Michael Rosner
Norbert E. Fuchs (Eds.)

LNAI 7175

Controlled Natural Language

Second International Workshop, CNL 2010
Marettimo Island, Italy, September 2010
Revised Papers

Springer

Lecture Notes in Artificial Intelligence 7175

Subseries of Lecture Notes in Computer Science

Michael Rosner Norbert E. Fuchs (Eds.)

Controlled Natural Language

Second International Workshop, CNL 2010
Marettimo Island, Italy, September 13-15, 2010
Revised Papers

Springer

Volume Editors

Michael Rosner
University of Malta
Department of Intelligent Computer Systems
Msida, Malta
E-mail: mike.rosner@um.edu.mt

Norbert E. Fuchs
University of Zurich
Department of Informatics
and Institute of Computational Linguistics
Zurich, Switzerland
E-mail: fuchs@ifi.uzh.ch

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-31174-1 e-ISBN 978-3-642-31175-8
DOI 10.1007/978-3-642-31175-8
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012939779

CR Subject Classification (1998): F.4.3, I.2.4, I.2.7, F.4, I.2, J.1, J.5, H.3, H.4

LNCS Sublibrary: SL 7 – Artificial Intelligence

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The idea of holding a second workshop on Controlled Natural Language (CNL 2010) was conceived at the end of the first workshop held in 2009, and put into practice a few months later when Norbert Fuchs asked for my collaboration. After forming a programme committee, a call for submissions of extended abstracts was sent out in April 2010. We decided that given the relatively young and fluid status of Controlled Natural Languages (CNLs) as a discipline, it would be unwise to focus on any one topic and so the call was oriented towards broad coverage of the field, stressing theoretical and practical aspects of CNLs, relations to other knowledge representation languages, tool support, and applications.

Altogether 17 extended abstracts were submitted, of which the programme committee accepted 12. In contrast and in addition to the 2009 workshop, the programme included four tutorials that were intended to attract students without extensive experience in the area. These were:

- Rolf Schwitter, Controlled Natural Languages for Knowledge Representation
- Tobias Kuhn, An Introduction to AceWiki
- Aarne Ranta, Multilingual Controlled Language Packages: Implementation and Applications
- Norbert E. Fuchs, First-Order Reasoning for Attempto Controlled English

The workshop took place 13–15 September 2010, once again at the magical venue of Marettimo Island, where a collection of researchers that included a number of PhD students successfully recreated that excellent combination of presentations, discussions, recreation, and Mediterranean lifestyle that characterised the first workshop.

Revised versions of the originally submitted abstracts were published in 2010 as a pre-proceedings (M. Rosner and N. E. Fuchs, editors. CNL 2010 Second Workshop on Controlled Natural Languages, volume 622 of CEUR Workshop Proceedings, September 2010. http://ceur-ws.org/Vol-622/). Authors of the original 12 papers were then asked to submit extended versions of their original revised abstracts for a definitive version of the proceedings. Tutorial authors were also invited to contribute. The result is the present volume containing ten extended papers.

As programme co-chair I would like to take the opportunity to thank the many people without whose support realisation of the workshop and creation of the proceedings would not have been possible. First and foremost I have to thank Norbert Fuchs for his inspiration, his unwavering support and generosity, and also for his excellent sense of minimal but efficient administration which enabled us to organise such an event at close to zero cost. He also took more than his fair share of the burden with respect to the numerous communications, written and spoken, that have led to the completion of this volume.

Secondly, I thank the members of the programme committee who carefully reviewed all submissions: Johan Bos (University of Groningen, The Netherlands), Peter E. Clark (The Boeing Company, Seattle, USA), Danica Damljanovic (University of Sheffield, UK), Norbert E. Fuchs (University of Zurich, Switzerland), Albert Gatt (University of Malta, Malta), Alfio Gliozzo (CNR-ISTC Rome, Italy), Siegfried Handschuh (DERI, University of Galway, Ireland), Stefan Hoefler (University of Zurich, Switzerland), Kaarel Kaljurand (University of Zurich, Switzerland), Peter Koepke (University of Bonn, Germany), Tobias Kuhn (University of Malta, Malta), Gordon Pace (University of Malta, Malta), Stephen Pulman (University of Oxford, UK), Aarne Ranta (Chalmers University of Technology, Sweden), Uta Schwertel (Information Multimedia Communication AG, Saarbrücken, Germany), Rolf Schwitter (Macquarie University, Australia), Donia Scott (University of Sussex, UK), Harold Somers (Dublin City University, Ireland), Geoff Sutcliffe (University of Miami, USA), Silvie Spreeuwenberg (LibRT B.V., Amsterdam, The Netherlands), Yorick Wilks (University of Sheffield, UK), Adam Wyner (University of London, UK).

Thirdly, I have already alluded to the wonderful physical environment in which the first two CNL workshops were held. The superb, invisible organisation behind the breakfasts, coffee breaks, and banquet were a testimony to the dedication of the people of the Marettimo Residence (Fausto and Federica Goppo, Giacomo Sardina, Vito Torrente, Teresa Vaccaro) for hosting the workshop. I must also mention the crucial role of Vito Vaccaro and L'Associazione Culturale, Sportiva, Ricreativa, Turistica Marettimo for generous support. To all of these local people, and to those fishermen whose names we do not know but who embody the spirit of the Mediterranean, *tante grazie per tutto*.

Finally, I am delighted to announce that Tobias Kuhn has taken up the gauntlet of chairing the third workshop on Controlled Natural Languages—CNL2012. This will be held at the University of Zurich in August 2012.

January 2012

Mike Rosner
Norbert E. Fuchs

Organisation

Programme Committee

Johan Bos	University of Groningen, The Netherlands
Peter Clark	The Boeing Company
Danica Damljanovic	University of Sheffield, UK
Norbert E. Fuchs	University of Zurich, Switzerland
Albert Gatt	University of Malta
Alfio Gliozzo	CNR - ISTC, Italy
Siegfried Handschuh	DERI, Galway, Ireland
Stefan Hoefler	University of Zurich, Switzerland
Kaarel Kaljurand	University of Zurich, Switzerland
Peter Koepke	University of Bonn, Germany
Tobias Kuhn	University of Zurich, Switzerland
Gordon Pace	University of Malta
Stephen Pulman	Oxford University, UK
Aarne Ranta	Chalmers University of Technology, Sweden
Mike Rosner	University of Malta
Uta Schwertel	IMC, Saarbrücken, Germany
Rolf Schwitter	Macquarie University, Australia
Donia Scott	University of Sussex, UK
Harold Somers	Dublin City University, Ireland
Silvie Spreeuwenberg	LibRT B.V., Amsterdam, The Netherlands
Geoff Sutcliffe	University of Miami, USA
Yorick Wilks	University of Sheffield, UK
Adam Wyner	University College London, UK

Additional Reviewers

Angelov, Krasimir
Dantuluri, Pradeep
Davis, Brian

Table of Contents

Typeful Ontologies with Direct Multilingual Verbalization

Krasimir Angelov and Ramona Enache

Department of Computer Science and Engineering,
Chalmers University of Technology and University of Gothenburg
{krasimir,ramona.enache}@chalmers.se

Abstract. We have developed a methodology for representation of ontologies in a strictly typed language with dependent types. The methodology is supported by an experiment where we translated SUMO (Suggested Upper-Merged Ontology) to GF (Grammatical Framework). The representation of SUMO in GF preserves the expressivity of the original ontology, adding to this the advantages of a type system and built-in support for natural language generation. SUMO is the largest open-source ontology describing over 10,000 concepts and the relations between them, along with a number of first-order axioms, which are further on used in performing automated reasoning on the ontology. GF is a type-theoretical grammar formalism mainly used for natural language applications. Through the logical framework that it incorporates, GF allows a consistent ontology representation, and thanks to its grammatical features the ontology is directly verbalized in a number of controlled natural languages.

