \text{whin: }\boxed{1}\\ \text{whout: }\boxed{1}\end{array}\right) \xrightarrow{:} adj\_coord$$

$$(\underline{30})\quad v\left(\begin{array}{l}\text{be: +}\\ \text{copula: +}\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{array}\right) \xrightarrow{:} adjc\left(\begin{array}{l}\text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{array}\right)$$

Noun Phrases

Noun phrases are represented by *np* and can consist of proper names, variables, pronouns, and different noun constructs:

$$(\underline{31})\quad np\left(\begin{array}{l}\text{def: +}\\ \text{embv: }\boxed{1}\\ \text{exist: +}\\ \text{id: }\boxed{2}\\ \text{of: –}\\ \text{pl: –}\\ \text{rel: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{array}\right) \xrightarrow{:} prop\left(\begin{array}{l}\text{gender: }\boxed{6}\\ \text{human: }\boxed{7}\\ \text{id: }\boxed{2}\end{array}\right) \gg \left(\begin{array}{l}\text{gender: }\boxed{6}\\ \text{hasvar: –}\\ \text{human: }\boxed{7}\\ \text{id: }\boxed{2}\\ \text{type: prop}\end{array}\right) \; relcl\left(\begin{array}{l}\text{embv: }\boxed{1}\\ \text{human: }\boxed{7}\\ \text{rel: }\boxed{3}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{array}\right)$$

$$(\underline{32})\quad np\left(\begin{array}{l}\text{def: +}\\ \text{exist: +}\\ \text{id: }\boxed{1}\\ \text{of: –}\\ \text{pl: –}\\ \text{whin: }\boxed{2}\\ \text{whout: }\boxed{2}\end{array}\right) \xrightarrow{:} \#\boxed{1} \;\; newvar\left(\text{var: }\boxed{3}\right) > \left(\begin{array}{l}\text{hasvar: +}\\ \text{id: }\boxed{1}\\ \text{type: var}\\ \text{var: }\boxed{3}\end{array}\right)$$

$$(\underline{33})\quad np\left(\begin{array}{l}\text{def: +}\\ \text{exist: +}\\ \text{id: }\boxed{1}\\ \text{of: –}\\ \text{pl: –}\\ \text{whin: }\boxed{2}\\ \text{whout: }\boxed{2}\end{array}\right) \xrightarrow{:} \underline{def\_noun\_sg}\left(\text{noun: }\boxed{3}\right) \; \underline{ref}\left(\text{text: }\boxed{4}\right) < \left(\begin{array}{l}\text{gender: }\boxed{5}\\ \text{hasvar: +}\\ \text{human: }\boxed{6}\\ \text{id: }\boxed{1}\\ \text{noun: }\boxed{3}\\ \text{type: noun}\\ \text{var: }\boxed{4}\end{array}\right) > \left(\begin{array}{l}\text{gender: }\boxed{5}\\ \text{hasvar: –}\\ \text{human: }\boxed{6}\\ \text{id: }\boxed{1}\\ \text{type: ref}\end{array}\right)$$

$$(\underline{34})\quad np\left(\begin{array}{l}\text{def: +}\\ \text{exist: +}\\ \text{id: }\boxed{1}\\ \text{of: –}\\ \text{pl: –}\\ \text{whin: }\boxed{2}\\ \text{whout: }\boxed{2}\end{array}\right) \xrightarrow{:} \underline{def\_noun\_sg}\left(\text{noun: }\boxed{3}\right) < \left(\begin{array}{l}\text{gender: }\boxed{4}\\ \text{human: }\boxed{5}\\ \text{id: }\boxed{1}\\ \text{noun: }\boxed{3}\\ \text{type: noun}\end{array}\right) > \left(\begin{array}{l}\text{gender: }\boxed{4}\\ \text{hasvar: –}\\ \text{human: }\boxed{5}\\ \text{id: }\boxed{1}\\ \text{type: ref}\end{array}\right)$$

$$(\underline{35})\quad np\left(\begin{array}{l}\text{def: }+\\\text{exist: }+\\\text{id: }\boxed{1}\\\text{of: }-\\\text{pl: }-\\\text{whin: }\boxed{2}\\\text{whout: }\boxed{2}\end{array}\right)\ \overset{:}{\longrightarrow}\ \underline{ref}\left(\text{text: }\boxed{3}\right)\ <\left(\begin{array}{l}\text{gender: }\boxed{4}\\\text{hasvar: }+\\\text{human: }\boxed{5}\\\text{id: }\boxed{1}\\\text{var: }\boxed{3}\end{array}\right)\ >\left(\begin{array}{l}\text{gender: }\boxed{4}\\\text{hasvar: }-\\\text{human: }\boxed{5}\\\text{id: }\boxed{1}\\\text{type: ref}\end{array}\right)$$

$$(\underline{36})\quad np\left(\begin{array}{l}\text{def: }+\\\text{exist: }+\\\text{id: }\boxed{1}\\\text{of: }-\\\text{pl: }-\\\text{refl: }+\\\text{subj: }\boxed{1}\\\text{whin: }\boxed{2}\\\text{whout: }\boxed{2}\end{array}\right)\ \overset{:}{\longrightarrow}\ \underline{pron}\left(\begin{array}{l}\text{gender: }\boxed{3}\\\text{human: }\boxed{4}\\\text{refl: }+\end{array}\right)\ <\left(\begin{array}{l}\text{gender: }\boxed{3}\\\text{human: }\boxed{4}\\\text{id: }\boxed{1}\end{array}\right)$$

$$(\underline{37})\quad np\left(\begin{array}{l}\text{case: }\boxed{1}\\\text{def: }+\\\text{exist: }+\\\text{id: }\boxed{2}\\\text{of: }-\\\text{pl: }-\\\text{refl: }-\\\text{subj: }\boxed{3}\\\text{whin: }\boxed{4}\\\text{whout: }\boxed{4}\end{array}\right)\ \overset{:}{\longrightarrow}\ \underline{pron}\left(\begin{array}{l}\text{case: }\boxed{1}\\\text{gender: }\boxed{5}\\\text{human: }\boxed{6}\\\text{refl: }-\end{array}\right)\ <^{+}\left(\begin{array}{l}\text{gender: }\boxed{5}\\\text{human: }\boxed{6}\\\text{id: }\boxed{2}\end{array}\right)^{-}\left(\text{id: }\boxed{3}\right)\ >\left(\begin{array}{l}\text{gender: }\boxed{5}\\\text{hasvar: }-\\\text{human: }\boxed{6}\\\text{id: }\boxed{2}\\\text{type: pron}\end{array}\right)$$

$$(\underline{38})\quad np\left(\begin{array}{l}\text{embv: }\boxed{1}\\\text{exist: }\boxed{2}\\\text{id: }\boxed{3}\\\text{of: }\boxed{4}\\\text{pl: }-\\\text{rel: }\boxed{5}\\\text{subj: }\boxed{6}\\\text{whin: }\boxed{7}\\\text{whout: }\boxed{8}\end{array}\right)\ \overset{:}{\longrightarrow}\ quant\left(\text{exist: }\boxed{2}\right)\ nc\left(\begin{array}{l}\text{embv: }\boxed{1}\\\text{id: }\boxed{3}\\\text{of: }\boxed{4}\\\text{rel: }\boxed{5}\\\text{subj: }\boxed{6}\\\text{whin: }\boxed{7}\\\text{whout: }\boxed{8}\end{array}\right)$$

$$(\underline{39})\quad np\left(\begin{array}{l}\text{embv: }\boxed{1}\\\text{exist: }\boxed{2}\\\text{id: }\boxed{3}\\\text{of: }-\\\text{pl: }-\\\text{rel: }\boxed{4}\\\text{whin: }\boxed{5}\\\text{whout: }\boxed{6}\end{array}\right)\ \overset{:}{\longrightarrow}\ \#\boxed{3}\ \ ipron\left(\begin{array}{l}\text{exist: }\boxed{2}\\\text{human: }\boxed{7}\end{array}\right)\ opt\_newvar\left(\begin{array}{l}\text{hasvar: }\boxed{8}\\\text{var: }\boxed{9}\end{array}\right)\ >\left(\begin{array}{l}\text{hasvar: }\boxed{8}\\\text{human: }\boxed{7}\\\text{id: }\boxed{3}\\\text{type: ipron}\\\text{var: }\boxed{9}\end{array}\right)\ relcl\left(\begin{array}{l}\text{embv: }\boxed{1}\\\text{human: }\boxed{7}\\\text{rel: }\boxed{4}\\\text{subj: }\boxed{3}\\\text{whin: }\boxed{5}\\\text{whout: }\boxed{6}\end{array}\right)$$

$$(\underline{40})\quad np\left(\begin{array}{l}\text{copula: }-\\\text{exist: }+\\\text{id: }\boxed{1}\\\text{of: }-\\\text{pl: }+\\\text{whin: }\boxed{2}\\\text{whout: }\boxed{2}\end{array}\right)\ \overset{:}{\longrightarrow}\ num\_quant\ \ \underline{num}\ \ opt\_adj\_coord\ \ \#\boxed{1}\ \ \underline{noun\_pl}$$

$$(\underline{41})\quad np\left(\begin{array}{l}\text{copula: }-\\\text{exist: }+\\\text{id: }\boxed{1}\\\text{of: }-\\\text{pl: }-\\\text{whin: }\boxed{2}\\\text{whout: }\boxed{2}\end{array}\right)\ \overset{:}{\longrightarrow}\ num\_quant\ \ [\,1\,]\ \ \#\boxed{1}\ \ opt\_adj\_coord\ \ \underline{noun\_sg}\left(\begin{array}{l}\text{gender: }\boxed{3}\\\text{human: }\boxed{4}\\\text{text: }\boxed{5}\end{array}\right)\ >\left(\begin{array}{l}\text{gender: }\boxed{3}\\\text{hasvar: }-\\\text{human: }\boxed{4}\\\text{id: }\boxed{1}\\\text{noun: }\boxed{5}\\\text{type: noun}\end{array}\right)$$

$$(42)\quad np\left(\begin{array}{l}\text{exist: }+\\\text{id: }\boxed{1}\\\text{of: }-\\\text{pl: }-\\\text{whout: }+\end{array}\right)\ \overset{:}{\longrightarrow}\ \#\boxed{1}\ \ [\,\text{what}\,]\ >\left(\begin{array}{l}\text{hasvar: }-\\\text{human: }-\\\text{id: }\boxed{1}\\\text{type: wh}\end{array}\right)$$

$$(\underline{43})\quad np\left(\begin{array}{l}\text{exist: }+\\\text{id: }\boxed{1}\\\text{of: }-\\\text{pl: }-\\\text{whout: }+\end{array}\right)\ \overset{:}{\longrightarrow}\ \#\boxed{1}\ \ [\,\text{who}\,]\ >\left(\begin{array}{l}\text{hasvar: }-\\\text{human: }+\\\text{id: }\boxed{1}\\\text{type: wh}\end{array}\right)$$

$$(\underline{44})\quad np\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{exist: }+\\ \text{id: }\boxed{2}\\ \text{of: }\boxed{3}\\ \text{pl: }-\\ \text{rel: }\boxed{4}\\ \text{subj: }\boxed{5}\\ \text{whout: }+\end{pmatrix} \xrightarrow{:} [\,\text{which}\,]\quad nc\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{id: }\boxed{2}\\ \text{of: }\boxed{3}\\ \text{rel: }\boxed{4}\\ \text{subj: }\boxed{5}\\ \text{whin: }+\\ \text{whout: }+\end{pmatrix}$$

$$(\underline{45})\quad np\begin{pmatrix}\text{exist: }+\\ \text{id: }\boxed{1}\\ \text{of: }-\\ \text{pl: }+\\ \text{whout: }+\end{pmatrix} \xrightarrow{:} [\,\text{which}\,]\quad opt\_adj\_coord\quad \#\boxed{1}\quad \underline{noun\_pl}$$

The category *nc* represents nouns optionally followed by variables, relative clauses, and prepositional phrases using "of":

$$(\underline{46})\quad nc\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{id: }\boxed{2}\\ \text{of: }-\\ \text{rel: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \xrightarrow{:} n\begin{pmatrix}\text{gender: }\boxed{6}\\ \text{human: }\boxed{7}\\ \text{id: }\boxed{2}\\ \text{text: }\boxed{8}\end{pmatrix}\quad opt\_newvar\begin{pmatrix}\text{hasvar: }\boxed{9}\\ \text{var: }\boxed{10}\end{pmatrix} > \begin{pmatrix}\text{gender: }\boxed{6}\\ \text{hasvar: }\boxed{9}\\ \text{human: }\boxed{7}\\ \text{id: }\boxed{2}\\ \text{noun: }\boxed{8}\\ \text{type: noun}\\ \text{var: }\boxed{10}\end{pmatrix}\quad relcl\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{human: }\boxed{7}\\ \text{rel: }\boxed{3}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

$$(\underline{47})\quad nc\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{id: }\boxed{2}\\ \text{of: }+\\ \text{rel: }\boxed{3}\\ \text{subj: }\boxed{4}\\ \text{whin: }\boxed{5}\\ \text{whout: }\boxed{6}\end{pmatrix} \xrightarrow{\sim} n\begin{pmatrix}\text{gender: }\boxed{7}\\ \text{human: }\boxed{8}\\ \text{id: }\boxed{2}\\ \text{text: }\boxed{9}\end{pmatrix} > \begin{pmatrix}\text{gender: }\boxed{7}\\ \text{hasvar: }-\\ \text{human: }\boxed{8}\\ \text{id: }\boxed{2}\\ \text{noun: }\boxed{9}\\ \text{type: noun}\end{pmatrix}\quad [\,\text{of}\,]\quad np\begin{pmatrix}\text{case: acc}\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{3}\\ \text{subj: }\boxed{4}\\ \text{whin: }\boxed{5}\\ \text{whout: }\boxed{6}\end{pmatrix}$$

The category *n* stands for nouns that are preceded by an optional adjective coordination:

$$(\underline{48})\quad n\begin{pmatrix}\text{gender: }\boxed{1}\\ \text{human: }\boxed{2}\\ \text{id: }\boxed{3}\\ \text{text: }\boxed{4}\end{pmatrix} \xrightarrow{:} opt\_adj\_coord\quad \#\boxed{3}\quad \underline{noun\_sg}\begin{pmatrix}\text{gender: }\boxed{1}\\ \text{human: }\boxed{2}\\ \text{text: }\boxed{4}\end{pmatrix}$$

New variables, optional and mandatory, are represented by *opt_newvar* and *newvar*, respectively:

$$(\underline{49})\quad opt\_newvar\begin{pmatrix}\text{hasvar: }-\end{pmatrix} \xrightarrow{:}$$

$$(\underline{50})\quad opt\_newvar\begin{pmatrix}\text{hasvar: }+\\ \text{var: }\boxed{1}\end{pmatrix} \xrightarrow{:} newvar\begin{pmatrix}\text{var: }\boxed{1}\end{pmatrix}$$

$$(\underline{51})\quad newvar\begin{pmatrix}\text{var: }\boxed{1}\end{pmatrix} \xrightarrow{:} \underline{var}\begin{pmatrix}\text{text: }\boxed{1}\end{pmatrix}\quad \not<\begin{pmatrix}\text{hasvar: }+\\ \text{var: }\boxed{1}\end{pmatrix}$$

Proper names can either require the definite article "the" or not, and are represented by the category *prop*:

$$(\underline{52})\quad prop\begin{pmatrix}\text{gender: }\boxed{1}\\ \text{human: }\boxed{2}\\ \text{id: }\boxed{3}\end{pmatrix} \xrightarrow{:} \underline{prop\_sg}\begin{pmatrix}\text{gender: }\boxed{1}\\ \text{human: }\boxed{2}\\ \text{text: }\boxed{3}\end{pmatrix}$$

$$(53)\quad prop\begin{pmatrix}\text{gender: }\boxed{1}\\ \text{human: }\boxed{2}\\ \text{id: }\boxed{3}\end{pmatrix} \xrightarrow{:} \underline{propdef\_sg}\begin{pmatrix}\text{gender: }\boxed{1}\\ \text{human: }\boxed{2}\\ \text{text: }\boxed{3}\end{pmatrix}$$

Adjectives

Adjectives can only be coordinated by “and”, and are represented by *opt_adj_coord* for the optional case and by *adj_coord* if mandatory:

(54) $$\mathit{opt\_adj\_coord} \xrightarrow{:}$$

(55) $$\mathit{opt\_adj\_coord} \xrightarrow{:} \mathit{adj\_coord}$$

(56) $$\mathit{adj\_coord} \xrightarrow{:} \mathit{adj}$$

(57) $$\mathit{adj\_coord} \xrightarrow{:} \mathit{adj}\ [\,\text{and}\,]\ \mathit{adj\_coord}$$

Uncoordinated adjectives are represented by *adj* and can be used in positive, comparative and superlative forms:

(58) $$\mathit{adj} \xrightarrow{:} \underline{\mathit{adj\_itr}}$$

(59) $$\mathit{adj} \xrightarrow{:} [\,\text{more}\,]\ \underline{\mathit{adj\_itr}}$$

(60) $$\mathit{adj} \xrightarrow{:} \underline{\mathit{adj\_itr\_comp}}$$

(61) $$\mathit{adj} \xrightarrow{:} [\,\text{most}\,]\ \underline{\mathit{adj\_itr}}$$

(62) $$\mathit{adj} \xrightarrow{:} \underline{\mathit{adj\_itr\_sup}}$$

The category *adjc* stands for more complicated adjective constructions including nested noun phrases that represent a comparison object:

(63) $$\mathit{adjc}\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \xrightarrow{:} [\,\text{as}\,]\ \underline{\mathit{adj\_itr}}\ [\,\text{as}\,]\ \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

(64) $$\mathit{adjc}\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \xrightarrow{:} \underline{\mathit{adj\_itr\_comp}}\ [\,\text{than}\,]\ \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

(65) $$\mathit{adjc}\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \xrightarrow{:} [\,\text{more}\,]\ \underline{\mathit{adj\_itr}}\ [\,\text{than}\,]\ \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

(66) $$\mathit{adjc}\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \xrightarrow{:} \underline{\mathit{adj\_tr}}\begin{pmatrix}\text{prep: }\boxed{6}\end{pmatrix}\ \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

(67) $$\mathit{adjc}\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \xrightarrow{:} [\,\text{more}\,]\ \underline{\mathit{adj\_tr}}\begin{pmatrix}\text{prep: }\boxed{6}\end{pmatrix}\ \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

$$(68)\quad \mathit{adjc}\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \overset{:}{\longrightarrow} [\,\text{most}\,]\;\; \underline{\mathit{adj\_tr}}\Big(\text{prep: }\boxed{6}\Big)\;\; \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

$$(\underline{69})\quad \mathit{adjc}\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \overset{:}{\longrightarrow} [\,\text{as}\,]\;\; \underline{\mathit{adj\_tr}}\Big(\text{prep: }\boxed{6}\Big)\;\; \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }-\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{7}\end{pmatrix}\;\; [\,\text{as}\,]\;\; \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{7}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

$$(\underline{70})\quad \mathit{adjc}\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \overset{:}{\longrightarrow} [\,\text{as}\,]\;\; \underline{\mathit{adj\_tr}}\Big(\text{prep: }\boxed{6}\Big)\;\; \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }-\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{7}\end{pmatrix}\;\; [\,\text{as}\,]\;\; \underline{\mathit{adj\_prep}}\Big(\text{prep: }\boxed{6}\Big)\;\; \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{7}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

$$(71)\quad \mathit{adjc}\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \overset{:}{\longrightarrow} [\,\text{more}\,]\;\; \underline{\mathit{adj\_tr}}\Big(\text{prep: }\boxed{6}\Big)\;\; \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }-\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{7}\end{pmatrix}\;\; [\,\text{than}\,]\;\; \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{7}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

$$(\underline{72})\quad \mathit{adjc}\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \overset{:}{\longrightarrow} \underline{\mathit{adj\_tr\_comp}}\Big(\text{prep: }\boxed{6}\Big)\;\; \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }-\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{7}\end{pmatrix}\;\; [\,\text{than}\,]\;\; \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{7}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

$$(73)\quad \mathit{adjc}\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \overset{:}{\longrightarrow} [\,\text{more}\,]\;\; \underline{\mathit{adj\_tr}}\Big(\text{prep: }\boxed{6}\Big)\;\; \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }-\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{7}\end{pmatrix}\;\; [\,\text{than}\,]\;\; \underline{\mathit{adj\_prep}}\Big(\text{prep: }\boxed{6}\Big)\;\; \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{7}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

$$(\underline{74})\quad \mathit{adjc}\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \overset{:}{\longrightarrow} \underline{\mathit{adj\_tr\_comp}}\Big(\text{prep: }\boxed{6}\Big)\;\; \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }-\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{7}\end{pmatrix}\;\; [\,\text{than}\,]\;\; \underline{\mathit{adj\_prep}}\Big(\text{prep: }\boxed{6}\Big)\;\; \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{copula: }-\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{7}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