Keywords: ontologies, type theory, knowledge representation, automated reasoning, natural language generation.

1 Introduction

The constantly growing amount of formal knowledge has brought about the necessity of a coherent and unambiguous representation of ontologies which can further be processed automatically. As a consequence, a number of ontology description languages like KIF [1], OWL [2], CycL [3] and Gellish [4] has emerged. However, the focus in all these languages is on the knowledge representation and consequently, they are mainly descriptive, leaving tasks such as consistency checking or natural language generation to external tools. Moreover, most languages are based on some kind of untyped first-order logic with predicates which occasionally allows higher-order constructions. They aim to maximize the expressivity with the cost of allowing set theoretical paradoxes to be expressed (Section 3). Also, because of the lack of a type system, one can easily extend such ontologies with axioms which are not well-formed. Although type information in these languages is often provided in the form of logical assertions, the validation of correctness is left to a reasoner which may or may not be able to find all problems. Even with a complete and decidable reasoner, if the ontological language has the open-world assumption, a potential problem might be left undiscovered, if it

M. Rosner and N.E. Fuchs (Eds.): CNL 2010, LNAI 7175, pp. 1–20, 2012.

is not stated explicitly that certain classes in the ontology are disjoint. This is a problem when dealing with large coverage ontologies. Or, for example, a predicate could be applied to an argument of the wrong type, or a small change in the signature of a function could lead to the update of all its occurrences. If all these checks are manual, then this is a resource-consuming and error-prone process. In contrast, database systems are equipped with rigid database schemas which ensure that the information is always kept consistent. The programming languages community was also dealing with that from the very beginning of computer science and has developed many different type systems.

We have developed a methodology for encoding of ontologies in strictly typed language based on type theory with dependent types. As a proof-of-concept, an experiment with SUMO [5], the largest open-source ontology available today. The implementation language of choice is GF [6]. The result is a controlled language which could be used to formulate new axioms in SUMO or to render existing axioms in natural language. Further on, we analyzed the difference of expressivity compared to the original ontology and also the other benefits that one can get from encoding an ontology in GF.

SUMO consists of 2 upper-level ontologies (*Merge*, *Mid-level-ontology*) describing general concepts, and 29 domain-specific ontologies for finances (*FinancialOntology*), geographical concepts (*Geography*), and others. The ontology is written in a dialect of KIF (Knowledge Interchange Format [1]), called SUO-KIF, which permits the declaration of concepts in a human-readable form, featuring support for expressing first-order predicate calculus constructions. However, due to the modelling of the hierarchy in SUMO, which treats functions and relations as ordinary concepts, it is possible to express second-order logic constructions in SUO-KIF, such as quantification over functions and relations.

The SUMO ontology has natural language translations for the *Merge* module in 12 languages. The translations are based on a set of string templates, which are combined by concatenation. They are hand-written and cover the ontology partially. However, the templates are not expressive enough to handle various natural language phenomena such as case and gender agreement or phonetic mutations. We will show how these problems were solved by using GF.

GF is a type-theoretical grammar formalism which distinguishes between abstract and concrete syntax. The abstract syntax is a logical framework based on Martin-Löf's type theory [7], in which the application domain can be described in an abstract language-neutral manner. The concrete syntax is a mapping of the abstract syntax into some controlled natural language. Since it is possible to have multiple concrete syntaxes, linked to the same abstract syntax, the abstract syntax acts as a semantic interlingua which allows simultaneous translation into multiple controlled languages.

We consider the abstract syntax of GF as a kind of ontology description language and translate some of the axioms from SUMO to statements in the abstract syntax of a grammar. Other axioms, those related to the natural language generation from SUMO, are used to generate the concrete syntax. The rest of the axioms are just converted to abstract syntax trees and used in the automated theorem prover for reasoning.

The development of a grammar for a new controlled natural language from scratch would involve an ad hoc implementation for low-level linguistic details such as word

order, agreement, etc. This is simplified by using the resource grammar library [8] developed in GF. The library provides an abstract syntax for common general-purpose natural language constructions and concrete syntaxes corresponding to 16 languages. The usage of the library ensures that the rendering is always syntactically correct and reduces the development effort for new application grammars. The resource grammar library was used for the generation of the concrete syntax of SUMO.

Another advantage of GF is the portability of the grammars, via PGF [9] – a runtime binary format, which can be used by applications written in Haskell, Java, JavaScript, C and Python - through the GF runtime system. In this way, the GF grammars can be embedded in user applications. GF has been used in various large-scale projects such as the dialogue system research project TALK [10], the educational project WebALT [11], the verification tool KeY [12], and the project in multilingual translation MOLTO [13].

Regarding the automated reasoning and the checking for consistency [14], SUMO was mapped to TPTP-FOF [15], a standard untyped first-order logic language, which is accepted by most theorem provers. There is an annual competition held during the premier conference in automated deduction, CADE[1], which awards prizes for finding inconsistencies in one of the two upper ontologies from SUMO, based on these mappings[2]. A similar translation from SUMO-GF to TPTP is provided. The translated ontology is checked for consistency and is used for making inferences on the abstract syntax trees or natural language, with the aid of an automated theorem prover (Section 6).

SUMO is also associated with a knowledge engineering environment – Sigma [16], which can be used for intelligent browsing of the ontology, optimized natural language generation and automated reasoning [17]. An alternative system with similar capabilities is the KSMSA browser[3]. The web user interface of GF also evolved in the direction of ontology browsing. Since his interface is still under development, we will give an overview of it in Section 7.

From the total number of ontologies that SUMO provides, 17 were translated into GF. These are: *Merge* and *Mid-level-ontology* – the upper ontologies and 12 domain ontologies. The remaining ontologies can also be ported to GF using the same techniques, in a semi-automatic way.

The advantages of representing the SUMO ontology in GF are the possibility to type-check the axioms and the definitions at an early stage and also to generate natural language of a higher syntactical quality. The translation to GF, is also an in-depth analysis of SUMO and the benefits that a type system in general, and GF in particular, could bring to ontology development.