Relative Clauses

Relative clauses are represented by *relcl*. They start with a relative pronoun and are always optional:

$$(\underline{75})\quad \mathit{relcl}\begin{pmatrix}\text{whin: }\boxed{1}\\ \text{whout: }\boxed{1}\end{pmatrix} \overset{:}{\longrightarrow}$$

$$(\underline{76})\quad \mathit{relcl}\begin{pmatrix}\text{embv: }+\\ \text{human: }\boxed{1}\\ \text{rel: }+\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{4}\end{pmatrix} \overset{:}{\longrightarrow} \mathit{relpron}\begin{pmatrix}\text{human: }\boxed{1}\\ \text{relpron: }\boxed{5}\end{pmatrix}\;\; \mathit{relcl1}\begin{pmatrix}\text{human: }\boxed{1}\\ \text{relpron: }\boxed{5}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{4}\end{pmatrix}$$

Like sentences and verb phrases, relative clauses can be coordinated by “or” (*relcl1*) and “and” (*relcl2*):

$$(77)\quad \mathit{relcl1}\left(\begin{array}{l}\text{human: }\boxed{1}\\ \text{relpron: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{array}\right) \xrightarrow{\sim} /\!/\ \mathit{relcl2}\left(\begin{array}{l}\text{human: }\boxed{1}\\ \text{rel: }-\\ \text{relpron: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{6}\end{array}\right)\ \mathit{or\_relpron}\left(\begin{array}{l}\text{human: }\boxed{1}\\ \text{relpron: }\boxed{2}\end{array}\right)\ \mathit{relcl1}\left(\begin{array}{l}\text{human: }\boxed{1}\\ \text{relpron: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{6}\\ \text{whout: }\boxed{5}\end{array}\right)$$

$$(\underline{78})\quad \mathit{relcl1}\left(\begin{array}{l}\text{human: }\boxed{1}\\ \text{relpron: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{array}\right) \xrightarrow{:} \mathit{relcl2}\left(\begin{array}{l}\text{human: }\boxed{1}\\ \text{relpron: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{array}\right)$$

$$(\underline{79})\quad \mathit{relcl2}\left(\begin{array}{l}\text{human: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{relpron: }\boxed{3}\\ \text{subj: }\boxed{4}\\ \text{whin: }\boxed{5}\\ \text{whout: }\boxed{6}\end{array}\right) \xrightarrow{:} \mathit{vp}\left(\begin{array}{l}\text{pl: }-\\ \text{rel: }-\\ \text{subj: }\boxed{4}\\ \text{whin: }\boxed{5}\\ \text{whout: }\boxed{7}\end{array}\right)\ \mathit{and\_relpron}\left(\begin{array}{l}\text{human: }\boxed{1}\\ \text{relpron: }\boxed{3}\end{array}\right)\ \mathit{relcl2}\left(\begin{array}{l}\text{human: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{relpron: }\boxed{3}\\ \text{subj: }\boxed{4}\\ \text{whin: }\boxed{7}\\ \text{whout: }\boxed{6}\end{array}\right)$$

$$(\underline{80})\quad \mathit{relcl2}\left(\begin{array}{l}\text{rel: }\boxed{1}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{4}\end{array}\right) \xrightarrow{:} \mathit{vp}\left(\begin{array}{l}\text{pl: }-\\ \text{rel: }\boxed{1}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{4}\end{array}\right)$$

$$(\underline{81})\quad \mathit{relcl2}\left(\begin{array}{l}\text{rel: }\boxed{1}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{4}\end{array}\right) \xrightarrow{\sim} \mathit{np}\left(\begin{array}{l}\text{case: nom}\\ \text{copula: }-\\ \text{embv: }\boxed{5}\\ \text{id: }\boxed{6}\\ \text{pl: }\boxed{7}\\ \text{refl: }-\\ \text{rel: }-\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{8}\end{array}\right)\ \mathit{aux}\left(\begin{array}{l}\text{be: }-\\ \text{pl: }\boxed{7}\end{array}\right)\ \mathit{verb}\left(\begin{array}{l}\text{be: }-\\ \text{pl: }\boxed{7}\\ \text{vcat: tr}\\ \text{vform: inf}\end{array}\right)\ \mathit{vmod}\left(\begin{array}{l}\text{copula: }-\\ \text{embv: }\boxed{5}\\ \text{rel: }\boxed{1}\\ \text{subj: }\boxed{6}\\ \text{whin: }\boxed{8}\\ \text{whout: }\boxed{4}\end{array}\right)$$

$$(\underline{82})\quad \mathit{relcl2}\left(\begin{array}{l}\text{rel: }\boxed{1}\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{4}\end{array}\right) \xrightarrow{\sim} \mathit{np}\left(\begin{array}{l}\text{case: nom}\\ \text{copula: }-\\ \text{embv: }\boxed{5}\\ \text{id: }\boxed{6}\\ \text{pl: }\boxed{7}\\ \text{refl: }-\\ \text{rel: }-\\ \text{subj: }\boxed{2}\\ \text{whin: }\boxed{3}\\ \text{whout: }\boxed{8}\end{array}\right)\ \mathit{verb}\left(\begin{array}{l}\text{be: }-\\ \text{pl: }\boxed{7}\\ \text{vcat: tr}\\ \text{vform: fin}\end{array}\right)\ \mathit{vmod}\left(\begin{array}{l}\text{copula: }-\\ \text{embv: }\boxed{5}\\ \text{rel: }\boxed{1}\\ \text{subj: }\boxed{6}\\ \text{whin: }\boxed{8}\\ \text{whout: }\boxed{4}\end{array}\right)$$

Relative pronouns are represented by *relpron* and can be either “that”, “who” or “which”:

$$(83)\quad \mathit{relpron}\left(\text{relpron: that}\right) \xrightarrow{:} [\,\text{that}\,]$$

$$(\underline{84})\quad \mathit{relpron}\left(\begin{array}{l}\text{human: }+\\ \text{relpron: who}\end{array}\right) \xrightarrow{:} [\,\text{who}\,]$$

$$(85)\quad \mathit{relpron}\left(\begin{array}{l}\text{human: }-\\ \text{relpron: which}\end{array}\right) \xrightarrow{:} [\,\text{which}\,]$$

The categories *or_relpron* and *and_relpron* define shortcuts — e.g. “or that” as one token — for better usability inside of the predictive editor:

$$(86)\quad \mathit{or\_relpron}\left(\begin{array}{l}\text{human: }\boxed{1}\\ \text{relpron: }\boxed{2}\end{array}\right) \xrightarrow{:} [\,\text{or}\,]\ \mathit{relpron}\left(\begin{array}{l}\text{human: }\boxed{1}\\ \text{relpron: }\boxed{2}\end{array}\right)$$

$$(87)\quad \mathit{or\_relpron}\left(\text{relpron: that}\right) \xrightarrow{:} [\,\text{or that}\,]$$

$$(88)\quad \mathit{or\_relpron}\left(\begin{array}{l}\text{human: }+\\ \text{relpron: who}\end{array}\right) \xrightarrow{:} [\,\text{or who}\,]$$

(89) $$\mathit{or\_relpron}\begin{pmatrix}\text{human: –}\\ \text{relpron: which}\end{pmatrix} \overset{:}{\longrightarrow} [\,\text{or which}\,]$$

(90) $$\mathit{and\_relpron}\begin{pmatrix}\text{human: }\boxed{1}\\ \text{relpron: }\boxed{2}\end{pmatrix} \overset{:}{\longrightarrow} [\,\text{and}\,]\quad \mathit{relpron}\begin{pmatrix}\text{human: }\boxed{1}\\ \text{relpron: }\boxed{2}\end{pmatrix}$$

(91) $$\mathit{and\_relpron}\begin{pmatrix}\text{relpron: that}\end{pmatrix} \overset{:}{\longrightarrow} [\,\text{and that}\,]$$

(92) $$\mathit{and\_relpron}\begin{pmatrix}\text{human: +}\\ \text{relpron: who}\end{pmatrix} \overset{:}{\longrightarrow} [\,\text{and who}\,]$$

(93) $$\mathit{and\_relpron}\begin{pmatrix}\text{human: –}\\ \text{relpron: which}\end{pmatrix} \overset{:}{\longrightarrow} [\,\text{and which}\,]$$

Verb Phrase Modifiers

Verb phrase modifiers are represented by *vmod* and the auxiliary category *vmod_x*, and are always optional:

(94) $$\mathit{vmod}\begin{pmatrix}\text{whin: }\boxed{1}\\ \text{whout: }\boxed{1}\end{pmatrix} \overset{:}{\longrightarrow}$$

(95) $$\mathit{vmod}\begin{pmatrix}\text{copula: }\boxed{1}\\ \text{embv: –}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \overset{:}{\longrightarrow} \mathit{adv\_coord}\begin{pmatrix}\text{copula: }\boxed{1}\end{pmatrix}\quad \mathit{vmod\_x}\begin{pmatrix}\text{copula: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

(96) $$\mathit{vmod}\begin{pmatrix}\text{copula: }\boxed{1}\\ \text{embv: –}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \overset{:}{\longrightarrow} \mathit{pp}\begin{pmatrix}\text{embv: }\boxed{6}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{7}\end{pmatrix}\quad \mathit{vmod}\begin{pmatrix}\text{copula: }\boxed{1}\\ \text{embv: }\boxed{6}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{7}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

(97) $$\mathit{vmod\_x}\begin{pmatrix}\text{whin: }\boxed{1}\\ \text{whout: }\boxed{1}\end{pmatrix} \overset{:}{\longrightarrow}$$

(98) $$\mathit{vmod\_x}\begin{pmatrix}\text{copula: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \overset{:}{\longrightarrow} \mathit{pp}\begin{pmatrix}\text{embv: }\boxed{6}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{7}\end{pmatrix}\quad \mathit{vmod}\begin{pmatrix}\text{copula: }\boxed{1}\\ \text{embv: }\boxed{6}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{7}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

The category *pp* represents prepositional phrases:

(99) $$\mathit{pp}\begin{pmatrix}\text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix} \overset{:}{\longrightarrow} \underline{\mathit{prep}}\quad \mathit{np}\begin{pmatrix}\text{case: acc}\\ \text{embv: }\boxed{1}\\ \text{rel: }\boxed{2}\\ \text{subj: }\boxed{3}\\ \text{whin: }\boxed{4}\\ \text{whout: }\boxed{5}\end{pmatrix}$$

Adverbs can be coordinated by “and”, which is represented by *adv_coord*:

(100) $$\mathit{adv\_coord}\begin{pmatrix}\text{copula: –}\end{pmatrix} \overset{:}{\longrightarrow} \mathit{adv\_phrase}$$

(101) $$\mathit{adv\_coord}\begin{pmatrix}\text{copula: –}\end{pmatrix} \overset{:}{\longrightarrow} \mathit{adv\_phrase}\quad [\,\text{and}\,]\quad \mathit{adv\_coord}$$

Adverbial phrases are represented by *adv_phrase*, and can be in positive, comparative or superlative form:

(102) $$\mathit{adv\_phrase} \overset{:}{\longrightarrow} \underline{\mathit{adv}}$$

(103) $\mathit{adv\_phrase} \overset{:}{\longrightarrow} [\,\text{more}\,]\ \ \underline{\mathit{adv}}$

(104) $\mathit{adv\_phrase} \overset{:}{\longrightarrow} \underline{\mathit{adv\_comp}}$

(105) $\mathit{adv\_phrase} \overset{:}{\longrightarrow} [\,\text{most}\,]\ \ \underline{\mathit{adv}}$

(106) $\mathit{adv\_phrase} \overset{:}{\longrightarrow} \underline{\mathit{adv\_sup}}$

Verbs

The category *verb* represents main verbs, which can be intransitive or transitive:

($\underline{107}$) $\mathit{verb}\begin{pmatrix}\text{be: }-\\ \text{pl: }-\\ \text{vcat: itr}\\ \text{vform: fin}\end{pmatrix} \overset{:}{\longrightarrow} \underline{\mathit{iv\_finsg}}$

($\underline{108}$) $\mathit{verb}\begin{pmatrix}\text{be: }-\\ \text{pl: }+\\ \text{vcat: itr}\\ \text{vform: fin}\end{pmatrix} \overset{:}{\longrightarrow} \underline{\mathit{iv\_infpl}}$

($\underline{109}$) $\mathit{verb}\begin{pmatrix}\text{be: }-\\ \text{vcat: itr}\\ \text{vform: inf}\end{pmatrix} \overset{:}{\longrightarrow} \underline{\mathit{iv\_infpl}}$

($\underline{110}$) $\mathit{verb}\begin{pmatrix}\text{be: }-\\ \text{pl: }-\\ \text{vcat: tr}\\ \text{vform: fin}\end{pmatrix} \overset{:}{\longrightarrow} \underline{\mathit{tv\_finsg}}$

($\underline{111}$) $\mathit{verb}\begin{pmatrix}\text{be: }-\\ \text{pl: }+\\ \text{vcat: tr}\\ \text{vform: fin}\end{pmatrix} \overset{:}{\longrightarrow} \underline{\mathit{tv\_infpl}}$

($\underline{112}$) $\mathit{verb}\begin{pmatrix}\text{be: }-\\ \text{vcat: tr}\\ \text{vform: inf}\end{pmatrix} \overset{:}{\longrightarrow} \underline{\mathit{tv\_infpl}}$

($\underline{113}$) $\mathit{verb}\begin{pmatrix}\text{be: }+\\ \text{vcat: tr}\end{pmatrix} \overset{:}{\longrightarrow} \underline{\mathit{tv\_pp}}$

Auxiliary verbs are represented by *aux*, which includes negation markers:

($\underline{114}$) $\mathit{aux}\begin{pmatrix}\text{be: }+\\ \text{exist: }+\\ \text{pl: }-\end{pmatrix} \overset{:}{\longrightarrow} [\,\text{is}\,]$

(115) $\mathit{aux}\begin{pmatrix}\text{be: }+\\ \text{exist: }-\\ \text{pl: }-\end{pmatrix} \overset{:}{\longrightarrow} /\!\!/\ \ [\,\text{is not}\,]$

($\underline{116}$) $\mathit{aux}\begin{pmatrix}\text{be: }+\\ \text{exist: }-\\ \text{pl: }-\end{pmatrix} \overset{:}{\longrightarrow} /\!\!/\ \ [\,\text{is}\,]\ \ [\,\text{not}\,]$

($\underline{117}$) $\mathit{aux}\begin{pmatrix}\text{be: }+\\ \text{exist: }+\\ \text{pl: }+\end{pmatrix} \overset{:}{\longrightarrow} [\,\text{are}\,]$

(118) $\mathit{aux}\begin{pmatrix}\text{be: }+\\ \text{exist: }-\\ \text{pl: }+\end{pmatrix} \overset{:}{\longrightarrow} /\!\!/\ \ [\,\text{are not}\,]$

($\underline{119}$) $\mathit{aux}\begin{pmatrix}\text{be: }+\\ \text{exist: }-\\ \text{pl: }+\end{pmatrix} \overset{:}{\longrightarrow} /\!\!/\ \ [\,\text{are}\,]\ \ [\,\text{not}\,]$

($\underline{120}$) $aux\begin{pmatrix}\text{be: }-\\ \text{exist: }-\\ \text{pl: }-\end{pmatrix} \xrightarrow{:}$ // [does not]

($\underline{121}$) $aux\begin{pmatrix}\text{be: }-\\ \text{exist: }-\\ \text{pl: }+\end{pmatrix} \xrightarrow{:}$ // [do not]

(122) $aux\begin{pmatrix}\text{be: }-\\ \text{exist: }-\end{pmatrix} \xrightarrow{:}$ // [can]

(123) $aux\begin{pmatrix}\text{be: }-\\ \text{exist: }-\end{pmatrix} \xrightarrow{:}$ // [should]

(124) $aux\begin{pmatrix}\text{be: }-\\ \text{exist: }-\end{pmatrix} \xrightarrow{:}$ // [must]

(125) $aux\begin{pmatrix}\text{be: }-\\ \text{exist: }-\\ \text{pl: }-\end{pmatrix} \xrightarrow{:}$ // [has to]

(126) $aux\begin{pmatrix}\text{be: }-\\ \text{exist: }-\\ \text{pl: }+\end{pmatrix} \xrightarrow{:}$ // [have to]

(127) $aux\begin{pmatrix}\text{be: }+\\ \text{exist: }-\end{pmatrix} \xrightarrow{:}$ // [can] [be]

(128) $aux\begin{pmatrix}\text{be: }+\\ \text{exist: }-\end{pmatrix} \xrightarrow{:}$ // [should] [be]

(129) $aux\begin{pmatrix}\text{be: }+\\ \text{exist: }-\end{pmatrix} \xrightarrow{:}$ // [must] [be]

(130) $aux\begin{pmatrix}\text{be: }+\\ \text{exist: }-\\ \text{pl: }-\end{pmatrix} \xrightarrow{:}$ // [has to] [be]

(131) $aux\begin{pmatrix}\text{be: }+\\ \text{exist: }-\\ \text{pl: }+\end{pmatrix} \xrightarrow{:}$ // [have to] [be]

(132) $aux\begin{pmatrix}\text{be: }+\\ \text{exist: }-\end{pmatrix} \xrightarrow{:}$ // [cannot] [be]

(133) $aux\begin{pmatrix}\text{be: }+\\ \text{exist: }-\end{pmatrix} \xrightarrow{:}$ // [can] [not] [be]

(134) $aux\begin{pmatrix}\text{be: }+\\ \text{exist: }-\end{pmatrix} \xrightarrow{:}$ // [should] [not] [be]

(135) $aux\begin{pmatrix}\text{be: }+\\ \text{exist: }-\\ \text{pl: }-\end{pmatrix} \xrightarrow{:}$ // [does not] [have to] [be]

(136) $aux\begin{pmatrix}\text{be: }+\\ \text{exist: }-\\ \text{pl: }+\end{pmatrix} \xrightarrow{:}$ // [do not] [have to] [be]

(137) $aux\begin{pmatrix}\text{be: }-\\ \text{exist: }-\\ \text{pl: }-\end{pmatrix} \xrightarrow{:}$ // [cannot]

(138) $aux\begin{pmatrix}\text{be: }-\\ \text{exist: }-\\ \text{pl: }-\end{pmatrix} \xrightarrow{:}$ // [can] [not]

(139) $aux\begin{pmatrix}\text{be: }-\\ \text{exist: }-\\ \text{pl: }-\end{pmatrix} \xrightarrow{:}$ // [should] [not]

(140) $aux\begin{pmatrix}\text{be: }-\\ \text{exist: }-\\ \text{pl: }-\end{pmatrix} \overset{:}{\longrightarrow} /\!\!/ \quad [\,\text{does not}\,] \quad [\,\text{have to}\,]$

(141) $aux\begin{pmatrix}\text{be: }-\\ \text{exist: }-\\ \text{pl: }+\end{pmatrix} \overset{:}{\longrightarrow} /\!\!/ \quad [\,\text{do not}\,] \quad [\,\text{have to}\,]$

Quantifiers

Existential and universal quantifiers are represented by *quant*:

($\underline{142}$) $quant\begin{pmatrix}\text{exist: }+\end{pmatrix} \overset{:}{\longrightarrow} [\,\text{a}\,]$

(143) $quant\begin{pmatrix}\text{exist: }+\end{pmatrix} \overset{:}{\longrightarrow} [\,\text{an}\,]$

($\underline{144}$) $quant\begin{pmatrix}\text{exist: }-\end{pmatrix} \overset{:}{\longrightarrow} /\!\!/ \quad [\,\text{every}\,]$

(145) $quant\begin{pmatrix}\text{exist: }-\end{pmatrix} \overset{:}{\longrightarrow} /\!\!/ \quad [\,\text{no}\,]$

The category *num_quant* stands for numerical quantifiers:

(146) $num\_quant \overset{:}{\longrightarrow} [\,\text{at least}\,]$

(147) $num\_quant \overset{:}{\longrightarrow} [\,\text{at most}\,]$

(148) $num\_quant \overset{:}{\longrightarrow} [\,\text{less than}\,]$

(149) $num\_quant \overset{:}{\longrightarrow} [\,\text{more than}\,]$

($\underline{150}$) $num\_quant \overset{:}{\longrightarrow} [\,\text{exactly}\,]$

Indefinite Pronouns

Indefinite pronouns are represented by *ipron*:

(151) $ipron\begin{pmatrix}\text{exist: }+\\ \text{human: }-\end{pmatrix} \rightarrow [\,\text{something}\,]$

($\underline{152}$) $ipron\begin{pmatrix}\text{exist: }+\\ \text{human: }+\end{pmatrix} \rightarrow [\,\text{somebody}\,]$

(153) $ipron\begin{pmatrix}\text{exist: }-\\ \text{human: }-\end{pmatrix} \rightarrow /\!\!/ \quad [\,\text{everything}\,]$

($\underline{154}$) $ipron\begin{pmatrix}\text{exist: }-\\ \text{human: }+\end{pmatrix} \rightarrow /\!\!/ \quad [\,\text{everybody}\,]$

(155) $ipron\begin{pmatrix}\text{exist: }-\\ \text{human: }-\end{pmatrix} \rightarrow /\!\!/ \quad [\,\text{nothing}\,]$

(156) $ipron\begin{pmatrix}\text{exist: }-\\ \text{human: }+\end{pmatrix} \rightarrow /\!\!/ \quad [\,\text{nobody}\,]$

Anaphoric Pronouns

The category *pron* represents reflexive and irreflexive anaphoric pronouns:

(157) $\underline{\mathit{pron}}\begin{pmatrix}\text{human: –}\\ \text{refl: +}\end{pmatrix} \rightarrow [\,\text{itself}\,]$

(158) $\underline{\mathit{pron}}\begin{pmatrix}\text{gender: masc}\\ \text{human: +}\\ \text{refl: +}\end{pmatrix} \rightarrow [\,\text{himself}\,]$

(159) $\underline{\mathit{pron}}\begin{pmatrix}\text{gender: fem}\\ \text{human: +}\\ \text{refl: +}\end{pmatrix} \rightarrow [\,\text{herself}\,]$

(160) $\underline{\mathit{pron}}\begin{pmatrix}\text{human: –}\\ \text{refl: –}\end{pmatrix} \rightarrow [\,\text{it}\,]$

(161) $\underline{\mathit{pron}}\begin{pmatrix}\text{case: nom}\\ \text{gender: masc}\\ \text{human: +}\\ \text{refl: –}\end{pmatrix} \rightarrow [\,\text{he}\,]$

(162) $\underline{\mathit{pron}}\begin{pmatrix}\text{case: acc}\\ \text{gender: masc}\\ \text{human: +}\\ \text{refl: –}\end{pmatrix} \rightarrow [\,\text{him}\,]$

(163) $\underline{\mathit{pron}}\begin{pmatrix}\text{case: nom}\\ \text{gender: fem}\\ \text{human: +}\\ \text{refl: –}\end{pmatrix} \rightarrow [\,\text{she}\,]$

(164) $\underline{\mathit{pron}}\begin{pmatrix}\text{case: acc}\\ \text{gender: fem}\\ \text{human: +}\\ \text{refl: –}\end{pmatrix} \rightarrow [\,\text{her}\,]$

A.3 Lexical Rules of ACE Codeco

Below, the lexical rules (i.e. lexicon entries) are shown, which are used for evaluation purposes.