2 The Abstract Syntax of SUMO-GF

The two languages SUO-KIF and GF have been created for different purposes and have evolved in different ways. It is not surprising that the translation of SUMO from SUO-KIF to an abstract syntax in GF is not trivial. Still we will show that the different

[1] http://www.cadeinc.org/

[2] http://www.cs.miami.edu/~tptp/Challenges/SUMOChallenge/

[3] http://virtual.cvut.cz/ksmsaWeb/main

ontological concepts - from classes and taxonomical relations to complex logical axioms have natural representations in GF.

2.1 The Taxonomy

The most central component of every ontology is the taxonomy of classes, and this is the starting point from where we begin the ontology modelling in GF.

Knowledge representation languages like OWL, KIF and CycL do not set a sharp border between classes and instances. In fact, the classes are just instances of one special class which is the class of all classes. In SUMO the special class is called *Class* and there is a predicate *subclass* which is used to assert the taxonomical relations. For example, the axiom:

$$(subclass\ Human\ Hominid) \tag{1}$$

asserts that the class *Human* is a subclass of *Hominid*. Furthermore, there is an axiom stating that everything that is a subclass of *Entity* is also an instance of *Class* and viceversa:

$$\begin{aligned}(<=>\ &(instance\ ?CLASS\ Class)\\ &(subclass\ ?CLASS\ Entity))\end{aligned}$$

Since the *subclass* relation is transitive and *Entity* is the most general class, from the axiom:

$$(subclass\ Class\ SetOrClass)$$

it follows that *Class* is itself an instance of *Class*:

$$(instance\ Class\ Class)$$

This kind of cyclic relations were proven to be inconsistent because they lead to different kinds of paradoxes (Section 3). The other two popular languages OWL and CycL are not exceptions and similar examples could be constructed in them as well. This seems to be a common mistake because the first version of Martin-Löf's [18] type theory suffered from the same inconsistency which was first demonstrated with Girard's paradox [19]. The problem was resolved in the later versions of the theory [7] by introducing the concepts of small and big types. In the context of SUMO, this would be translated as a restriction which states that *Class* cannot be an instance of *Class* because it is too big to fit as an instance of itself. The abstract syntax of GF is a logical framework consistent with the modern type theory, so if we want to model ontologies like SUMO in GF we have to resolve the conflict.

GF distinguishes between values and types. Every value belongs to some type but none of the types could be a value as well, so it is not possible for a type to belong to another type. The solution for the cyclic relation in SUMO is to declare that *Class* is a type:

$$\textbf{cat}\ Class;$$

Now the classes will be values of type *Class*. For instance:

$$\begin{aligned}\textbf{fun}\ &\mathit{Entity} : \mathit{Class};\\ &\mathit{Hominid} : \mathit{Class};\\ &\mathit{Human} : \mathit{Class};\end{aligned}$$

Essentially, we cut the class *Class* from the common hierarchy and move it to another level (also known as universe in type theory).

Once we have a way to define classes in the abstract syntax we could also define the taxonomy. In SUMO, the taxonomy is encoded by using the *subclass* predicate. In GF, we can translate *subclass* either as a function or as a type. Since we want to be able to statically check the axioms for well-formedness we choose to represent the predicate as a type:

$$\textbf{cat}\ \mathit{SubClass}\ \mathit{Class}\ \mathit{Class};$$

then the human-hominid relation could be asserted as:

$$\textbf{fun}\ \mathit{Human\_Class} : \mathit{SubClass}\ \mathit{Human}\ \mathit{Hominid}; \tag{2}$$

Here, the *SubClass* type is an example of a dependent type. The dependent types are not just simple identifiers, but have in addition indices of some type. In this case, *SubClass* is a type indexed by two values of type *Class*. In the case of *Human_Class* those are *Human* and *Hominid*.

Note that while in the original SUMO axiom (1) we had just a logical assertion, in GF we have to assign an unique identifier (*Human_Class*) to it. In type theory this is deeply rooted in Curry—Howard's correspondence, but it is interesting that a similar kind of "labeling" of assertions is now emerging in OWL via the Named Graphs standard [20].

Semantically the *subclass* predicate in SUMO encodes the reflexive transitive closure of the taxonomic relation, while the immediate subclass relation is encoded using the predicate *immediateSubclass*. To take this into account we define one more type:

$$\textbf{cat}\ \mathit{Inherits}\ \mathit{Class}\ \mathit{Class};$$

Strictly speaking the *SubClass* type is the translation of the predicate *immediateSubclass* and *Inherits* is the translation of *subclass*. However we choose to read simple *subclass* axioms such as (1) as assertions for immediate subclassing and thus the conversion tool will generate the *SubClass* type in GF. The reason for this is that this would let us do some reasoning with the ontology by using only the tools that are already available in GF. Our intuition is that this still preserves the principal information from SUMO.

From the atomic *SubClass* axioms we can easily infer the reflexive-transitive closure *Inherits*. All that is needed is to add two inference rules. The inference rules in type theory are nothing else but functions with some specific type signatures:

$$\begin{aligned}\textbf{fun}\ &\mathit{inhz} : (c : \mathit{Class}) \to \mathit{Inherits}\ c\ c;\\ &\mathit{inhs} : (c_1, c_2, c_3 : \mathit{Class}) \to \mathit{SubClass}\ c_1\ c_2\\ &\qquad \to \mathit{Inherits}\ c_2\ c_3 \to \mathit{Inherits}\ c_1\ c_3;\end{aligned}$$

The type of function *inhz* states that every class c inherits itself, i.e. this is the reflexivity axiom. The second function *inhs* expresses the transitivity over *SubClass*, i.e. if c_1 is a subclass of c_2, and c_2 inherits c_3 then c_1 inherits c_3.

The inference rules can be applied using the inference engine built into GF. For example, from the GF shell the user can use the `gt` command to generate an expression of a given type:

```
SUMO> gt -cat="Inherits Human Hominid"
(inhs Human Hominid Hominid Human_Class (inhz Hominid))
```

In type theory the types are seen as logical propositions and the existence of a value of a given type is interpreted as an evidence for the validity of the proposition. The value is also a constructive recipe for building the proof from the axioms in the theory. In Section 2.4 we will use it to generate explanations in natural language for the proofs.