(165) $\underline{\mathit{prop\_sg}}\begin{pmatrix}\text{gender: fem}\\ \text{human: +}\\ \text{text: Mary}\end{pmatrix} \rightarrow [\,\text{Mary}\,]$

(166) $\underline{\mathit{def\_noun\_sg}}\begin{pmatrix}\text{noun: woman}\end{pmatrix} \rightarrow [\,\text{the woman}\,]$

(167) $\underline{\mathit{ref}}\begin{pmatrix}\text{text: X}\end{pmatrix} \rightarrow [\,\text{X}\,]$

(168) $\underline{\mathit{num}} \rightarrow [\,2\,]$

(169) $\underline{\mathit{noun\_pl}} \rightarrow [\,\text{women}\,]$

(170) $\underline{\mathit{noun\_sg}}\begin{pmatrix}\text{gender: fem}\\ \text{human: +}\\ \text{text: woman}\end{pmatrix} \rightarrow [\,\text{woman}\,]$

(171) $\underline{\mathit{var}}\begin{pmatrix}\text{text: X}\end{pmatrix} \rightarrow [\,\text{X}\,]$

(172) $\underline{\mathit{iv\_finsg}} \rightarrow [\,\text{waits}\,]$

(173) $\underline{\mathit{iv\_infpl}} \rightarrow [\,\text{wait}\,]$

(174) *tv_finsg* → [asks]

(175) *tv_infpl* → [ask]

(176) *tv_pp* → [asked]

(177) *adj_itr* → [young]

(178) *adj_itr_comp* → [younger]

(179) *adj_tr*(prep: about) → [mad-about]

(180) *adj_tr_comp*(prep: about) → [madder-about]

(181) *adj_prep*(prep: about) → [about]

(182) *prep* → [for]

(183) *adv* → [early]

APPENDIX B

Ontograph Resources

This appendix chapter shows the resources that have been used for the two experiments described in Chapter 5, i.e. the ontographs, the corresponding statements, and (in the case of the second experiment) the language description sheets. First, the resources for the first experiment are shown (Section B.1), and then the ones for the second experiment (Section B.2).

The resources shown in this appendix chapter can also be found on the web[1] in different formats. All these resources can be reused freely under the terms of the Creative Commons Attribution License[2]. Thus, everyone is free to use and modify the shown ontographs if the work is properly attributed. In an experimental setting, the attribution can be dropped.

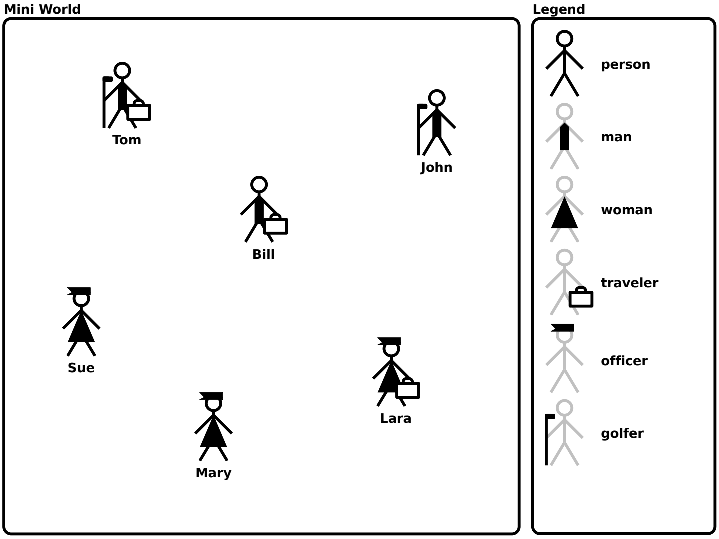
B.1 Resources of the first Ontograph Experiment

The first ontograph experiment used four ontographs 1X, 2X, 3X and 4X. Each ontograph has two series of statements (a and b) each of which consists of 10 statements in ACE. Only series a has been used for the experiment. Some of the statement are true and have a plus sign (+) in the identifier. The others are false and their identifiers have a minus sign (−).

[1] http://attempto.ifi.uzh.ch/site/docs/ontograph/

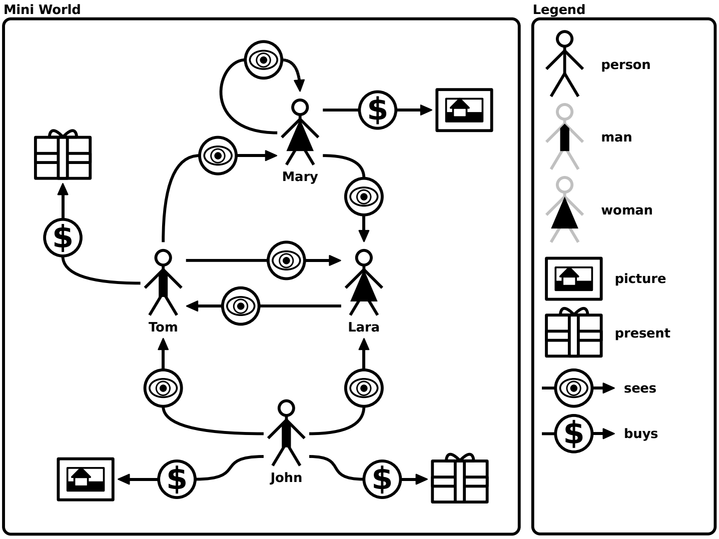
[2] http://creativecommons.org/licenses/by/3.0/

Ontograph Table 1X



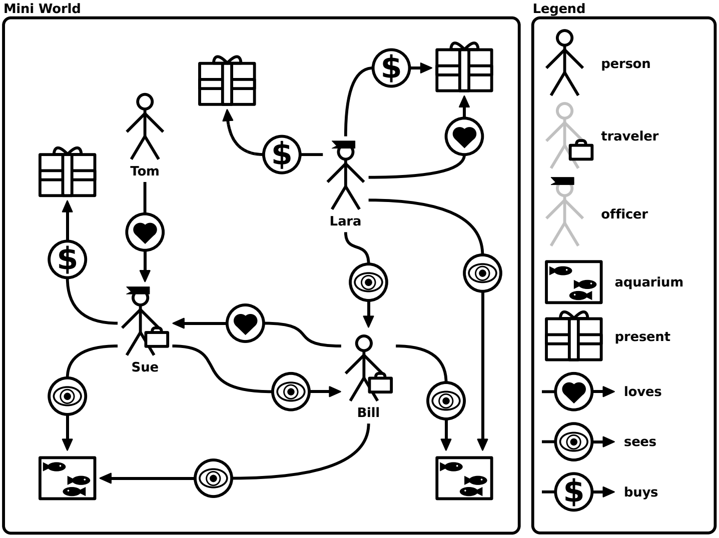
ID	ACE
1a−	Mary is a traveler.
1b+	John is a golfer.
2a+	Bill is not a golfer.
2b−	Lara is not an officer.
3a+	Mary is an officer or is a woman.
3b−	John is a woman or is a traveler.
4a−	Sue is an officer and is a traveler.
4b+	Tom is a man and is a golfer.
5a−	Every traveler is a man.
5b+	Every officer is a woman.
6a+	No golfer is a woman.
6b−	No traveler is a golfer.
7a+	Every woman is an officer and every officer is a woman.
7b−	Every golfer is a man and every man is a golfer.
8a−	Every traveler who is not a woman is a golfer.
8b+	Every man who is not a golfer is a traveler.
9a+	Every man is a golfer or is a traveler.
9b−	Every traveler is a golfer or is an officer.
10a+	Every woman who is a golfer is a traveler.
10b+	Every officer who is a man is a golfer.

Ontograph Table 2X



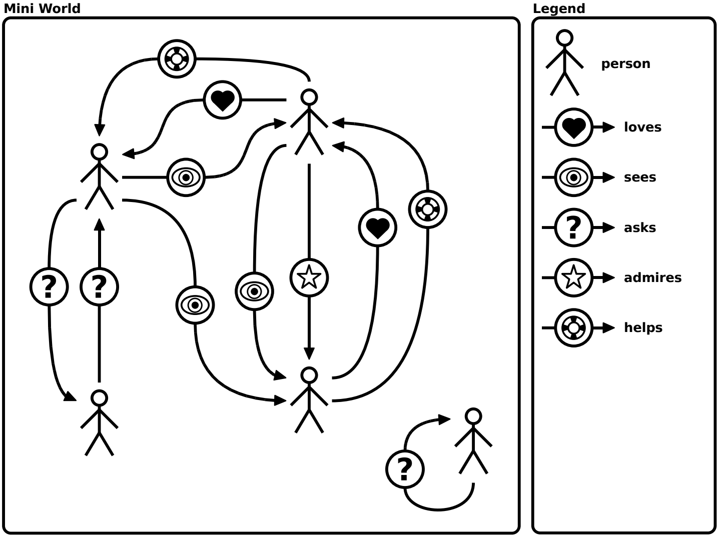
ID	ACE
1a+	John sees Tom.
1b−	Lara sees Mary.
2a+	Mary does not see Tom.
2b−	Tom does not see Lara.
3a−	Tom buys a picture.
3b+	John buys a present.
4a−	John sees no woman.
4b+	Mary sees no man.
5a+	Tom sees every woman.
5b−	Lara sees every man.
6a+	Tom sees nothing but women.
6b−	John sees nothing but men.
7a+	Lara buys nothing but presents.
7b+	Lara buys nothing but pictures.
8a−	No woman sees herself.
8b+	No man sees himself.
9a+	Every woman buys nothing but pictures.
9b−	Every man buys nothing but presents.
10a+	No man who buys a picture is seen by a woman.
10b−	No woman who buys a picture is seen by a man.

Ontograph Table 3X



ID	ACE
1a+	Everything that loves something is a person.
1b−	Everything that sees something is an officer.
2a−	Everything that is loved by something is a person.
2b+	Everything that is bought by something is a present.
3a+	Everything that sees something is an officer or is a traveler.
3b−	Everything that loves something is a traveler or is an officer.
4a−	Everything that is bought by something is an aquarium or is an officer.
4b+	Everything that is seen by something is a traveler or is an aquarium.
5a−	Everything buys at most 1 thing.
5b+	Everything loves at most 1 thing.
6a+	Everything is bought by at most 1 thing.
6b−	Everything is loved by at most 1 thing.
7a−	Lara sees at least 2 persons.
7b+	Bill sees at least 2 aquariums.
8a+	Bill is seen by at most 1 traveler.
8b−	Sue is loved by at most 1 person.
9a−	Every officer is loved by at least 2 persons.
9b+	Every aquarium is seen by at least 2 persons.
10a+	Every officer sees exactly 1 traveler.
10b−	Every traveler loves exactly 1 person.

Ontograph Table 4X



ID	ACE
1a+	If X asks Y then Y asks X.
1b−	If X helps Y then Y helps X.
2a+	If X sees Y then Y does not see X.
2b−	If X asks Y then Y does not ask X.
3a−	Nothing asks itself.
3b+	Nothing sees itself.
4a−	If X loves something that loves Y then X loves Y.
4b+	If X sees something that sees Y then X sees Y.
5a+	If X admires Y then X sees Y.
5b−	If X sees Y then X admires Y.
6a−	If X helps Y then Y admires X.
6b+	If X loves Y then Y sees X.
7a−	If X admires Y then X does not see Y.
7b+	If X loves Y then X does not admire Y.
8a+	If X admires Y then Y does not see X.
8b−	If X sees Y then Y does not love X.
9a−	If X admires Y then X sees Y. If X sees Y then X admires Y.
9b+	If X loves Y then X helps Y. If X helps Y then X loves Y.
10a+	If X sees something that admires Y then X sees Y.
10b−	If X asks something that sees Y then X asks Y.

B.2 Resources of the second Ontograph Experiment

The second ontograph experiment used four series of ontographs (1, 2, 3 and 4) each consisting of three ontographs (A, B and C). Each ontograph has two series of statements (a and b) each of which consists of 10 statements. These statements are built according to certain patterns, and each statement is expressed in ACE and in MLL. Some of the statement are true and have a plus sign (+) in the identifier. The others are false and their identifiers have a minus sign (−). The ontographs B and C of each series and the respective statements are structurally equivalent in the sense that they can be derived from each other by one-to-one replacements of the names of the individuals, types and relations.

The participants of the experiment received a printed language description sheet for ACE and another one for MLL. Because these description sheets only describe the subset of the language that was used for the given series, each series has its own description sheets. For the sake of neutrality, ACE was called "language A" and MLL was called "language B". The description sheets used for the experiment were in German.

In the following Sections B.2.1 to B.2.4, the different series are introduced and their statement patterns are listed. English translations of the used language description sheets are shown for each series. Furthermore, the ontographs are shown, together with their statements in ACE and MLL. Section B.2.5, finally, shows the questionnaire that the participants had to fill out after the experiment.

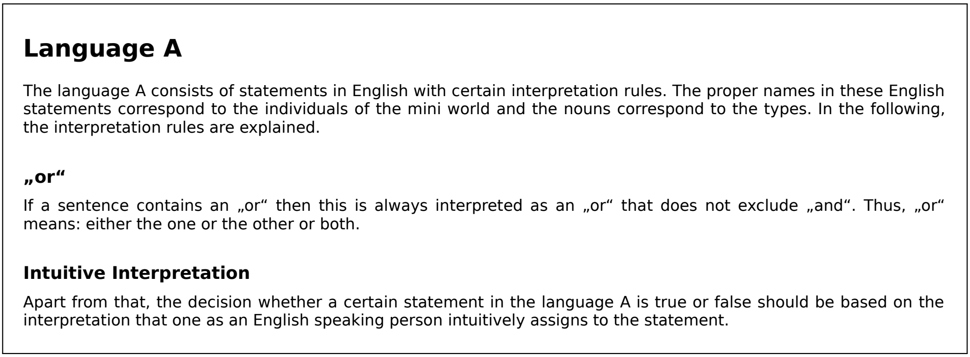
B.2.1 Ontograph Series 1

The statements of this series only contain individuals and types, but no relations.

Statement patterns

ID	Name	ACE Pattern	MLL Pattern
1	individual type	$\langle I \rangle$ is a $\langle T \rangle$.	$\langle I \rangle$ **HasType** $\langle T \rangle$
2	negative individual type	$\langle I \rangle$ is not a $\langle T \rangle$.	$\langle I \rangle$ **HasType** **not** $\langle T \rangle$
3	disjunctive individual type	$\langle I \rangle$ is a $\langle T_1 \rangle$ or is a $\langle T_2 \rangle$.	$\langle I \rangle$ **HasType** $\langle T_1 \rangle$ **or** $\langle T_2 \rangle$
4	conjunctive individual type	$\langle I \rangle$ is a $\langle T_1 \rangle$ and is a $\langle T_2 \rangle$.	$\langle I \rangle$ **HasType** $\langle T_1 \rangle$ **and** $\langle T_2 \rangle$
5	subtype	Every $\langle T_1 \rangle$ is a $\langle T_2 \rangle$.	$\langle T_1 \rangle$ **SubTypeOf** $\langle T_2 \rangle$
6	disjoint types	No $\langle T_1 \rangle$ is a $\langle T_2 \rangle$.	$\langle T_1 \rangle$ **DisjointWith** $\langle T_2 \rangle$
7	equivalent types	Every $\langle T_1 \rangle$ is a $\langle T_2 \rangle$ and every $\langle T_2 \rangle$ is a $\langle T_1 \rangle$.	$\langle T_1 \rangle$ **EquivalentTo** $\langle T_2 \rangle$
8	complex subtype	Every $\langle T_1 \rangle$ who is not a $\langle T_2 \rangle$ is a $\langle T_3 \rangle$.	$\langle T_1 \rangle$ **and** (**not** $\langle T_2 \rangle$) **SubTypeOf** $\langle T_3 \rangle$
9	complex supertype	Every $\langle T_1 \rangle$ is a $\langle T_2 \rangle$ or is a $\langle T_3 \rangle$.	$\langle T_1 \rangle$ **SubTypeOf** $\langle T_2 \rangle$ **or** $\langle T_3 \rangle$
10	complex subtype/supertype	Nobody who is a $\langle T_1 \rangle$ or who is a $\langle T_2 \rangle$ is a $\langle T_3 \rangle$ and is a $\langle T_4 \rangle$.	$\langle T_1 \rangle$ **or** $\langle T_2 \rangle$ **SubTypeOf** **not** ($\langle T_3 \rangle$ **and** $\langle T_4 \rangle$)

ACE Description Sheet



Language A

The language A consists of statements in English with certain interpretation rules. The proper names in these English statements correspond to the individuals of the mini world and the nouns correspond to the types. In the following, the interpretation rules are explained.

„or“

If a sentence contains an „or“ then this is always interpreted as an „or“ that does not exclude „and“. Thus, „or“ means: either the one or the other or both.

Intuitive Interpretation

Apart from that, the decision whether a certain statement in the language A is true or false should be based on the interpretation that one as an English speaking person intuitively assigns to the statement.

MLL Description Sheet

Language B

The language B consists of statements of the forms described and explained below. These statements are composed of keywords and of the names of the individuals and types of the respective mini world. The used keywords are „HasType“, „SubTypeOf“, „DisjointWith“, „EquivalentTo“, „not“, „and“, and „or“.

Statements

Every statement can either be true or false. Every statement of the language B has the form of one of the four schemes described here. Note that types can be complex (see the next section).

HasType-statements	
scheme:	*Individual* ***HasType*** *Type*
example:	John **HasType** golfer
explanation:	A HasType-statement requires an individual and a type and it states that the given individual belongs to the given type. The example above states that John is a golfer.

DisjointWith-statements	
scheme:	*Type1* **DisjointWith** *Type2*
example:	woman **DisjointWith** golfer
explanation:	A DisjointWith-statement requires two types and states that no individual belongs to the first type and at the same time to the second type. The example above states that no individual is a woman and is a golfer.

SubTypeOf-statements	
scheme:	*Type1* **SubTypeOf** *Type2*
example:	golfer **SubTypeOf** man
explanation:	A SubTypeOf-statement requires two types and states that every individual that belongs to the first type also belongs to the second type (but not necessarily the other way round). The example above states that every individual that is a golfer is also a man.

EquivalentTo-statements	
scheme:	*Type1* **EquivalentTo** *Type2*
example:	golfer **EquivalentTo** man
explanation:	An EquivalentTo-statement requires two types and states that every individual that belongs to the first type also belongs to the second type and vice versa. The example above states that every individual that is a golfer is also a man and vice versa.