Some of the classes in SUMO have two or more superclasses. For instance *Human* is both a *CognitiveAgent* and a *Hominid*. In other situations it is necessary to quantify over instances of the union of two or more classes. For that purpose we added two of the primitive operations from description logic – intersection and union of classes:

$$\begin{aligned} &\textbf{fun}\ \mathit{both} : \mathit{Class} \to \mathit{Class} \to \mathit{Class}; && \text{– intersection} \\ &\quad \mathit{either} : \mathit{Class} \to \mathit{Class} \to \mathit{Class}; && \text{– union} \end{aligned}$$

With the help of these primitives, the full definition of the class *Human* is:

$$\textbf{fun}\ \mathit{Human\_Class} : \mathit{SubClass}\ \mathit{Human}\ (\mathit{both}\ \mathit{CognitiveAgent}\ \mathit{Hominid}); \qquad (3)$$

The reasoning with these two new primitives can be axiomatized with three new inference rules:

$$\begin{aligned} &\textbf{fun}\ \mathit{bothL} : (c_1, c_2 : \mathit{Class}) \to \mathit{SubClass}\ (\mathit{both}\ c_1\ c_2)\ c_1; \\ &\quad \mathit{bothR} : (c_1, c_2 : \mathit{Class}) \to \mathit{SubClass}\ (\mathit{both}\ c_1\ c_2)\ c_2; \\ &\quad \mathit{eitherC} : (c_1, c_2, c_3 : \mathit{Class}) \to \\ &\qquad \mathit{SubClass}\ c_1\ c_3 \to \mathit{SubClass}\ c_2\ c_3 \to \mathit{SubClass}\ (\mathit{either}\ c_1\ c_2)\ c_3; \end{aligned}$$

The first two state that the intersection class of any two classes c_1 and c_2 is a subclass of both c_1 (function *bothL*) and c_2 (function *bothR*). The third function (*eitherC*) states that if two classes c_1 and c_2 are both subclasses of c_3, then their union class is also a subclass of c_3. Now, with the extended definition for *Human* (3), the proof that every Human is a kind of Hominid will use the function *bothR*:

```
SUMO> gt -cat="Inherits Human Hominid"
(inhs Human (both CognitiveAgent Hominid) Hominid Human_Class
(inhs (both CognitiveAgent Hominid) Hominid Hominid
(bothR CognitiveAgent Hominid) (inhz Hominid)))
```

At least in some cases, the criterion which distinguishes the members of a given class from the super class is formally specified. In this case the criterion is specified in SUMO as an axiom. In our encoding we found it handy to use an encoding which uses the

KappaFn function. *KappaFn* is a function in SUMO which takes a logical formula and returns the class of all instances for which the formula is valid. The type of the function in GF is:

$$\textbf{fun}\ KappaFn : (c : Class) \to (\,Var\ c \to Formula) \to Class; \tag{4}$$

It takes as arguments the superclass c and the logical formula and returns the subclass. The type $(\,Var\ c \to Formula)$ indicates that the argument itself is a function which takes a variable of class c and returns a formula. Every instance of c for which the formula is true is also a member of the new subclass. Using *KappaFn* it is trivial to define the class *NegativeRealNumber* as a subclass of *RealNumber*:

$$\begin{aligned}&\textbf{fun}\ NegativeRealNumber : Class;\\ &\textbf{def}\ NegativeRealNumber = KappaFn\ RealNumber\ (\backslash N \to lessThan\ \ldots);\end{aligned}$$

Again for the inference of the transitive closure to work we need an inference rule:

$$\begin{aligned}\textbf{fun}\ kappa : (c : Class) \to (p : Var\ c \to Formula) \to\\ SubClass\ (KappaFn\ c\ p)\ c;\end{aligned}$$

which defines the semantics of *KappaFn*, i.e. that the new class is a subclass of the argument of the function.

2.2 Instances

Once we have the taxonomy of the ontology we can proceed with adding some instances. Similarly with the classes we will distinguish between direct instances of a class and generalized instances. The instances will be defined as values of one of the following types:

$$\begin{aligned}\textbf{cat}\ &El\ Class;\\ &Ind\ Class;\end{aligned}$$

The type $Ind\ c$ is assigned to all instances with principal class c, while $El\ c$ is the type of all direct instances of c together with the instances of its subclasses. There is an injection between this two types:

$$\textbf{fun}\ el : (c_1, c_2 : Class) \to Inherits\ c_1\ c_2 \to Ind\ c_1 \to El\ c_2;$$

The function *el* injects an instance with principal class c_1 into the type of the generalized instances of c_2, if there is an evidence that c_1 is a subclass of c_2 (the argument $Inherits\ c_1\ c_2$). For example in the CountriesAndRegions module of SUMO there is an instance for the city of London:

$$\textbf{fun}\ LondonUnitedKingdom : Ind\ EuropeanCity;$$

The class *EuropeanCity* is a subclass of *City* so it is possible to do the coercion. The following expression is the injection of *LondonUnitedKingdom* into the generalized instances of *City*:

$$\begin{array}{l} el\ EuropeanCity\ City \\ \quad (inhs\ EuropeanCity\ City\ City\ EuropeanCity\_Class\ (inhz\ City)) \\ \quad LondonUnitedKingdom \end{array}$$

2.3 Functions, Predicates and Logical Formulas

In SUMO, all functions and predicates are represented as instances of a descendant of *Relation*, and the expected classes of the arguments and the result are stated as axioms in the ontology. For example the definition of the *AbsoluteValueFn* function is:

$$\begin{array}{l} (instance\ AbsoluteValueFn\ UnaryFunction) \\ (domain\ AbsoluteValueFn\ 1\ RealNumber) \\ (range\ AbsoluteValueFn\ NonnegativeRealNumber) \end{array}$$

Here the predicates *domain* and *range* specify the class of the first argument and the class of the returned value. The class of *AbsoluteValueFn* itself is *UnaryFunction* which encodes the fact that this is a function with only one argument. From this SUMO axioms we generate a type signature in GF:

$$\textbf{fun}\ AbsoluteValueFn : El\ RealNumber \to Ind\ NonnegativeRealNumber;$$

Note that with our implementation we impose the closed world assumption. The argument of *AbsoluteValueFn* is declared of type $El\ RealNumber$, and the only way to construct a value of that type is to combine an instance of some subclass c of *RealNumber* with a proof object of type:

$$Inherits\ c\ RealNumber$$

If this object cannot be constructed from the current state of the knowledge base then the application of *AbsoluteValueFn* is not possible.

The predicates are declared in a way very similar to the functions. The only difference is that while the functions return some instance, the predicates are used to create logical formulas. In the original ontology, there is already a class called *Formula* which represents the class of all well-formed SUO-KIF formulas. In principle the predicates could return $Ind\ Formula$ but there are two reasons for which we choose not to do that. The first reason is that if *Formula* is kept as a class then this would allow quantification over logical formulas which is not supported in first-order logic. The second reason is that when the logical axioms are translated to natural language then *Formula* will correspond syntactically to a sentence while *Ind* corresponds to a noun phrase, and this would make the verbalization of the ontology difficult. Instead we declared *Formula* as a type:

$$\textbf{cat}\ Formula;$$

The last piece that is needed to be able to write logical axioms in GF is to add the standard logical quantifiers and connectives:

$$\begin{aligned}
&\textbf{cat } \mathit{Var}\ \mathit{Class};\\
&\textbf{fun } \mathit{var} : (c_1, c_2 : \mathit{Class}) \to \mathit{Inherits}\ c_1\ c_2 \to \mathit{Var}\ c_1 \to \mathit{El}\ c_2;\\
&\\
&\textbf{fun } \mathit{exists} : (c : \mathit{Class}) \to (\mathit{Var}\ c \to \mathit{Formula}) \to \mathit{Formula};\\
&\quad \mathit{forall} : (c : \mathit{Class}) \to (\mathit{Var}\ c \to \mathit{Formula}) \to \mathit{Formula};\\
&\textbf{fun } \mathit{not} : \mathit{Formula} \to \mathit{Formula};\\
&\quad \mathit{and}, \mathit{or}, \mathit{impl}, \mathit{equiv} : \mathit{Formula} \to \mathit{Formula} \to \mathit{Formula};
\end{aligned}$$

The only specific thing here is how the variables are introduced by the quantifiers. The first argument of the quantifier (function *exists* or *forall*) is the class over which the function quantifies. The second argument is the formula over which it scopes. The quantified variable itself is a high-order argument of type $\mathit{Var}\ c$. This type plays a role similar to the role of El. While the former denotes some known instance, for Var we neither know the instance, nor its principal class. This is reflected for example in natural language generation where the grammatical gender is deduced from the class of the variable instead of the instance itself. This special treatment of variables allows the generation of more fluent natural language. Still the *var* function allows the coercion from type *Var* to *El*.