Type Operators

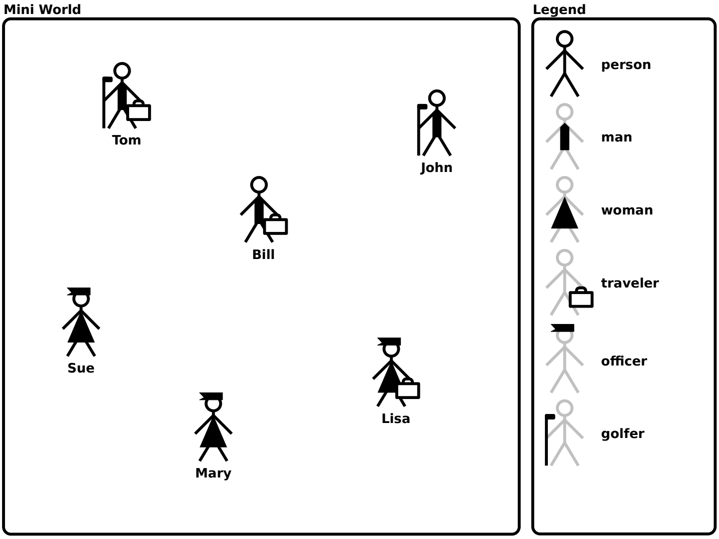
Every type (simple or complex) stands for a certain group of individuals. „woman“ and „golfer“ are examples of simple types. Apart from simple types, there are complex types that are composed by the type operators described here. „woman **or** golfer“ is an example of a complex type. Note that such complex types can be nested. In this case, parentheses are used to clarify the structure, for example „**not** (golfer **and** man)“.

not-operator	
scheme:	**not** *Type*
example:	**not** golfer
explanation:	The not-operator requires just one type. The resulting complex type stands for all individuals that do not belong to the given type. The example above stands for all individuals that are not golfers.

or-operator	
scheme:	*Type1* **or** *Type2*
example:	woman **or** golfer
explanation:	The or-operator requires two types. The resulting complex type stands for all individuals that belong to the first type or to the second type or to both. The example above stands for all individuals that are women or golfers or both.

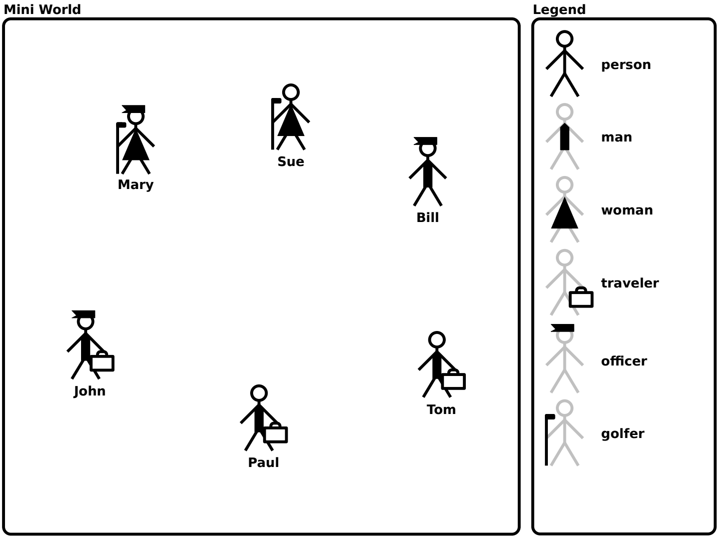
and-operator	
scheme:	*Type1* **and** *Type2*
example:	golfer **and** man
explanation:	The and-operator requires two types. The resulting complex type stands for all individuals that belong to the first type and at the same time to the second type. The example above stands for all individuals that are golfers and men.

Ontograph Table 1A



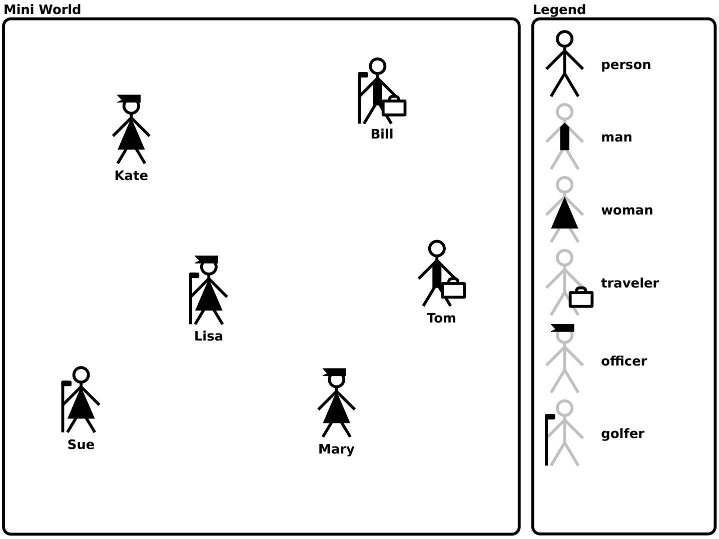
ID	ACE	MLL
1a−	Mary is a traveler.	Mary **HasType** traveler
1b+	John is a golfer.	John **HasType** golfer
2a+	Bill is not a golfer.	Bill **HasType** **not** golfer
2b−	Lisa is not an officer.	Lisa **HasType** **not** officer
3a+	Mary is an officer or is a golfer.	Mary **HasType** officer **or** golfer
3b−	John is a woman or is a traveler.	John **HasType** woman **or** traveler
4a−	Sue is an officer and is a traveler.	Sue **HasType** officer **and** traveler
4b+	Tom is a man and is a golfer.	Tom **HasType** man **and** golfer
5a−	Every man is a golfer.	man **SubTypeOf** golfer
5b+	Every golfer is a man.	golfer **SubTypeOf** man
6a+	No golfer is a woman.	golfer **DisjointWith** woman
6b−	No traveler is a golfer.	traveler **DisjointWith** golfer
7a+	Every woman is an officer and every officer is a woman.	woman **EquivalentTo** officer
7b−	Every golfer is a man and every man is a golfer.	golfer **EquivalentTo** man
8a−	Every traveler who is not a woman is a golfer.	traveler **and** (**not** woman) **SubTypeOf** golfer
8b+	Every man who is not a golfer is a traveler.	man **and** (**not** golfer) **SubTypeOf** traveler
9a+	Every man is a golfer or is a traveler.	man **SubTypeOf** golfer **or** traveler
9b−	Every traveler is a golfer or is an officer.	traveler **SubTypeOf** golfer **or** officer
10a+	Nobody who is a man or who is a golfer is an officer and is a traveler.	man **or** golfer **SubTypeOf** **not** (officer **and** traveler)
10b−	Nobody who is a traveler or who is an officer is a man and is a golfer.	traveler **or** officer **SubTypeOf** **not** (man **and** golfer)

Ontograph Table 1B



ID	ACE	MLL
1a+	John is a traveler.	John **HasType** traveler
1b−	Mary is a traveler.	Mary **HasType** traveler
2a−	Bill is not an officer.	Bill **HasType** **not** officer
2b+	Sue is not an officer.	Sue **HasType** **not** officer
3a−	Tom is an officer or is a golfer.	Tom **HasType** officer **or** golfer
3b+	Paul is a golfer or is a man.	Paul **HasType** golfer **or** man
4a−	Mary is a woman and is a traveler.	Mary **HasType** woman **and** traveler
4b+	John is a man and is a traveler.	John **HasType** man **and** traveler
5a−	Every man is a traveler.	man **SubTypeOf** traveler
5b+	Every traveler is a man.	traveler **SubTypeOf** man
6a+	No traveler is a golfer.	traveler **DisjointWith** golfer
6b−	No officer is a woman.	officer **DisjointWith** woman
7a+	Every woman is a golfer and every golfer is a woman.	woman **EquivalentTo** golfer
7b−	Every traveler is a man and every man is a traveler.	traveler **EquivalentTo** man
8a−	Every officer who is not a traveler is a golfer.	officer **and** (**not** traveler) **SubTypeOf** golfer
8b+	Every man who is not a traveler is an officer.	man **and** (**not** traveler) **SubTypeOf** officer
9a+	Every man is a traveler or is an officer.	man **SubTypeOf** traveler **or** officer
9b−	Every officer is a man or is a traveler.	officer **SubTypeOf** man **or** traveler
10a−	Nobody who is a golfer or who is a traveler is a man and is an officer.	golfer **or** traveler **SubTypeOf** **not** (man **and** officer)
10b+	Nobody who is a man or who is a traveler is an officer and is a golfer.	man **or** traveler **SubTypeOf** **not** (officer **and** golfer)

Ontograph Table 1C



ID	ACE	MLL
1a+	Lisa is an officer.	Lisa **HasType** officer
1b−	Bill is an officer.	Bill **HasType** officer
2a−	Sue is not a golfer.	Sue **HasType** **not** golfer
2b+	Tom is not a golfer.	Tom **HasType** **not** golfer
3a−	Mary is a golfer or is a traveler.	Mary **HasType** golfer **or** traveler
3b+	Kate is a traveler or is a woman.	Kate **HasType** traveler **or** woman
4a−	Bill is a man and is an officer.	Bill **HasType** man **and** officer
4b+	Lisa is a woman and is an officer.	Lisa **HasType** woman **and** officer
5a−	Every woman is an officer.	woman **SubTypeOf** officer
5b+	Every officer is a woman.	officer **SubTypeOf** woman
6a+	No officer is a traveler.	officer **DisjointWith** traveler
6b−	No golfer is a man.	golfer **DisjointWith** man
7a+	Every man is a traveler and every traveler is a man.	man **EquivalentTo** traveler
7b−	Every officer is a woman and every woman is an officer.	officer **EquivalentTo** woman
8a−	Every golfer who is not an officer is a traveler.	golfer **and** (**not** officer) **SubTypeOf** traveler
8b+	Every woman who is not an officer is a golfer.	woman **and** (**not** officer) **SubTypeOf** golfer
9a+	Every woman is an officer or is a golfer.	woman **SubTypeOf** officer **or** golfer
9b−	Every golfer is a woman or is an officer.	golfer **SubTypeOf** woman **or** officer
10a−	Nobody who is a traveler or who is an officer is a woman and is a golfer.	traveler **or** officer **SubTypeOf** **not** (woman **and** golfer)
10b+	Nobody who is a woman or who is an officer is a golfer and is a traveler.	woman **or** officer **SubTypeOf** **not** (golfer **and** traveler)

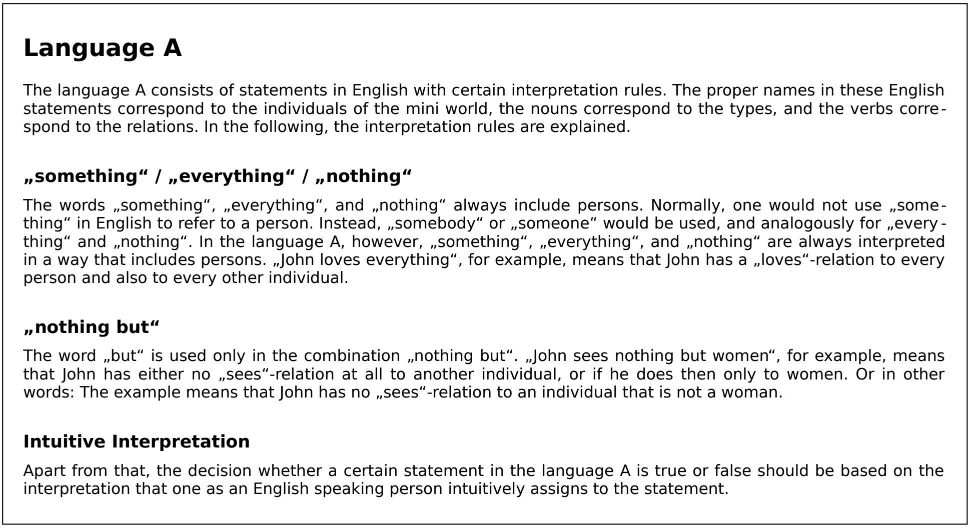
B.2.2 Ontograph Series 2

The statements of this series contain relations with different kinds of simple quantifications.

Statement patterns

ID	Name	ACE Pattern	MLL Pattern
1	relation instance	$\langle I_1 \rangle$ $\langle R \rangle$ $\langle I_2 \rangle$.	$\langle I_1 \rangle$ $\langle R \rangle$ $\langle I_2 \rangle$
2	negative relation instance	$\langle I_1 \rangle$ does not $\langle R \rangle$ $\langle I_2 \rangle$.	$\langle I_1 \rangle$ **not** $\langle R \rangle$ $\langle I_2 \rangle$
3	concrete existential statement	$\langle I \rangle$ $\langle R \rangle$ a $\langle T \rangle$.	$\langle I \rangle$ **HasType** $\langle R \rangle$ **some** $\langle T \rangle$
4	concrete negative statement 1	$\langle I \rangle$ $\langle R \rangle$ no $\langle T \rangle$.	$\langle I \rangle$ **HasType** **not** ($\langle R \rangle$ **some** $\langle T \rangle$)
5	concrete negative statement 2	$\langle I \rangle$ $\langle R \rangle$ something that is not a $\langle T \rangle$.	$\langle I \rangle$ **HasType** $\langle R \rangle$ **some** (**not** $\langle T \rangle$)
6	concrete exception statement	$\langle I \rangle$ $\langle R \rangle$ nothing but $\langle T \rangle$.	$\langle I \rangle$ **HasType** $\langle R \rangle$ **only** $\langle T \rangle$
7	general existential statement 1	Every $\langle T_1 \rangle$ $\langle R \rangle$ a $\langle T_2 \rangle$.	$\langle T_1 \rangle$ **SubTypeOf** $\langle R \rangle$ **some** $\langle T_2 \rangle$
8	general existential statement 2	Everything that $\langle R \rangle$ a $\langle T_1 \rangle$ is a $\langle T_2 \rangle$.	$\langle R \rangle$ **some** $\langle T_1 \rangle$ **SubTypeOf** $\langle T_2 \rangle$
9	general exception statement 1	Every $\langle T_1 \rangle$ $\langle R \rangle$ nothing but $\langle T_2 \rangle$.	$\langle T_1 \rangle$ **SubTypeOf** $\langle R \rangle$ **only** $\langle T_2 \rangle$
10	general exception statement 2	Everything that $\langle R \rangle$ nothing but $\langle T_1 \rangle$ is a $\langle T_2 \rangle$.	$\langle R \rangle$ **only** $\langle T_1 \rangle$ **SubTypeOf** $\langle T_2 \rangle$

ACE Description Sheet



Language A

The language A consists of statements in English with certain interpretation rules. The proper names in these English statements correspond to the individuals of the mini world, the nouns correspond to the types, and the verbs correspond to the relations. In the following, the interpretation rules are explained.

„something“ / „everything“ / „nothing“

The words „something“, „everything“, and „nothing“ always include persons. Normally, one would not use „something“ in English to refer to a person. Instead, „somebody“ or „someone“ would be used, and analogously for „everything“ and „nothing“. In the language A, however, „something“, „everything“, and „nothing“ are always interpreted in a way that includes persons. „John loves everything“, for example, means that John has a „loves“-relation to every person and also to every other individual.

„nothing but“

The word „but“ is used only in the combination „nothing but“. „John sees nothing but women“, for example, means that John has either no „sees“-relation at all to another individual, or if he does then only to women. Or in other words: The example means that John has no „sees“-relation to an individual that is not a woman.

Intuitive Interpretation

Apart from that, the decision whether a certain statement in the language A is true or false should be based on the interpretation that one as an English speaking person intuitively assigns to the statement.

MLL Description Sheet

Language B

The language B consists of statements of the forms described and explained below. These statements are composed of keywords and of the names of the individuals, types, and relations of the respective mini world. The used keywords are „HasType“, „SubTypeOf“, „not“, „some“, and „only“.

Statements

Every statement can either be true or false. Every statement of the language B has the form of one of the four schemes described here. Note that types can be complex (see the next section).

Positive simple statements	
scheme:	*Individual1* *Relation* *Individual2*
example:	John sees Mary
explanation:	A positive simple statement consists of two individuals and one relation and states that the first individual has the given relation to the second individual. The example above states that John has a „sees”-relation to Mary.

Negative simple statements	
scheme:	*Individual1* **not** *Relation* *Individual2*
example:	Mary **not** helps Bill
explanation:	A negative simple statement consists of two individuals and one relation where the relation is preceded by the keyword „not“. Such a statement states that the first individual does not have the given relation to the second individual. The example above states that Mary does not have a „helps“-relation to Bill.

HasType-statements	
scheme:	*Individual* ***HasType*** *Type*
example:	John **HasType** man
explanation:	A HasType-statement requires an individual and a type and it states that the given individual belongs to the given type. The example above states that John is a man.

SubTypeOf-statements	
scheme:	*Type1* **SubTypeOf** *Type2*
example:	golfer **SubTypeOf** man
explanation:	A SubTypeOf-statement requires two types and states that every individual that belongs to the first type also belongs to the second type (but not necessarily the other way round). The example above states that every individual that is a golfer is also a man.

Type Operators

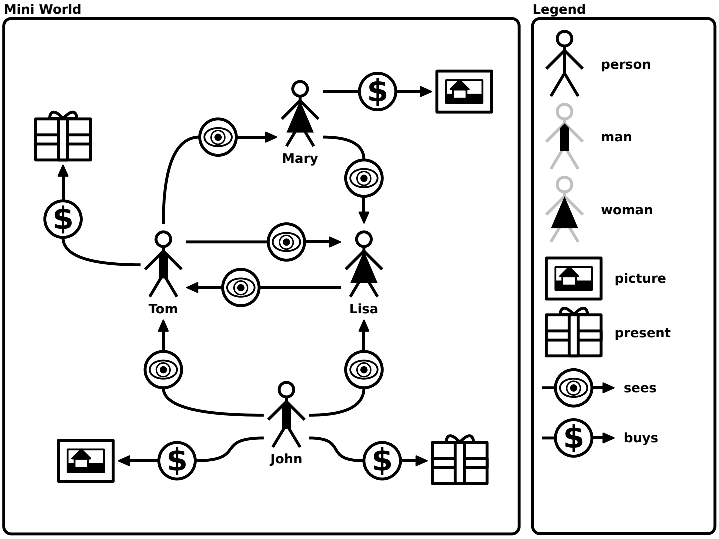
Every type (simple or complex) stands for a certain group of individuals. „woman“ and „golfer“ are examples of simple types. Apart from simple types, there are complex types that are composed by the type operators described here. „sees **only** golfer“ is an example of a complex type. Note that such complex types can be nested. In this case, parentheses are used to clarify the structure, for example „**not** (loves **some** woman)“.

not-operator	
scheme:	**not** *Type*
example:	**not** golfer
explanation:	The not-operator requires just one type. The resulting complex type stands for all individuals that do not belong to the given type. The example above stands for all individuals that are not golfers.

some-operator	
scheme:	*Relation* **some** *Type*
example:	loves **some** woman
explanation:	The some-operator requires a relation and a type. The resulting complex type stands for all individuals that have the given relation to at least one individual of the given type. The example above stands for all individuals that have a „loves“-relation to at least one woman.

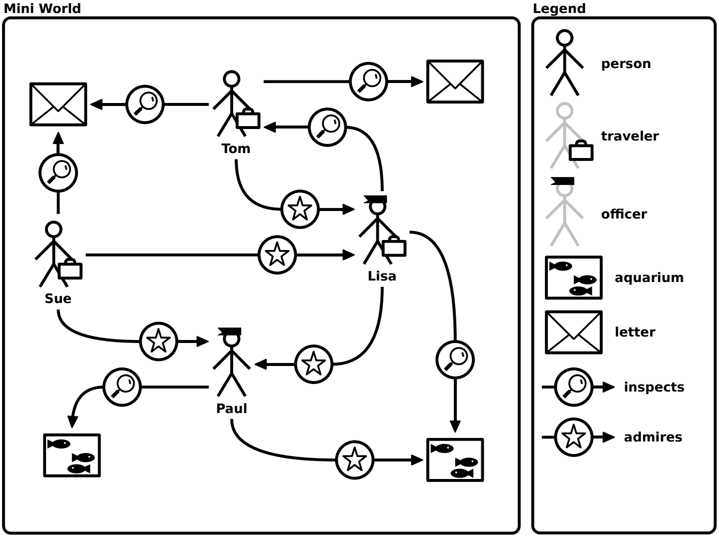
only-operator	
scheme:	*Relation* **only** *Type*
example:	helps **only** woman
explanation:	The only-operator requires a relation and a type. The resulting complex type stands for all individuals that either have no corresponding relation to another individual at all, or if they do then only to individuals of the given type. The example above stands for all individuals that have „helps“-relations (if present at all) only to women. Thus, the example includes all individuals except those that have a „helps“-relation to an individual that is not a woman.