With the usage of quantifiers and connectives all axioms from SUMO, which were not already converted to type signatures in GF, can be converted to abstract syntax trees. For example the SUO-KIF formula:

$$\begin{aligned}
&(=> (\mathit{instance}\ ?P\ \mathit{Wading})\\
&\quad (\mathit{exists}\ (?W)\ (\mathit{and}\ (\mathit{instance}\ ?W\ \mathit{BodyOfWater})\ (\mathit{located}\ ?P\ ?W))))
\end{aligned}$$

is converted to the following abstract syntax tree in GF:

$$\mathit{forall}\ \mathit{Wading}\ (\backslash P \to \mathit{exists}\ \mathit{BodyOfWater}\ (\backslash W \to \quad \mathit{located}\ (\mathit{var}\ P)\ (\mathit{var}\ W)))$$

Note that this is more than just a syntactic conversion because the quantifiers in GF expect explicit class information while in SUMO this is encoded with *instance* predicates.

2.4 Proofs in Natural Language

As it was mentioned in Section 2.1, the proofs in GF are explicitly represented as abstract syntax trees. Since the abstract syntax trees could also have linearizations in the concrete syntax, it is possible to render the proofs in the same controlled natural language encoding the ontology. For example the following command in the GF shell:

```
SUMO> gt -cat="Inherits Human Primate" | l -lang=SUMOEng
```

will derive a proof for $\mathit{Inherits}\ \mathit{Human}\ \mathit{Primate}$ and will linearize the proof in English. The text contains some HTML tags, so when it is rendered in a web browser it looks like a bullet list:

– *human is a subclass of both cognitive agent and hominid*
– *hominid is a subclass of primate*

The natural language rendering can be used to generate end-user explanations for the inferences in the ontology.

3 Russell's Paradox

Russell's paradox [21] was first discovered in naïve set theory. It stems from the assumption that for every logical proposition there is a set of entities which satisfy the proposition. This was shown to be inconsistent with the example of the set of all sets which are not members of themselves. Such a set cannot exist because then it will be simultaneously a member and not a member of itself. The design of SUMO follows naïve set theory and the *KappaFn* function is exactly the way to build sets from propositions. Using the function, the paradox can be expressed as:

$$(instance\ (KappaFn\ "x"\ (not\ (instance\ x\ x)))$$
$$(KappaFn\ "x"\ (not\ (instance\ x\ x))))$$

The reasoning with SUMO is sound only because the *KappaFn* function is not axiomatised and the automated theorem provers cannot make any inferences.

The paradox is principally avoided in the GF translation by first discarding the predicate *instance* and second by making the class *Class* into a type. This results into a completely different signature for KappaFn (4) which would make the above statement incorrect even if we still had the predicate *instance*.

4 Verbalization

Apart from the advantages that the GF type system provides, for the natural language generation the benefits of using GF are considerably more substantial. The present work deals with the generation of natural language for the two upper ontologies - *Merge* and *Mid-level-ontology* in 3 languages: English, Romanian and French, as a proof-of-concept for the capabilities of GF to host a controlled language for ontologies.

For English, about 7,000 concepts and relations have been translated to natural language. For Romanian and French, only a small number of examples, that illustrate the advantages of GF over a template-based generation, were built. This is due to the fact that there are no large coverage lexicons for those languages in GF yet.

A typical SUMO template is the predicate `age` expressed in English:

```
(format en age "the &%age of %1 is %n %2")
```

where %n will be replaced with "not" for the negation of the predicate, and with the empty string for the affirmative form. The structure of the templates is rather simple, and works reasonably just for morphologically simple languages, such as English. The templates do not take into account the presence of declension forms for nouns, of the

gender agreement with verbs and prepositions or the various moods of a sentence, depending on its usage.

This solution is not compositional and leads to incorrect constructions in languages with a rich morphology such as Romanian. For example the verbalization of "the inverse of the square root of X" in Romanian would require the combination of two templates and would render: *inversa lui rădăcina pătrată a lui X*, which is considerably different from the correct form - *inversa rădăcinii pătrate a lui X*. One reason is that the translation of "square root" should be in Genitive case, whereas the template only has the Nominative one, and in Romanian the two forms are different. The second is the matter of the possessive preposition, which in Romanian needs to agree with its object. The template provides the masculine form as default, but *rădăcina pătrată a lui X* is feminine. For French, although nouns do not have multiple declension forms, there is an agreement in gender and number between nouns and other parts of speech that determines them, which cannot be handled by the SUMO templates. In addition to this, for French there is also the problem of phonetic mutations, such as for the usage of a verb with negative polarity. In case that the verb starts with a vowel, the form of the particles used to express negation changes, and this is a mutation that SUMO doesn't handle, because the templates provide only one value for the particles. It goes without saying that the French and Romanian resource grammars offer solutions for these problems, so that the natural language generation in SUMO-GF is syntactically correct for compositions of patterns also.

Moreover, the feature that shows best the advantage of a typed system in general, and of GF, in particular, over sets of templates is the assignment of a gender to the variables, according to the gender of their type, for languages that have gender agreement [22]. This is a very common feature for Romance and Slavic languages, where there is a gender differentiation. The SUMO templates simply assume that all the variables have masculine gender, while in GF, the wrapper function `var`, that has access to the class of the variable also, would assign a proper gender to the variable. Since variables can only be used after being wrapped with `var`, they will have a correct gender for any usage in a quantified formula. This behaviour shows the importance of separating between variables and instances of a class. If `Var` and `Ind` or `El` would have been unified in the same category, we could not use a wrapper function to change the gender, since we might accidentally change the gender of an ordinary instance.

An example of how the gender variation feature works in the current implementation is the GF axiom:

$$forall\ Animal\ (\backslash A \rightarrow exists\ Animal\ (\backslash B \rightarrow smaller\ (var B)\ (var A)))$$

which would be linearized in French as:

pour chaque animal A il existe un animal B `tel` *que B est plus* `petit` *que A*

where animal is of masculine gender in French. For a type of feminine gender, such as house we would have that:

$$forall\ House\ (\backslash A \rightarrow exists\ House\ (\backslash B \rightarrow smaller\ (var B)\ (var A)))$$

which would be linearized in French as:

pour chaque maison A il existe une maison B `telle` *que B est plus* `petite` *que A*

The axioms are not taken from SUMO, but they are just two examples that illustrate this linguistic feature, and would not probably hold in general, as the set of animals and the set of houses are finite, and hence noetherian.