Ontograph Table 2A



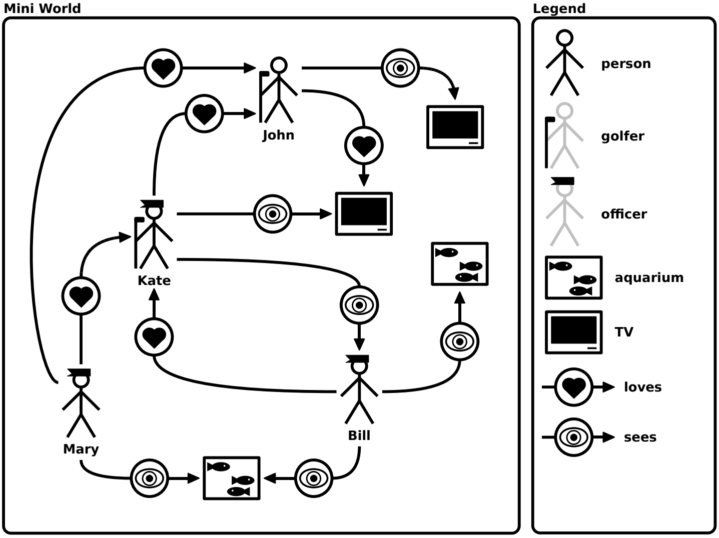
ID	ACE	MLL
1a−	Lisa sees Mary.	Lisa sees Mary
1b+	John sees Tom.	John sees Tom
2a+	Mary does not see Tom.	Mary **not** sees Tom
2b−	Tom does not see Lisa.	Tom **not** sees Lisa
3a−	Tom buys a picture.	Tom **HasType** buys **some** picture
3b+	John buys a present.	John **HasType** buys **some** present
4a+	Mary sees no man.	Mary **HasType** **not** (sees **some** man)
4b−	John sees no woman.	John **HasType** **not** (sees **some** woman)
5a+	John buys something that is not a present.	John **HasType** buys **some** (**not** present)
5b−	Tom sees something that is not a woman.	Tom **HasType** sees **some** (**not** woman)
6a−	John sees nothing but men.	John **HasType** sees **only** man
6b+	Tom sees nothing but women.	Tom **HasType** sees **only** woman
7a+	Every man buys a present.	man **SubTypeOf** buys **some** present
7b−	Every woman buys a picture.	woman **SubTypeOf** buys **some** picture
8a+	Everything that buys a present is a man.	buys **some** present **SubTypeOf** man
8b−	Everything that sees a woman is a man.	sees **some** woman **SubTypeOf** man
9a−	Every man buys nothing but presents.	man **SubTypeOf** buys **only** present
9b+	Every woman buys nothing but pictures.	woman **SubTypeOf** buys **only** picture
10a+	Everything that buys nothing but pictures is a woman.	buys **only** picture **SubTypeOf** woman
10b−	Everything that sees nothing but women is a man.	sees **only** woman **SubTypeOf** man

Ontograph Table 2B



ID	ACE	MLL
1a−	Tom inspects Lisa.	Tom inspects Lisa
1b+	Lisa inspects Tom.	Lisa inspects Tom
2a+	Lisa does not admire Tom.	Lisa **not** admires Tom
2b−	Tom does not admire Lisa.	Tom **not** admires Lisa
3a+	Lisa admires an officer.	Lisa **HasType** admires **some** officer
3b−	Lisa admires a traveler.	Lisa **HasType** admires **some** traveler
4a+	Paul admires no officer.	Paul **HasType** **not** (admires **some** officer)
4b−	Lisa inspects no traveler.	Lisa **HasType** **not** (inspects **some** traveler)
5a+	Sue admires something that is not a traveler.	Sue **HasType** admires **some** (**not** traveler)
5b−	Paul admires something that is not an aquarium.	Paul **HasType** admires **some** (**not** aquarium)
6a+	Tom inspects nothing but letters.	Tom **HasType** inspects **only** letter
6b−	Lisa inspects nothing but travelers.	Lisa **HasType** inspects **only** traveler
7a−	Every traveler inspects a letter.	traveler **SubTypeOf** inspects **some** letter
7b+	Every officer inspects an aquarium.	officer **SubTypeOf** inspects **some** aquarium
8a+	Everything that inspects an aquarium is an officer.	inspects **some** aquarium **SubTypeOf** officer
8b−	Everything that admires a person is an officer.	admires **some** person **SubTypeOf** officer
9a−	Every officer inspects nothing but aquariums.	officer **SubTypeOf** inspects **only** aquarium
9b+	Every traveler admires nothing but officers.	traveler **SubTypeOf** admires **only** officer
10a−	Everything that admires nothing but persons is an officer.	admires **only** person **SubTypeOf** officer
10b+	Everything that inspects nothing but letters is a traveler.	inspects **only** letter **SubTypeOf** traveler

Ontograph Table 2C



ID	ACE	MLL
1a−	Bill sees Kate.	Bill sees Kate
1b+	Kate sees Bill.	Kate sees Bill
2a+	Kate does not love Bill.	Kate not loves Bill
2b−	Bill does not love Kate.	Bill not loves Kate
3a+	Kate loves a golfer.	Kate **HasType** loves **some** golfer
3b−	Kate loves an officer.	Kate **HasType** loves **some** officer
4a+	John loves no golfer.	John **HasType** **not** (loves **some** golfer)
4b−	Kate sees no officer.	Kate **HasType** **not** (sees **some** officer)
5a+	Mary loves something that is not an officer.	Mary **HasType** loves **some** (**not** officer)
5b−	John loves something that is not a TV.	John **HasType** loves **some** (**not** TV)
6a+	Bill sees nothing but aquariums.	Bill **HasType** sees **only** aquarium
6b−	Kate sees nothing but officers.	Kate **HasType** sees **only** officer
7a−	Every officer sees an aquarium.	officer **SubTypeOf** sees **some** aquarium
7b+	Every golfer sees a TV.	golfer **SubTypeOf** sees **some** TV
8a+	Everything that sees a TV is a golfer.	sees **some** TV **SubTypeOf** golfer
8b−	Everything that loves a person is a golfer.	loves **some** person **SubTypeOf** golfer
9a−	Every golfer sees nothing but TVs.	golfer **SubTypeOf** sees **only** TV
9b+	Every officer loves nothing but golfers.	officer **SubTypeOf** loves **only** golfer
10a−	Everything that loves nothing but persons is a golfer.	loves **only** person **SubTypeOf** golfer
10b+	Everything that sees nothing but aquariums is an officer.	sees **only** aquarium **SubTypeOf** officer

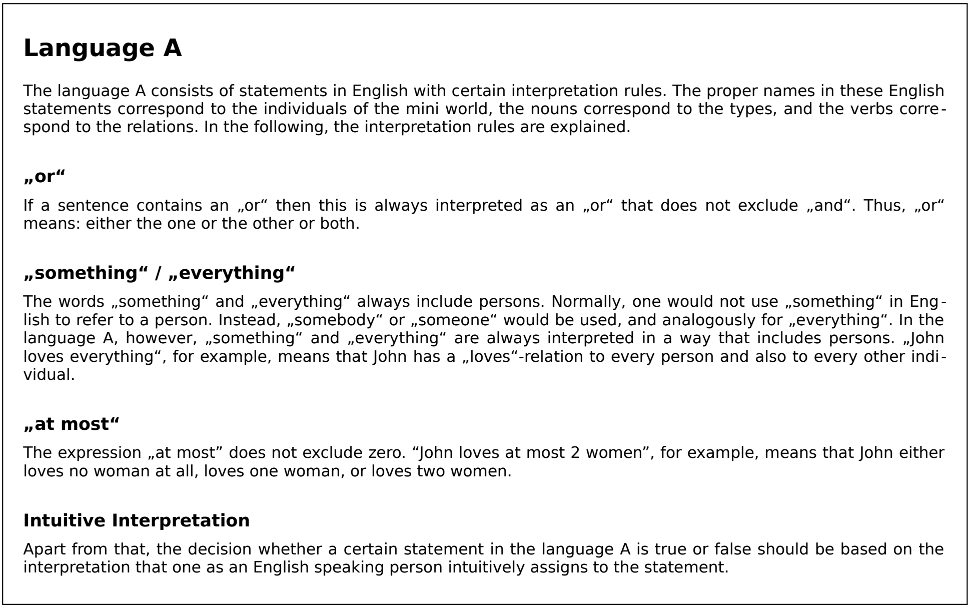
B.2.3 Ontograph Series 3

The statements of this series consist of domain, range, and cardinality restrictions.

Statement patterns

ID	Name	ACE Pattern	MLL Pattern
1	domain	Everything that $\langle R\rangle$ something is a $\langle T\rangle$.	$\langle R\rangle$ **HasDomain** $\langle T\rangle$
2	range	Everything that is $\langle R\rangle$ by something is a $\langle T\rangle$.	$\langle R\rangle$ **HasRange** $\langle T\rangle$
3	complex domain	Everything that $\langle R\rangle$ something is a $\langle T_1\rangle$ or is a $\langle T_2\rangle$.	$\langle R\rangle$ **HasDomain** $\langle T_1\rangle$ **or** $\langle T_2\rangle$
4	complex range	Everything that is $\langle R\rangle$ by something is a $\langle T_1\rangle$ or is a $\langle T_2\rangle$.	$\langle R\rangle$ **HasRange** $\langle T_1\rangle$ **or** $\langle T_2\rangle$
5	minimal cardinality	$\langle I\rangle$ $\langle R\rangle$ at least 2 $\langle T\rangle$.	$\langle I\rangle$ **HasType** $\langle R\rangle$ **min** 2 $\langle T\rangle$
6	maximal cardinality	$\langle I\rangle$ $\langle R\rangle$ at most 1 $\langle T\rangle$.	$\langle I\rangle$ **HasType** $\langle R\rangle$ **max** 1 $\langle T\rangle$
7	general minimal cardinality 1	Every $\langle T_1\rangle$ $\langle R\rangle$ at least 2 $\langle T_2\rangle$.	$\langle T_1\rangle$ **SubTypeOf** $\langle R\rangle$ **min** 2 $\langle T_2\rangle$
8	general minimal cardinality 2	Everything that $\langle R\rangle$ at least 2 $\langle T_1\rangle$ is a $\langle T_2\rangle$.	$\langle R\rangle$ **min** 2 $\langle T_1\rangle$ **SubTypeOf** $\langle T_2\rangle$
9	general maximal cardinality	Every $\langle T_1\rangle$ $\langle R\rangle$ at most 1 $\langle T_2\rangle$.	$\langle T_1\rangle$ **SubTypeOf** $\langle R\rangle$ **max** 1 $\langle T_2\rangle$
10	complex maximal cardinality	Everything that is a $\langle T_1\rangle$ or that is a $\langle T_2\rangle$ $\langle R\rangle$ at most 1 $\langle T_3\rangle$.	$\langle T_1\rangle$ **or** $\langle T_2\rangle$ **SubTypeOf** $\langle R\rangle$ **max** 1 $\langle T_3\rangle$

ACE Description Sheet



Language A

The language A consists of statements in English with certain interpretation rules. The proper names in these English statements correspond to the individuals of the mini world, the nouns correspond to the types, and the verbs correspond to the relations. In the following, the interpretation rules are explained.

„or“

If a sentence contains an „or“ then this is always interpreted as an „or“ that does not exclude „and“. Thus, „or“ means: either the one or the other or both.

„something“ / „everything“

The words „something“ and „everything“ always include persons. Normally, one would not use „something“ in English to refer to a person. Instead, „somebody“ or „someone“ would be used, and analogously for „everything“. In the language A, however, „something“ and „everything“ are always interpreted in a way that includes persons. „John loves everything“, for example, means that John has a „loves“-relation to every person and also to every other individual.

„at most“

The expression „at most” does not exclude zero. “John loves at most 2 women”, for example, means that John either loves no woman at all, loves one woman, or loves two women.

Intuitive Interpretation

Apart from that, the decision whether a certain statement in the language A is true or false should be based on the interpretation that one as an English speaking person intuitively assigns to the statement.

MLL Description Sheet

Language B

The language B consists of statements of the forms described and explained below. These statements are composed of keywords and of the names of the individuals, types, and relations of the respective mini world. The used keywords are „HasType“, „SubTypeOf“, „HasDomain“, „HasRange“, „or“, „min“, and „max“.

Statements

Every statement can either be true or false. Every statement of the language B has the form of one of the four schemes described here. Note that types can be complex (see the next section).

HasType-statement	
scheme:	*Individual* ***HasType*** *Type*
example:	John **HasType** golfer
explanation:	A HasType-statement requires an individual and a type and it states that the given individual belongs to the given type. The example above states that John is a golfer.

HasDomain-statement	
scheme:	*Relation* **HasDomain** *Type*
example:	buys **HasDomain** golfer
explanation:	A HasDomain-statement requires a relation and a type. Such a statement states that whenever an individual has the given relation to another individual then the first individual belongs to the given type. The example above states that every individual that has a „buys“-relation to another individual is a golfer.

HasRange-statement	
scheme:	*Relation* **HasRange** *Type*
example:	loves **HasRange** woman
explanation:	A HasRange-statement requires a relation and a type. Such a statement states that whenever an individual has the given relation to another individual then the second individual belongs to the given type. The example above states that every individual to which another individual has a „loves“-relation is a woman.

SubTypeOf-statement	
scheme:	*Type1* **SubTypeOf** *Type2*
example:	golfer **SubTypeOf** man
explanation:	A SubTypeOf-statement requires two types and states that every individual that belongs to the first type also belongs to the second type (but not necessarily the other way round). The example above states that every individual that is a golfer is also a man.

Type Operators

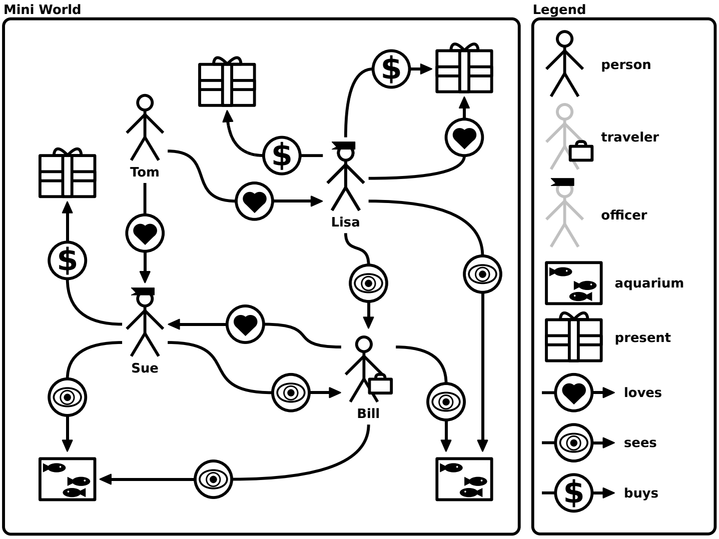
Every type (simple or complex) stands for a certain group of individuals. „traveler“ and „aquarium“ are examples of simple types. Apart from simple types, there are complex types that are composed by the type operators described here. „buys **min** 2 presents“ is an example of a complex type.

or-operator	
scheme:	*Type1* **or** *Type2*
example:	woman **or** golfer
explanation:	The or-operator requires two types. The resulting complex type stands for all individuals that belong to the first type or to the second type or to both. The example above stands for all individuals that are women or golfers or both.

min-operator	
scheme:	*Relation* **min** *Number* *Type*
example:	loves **min** 2 woman
explanation:	The min-operator requires a relation, a number, and a type. The resulting complex type stands for all individuals that have the given relation to at least the given number of individuals of the given type. The example above stands for all individuals that have a „loves“-relation to at least two women.

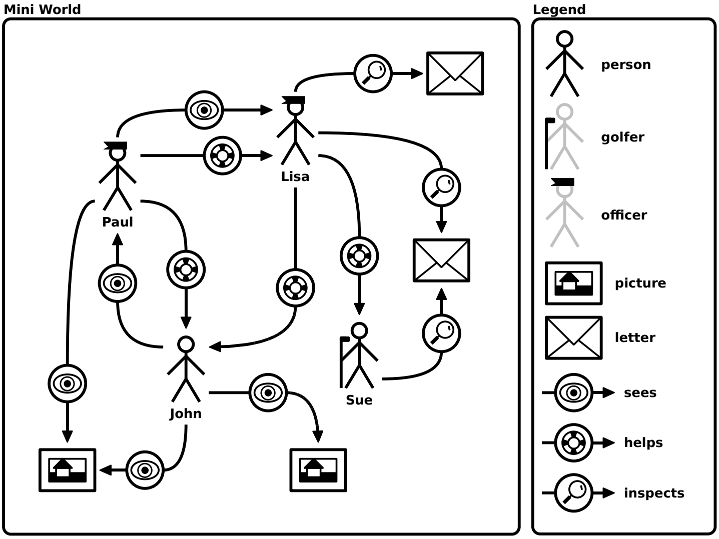
max-operator	
scheme:	*Relation* **max** *Number* *Type*
example:	sees **max** 2 man
explanation:	The max-operator requires a relation, a number, and a type. The resulting complex type stands for all individuals that have the given relation to a maximum of the given number of individuals of the given type. This includes the individuals that have no corresponding relation at all. The example above stands for all individuals that have a „sees“-relation to at most two men.

Ontograph Table 3A



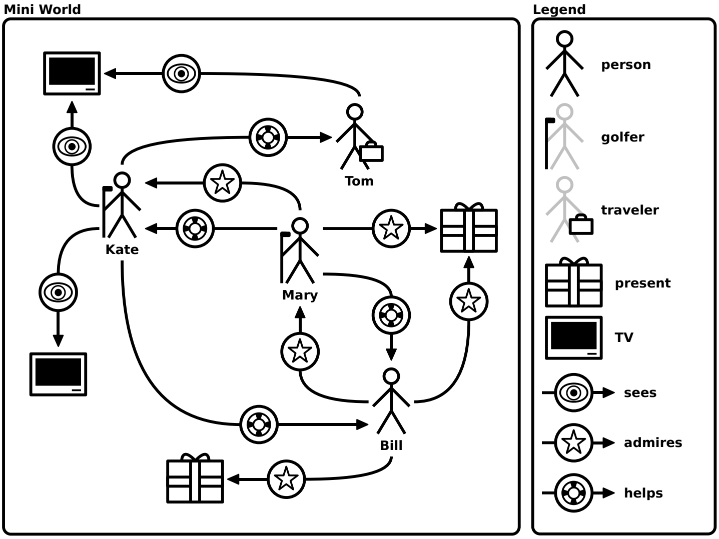
ID	ACE	MLL
1a−	Everything that sees something is an officer.	sees **HasDomain** officer
1b+	Everything that loves something is a person.	loves **HasDomain** person
2a−	Everything that is loved by something is a person.	loves **HasRange** person
2b+	Everything that is bought by something is a present.	buys **HasRange** present
3a−	Everything that loves something is a traveler or is an officer.	loves **HasDomain** traveler **or** officer
3b+	Everything that sees something is an officer or is a traveler.	sees **HasDomain** officer **or** traveler
4a+	Everything that is seen by something is a traveler or is an aquarium.	sees **HasRange** traveler **or** aquarium
4b−	Everything that is bought by something is an aquarium or is an officer.	buys **HasRange** aquarium **or** officer
5a+	Tom loves at least 2 officers.	Tom **HasType** loves **min** 2 officer
5b−	Sue sees at least 2 persons.	Sue **HasType** sees **min** 2 person
6a−	Lisa buys at most 1 present.	Lisa **HasType** buys **max** 1 present
6b+	Bill loves at most 1 person.	Bill **HasType** loves **max** 1 person
7a+	Every traveler sees at least 2 aquariums.	traveler **SubTypeOf** sees **min** 2 aquarium
7b−	Every officer buys at least 2 presents.	officer **SubTypeOf** buys **min** 2 present
8a+	Everything that buys at least 2 presents is an officer.	buys **min** 2 present **SubTypeOf** officer
8b−	Everything that loves at least 2 officers is a traveler.	love **min** 2 officer **SubTypeOf** traveler
9a+	Every officer sees at most 1 aquarium.	officer **SubTypeOf** sees **max** 1 aquarium
9b−	Every person buys at most 1 present.	person **SubTypeOf** buys **max** 1 present
10a−	Everything that is a traveler or that is an officer sees at most 1 aquarium.	traveler **or** officer **SubTypeOf** sees **max** 1 aquarium
10b+	Everything that is an officer or that is a traveler loves at most 1 person.	officer **or** traveler **SubTypeOf** loves **max** 1 person

Ontograph Table 3B



ID	ACE	MLL
1a−	Everything that inspects something is an officer	inspects **HasDomain** officer
1b+	Everything that helps something is an officer.	helps **HasDomain** officer
2a+	Everything that is inspected by something is a letter.	inspects **HasRange** letter
2b−	Everything that is seen by something is an officer.	sees **HasRange** officer
3a+	Everything that inspects something is a golfer or is an officer.	inspects **HasDomain** golfer **or** officer
3b−	Everything that sees something is an officer or is a golfer.	sees **HasDomain** officer **or** golfer
4a+	Everything that is seen by something is an officer or is a picture.	sees **HasRange** officer **or** picture
4b−	Everything that is helped by something is a golfer or is an officer.	helps **HasRange** golfer **or** officer
5a+	Lisa inspects at least 2 letters.	Lisa **HasType** inspects **min** 2 letter
5b−	Paul sees at least 2 persons.	Paul **HasType** sees **min** 2 person
6a−	Lisa helps at most 1 person.	Lisa **HasType** helps **max** 1 person
6b+	John sees at most 1 officer.	John **HasType** sees **max** 1 officer
7a+	Every officer helps at least 2 persons.	officer **SubTypeOf** helps **min** 2 person
7b−	Every officer inspects at least 2 letters.	officer **SubTypeOf** inspects **min** 2 letter
8a−	Everything that sees at least 2 pictures is an officer.	sees **min** 2 picture **SubTypeOf** officer
8b+	Everything that inspects at least 2 letters is an officer.	inspects **min** 2 letter **SubTypeOf** officer
9a−	Every person inspects at most 1 letter.	person **SubTypeOf** inspects **max** 1 letter
9b+	Every person helps at most 1 officer.	person **SubTypeOf** helps **max** 1 officer
10a+	Everything that is an officer or that is a golfer sees at most 1 picture.	officer **or** golfer **SubTypeOf** sees **max** 1 picture
10b−	Everything that is a golfer or that is an officer inspects at most 1 letter.	golfer **or** officer **SubTypeOf** inspects **max** 1 letter