The examples, although few, show the advantages of GF in developing a set of multilingual aligned syntactically-correct controlled languages for describing ontologies.

Besides axioms, we can also generate natural language for `SubClass`, `Ind` declarations and higher-order functions. For example:

beverage is a subclass of food
blue is an instance of primary color
"x is equal to y" is an equivalence relation

Our work provides natural language generation in English for the two biggest modules *Merge* and *Mid-level-ontology* and two domain specific: *Elements* - featuring chemical substances and *Mondial* - featuring countries and cities of the world. As mentioned before, a total of almost 7 000 objects and 500 relations from SUMO were verbalized. This process is done automatically for objects and semi-automatically for relations, and uses the GF resource grammar.

The automatic process takes advantage of the camel case representation of SUMO concepts. For example, *BodyOfWater* will be rendered as ”body of water” and parsed by GF as a noun phrase. Instances are parsed as GF noun phrases, while classes are parsed as GF common nouns, which are similar to noun phrases, only that they have variable number, gender and other morphological features. In this way, the representation of *BodyOfWater* will also contain the plural form "bodies of water", which we can use for generating natural language constructions. For functions and predicates the missing arguments are replaced by some dummy variables and the procedure is semi-automatical, using the original SUMO templates and hand-written verbalizations which are further on parsed as noun phrases for functions and clauses with polarity for predicates. For example, the binary predicate `parent` will be verbalized as "o1 is the parent of o2" and parsed to a GF abstract syntax tree. This method allows generalizations, so the 2 negative forms are "o1 is not the parent of o2" and "o1 isn’t the parent of o2" are automatically obtained from this. For the two domain specific ontologies, the information is extracted from the SUMO predicate `name` that gives the English verbalization of the concepts. As a result, our approach renders verbalization of a large number of entries from the ontology, with a high rate of automation, ensuring syntactical correctness of the generated phrases. For example :

For every unique list LIST, every positive integer NUMBER2 and every positive integer NUMBER1, we have that if the element with number NUMBER1 in LIST is equal to the element with number NUMBER2 in LIST, then NUMBER1 is equal to NUMBER2.

For the same axiom, the SUMO templates generate:

For all unique list ?LIST holds for all ?NUMBER1, ?NUMBER2 holds if ?NUMBER1th element of ?LIST is equal to ?NUMBER2th element of ?LIST, then ?NUMBER1 is equal to ?NUMBER2

The optimized natural language generation mechanism from the Sigma system would render the axiom as:

* *If a list is an instance of unique list*
* *then for all a positive integer and positive integer*
  - *if positive integerth element of list is equal to positive integerth element of list*
  - *then positive integer is equal to positive integer*

Further optimizing of the code by anaphora generation and a list-like structure of the arguments for better readibility is possible, like in the proof rendering.

5 Evaluation

During the translation of SUMO to GF, we discovered a number of small inconsistencies in the original ontologies like mismatches between instances and classes, usage of undefined objects and usage of functions with a wrong number of arguments. This represents almost 8% of the total number of axioms from SUMO and was determined automatically during the type-checking phase. In addition to this, we left out the higher-order logic constructions such as quantifications on `Formula` or axioms with higher-order functions.

However, there are some types of axioms which could not be ported to SUMO-GF, such as the ones that use quantification over classes, negative type declarations and axioms which use the predicates `subclass`, `range` or `domain`. In addition to this, we mention the class of axioms which feature conditional type declarations. For example:

$$(=> \ (and \ (instance \ ?DRINK \ Drinking) \\ (patient \ ?DRINK \ ?BEV)) \\ (instance \ ?BEV \ ?Beverage))$$

The type declaration for *BEV* appears as a consequence of the fact that it is used in the process of *Drinking*. The total number of axioms which are lost in translation is about 23%. Our observations suggests that those axioms could be paraphrased and incorporated in the type system but this would require manual work with every axiom.

6 Automated Reasoning in SUMO-GF

Since SUMO offers a generous amount of information in a first-order logic format, it represents an excellent source for automated reasoning, especially in the context of SUMO-GF where one can perform automated reasoning on natural language.

As mentioned before, the TPTP-FOF translations of the 2 upper SUMO ontologies are used yearly in the ATP competition. We have shown already in Section 2 that a limited kind of ontological reasoning is possible by using GF alone. Unfortunately, the reasoner in GF is not as optimized as current state of the art theorem provers. However, to take advantage of the tools that already exists, we translated the 17 SUMO-GF ontologies to TPTP-FOF, checked them for consistency and used them for solving small inferences.

Since TPTP is an untyped system, whereas GF is strongly typed, the information about types needs to be reformulated, with the aid of an additional predicate `hasType`, that resembles the original `instance` predicate from SUMO.

For subclasses, the translation reflects the possibility of coercing from the subclass to the superclass:

$$\textbf{fun}\ \mathit{Adjective\_Class} : \mathit{SubClass\ Adjective\ Word};$$

and would be translated to TPTP as:

```
fof(axMerge2, axiom, (![X]:
    (hasType(type_Adjective, X)=>hasType(type_Word, X)))).
```

For instance declarations, we have a simpler translation pattern:

$$\textbf{fun}\ \mathit{Flat} : \mathit{Ind\ ShapeAttribute};$$

will be translated to TPTP as:

```
fof(axMerge686, axiom,
     hasType(type_ShapeAttribute, inst_Flat)).
```

A more commonly used approach for expressing typing declarations in first-order logic is to create a predicate for each type, like:

```
type_ShapeAttribute (inst_Flat)
```

We did not choose this method, since the SUMO classes are not just used as types, in typing declarations, but also as arguments for some functions. By using classes as predicates, one couldn't unify the two occurrences in first-order logic.

The functions that manipulate *Formula* objects, such as *not*, *and*, *or*, *impl* and *equiv* have been translated into their corresponding first-order logic operators that are predefined in TPTP: $\sim$, $\&$, $|$ and $\Rightarrow$. For the *both* and *either* constructors, the built-in $\&$ and $|$ are used again:

$$\textbf{fun}\ \mathit{Togo} : \mathit{Ind}\ (\mathit{both\ Country\ Nation});$$

will be translated to TPTP as:

```
fof(axmond72, axiom,
     hasType(type_Country, inst_Togo) &
               hasType(type_Nation, inst_Togo)).
```

As for the equality operator *equal*, the situation is more complicated. In SUMO, because of the structure of the concepts, it could basically take any arguments, like classes, and relations and instances. In GF the *equal* function would just take arguments of type *El Entity*, so it would not be possible to test the equality of formulas, functions or classes. In SUMO, *equal* is defined as an *EquivalenceRelation*, with some extra axioms, for the various kinds of arguments that it might take. For instances, the axiom, that verifies a property of equal objects:

$$\begin{array}{l}(=>\ (\mathit{equal}\ ?\mathit{THING1}\ ?\mathit{THING2})\\ \quad(\mathit{forall}\ (?\mathit{CLASS})\\ \qquad(<=>\ (\mathit{instance}\ ?\mathit{THING1}\ ?\mathit{CLASS})\\ \qquad\quad(\mathit{instance}\ ?\mathit{THING2}\ ?\mathit{CLASS}))))\end{array}$$

could not be translated to GF, as it contains a variable type declaration and quantification over a class. Moreover, a more solid interpretation of equality would be using at least a congruence relation, not just an equivalence one. SUMO does not have the concept of congruence, while theorem provers that can process first-order logic with equality, usually have specific mechanisms for dealing with the built-in equality from TPTP [23]. For these reasons, the translation from GF to TPTP, uses the default TPTP equality for the `equal` function.