Ontograph Table 3C



ID	ACE	MLL
1a−	Everything that sees something is a golfer.	sees **HasDomain** golfer
1b+	Everything that helps something is a golfer.	helps **HasDomain** golfer
2a+	Everything that is seen by something is a TV.	sees **HasRange** TV
2b−	Everything that is admired by something is a golfer.	admires **HasRange** golfer
3a+	Everything that sees something is a traveler or is a golfer.	sees **HasDomain** traveler **or** golfer
3b−	Everything that admires something is a golfer or is a traveler.	admires **HasDomain** golfer **or** traveler
4a+	Everything that is admired by something is a golfer or is a present.	admires **HasRange** golfer **or** present
4b−	Everything that is helped by something is a traveler or is a golfer.	helps **HasRange** traveler **or** golfer
5a+	Kate sees at least 2 TVs.	Kate **HasType** sees **min** 2 TV
5b−	Mary admires at least 2 persons.	Mary **HasType** admires **min** 2 person
6a−	Kate helps at most 1 person.	Kate **HasType** helps **max** 1 person
6b+	Bill admires at most 1 golfer.	Bill **HasType** admires **max** 1 golfer
7a+	Every golfer helps at least 2 persons.	golfer **SubTypeOf** helps **min** 2 person
7b−	Every golfer sees at least 2 TVs.	golfer **SubTypeOf** sees **min** 2 TV
8a−	Everything that admires at least 2 presents is a golfer.	admires **min** 2 present **SubTypeOf** golfer
8b+	Everything that sees at least 2 TVs is a golfer.	sees **min** 2 TV **SubTypeOf** golfer
9a−	Every person sees at most 1 TV.	person **SubTypeOf** sees **max** 1 TV
9b+	Every person helps at most 1 golfer.	person **SubTypeOf** helps **max** 1 golfer
10a+	Everything that is a golfer or that is a traveler admires at most 1 present.	golfer **or** traveler **SubTypeOf** admires **max** 1 present
10b−	Everything that is a traveler or that is a golfer sees at most 1 TV.	traveler **or** golfer **SubTypeOf** sees **max** 1 TV

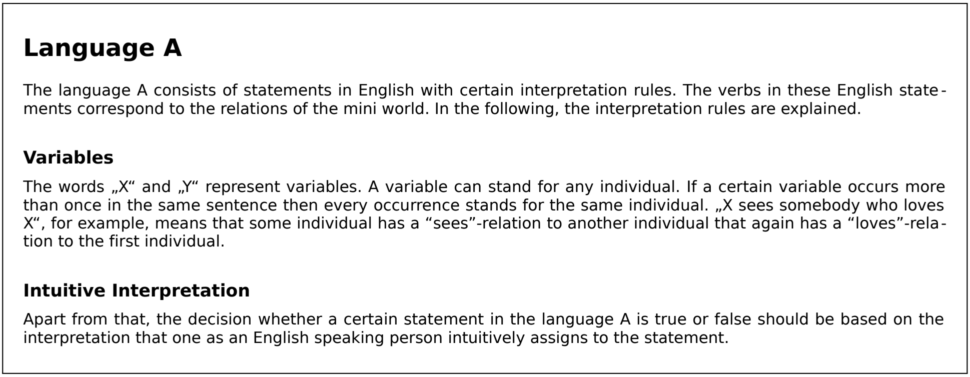
B.2.4 Ontograph Series 4

The statements of this series are only about relations and not about individuals or types.

Statement patterns

ID	Name	ACE Pattern	MLL Pattern
1	symmetric relation	If X $\langle R\rangle$ Y then Y $\langle R\rangle$ X.	$\langle R\rangle$ **IsSymmetric**
2	asymmetric relation	If X $\langle R\rangle$ Y then Y does not $\langle R\rangle$ X.	$\langle R\rangle$ **IsAsymmetric**
3	transitive relation	If X $\langle R\rangle$ somebody who $\langle R\rangle$ Y then X $\langle R\rangle$ Y.	$\langle R\rangle$ **IsTransitive**
4	subrelation 1	If X $\langle R_1\rangle$ Y then X $\langle R_2\rangle$ Y.	$\langle R_1\rangle$ **SubRelationOf** $\langle R_2\rangle$
5	subrelation 2	If X $\langle R_1\rangle$ Y then X $\langle R_2\rangle$ Y.	$\langle R_1\rangle$ **SubRelationOf** $\langle R_2\rangle$
6	inverse subrelation	If X $\langle R_1\rangle$ Y then Y $\langle R_2\rangle$ X.	$\langle R_1\rangle$ **SubRelationOf** **inverse** $\langle R_2\rangle$
7	disjoint relations	If X $\langle R_1\rangle$ Y then X does not $\langle R_2\rangle$ Y.	$\langle R_1\rangle$ **DisjointWith** $\langle R_2\rangle$
8	inverse disjoint relations	If X $\langle R_1\rangle$ Y then Y does not $\langle R_2\rangle$ X.	$\langle R_1\rangle$ **DisjointWith** **inverse** $\langle R_2\rangle$
9	equivalent relations	If X $\langle R_1\rangle$ Y then X $\langle R_2\rangle$ Y. If X $\langle R_2\rangle$ Y then X $\langle R_1\rangle$ Y.	$\langle R_1\rangle$ **EquivalentTo** $\langle R_2\rangle$
10	inverse equivalent relations	If X $\langle R_1\rangle$ Y then Y $\langle R_2\rangle$ X. If Y $\langle R_2\rangle$ X then X $\langle R_1\rangle$ Y.	$\langle R_1\rangle$ **EquivalentTo** **inverse** $\langle R_2\rangle$

ACE Description Sheet



Language A

The language A consists of statements in English with certain interpretation rules. The verbs in these English statements correspond to the relations of the mini world. In the following, the interpretation rules are explained.

Variables

The words „X“ and „Y“ represent variables. A variable can stand for any individual. If a certain variable occurs more than once in the same sentence then every occurrence stands for the same individual. „X sees somebody who loves X“, for example, means that some individual has a “sees”-relation to another individual that again has a “loves”-relation to the first individual.

Intuitive Interpretation

Apart from that, the decision whether a certain statement in the language A is true or false should be based on the interpretation that one as an English speaking person intuitively assigns to the statement.

MLL Description Sheet

Language B

The language B consists of statements of the forms described and explained below. These statements are composed of keywords and of the names of the relations of the respective mini world. The used keywords are „SubRelationOf“, „DisjointWith“, „EquivalentTo“, „IsSymmetric“, „IsAsymmetric“, „IsTransitive“, and „inverse“.

Statements

Every statement can either be true or false. Every statement of the language B has the form of one of the six schemes described here. Note that relations can be inverse (see the next section).

SubRelationOf-statements	
scheme:	*Relation1* **SubRelationOf** *Relation2*
example:	helps **SubRelationOf** loves
explanation:	A SubRelationOf-statement requires two relations. Such a statement states that whenever two individuals are connected by the first relation then these two individuals are in the same direction also connected by the second relation (but not necessarily the other way round). The example above states that whenever two individuals are connected by a „helps“-relation then they are in the same direction also connected by a „loves“-relation.

DisjointWith-statements	
scheme:	*Relation1* **DisjointWith** *Relation2*
example:	admires **DisjointWith** helps
explanation:	A DisjointWith-statement requires two relations. Such a statement states that whenever two individuals are connected by the first relation then these two individuals are never in the same direction connected by the second relation. The example above states that whenever two individuals are connected by an „admires“-relation then they are never in the same direction connected by a „helps“-relation.

EquivalentTo-statements	
scheme:	*Relation1* **EquivalentTo** *Relation2*
example:	sees **EquivalentTo** asks
explanation:	An EquivalentTo-statement requires two relations. Such a statement states that whenever two individuals are connected by the first relation then these two individuals are in the same direction also connected by the second relation, and vice versa. The example above states that whenever two individuals are connected by a „sees“-relation then they are in the same direction also connected by an „asks“-relation and vice versa.

IsSymmetric-statements	
scheme:	*Relation* **IsSymmetric**
example:	sees **IsSymmetric**
explanation:	An IsSymmetric-statement requires just one relation. Such a statement states that whenever an individual has the given relation to another individual then the second individual has the same relation to the first individual as well. The example above states that whenever an individual has a „sees“-relation to another individual then the second individual has a „sees“-relation to the first individual as well.

IsAsymmetric-statements	
scheme:	*Relation* **IsAsymmetric**
example:	admires **IsAsymmetric**
explanation:	An IsAsymmetric-statement requires just one relation. Such a statement states that whenever an individual has the given relation to another individual then the second individual never has the same relation to the first individual. The example above states that whenever an individual has an „admires“-relation to another individual then the second individual never has an „admires“-relation to the first individual.

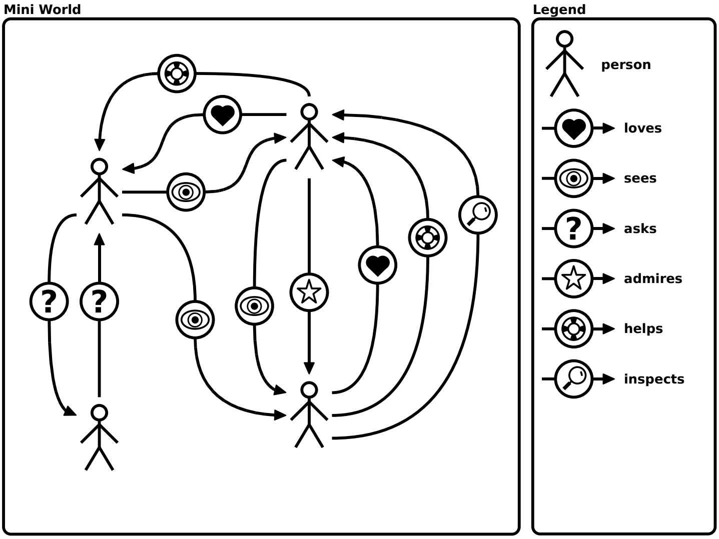
IsTransitive-statements	
scheme:	*Relation* **IsTransitive**
example:	asks **IsTransitive**
explanation:	An IsTransitive-statement requires just one relation. Such a statement states that whenever an individual has the given relation to another individual that again has the same relation to a third individual then the first individual has the same relation to the third individual as well. The example above states that whenever an individual has an „asks“-relation to another individual that has an „asks“-relation to a third individual then the first individual has an „asks“-relation to the third individual as well.

inverse-operator

Wherever a relation can occur in the statements, there can also be an inverse relation. Such inverse relations are built by the operator „inverse“ described here.

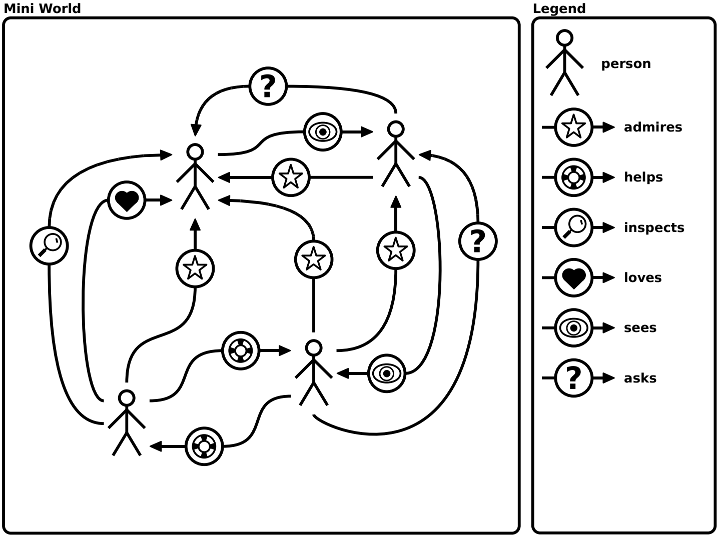
inverse-operator	
scheme:	**inverse** *Relation*
example:	**inverse** sees
explanation:	The inverse-operator requires just one relation. The resulting complex relation represents the inverse relation that connects the same individuals but in the inverse direction. The example above represents the inverse relation of the „sees“-relation and one could call it the „is seen by“-relation.

Ontograph Table 4A



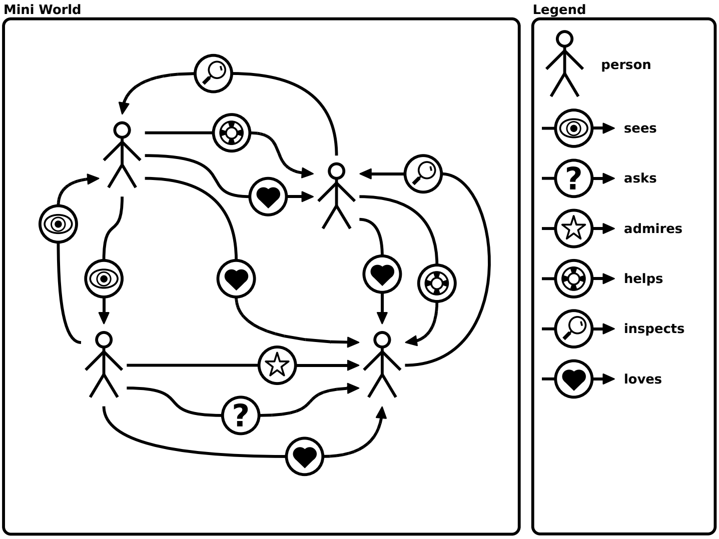
ID	ACE	MLL
1a−	If X helps Y then Y helps X.	helps **IsSymmetric**
1b+	If X asks Y then Y asks X.	asks **IsSymmetric**
2a+	If X sees Y then Y does not see X.	sees **IsAsymmetric**
2b−	If X asks Y then Y does not ask X.	asks **IsAsymmetric**
3a+	If X sees somebody who sees Y then X sees Y.	sees **IsTransitive**
3b−	If X loves somebody who loves Y then X loves Y.	loves **IsTransitive**
4a+	If X admires Y then X sees Y.	admires **SubRelationOf** sees
4b−	If X sees Y then X admires Y.	sees **SubRelationOf** admires
5a+	If X inspects Y then X helps Y.	inspects **SubRelationOf** helps
5b−	If X helps Y then X inspects Y.	helps **SubRelationOf** inspects
6a−	If X helps Y then Y admires X.	helps **SubRelationOf** **inverse** admires
6b+	If X loves Y then Y sees X.	loves **SubRelationOf** **inverse** sees
7a+	If X loves Y then X does not admire Y.	loves **DisjointWith** admires
7b−	If X admires Y then X does not see Y.	admires **DisjointWith** sees
8a−	If X sees Y then Y does not love X.	sees **DisjointWith** **inverse** love
8b+	If X admires Y then Y does not see X.	admires **DisjointWith** **inverse** sees
9a−	If X admires Y then X sees Y. If X sees Y then X admires Y.	admires **EquivalentTo** sees
9b+	If X loves Y then X helps Y. If X helps Y then X loves Y.	loves **EquivalentTo** helps
10a−	If X inspects Y then Y sees X. If Y sees X then X inspects Y.	inspects **EquivalentTo** **inverse** sees
10b+	If X admires Y then Y inspects X. If Y inspects X then X admires Y.	admires **EquivalentTo** **inverse** inspects

Ontograph Table 4B



ID	ACE	MLL
1a+	If X helps Y then Y helps X.	helps **IsSymmetric**
1b−	If X admires Y then Y admires X.	admires **IsSymmetric**
2a+	If X asks Y then Y does not ask X.	asks **IsAsymmetric**
2b−	If X helps Y then Y does not help X.	helps **IsAsymmetric**
3a+	If X admires somebody who admires Y then X admires Y.	admires **IsTransitive**
3b−	If X sees somebody who sees Y then X sees Y.	sees **IsTransitive**
4a−	If X sees Y then X admires Y.	sees **SubRelationOf** admires
4b+	If X inspects Y then X admires Y.	inspects **SubRelationOf** admires
5a+	If X asks Y then X admires Y.	asks **SubRelationOf** admires
5b−	If X admires Y then X asks Y.	admires **SubRelationOf** asks
6a−	If X loves Y then Y admires X.	loves **SubRelationOf** inverse admires
6b+	If X sees Y then Y admires X.	sees **SubRelationOf** inverse admires
7a−	If X admires Y then X does not inspect Y.	admires **DisjointWith** inspects
7b+	If X sees Y then X does not admire Y.	sees **DisjointWith** admires
8a+	If X asks Y then Y does not admire X.	asks **DisjointWith** inverse admires
8b−	If X admires Y then Y does not see X.	admires **DisjointWith** inverse sees
9a+	If X loves Y then X inspects Y. If X inspects Y then X loves Y.	loves **EquivalentTo** inspects
9b−	If X asks Y then X admires Y. If X admires Y then X asks Y.	asks **EquivalentTo** admires
10a−	If X loves Y then Y admires X. If Y admires X then X loves Y.	loves **EquivalentTo** inverse admires
10b+	If X asks Y then Y sees X. If Y sees X then X asks Y.	asks **EquivalentTo** inverse sees

Ontograph Table 4C



ID	ACE	MLL
1a+	If X sees Y then Y sees X.	sees **IsSymmetric**
1b−	If X loves Y then Y loves X.	loves **IsSymmetric**
2a+	If X helps Y then Y does not help X.	helps **IsAsymmetric**
2b−	If X sees Y then Y does not see X.	sees **IsAsymmetric**
3a+	If X loves somebody who loves Y then X loves Y.	loves **IsTransitive**
3b−	If X inspects somebody who inspects Y then X inspects Y.	inspects **IsTransitive**
4a−	If X inspects Y then X loves Y.	inspects **SubRelationOf** loves
4b+	If X asks Y then X loves Y.	asks **SubRelationOf** loves
5a+	If X helps Y then X loves Y.	helps **SubRelationOf** loves
5b−	If X loves Y then X helps Y.	loves **SubRelationOf** helps
6a−	If X admires Y then Y loves X.	admires **SubRelationOf** **inverse** loves
6b+	If X inspects Y then Y loves X.	inspects **SubRelationOf** **inverse** loves
7a−	If X loves Y then X does not ask Y.	loves **DisjointWith** asks
7b+	If X inspects Y then X does not love Y.	inspects **DisjointWith** loves
8a+	If X helps Y then Y does not love X.	helps **DisjointWith** **inverse** loves
8b−	If X loves Y then Y does not inspect X.	loves **DisjointWith** **inverse** inspects
9a+	If X admires Y then X asks Y. If X asks Y then X admires Y.	admires **EquivalentTo** asks
9b−	If X helps Y then X loves Y. If X loves Y then X helps Y.	helps **EquivalentTo** loves
10a−	If X admires Y then Y loves X. If Y loves X then X admires Y.	admires **EquivalentTo** **inverse** loves
10b+	If X helps Y then Y inspects X. If Y inspects X then X helps Y.	helps **EquivalentTo** **inverse** inspects

B.2.5 Questionnaire

Below, the questionnaire is shown that the participants had to fill out after the second ontograph experiment. The items marked with a star "*" exist only for control reasons (the star is not part of the questionnaire) and the participants who checked one of these options had to be excluded from the data set.

1. What is your gender?

   □ male
   □ female

2. How old are you?

   ______ years

3. What do you study or what have you been studying? Please also indicate your minor subjects (if applicable).

   □ current studies in: ________________________________
   □ finished studies in: ________________________________
   □ I am not a student *

4. What is your level of English skills?

   □ no skills or almost no skills *
   □ little skills *
   □ good skills
   □ very good skills

5. How easy or hard to understand did you find the instructions?

   □ very easy to understand
   □ easy to understand
   □ hard to understand
   □ very hard to understand

6. How easy or hard to understand did you find the diagrams?

   □ very easy to understand
   □ easy to understand
   □ hard to understand
   □ very hard to understand

7. How easy or hard to understand did you find the statements in language A (the language that looks like English)?

   □ very easy to understand
   □ easy to understand
   □ hard to understand
   □ very hard to understand

8. How easy or hard to understand did you find the statements in language B?

   □ very easy to understand
   □ easy to understand
   □ hard to understand
   □ very hard to understand

Bibliography

[1] ACE 6.5 construction rules. Attempto group, Department of Informatics, University of Zurich. http://attempto.ifi.uzh.ch/site/docs/ace/6.5/ace_constructionrules.html, May 2009.