The existential and universal quantifiers from SUMO and GF, were translated as the built-in quantifiers from TPTP. The type declarations are expressed with the function `hasType` for consistency with the type declarations. For example, the axiom (**??**) was translated to TPTP as:

```
fof(axMid9, axiom, ![Var_P]:
 hasType(type_Wading, Var_P) => ?[Var_W]:
 hasType(type_BodyOfWater, Var_W) & f_located(Var_P,Var_W)).
```

A special case is the translation of higher-order axioms to TPTP. In this case, the function call is replaced by the definition of the function, rendering a construction in first-order logic. In this way the function name is used as a macro for its body. This is the same approach as in [14].

The resulting files have been checked with the first-order theorem prover**E** [24]. **E** is a multiple award-winning theorem prover which is freely available and is based on an equational superposition calculus. It provides support for first-order logic with equality. **E** has been used to check the consistency of the largest ontology currently available - ResearchCyC [25]. The TPTP translations of the GF files were tested for consistency with **E**, and no contradiction was found, given the time limit of 1 hour per file, which was exceeded for the upper-ontologies, due to the increased complexity of the axioms they contain.

Regarding typical inferences that could be solved on the existing data, we used the problems from the SUMO webpage [4].

The category of axioms that the SUMO to TPTP translation can express, but the SUMO-GF to TPTP cannot are mainly the ones that got lost in the SUMO to SUMO-GF translation. In addition to this, there are the nested predicates, quantifications over `Formula` and the class-forming function `KappaFn`. They are used in SUMO-GF only for language generation. The loss is almost 23% of the total number of the axioms. The coverage of the SUMO to GF to TPTP translation is comparable to the direct SUMO to TPTP translation. It is worth mentioning that the first translation yields to a slightly slower system because of the additional type declarations that need to be checked by the theorem prover. However, it is worth investigating if the results could be better, if one chooses the typed version of TPTP. [5].

7 End-User Interface

An important component of the GF distribution is the front-end user interface. While the grammarians are supposed to use the GF shell plus some development environment

[4] http://sigmakee.cvs.sourceforge.net/viewvc/sigmakee/KBs/tests/

[5] http://www.cs.miami.edu/$~$tptp/TPTP/Proposals/TypedFOF.html

for writing grammars, end users should have the option to use some more comfortable interface. GF comes with a generic web-based interface [26] which could be specialized further for particular applications. In relation with SUMO, the interface was extended with features which make the relation of the ontology with the concrete syntax more transparent.

While in Sigma the knowledge engineers are supposed to write the axioms in KIF, in GF they could do it in a controlled natural language. The problem with all controlled languages is that the user has to learn how to write content which is in the scope of the grammar. The successful use of predictive editors in helping the users build constructions within the bounds of the controlled language was investigated in [27]. In GF there is a similar predictive editor (fig. 1) which guides the authoring by generation of suggestions. In the example shown the user has just started adding a reference to a variable and the editor suggests the list of all variables in the current scope which start with "NU". The same kinds of suggestions are offered for every word in the sentence. Furthermore, the user could at any time select a phrase (the highlighted phrase on the picture) and

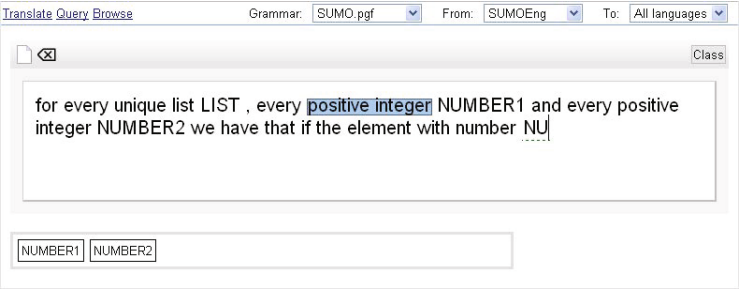


Fig. 1. Text editor for authoring SUMO axioms using controlled natural language

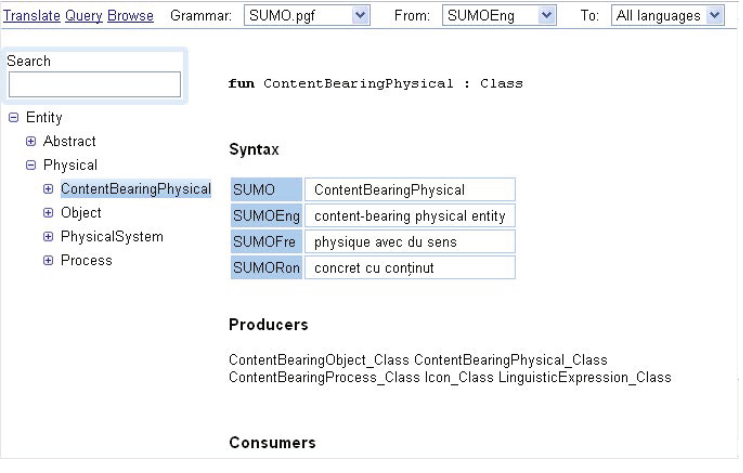


Fig. 2. Browser for combined ontology and syntax exploration

see in the upper-right corner the corresponding ontological type of the phrase (*Class* in this case). If the axiom is not well-formed, i.e. contains a type error for example, then the error is immediately reported and the corresponding phrase is underlined. This very much resembles an IDE for programming languages, except for the fact that the input is a kind of natural language.

For nontrivial ontologies of the scale of SUMO it is often helpful to have an overall view of the ontology. The browsing functionalities of Sigma very much fulfil the requirements. A similar browser (fig. 2) for the abstract syntax of the grammars was developed for GF. In the case of SUMO-GF, this corresponds exactly to the taxonomy plus the signatures of all functions and predicates. The user sees the class hierarchy on the left-hand side and can start with the exploration of any class, or could use the search box in the upper-left corner to find a class or function by name.

8 Related Work

At the moment there exists a large number of applications dealing with ontologies and building various applications on top of them. Regarding the languages that are used to encode ontologies, as mentioned before, the most popular ones do not have a type system.

A first exception is the programming language prototype Zhi#[6], which is a novel language for encoding ontologies. It has a static type-system and it is compiled to C#. It benefits from using the C# built-in types and functions, but also the syntax looks very much like C# and it is not very intuitive for most users.