[2] ACE 6.5 interpretation rules. Attempto group, Department of Informatics, University of Zurich. http://attempto.ifi.uzh.ch/site/docs/ace/6.5/ace_interpretationrules.html,, May 2009.

[3] ACE 6.5 syntax report. Attempto group, Department of Informatics, University of Zurich. http://attempto.ifi.uzh.ch/site/docs/ace/6.5/syntax_report.html, May 2009.

[4] Geert Adriaens and Dirk Schreors. From COGRAM to ALCOGRAM: Toward a controlled english grammar checker. In *Proceedings of the 14th Conference on Computational Linguistics*, volume 2, pages 595–601. Association for Computational Linguistics, 1992.

[5] Krasimir Angelov and Aarne Ranta. Implementing controlled languages in GF. In Norbert E. Fuchs, editor, *Proceedings of the Workshop on Controlled Natural Language (CNL 2009)*, volume 5972 of *Lecture Notes in Computer Science*, pages 82–101. Springer, 2010.

[6] Krzysztof R. Apt and Marc Bezem. Acyclic programs. In David H. D. Warren, editor, *Logic Programming: Proceedings of the 7th International Conference*, pages 617–633. MIT Press, 1990.

[7] Aristotle. Prior analytics, ca. 350 B.C. http://classics.mit.edu/Aristotle/prior.html.

[8] Sören Auer, Sebastian Dietzold, and Thomas Riechert. OntoWiki — a tool for social, semantic collaboration. In *The Semantic Web — ISWC 2006, Proceedings of the 5th*

International Semantic Web Conference, volume 4273 of *Lecture Notes in Computer Science*, pages 736–749. Springer, November 2006.

[9] Francis Bacon. Meditationes sacræ, de hæresibus. In Basil Montagu, editor, *The works of Francis Bacon, Baron of Veluram, Viscount St. Alban, and Lord High Chancellor of England*, new edition, volume 10, pages 307–309. W. Pickering, London, 1825.

[10] Raffaella Bernardi, Diego Calvanese, and Camilo Thorne. Lite natural language. In *Proceedings of the 7th International Workshop on Computational Semantics (IWCS-7)*, 2007.

[11] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5), May 2001.

[12] Abraham Bernstein and Esther Kaufmann. GINO — a guided input natural language ontology editor. In *The Semantic Web — ISWC 2006, Proceedings of the 5th International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 144–157. Springer, November 2006.

[13] Abraham Bernstein, Esther Kaufmann, Norbert E. Fuchs, and June von Bonin. Talking to the Semantic Web — a controlled english query interface for ontologies. In *Proceedings of the 14th Workshop on Information Technology and Systems*, pages 212–217, December 2004.

[14] Arendse Bernth. EasyEnglish: a tool for improving document quality. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 159–165. Association for Computational Linguistics, 1997.

[15] Patrick Blackburn and Johan Bos. *Representation and Inference for Natural Language. A First Course in Computational Semantics.* CSLI Publications, 2005.

[16] Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, and Mike Smith. OWL 2 Web Ontology Language — structural specification and functional-style syntax. World Wide Web Consortium. W3C Candidate Recommendation, `http://www.w3.org/TR/2009/CR-owl2-syntax-20090611/`, June 2009.

[17] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.

[18] Johan Bos. Computational semantics in discourse: Underspecification, resolution, and inference. *Journal of Logic, Language and Information*, 13(2):139–157, March 2004.

[19] Johan Bos. Let's not Argue about Semantics. In Nicoletta Calzolari, Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odjik, Stelios Piperidis, and Daniel Tapias, editors, *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, pages 2835–2840. European Language Resources Association (ELRA), 2008.

[20] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[21] Björn Bringert, Krasimir Angelov, and Aarne Ranta. Grammatical framework web service. In *EACL '09: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Demonstrations Session*, pages 9–12. Association for Computational Linguistics, 2009.

[22] Selmer Bringsjord, Konstantine Arkoudas, Micah Clark, Andrew Shilliday, Joshua Taylor, Bettina Schimanski, and Yingrui Yang. Reporting on some logic-based machine reading research. In Oren Etzioni, editor, *Machine Reading — Papers from the 2007 AAAI Spring Symposium*, number SS-07-06 in AAAI Spring Symposium Series, pages 23–28. AAAI Press, 2007.

[23] John Brooke. SUS: A quick and dirty usability scale. In Patrick W. Jordan, Bernard A. Weerdmeester, Bruce Thomas, and Ian L. McClelland, editors, *Usability evaluation in industry*, pages 189–194. Taylor and Francis, London, 1996.

[24] Bruce Buchanan, Georgia Sutherland, and Edward A. Feigenbaum. Heuristic Dendral: a program for generating explanatory hypotheses in organic chemistry. In Bernard Meltzer and Donald Michie, editors, *Proceedings of the Fourth Annual Machine Intelligence Workshop*, volume 4 of *Machine Intelligence*, pages 209–254. Edinburgh University Press, 1969.

[25] Michel Buffa, Fabien Gandon, Guillaume Ereteo, Peter Sander, and Catherine Faron. SweetWiki: A semantic wiki. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(1):84–97, 2008.

[26] Chen Chung Chang and H. Jerome Keisler. *Model Theory*, volume 73 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1973.

[27] Steve Chervak, Colin G. Drury, and James P. Ouellette. Field evaluation of simplified english for aircraft workcards. In *Proceedings of the 10th FAA/AAM Meeting on Human Factors in Aviation Maintenance and Inspection*, 1996.

[28] Noam Chomsky. On binding. *Linguistic Inquiry*, 11(1):1–46, 1980.

[29] Philipp Cimiano. ORAKEL: A natural language interface to an F-logic knowledge base. In *Proceedings of the 9th International Conference on Applications of Natural Language to Information Systems*, volume 3136 of *Lecture Notes in Computer Science*, pages 401–406. Springer, 2004.

[30] William J. Clancey. The knowledge level reinterpreted: Modeling how systems interact. *Machine Learning*, 4(3-4):285–291, 1989.

[31] Peter Clark, Shaw-Yi Chaw, Ken Barker, Vinay Chaudhri, Phil Harrison, James Fan, Bonnie John, Bruce Porter, Aaron Spaulding, John Thompson, and Peter Yeh. Capturing and answering questions posed to a knowledge-based system. In *K-CAP '07: Proceedings of the 4th International Conference on Knowledge Capture*, pages 63–70. ACM, 2007.

[32] Peter Clark, Phil Harrison, Thomas Jenkins, John Thompson, and Richard H. Wojcik. Acquiring and using world knowledge using a restricted subset of English. In Ingrid Russell and Zdravko Markov, editors, *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2005)*, pages 506–511. AAAI Press, 2005.

[33] Peter Clark, Phil Harrison, William R. Murray, and John Thompson. Naturalness vs. predictability: A key debate in controlled languages. In Norbert E. Fuchs, editor, *Proceedings of the Workshop on Controlled Natural Language (CNL 2009)*, volume 5972 of *Lecture Notes in Computer Science*, pages 65–81. Springer, 2010.

[34] Andrew B. Clegg and Adrian J. Shepherd. Benchmarking natural-language parsers for biological applications using dependency graphs. *BMC Bioinformatics*, 8(24), January 2007.

[35] Ron Cole, Joseph Mariani, Hans Uszkoreit, Giovanni Batista Varile, Annie Zaenen, Antonio Zampolli, and Victor Zue, editors. *Survey of the State of the Art in Human Language Technology.* Cambridge University Press, 1997.

[36] Michael A. Covington. GULP 3.1: An extension of Prolog for unification-based grammar. Research Report AI-1994-06, Artificial Intelligence Center, University of Georgia, Athens, GA, USA, 1994.

[37] Michael A. Covington. *Natural Language Processing for Prolog Programmers.* Prentice Hall, 1994.

[38] Anne Cregan, Rolf Schwitter, and Thomas Meyer. Sydney OWL Syntax - towards a controlled natural language syntax for OWL 1.1. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*, volume 258 of *CEUR Workshop Proceedings.* CEUR-WS, 2007.

[39] Veronica Dahl, Paul Tarau, and Renwei Li. Assumption grammars for processing natural language. In Lee Naish, editor, *Proceedings of the Fourteenth International Conference on Logic Programming*, pages 256–270. MIT Press, 1997.

[40] Kathleen Dahlgren and Joyce P. McDowell. Using commonsense knowledge to disambiguate prepositional phrase modifiers. In *Proceedings of the Proceedings of Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 589–593. AAAI Press, 1986.

[41] Brian Davis, Ahmad Ali Iqbal, Adam Funk, Valentin Tablan, Kalina Bontcheva, Hamish Cunningham, and Siegfried Handschuh. Roundtrip ontology authoring. In *Proceedings of the 7th International Semantic Web Conference (ISWC 2008)*, volume 5318 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2008.

[42] Juri Luca De Coi, Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Controlled English for reasoning on the Semantic Web. In François Bry and Jan Małuszyński, editors, *Semantic Techniques for the Web — The REWERSE Perspective*, volume 5500 of *Lecture Notes in Computer Science*, pages 276–308. Springer, 2009.

[43] Vania Dimitrova, Ronald Denaux, Glen Hart, Catherine Dolbear, Ian Holt, and Anthony G.Cohn. Involving Domain Experts in Authoring OWL Ontologies. In *Proceedings of the 7th International Semantic Web Conference (ISWC 2008)*, volume 5318 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2008.

[44] Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, February 1970.

[45] Gregor Erbach. ProFIT: prolog with features, inheritance and templates. In *Proceed-*

ings of the seventh conference on European chapter of the Association for Computational Linguistics, pages 180–187. Morgan Kaufmann Publishers, 1995.

[46] Christiane Fellbaum, editor. *WordNet — An Electronic Lexical Database*. MIT Press, 1998.

[47] Mark K. Fiegener. S&E degrees: 1966–2006. National Science Foundation, Division of Science Resources Statistics. `http://www.nsf.gov/statistics/nsf08321/pdf/nsf08321.pdf`, October 2008.

[48] Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Louis Nebert, Halle, Germany, 1879.

[49] Leora Friedberg. The impact of technological change on older workers: Evidence from data on computer use. *Industrial and Labor Relations Review*, 56(3):511–529, April 2003.

[50] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto Controlled English for knowledge representation. In Cristina Baroglio, Piero A. Bonatti, Jan Małuszyński, Massimo Marchiori, Axel Polleres, and Sebastian Schaffert, editors, *Reasoning Web — 4th International Summer School 2008*, volume 5224 of *Lecture Notes in Computer Science*, pages 104–124. Springer, 2008.

[51] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Discourse representation structures for ACE 6.5. Technical Report ifi-2009.04, Department of Informatics, University of Zurich, Switzerland, 2009.

[52] Norbert E. Fuchs and Uta Schwertel. Reasoning in Attempto Controlled English. In François Bry, Nicola Henze, and Jan Małuszyński, editors, *Principles and Practice of Semantic Web Reasoning — Proceedings of the International Workshop (PPSWR 2003)*, volume 2901 of *Lecture Notes in Computer Science*, pages 174–188. Springer, 2003.

[53] Norbert E. Fuchs, Uta Schwertel, and Sunna Torge. A natural language front-end to model generation. *Journal of Language and Computation*, 1(2):199–214, December 2000.

[54] Norbert E. Fuchs and Rolf Schwitter. Attempto Controlled English (ACE). In *Proceedings of the First International Workshop on Controlled Language Applications (CLAW 96)*. Katholieke Universiteit Leuven, Belgium, 1996.

[55] Norbert E. Fuchs and Rolf Schwitter. Web-annotations for humans and machines. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *The Semantic Web: Research and Applications — Proceedings of the 4th European Semantic Web Conference (ESWC 2007)*, volume 4519 of *Lecture Notes in Computer Science*, pages 458–472. Springer, 2007.

[56] Adam Funk, Brian Davis, Valentin Tablan, Kalina Bontcheva, and Hamish Cunningham. Controlled language IE components version 2. SEKT Project Deliverable D2.2.2, University of Sheffield, UK, 2007.

[57] Adam Funk, Valentin Tablan, Kalina Bontcheva, Hamish Cunningham, Brian Davis, and Siegfried Handschuh. CLOnE: Controlled language for ontology editing. In *Proceedings of the 6th International Semantic Web Conference and the 2nd Asian*

Semantic Web Conference (ISWC 2007 + ASWC 2007), volume 4825 of *Lecture Notes in Computer Science*. Springer, 2007.

[58] Gerald Gazdar and Chris Mellish. *Natural Language Processing in PROLOG*. Addison-Wesley, 1989.

[59] Niyu Ge, John Hale, and Eugene Charniak. A statistical approach to anaphora resolution. In *Proceedings of the Sixth Workshop on Very Large Corpora*, pages 161–171, 1998.

[60] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming — Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080. MIT Press, 1988.

[61] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3–4):365–385, August 1991.

[62] T. Grandon Gill. Early expert systems: Where are they now? *MIS Quarterly*, 19(1):51–81, 1995.

[63] Benjamin N. Grosof. Courteous logic programs: Prioritized conflict handling for rules. IBM Research Report RC 20836, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY, USA, December 1997.

[64] Dick Grune and Ceriel J.H. Jacobs. *Parsing Techniques — A Practical Guide*, second edition. Monographs in Computer Science. Springer Science+Business Media, 2008.

[65] Catalina Hallett, Donia Scott, and Richard Power. Composing questions through conceptual authoring. *Computational Linguistics*, 33(1):105–133, 2007.

[66] Terry Halpin. Business rule verbalization. In Anatoly Doroshenko, Terry Halpin, Stephen Liddle, and Heinrich C. Mayr, editors, *Information Systems Technology and its Applications — Proceedings of the 3rd International Conference ISTA 2004*, volume P-48 of *Lecture Notes in Informatics*, pages 39–52. Köllen Druck+Verlag, Bonn, Germany, 2004.

[67] Terry Halpin and Matthew Curland. Automated verbalization for ORM 2. In *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, volume 4278 of *Lecture Notes in Computer Science*, pages 1181–1190. Springer, 2006.

[68] Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1–3):335–346, June 1990.

[69] Glen Hart, Martina Johnson, and Catherine Dolbear. Rabbit: Developing a controlled natural language for authoring ontologies. In *Proceedings of the 5th European Semantic Web Conference (ESWC 2008)*, volume 5021 of *Lecture Notes in Computer Science*, pages 348–360. Springer, 2008.

[70] Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. *ACM Transactions on Database Systems (TODS)*, 3(2):105–147, 1978.

[71] David Hirtle. TRANSLATOR: A TRANSlator from LAnguage TO Rules. In *Proceedings of the Canadian Symposium on Text Analysis (CaSTA)*, 2006.

[72] James E. Hoard, Richard Wojcik, and Katherina Holzhauser. An automated grammar and style checker for writers of simplified english. In Patrik Holt and Noel Williams, editors, *Computers and Writing — State of the Art*, pages 278–296. Intellect, Oxford, UK, 1992.

[73] Matthew Horridge, Nick Drummond, John Goodwin, Alan L. Rector, Robert Stevens, and Hai Wang. The Manchester OWL syntax. In Bernardo Cuenca Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *Proceedings of the OWLED '06 Workshop on OWL: Experiences and Directions*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS, 2006.

[74] Sven Hurum. Handling scope ambiguities in English. In *Proceedings of the second Conference on Applied Natural Language Processing*, pages 58–65. Association for Computational Linguistics, 1988.

[75] Philip Inglesant, M. Angela Sasse, David Chadwick, and Lei Lei Shi. Expressions of expertness: The virtuous circle of natural language for access control policy specification. In *SOUPS '08: Proceedings of the 4th symposium on Usable privacy and security*, volume 337 of *ACM International Conference Proceeding Series*, pages 77–88. ACM, 2008.

[76] Bernard J. Jansen, Amanda Spink, and Tefko Saracevic. Failure analysis in query construction: Data and analysis from a large sample of Web queries. In *DL '98: Proceedings of the third ACM conference on Digital libraries*, pages 289–290. ACM, 1998.

[77] Mustafa Jarrar, Maria Keet, and Paolo Dongilli. Multilingual verbalization of ORM conceptual models and axiomatized ontologies. Technical report, STARLab, Vrije Universiteit Brussel, Belgium, February 2006.

[78] Christer Johansson, editor. *Proceedings of the Second Workshop on Anaphora Resolution (WAR II)*, volume 2 of *NEALT Proceedings Series*. Northern European Association for Language Technology (NEALT), 2008.

[79] Stephen C. Johnson. Yacc: Yet another compiler-compiler. Computer Science Technical Report 32, Bell Laboratories, Murray Hill, NJ, USA, July 1975.

[80] Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163, 1975.

[81] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*, second edition. Prentice Hall, 2009.

[82] Kaarel Kaljurand. *Attempto Controlled English as a Semantic Web Language*. PhD thesis, Faculty of Mathematics and Computer Science, University of Tartu, Estonia, December 2007.

[83] Kaarel Kaljurand. ACE View — an ontology and rule editor based on Attempto Controlled English. In Catherine Dolbear, Alan Ruttenberg, and Uli Sattler, editors, *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS, 2008.

[84] Kaarel Kaljurand. Paraphrasing controlled English texts. In Norbert E. Fuchs, ed-

itor, *Pre-Proceedings of the Workshop on Controlled Natural Language (CNL 2009)*, volume 448 of *CEUR Workshop Proceedings*. CEUR-WS, April 2009.

[85] Hans Kamp and Uwe Reyle. *From Discourse to Logic — Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*, volume 42 of *Studies in Linguistics and Philosophy*. Kluwer Academic Publishers, 1993.

[86] Ronald M. Kaplan and Joan Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, 1982.

[87] Robert T. Kasper and William C. Rounds. A logical semantics for feature structures. In *Proceedings of the 24th annual meeting on Association for Computational Linguistics*, pages 257–266. Association for Computational Linguistics, 1986.

[88] Esther Kaufmann and Abraham Bernstein. How useful are natural language interfaces to the Semantic Web for casual end-users? In *Proceedings of the 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference (ISWC 2007 + ASWC 2007)*, volume 4825 of *Lecture Notes in Computer Science*, pages 281–294. Springer, 2007.

[89] Donald E. Knuth. Backus normal form vs. backus naur form. *Communications of the ACM*, 7(12):735–736, 1964.

[90] Martin Krallinger, Florian Leitner, Carlos R. Penagos, and Alfonso Valencia. Overview of the protein-protein interaction annotation extraction task of BioCreative II. *Genome Biology*, 9(Suppl 2):S4, 2008.

[91] Markus Krötzsch, Denny Vrandečić, Max Völkel, Heiko Haller, and Rudi Studer. Semantic Wikipedia. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(4):251–261, December 2007.

[92] Tobias Kuhn, Loïc Royer, Norbert E. Fuchs, and Michael Schroeder. Improving text mining with controlled natural language: A case study for protein interations. In Ulf Leser, Barbara Eckman, and Felix Naumann, editors, *Data Integration in the Life Sciences — Proceedings of the Third International Workshop (DILS 2006)*, volume 4075 of *Lecture Notes in Bioinformatics*, pages 66–81. Springer, 2006.

[93] Tobias Kuhn and Rolf Schwitter. Writing support for controlled natural languages. In *Proceedings of the Australasian Language Technology Association Workshop 2008*, pages 46–54, December 2008.

[94] Christoph Lange. SWiM — a semantic wiki for mathematical knowledge management. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *The Semantic Web: Research and Applications — Proceedings of the 5th European Semantic Web Conference (ESWC 2008)*, volume 5021 of *Lecture Notes in Computer Science*, pages 832–837. Springer, 2008.

[95] Shalom Lappin and Herbert J. Leass. An algorithm for pronominal anaphora resolution. *Computational Linguistics*, 20(4):535–561, December 1994.

[96] Brenda Laurel, editor. *The Art of Human-Computer Interface Design*. Addison-Wesley, 1990.

[97] Gottfried W. Leibniz. Discours touchant la méthode de la certitude et de l'art d'inventer pour finir les disputes et pour faire en peu de temps de grands progrès, 1690. `http://fr.wikisource.org/wiki/Discours_touchant_la_m%C3%A9thode_de_la_certitude_et_l%E2%80%99art_d%E2%80%99inventer`.

[98] Douglas B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[99] Sergey Lukichev and Gerd Wagner. Verbalization of the REWERSE I1 rule markup language. Deliverable I1-D6, REWERSE, September 2006.

[100] Sergey Lukichev, Gerd Wagner, and Norbert E. Fuchs. Tool improvements and extensions 2. Deliverable I1-D11, REWERSE, March 2007.

[101] Martin Magnusson and Patrick Doherty. Temporal action logic for question answering in an adventure game. In Pei Wang, Ben Goertzel, and Stan Franklin, editors, *Artificial General Intelligence 2008: Proceedings of the First AGI Conference*, volume 171 of *Frontiers in Artificial Intelligence and Applications*, pages 236–247. IOS Press, February 2008.

[102] Frank Manola, Eric Miller, and Brian McBride. RDF primer. World Wide Web Consortium. W3C Candidate Recommendation, `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`, February 2004.

[103] Philippe Martin. The query and representation languages of WebKB-2. Distributed System Technology Centre, Griffith University. `http://www.webkb.org/doc/F_languages.html` (retrieved on 28 October 2009).

[104] Philippe Martin. Knowledge representation in CGLF, CGIF, KIF, Frame-CG and Formalized-English. In Uta Priss, Dan Corbett, and Galia Angelova, editors, *Conceptual Structures: Integration and Interfaces — Proceedings of the 10th International Conference on Conceptual Structures (ICCS 2002)*, volume 2393 of *Lecture Notes in Artificial Intelligence*, pages 77–91. Springer, 2002.

[105] John McDermott. R1: The formative years. *AI Magazine*, 2(2):21–29, 1981.

[106] Deborah L. McGuinness and Peter F. Patel-Schneider. Usability issues in knowledge representation systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference (AAAI-98, IAAI-98)*, pages 608–614. AAAI Press, 1998.

[107] Igor A. Mel'čuk. *Dependency Syntax: Theory and Practice*. State University of New York Press, 1988.

[108] Marvin Minsky. A framework for representing knowledge. Artificial Intelligence Memo 306, Massachusetts Institute of Technology, A.I. Laboratory, Cambridge, MA, USA, June 1974.

[109] Teruko Mitamura and Eric H. Nyberg. Controlled english for knowledge-based MT: Experience with the KANT system. In *Proceedings of the 6th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI95)*, pages 158–172. Centre for Computational Linguistics, Katholieke Universiteit Leuven, Belgium, 1995.

[110] Robert L. Mitchell. Cobol: Not dead yet. Computerworld. `http://www.computerworld.com/s/article/266156/Cobol_Not_Dead_Yet`, October 2006.

[111] Yusuke Miyao, Rune Sætre, Kenji Sagae, Takuya Matsuzaki, and Jun'ichi Tsujii. Task-oriented evaluation of syntactic parsers and their representations. In *Proceedings of ACL-08: HLT*, pages 46–54. Association for Computational Linguistics, June 2008.

[112] Moisés Salvador Meza Moreno and Björn Bringert. Interactive Multilingual Web Applications with Grammatical Framework. In Aarne Ranta and Bengt Nordström, editors, *Advances in Natural Language Processing — Proceedings of the 6th International Conference (GoTAL 2008)*, volume 5221 of *Lecture Notes in Artificial Intelligence*, pages 336–347. Springer, 2008.

[113] Eva-Martin Mueckstein. Controlled natural language interfaces: the best of three worlds. In *CSC '85: Proceedings of the 1985 ACM thirteenth annual conference on Computer Science*, pages 176–178. ACM, 1985.

[114] Gail C. Murphy, Mik Kersten, and Leah Findlater. How are java software developers using the Eclipse IDE? *IEEE Software*, 23(4):76–83, 2006.

[115] Hyun Namgoong and Hong-Gee Kim. Ontology-based controlled natural language editor using CFG with lexical dependency. In *Proceedings of the 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference (ISWC 2007 + ASWC 2007)*, volume 4825 of *Lecture Notes in Computer Science*, pages 353–366. Springer, 2007.

[116] Daniele Nardi and Ronald J. Brachman. An introduction to description logics. In Franz Baader, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 5–44. Cambridge University Press, 2003.

[117] Peter Naur, John W. Backus, Friedrich L. Bauer, Julien Green, Charles Katz, John McCarthy, Alan J. Perils, Heinz Rutishauser, Klaus Samelson, Bernard Vauquois, Joseph H. Wegstein, Adriaan van Wijngaarden, and Michael Woodger. Revised report on the algorithmic language ALGOL 60. *Communications of the ACM*, 6(1):1–17, January 1963.

[118] Ilkka Niemelä and Patrik Simons. Smodels - an implementation of the stable model and well-founded semantics for normal LP. In Jürgen Dix, Ulrich Fuhrbach, and Anil Nerode, editors, *Logic Programming and Nonmonotonic Reasoning — Proceedings of the 4th International Conference (LPNMR '97)*, volume 1265 of *Lecture Notes on Artificial Intelligence*, pages 421–430. Springer, 1997.

[119] Donald Nute. Defeasible logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 3, pages 353–395. Oxford University Press, 1994.

[120] Sharon O'Brien. Controlling controlled english — an analysis of several controlled language rule sets. In *Controlled Translation — Proceedings of the Joint Conference combining the 8th International Workshop of the European Association for Machine Translation and the 4th Controlled Language Application Workshop (EAMT-CLAW03)*, pages 105–114. Dublin City University, Ireland, 2003.

[121] Charles K. Ogden. *The A B C of Basic English (in Basic)*. Number 43 in Psyche

Miniatures General Series. K. Paul, Trench, Trubner, London, 1932.

[122] Terence J. Parr and Russell W. Quong. ANTLR: a predicated-LL(k) parser generator. *Software — Practice & Experience*, 25(7):789–810, July 1995.

[123] Charles S. Peirce. Existential graphs. In Charles Hartshorne and Paul Weiss, editors, *Collected Papers of Charles Sanders Peirce, Volume 4: The Simplest Mathematics*. Harvard University Press, 1932.

[124] Fernando Pereira. Extraposition grammars. *Computational Linguistics*, 7(4):243–256, 1981.

[125] Fernando Pereira and David H. D. Warren. Definite clause grammars for language analysis. In Barbara J. Grosz, Karen Sparck-Jones, and Bonnie L. Webber, editors, *Readings in Natural Language Processing*, pages 101–124. Morgan Kaufmann Publishers, 1986.

[126] Massimo Poesio. Semantic ambiguity and perceived ambiguity. In Kees van Deemter and Stanley Peters, editors, *Semantic Ambiguity and Underspecification*, number 55 in CSLI Lecture Notes, pages 159–201. CSLI Publications, 1996.

[127] Carl Pollard and Ivan Sag. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. Chicago University Press, 1994.

[128] Jonathan Pool. Can controlled languages scale to the Web? In *Proceedings of the 5th International Workshop on Controlled Language Applications (CLAW 2006)*, 2006.

[129] Richard Power, Donia Scott, and Roger Evans. What you see is what you meant: direct knowledge editing with natural language feedback. In Henri Prade, editor, *Proceeding of the 13th European Conference on Artificial Intelligence*, pages 677–681. John Wiley & Sons, 1998.

[130] Richard Power, Robert Stevens, Donia Scott, and Alan Rector. Editing OWL through generated CNL. In Norbert E. Fuchs, editor, *Pre-Proceedings of the Workshop on Controlled Natural Language (CNL 2009)*, volume 448 of *CEUR Workshop Proceedings*. CEUR-WS, April 2009.

[131] Ian Pratt-Hartmann. A two-variable fragment of English. *Journal of Logic, Language and Information*, 12(1):13–45, 2003.

[132] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. World Wide Web Consortium. W3C Recommendation, `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`, January 2008.

[133] Teodor C. Przymusinski. Stable semantics for disjunctive programs. *New Generation Computing*, 9(3–4):401–424, August 1991.

[134] Steven G. Pulman. Controlled language for knowledge representation. In *Proceedings of the First International Workshop on Controlled Language Applications (CLAW 96)*, pages 233–242. Katholieke Universiteit Leuven, Belgium, 1996.

[135] Axel Rauschmayer. Next-generation wikis: What users expect; how RDF helps. In Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel, editors, *Proceedings of the Third Workshop on Semantic Wikis — The Wiki Way of Semantics*, volume 360 of *CEUR Workshop Proceedings*. CEUR-WS, 2008.

[136] Alan L. Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, and Chris Wroe. OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In Enrico Motta, Nigel Shadbolt, Arthur Stutt, and Nicholas Gibbins, editors, *Engineering Knowledge in the Age of the SemanticWeb — Proceedings of the 14th International Conference (EKAW 2004)*, volume 3257 of *Lecture Notes in Artificial Intelligence*, pages 63–81. Springer, 2004.

[137] Howard Rheingold. An interview with Don Norman. In Brenda Laurel, editor, *The Art of Human-Computer Interface Design*, pages 5–10. Addison-Wesley, 1990.

[138] I. A. Richards and Christine M. Gibson. *English through Pictures.* Washington Square Press, 1945.

[139] Ronald G. Ross. *Principles of the Business Rule Approach.* Information Technology Series. Addison-Wesley, 2003.

[140] Ivan A. Sag and Thomas Wasow. *Syntactic Theory — A Formal Introduction.* Number 92 in CSLI Lecture Notes. CSLI Publications, 1999.

[141] Sebastian Schaffert. IkeWiki: A semantic wiki for collaborative knowledge management. In *15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'06)*, pages 388–396. IEEE Computer Society, 2006.

[142] Sebastian Schaffert, Julia Eder, Szaby Grünwald, Thomas Kurz, Mihai Radulescu, Rolf Sint, and Stephanie Stroka. KiWi — a platform for semantic social software. In Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel, editors, *Proceedings of the Fourth Workshop on Semantic Wikis — The Semantic Wiki Web*, volume 464 of *CEUR Workshop Proceedings.* CEUR-WS, 2009.

[143] Daniel Schwabe and Miguel Rezende da Silva. Unifying semantic wikis and semantic web applications. In Christian Bizer and Anupam Joshi, editors, *Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008)*, volume 401 of *CEUR Workshop Proceedings.* CEUR-WS, 2008.

[144] Uta Schwertel. *Plural Semantics for Natural Language Understanding — A Computational Proof-Theoretic Approach.* PhD thesis, University of Zurich, Switzerland, May 2005.

[145] Rolf Schwitter, Kaarel Kaljurand, Anne Cregan, Catherine Dolbear, and Glen Hart. A comparison of three controlled natural languages for OWL 1.1. In Kendall Clark and Peter F. Patel-Schneider, editors, *Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Directions*, volume 496 of *CEUR Workshop Proceedings.* CEUR-WS, 2008.

[146] Rolf Schwitter, Anna Ljungberg, and David Hood. ECOLE — a look-ahead editor for a controlled language. In *Controlled Translation — Proceedings of the Joint Conference combining the 8th International Workshop of the European Association for Machine Translation and the 4th Controlled Language Application Workshop (EAMT-CLAW03)*, pages 141–150. Dublin City University, Ireland, 2003.

[147] Rolf Schwitter and Marc Tilbrook. Controlled Natural Language meets the Seman-

tic Web. In Ash Asudeh, Cecile Paris, and Stephen Wan, editors, *Proceedings of the Australasian Language Technology Workshop 2004*, volume 2 of *ALTA Electronic Proceedings*, pages 55–62. Australasian Language Technology Association, 2004.

[148] Rolf Schwitter and Marc Tilbrook. Let's talk in Description Logic via controlled natural language. In *Proceedings of the Third International Workshop on Logic and Engineering of Natural Language Semantics (LENLS2006)*, June 2006.

[149] Semantics of business vocabulary and business rules (SBVR), v1.0. The Object Management Group. `http://www.omg.org/spec/SBVR/1.0/PDF`, January 2008.

[150] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The Semantic Web revisited. *IEEE Intelligent Systems*, 21(3):96–101, May 2006.

[151] Richard N. Shiffman, George Michel, Michael Krauthammer, Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Writing clinical practice guidelines in controlled natural language. In Norbert E. Fuchs, editor, *Proceedings of the Workshop on Controlled Natural Language (CNL 2009)*, volume 5972 of *Lecture Notes in Computer Science*, pages 265–280. Springer, 2010.

[152] Katharina Siorpaes and Martin Hepp. myOntology: The marriage of ontology engineering and collective intelligence. In *Proceedings of the International Workshop on Bridging the Gap between Semantic Web and Web 2.0*, pages 127–138. University of Kassel, Germany, 2007.

[153] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, June 2007.

[154] Douglas Skuce. An English-like language for qualitative scientific knowledge. In *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, pages 593–600, September 1975.

[155] Paul R. Smart, Jie Bao, Dave Braines, and Nigel R. Shadbolt. Development of a controlled natural language interface for semantic mediawiki. In Norbert E. Fuchs, editor, *Proceedings of the Workshop on Controlled Natural Language (CNL 2009)*, volume 5972 of *Lecture Notes in Computer Science*, pages 206–225. Springer, 2010.

[156] John F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations.* Brooks/Cole, 2000.

[157] John F. Sowa. Common logic controlled English (draft). `http://www.jfsowa.com/clce/specs.htm`, February 2004.

[158] John F. Sowa. Fads and fallacies about logic. *IEEE Intelligent Systems*, 22(2):84–87, March 2007.

[159] Silvie Spreeuwenberg and Keri Anderson Healy. SBVR's approach to controlled natural language. In Norbert E. Fuchs, editor, *Proceedings of the Workshop on Controlled Natural Language (CNL 2009)*, volume 5972 of *Lecture Notes in Computer Science*, pages 155–169. Springer, 2010.

[160] Mark Steedman. Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory*, 5(3):403–439, August 1987.

[161] Peter Strevens and Edward Johnson. SEASPEAK: A project in applied linguistics, language engineering, and eventually ESP for sailors. *ESP Journal*, 2(2):123–129, 1973.

[162] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, 25(1–2):161–197, March 1998.

[163] Geoff Sutcliffe. The CADE-21 automated theorem proving system competition. *AI Communications*, 21(1):71–81, January 2008.

[164] Geoff Sutcliffe, Christian B. Suttner, and Theodor Yemenis. The TPTP problem library. In Alan Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction (CADE-12)*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 252–266. Springer, 1994.

[165] Tommi Syrjänen. Lparse 1.0 user's manual. `http://www.tcs.hut.fi/Software/smodels/lparse.ps`, 2000.

[166] Valentin Tablan, Tamara Polajnar, Hamish Cunningham, and Kalina Bontcheva. User-friendly ontology authoring using a controlled language. In *Proceedings of LREC 2006 — 5th International Conference on Language Resources and Evaluation*, 2006.

[167] Talin. A summary of principles for user-interface design. `http://viridia.org/~talin/projects/ui_design.html`, August 1998.

[168] Roberto Tazzoli, Paolo Castagna, and Stefano E. Campanini. Towards a Semantic Wiki Wiki Web. In Jeremy J. Carroll, editor, *Poster Session at the 3rd International Semantic Web Conference (ISWC2004)*, November 2004.

[169] Harry R. Tennant, Kenneth M. Ross, Richard M. Saenz, Craig W. Thompson, and James R. Miller. Menu-based natural language understanding. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, pages 151–158. Association for Computational Linguistics, 1983.

[170] Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic reasoner: System description. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning — Proceedings of the Third International Joint Conference (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.

[171] Tania Tudorache, Jennifer Vendetti, and Natalya F. Noy. Web-Protégé: A lightweight OWL ontology editor for the Web. In Catherine Dolbear, Alan Ruttenberg, and Uli Sattler, editors, *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS, 2008.

[172] Charles A. Verbeke. Caterpillar fundamental English. *Training & Development Journal*, 27(2):36–40, February 1973.

[173] Christian Wagner. End users as expert system developers? *Journal of End User Computing*, 12(3):3–13, 2000.

[174] David L. Waltz. An english language question answering system for a large relational database. *Communications of the ACM*, 21(7):526–539, 1978.

[175] Chong Wang, Miao Xiong, Qi Zhou, and Yong Yu. PANTO: A portable natural language interface to ontologies. In Enrico Franconi, Michael Kifer, and Wolfgang

May, editors, *The Semantic Web: Research and Applications — Proceedings of the 4th European Semantic Web Conference (ESWC 2007)*, volume 4519 of *Lecture Notes in Computer Science*, pages 473–487. Springer, 2007.

[176] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, December 1945.

[177] Yorick Wilks, Xiuming Huang, and Dan Fass. Syntax, preference and right attachment. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, volume 2, pages 779–784. Morgan Kaufmann Publishers, 1985.

[178] Terry Winograd. *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, January 1971.

[179] William A. Woods. What's in a link: Foundations for semantic networks. In Daniel G. Bobrow and Allan Collins, editors, *Representation and Understanding: Studies in Cognitive Science*, pages 35–82. Academic Press, 1975.

[180] Adam Wyner, Krasimir Angelov, Guntis Barzdins, Danica Damljanovic, Brian Davis, Norbert Fuchs, Stefan Hoefler, Ken Jones, Kaarel Kaljurand, Tobias Kuhn, Martin Luts, Jonathan Pool, Mike Rosner, Rolf Schwitter, and John Sowa. On controlled natural languages: Properties and prospects. In Norbert E. Fuchs, editor, *Proceedings of the Workshop on Controlled Natural Language (CNL 2009)*, volume 5972 of *Lecture Notes in Computer Science*, pages 281–289. Springer, 2010.

Publications

Part of the content of this thesis has already been published in the form of the following scientific articles:

[181] Tobias Kuhn. AceRules: Executing rules in controlled natural language. In Massimo Marchiori, Jeff Z. Pan, and Christian de Sainte Marie, editors, *Web Reasoning and Rule Systems — First International Conference (RR 2007)*, volume 4524 of *Lecture Notes in Computer Science*, pages 299–308. Springer, 2007.

[182] Tobias Kuhn. AceWiki: A natural and expressive semantic wiki. In Duane Degler, mc schraefel, Jennifer Golbeck, Abraham Bernstein, and Lloyd Rutledge, editors, *Proceedings of the Fifth International Workshop on Semantic Web User Interaction (SWUI 2008) — Exploring HCI Challenges*, volume 543 of *CEUR Workshop Proceedings*. CEUR-WS, 2009.

[183] Tobias Kuhn. AceWiki: Collaborative ontology management in controlled natural language. In Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel, editors, *Proceedings of the Third Workshop on Semantic Wikis — The Wiki Way of Semantics*, volume 360 of *CEUR Workshop Proceedings*. CEUR-WS, 2008.

[184] Tobias Kuhn. Combining semantic wikis and controlled natural language. In Christian Bizer and Anupam Joshi, editors, *Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008)*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS, 2008.

[185] Tobias Kuhn. How controlled English can improve semantic wikis. In Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel, editors, *Proceedings of the Forth Semantic Wiki Workshop (SemWiki 2009)*, volume 464 of *CEUR Workshop Proceedings*. CEUR-WS, 2009.

[186] Tobias Kuhn. How to evaluate controlled natural languages. In Norbert E. Fuchs, editor, *Pre-Proceedings of the Workshop on Controlled Natural Language (CNL 2009)*, volume 448 of *CEUR Workshop Proceedings*. CEUR-WS, April 2009.

[187] Tobias Kuhn. An evaluation framework for controlled natural languages. In Norbert E. Fuchs, editor, *Proceedings of the Workshop on Controlled Natural Language (CNL 2009)*, volume 5972 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2010.

[188] Tobias Kuhn. Codeco: A grammar notation for controlled natural language in predictive editors. In Michael Rosner and Norbert E. Fuchs, editors, *Pre-Proceedings of the Second Workshop on Controlled Natural Languages (CNL 2010)*, volume 622 of *CEUR Workshop Proceedings*. CEUR-WS, 2010.

Curriculum Vitæ

Personal Information

Name	Tobias Kuhn
Date of birth	15 April 1981
Citizenship	Swiss

Education

Mar 2006 – Jan 2010 Doctoral student at the Department of Informatics and the Institute of Computational Linguistics of the University of Zurich; graduation as a Doctor in Informatics (Dr. Inform.)

Oct 2001 – Jan 2006 Studies in Economy and Computer Science ("Wirtschaftsinformatik") at the University of Zurich; graduation as a "Diplom-Informatiker" (Dipl. Inform.), equivalent to a Master of Science in Computer Science

1994 – 2001 High school ("Gymnasium") of the type mathematics and natural science in Zurich

1988 – 1994 Elementary school in Gutenswil and Volketswil (Switzerland)

Academic Activities and Teaching

Jun 2010 – Jul 2010 Guest at the Computer Science Department of the University of Chile in the PLEIAD group (Programming Languages and Environments for Intelligent, Adaptable and Distributed systems)

Oct 2006 – May 2008 Tutor at the Institute of Computational Linguistics of the University of Zurich for the lecture “Programming techniques in computational linguistics I/II”

Jan 2006 – Mar 2006 Guest at the Biotechnology Center of the Technical University Dresden in the Bioinformatics group

Mar 2005 – Jul 2005 Teaching assistant at the Department of Informatics of the University of Zurich for the lecture “Formal foundations of computer science”

Jul 2004 – Sept 2005 Internship and undergraduate research assistant at the Department for Computer Science of the ETH Zurich in the Global Information Systems group

Professional Experience (Non-academic)

Mar 2004 – May 2005 Several short employments at the Swiss Banking School (today called “Swiss Finance Institute”) as an assistant for the course “Fundamentals of Finance”

Nov 2002 – Dec 2005 Collaboration in two projects as a programmer at the department for nature conservation of the canton of Zurich

Jul 2001 – Mar 2002 Two temporary employments at Basoft (today part of Raiffeisen Switzerland) in Dietikon (Switzerland) as a customer supporter in the area of telebanking