A more notable example is CASL (Common Algebraic Specification Language) [28]. It introduces the notions of strongly-typed and structured ontologies and provides a strong formal structure for representing them. However, it deals only with the algebraic side of the specifications, whereas GF has a built-in natural language generation component, in addition to the robust type system.

Compared to these languages, GF is the only system which combines a strongly typed framework for ontology description with a direct multilingual verbalization. To our knowledge, the current work is the first representation of an ontology in type theory with dependent types. The benefits of dependent types are visible when expressing the concepts and relations from SUMO in GF, as they provide better control on their semantics. robustness to the representation.

Regarding natural language generation, there are many notable applications that verbalize ontologies. Most of them however, have only English as target. A notable exception is the KPML project [29], which provides natural language generation for 10 languages. Another interesting case is the Gellish ontology which provides direct verbalization for English, German and Dutch. However, there is considerably less progress for Slavic and Romance languages, due to their complexity. The GF approach has built-in mechanisms for verbalization via the resource grammars, which provide syntactically correct translations. Moreover, GF also has support for multilingual translation.

Regarding automatic reasoning, there has been work for checking the consistency of all the well-known ontologies. A notable example is the use of the E theorem prover for

[6] http://www.alexpaar.de/zhimantic/ZhiSharp.pdf

the ResearchCyC ontology [25]. However, SUMO is the most well-known case of an ontology which is checked for consistency every year, as part of a CADE competition. Compared to the official SUMO translation to TPTP, our approach has a comparable expressivity, rejecting the ill-typed axioms at an earlier stage.

The project OntoNat [30] provides automated reasoning for the SUMO ontology with KRHyper [31]. KRHyper is a theorem prover for first-order logic that implements hyper tableaux, and a version of it that deals with equality is also available [7]. The tool can answer questions posed in normal English, by using the wordnet mappings and a simple parser, in order to infer the SUMO expression that should be checked.

9 Future Work

The current work explores aspects of data modelling, compiling from an untyped system to a typed one and from a typed system to first-order logic, type inference, natural language generation, and automated reasoning. These directions can be extended in a more comprehensive manner and lead to stand-alone projects.

One interesting possibility would be to generate higher-quality natural language, following the idea[8] of truncating the hierarchy even more, separating `attributes` and `processes`. Instances of `attribute` and its subclasses can be linearized as adjective phrases, while instances and subclasses of `process` are to be linearized as verb phrases. In this way a predicate like:

$$(attribute\ ?X\ \ NonFullyFormed)$$

would not be linearized as *non fully formed is an attribute of X* but as *X is not fully formed*. For a predicate like:

$$(agent\ \ Reasoning\ ?A)$$

we would obtain *A reasons* instead of *A is an agent of reasoning*.

Another interesting application would be to build a user interface, where users could ask questions and get answers from the theorem prover via the GF to TPTP translation. If the prover provides a trace of the proof search, this could be converted back to a GF tree and used for generation of proof explanations in natural language. When dealing with more complex proofs, more work is needed for rendering readable natural language. A comprehensive reference for natural language generation from proofs is [32].

10 Conclusion

The work investigates the representation of upper ontologies in the type-theoretical functional language GF, which provides mechanisms for direct verbalization as a controlled language, having the SUMO ontology as a use case. The results obtained show

[7] `http://www.uni-koblenz.de/$~$bpelzer/ekrhyper/`
[8] `http://www.ontologyportal.org/student.html`

a consistent improvement from the multilingual natural language generation point of view, in terms of effort, scalability and syntactical correctness of the obtained text. Moreover, the type system, while still preserving a comparable coverage, prevents type errors that could lead to inconsistencies in the ontology. Also the editor makes it easier for users to interact with the ontology by adding content in natural language. From a knowledge engineering point of view, GF offers obvious advantages for encoding ontologies, as the framework defined and applied for SUMO is general enough to fit a large range of ontologies.

References

1. Ganesereth, M.R., Fikes, R.E.: Knowledge interchange format. Technical Report Logic-92-1, Stanford University (June 1992)
2. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL web ontology language reference (2009)
3. Cycorp: The syntax of CycL (2002)
4. Van Renssen, A.: Gellish: A Generic Extensible Ontological Language. PhD thesis, Delft University, PhD thesis (2005)
5. Niles, I., Pease, A.: Towards a standard upper ontology. In: FOIS 2001: Proceedings of the International Conference on Formal Ontology in Information Systems, pp. 2–9. ACM, New York (2001)
6. Ranta, A.: Grammatical Framework: A Type-Theoretical Grammar Formalism. The Journal of Functional Programming 14(2), 145–189 (2004)
7. Martin-Löf, P.: Constructive mathematics and computer programming. In: Cohen, Los, Pfeiffer, Podewski (eds.) Logic, Methodology and Philosophy of Science VI, pp. 153–175. North-Holland, Amsterdam (1982)
8. Ranta, A.: The GF resource grammar library. Linguistic Issues in Language Technology 2(2) (2009)
9. Angelov, K., Bringert, B., Ranta, A.: PGF: A Portable Run-Time Format for Type-Theoretical Grammars. Journal of Logic, Language and Information (2009)
10. Ljunglöf, P., Amores, G., Cooper, R., Hjelm, D., Lemon, O., Manchin, P., Perez, G., Ranta, A.: Multimodal Grammar Library, TALK. Talk and Look: Tools for Ambient Linguistic Knowledge. IST-507802. Deliverable 1.2b (2006)
11. Caprotti, O.: WebALT! Deliver Mathematics Everywhere. In: Proceedings of SITE 2006, Orlando, March 20-24 (2006)
12. Ahrendt, W., Baar, T., Beckert, B., Bubel, R., Giese, M., Hähnle, R., Menzel, W., Mostowski, W., Roth, A., Schlager, S., Schmitt, P.H.: The key tool. Technical report in computing science no. 2003-5, Department of Computing Science, Chalmers University and Göteborg University, Göteborg, Sweden (2003)
13. MOLTO - Multilingual Online Translation. European Project (2010-2012)
14. Pease, A., Sutcliffe, G.: First order reasoning on a large ontology. In: Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning on Large Theories, ESARLT (2007)
15. Sutcliffe, G.: The TPTP problem library and associated infrastructure. Journal of Automated Reasoning 43, 337–362 (2009) 10.1007/s10817-009-9143-8
16. Pease, A.: The sigma ontology development environment. Working Notes of the IJCAI 2003 Workshop on Ontology and Distributed Systems, vol. 71 (2003)

17. Trac, S., Sutcliffe, G., Pease, A.: Integration of the tptpworld into sigmakee. In: Proceedings of IJCAR 2008 Workshop on Practical Aspects of Automated Reasoning (PAAR 2008), vol. 373 (2009)
18. Martin-Löf, P.: A theory of types (1971) (unpublished)
19. Girard, J.Y.: Interpretation fonctionnelle et elimination des coupures de l'arithmetique d'ordre superieur, Paris (1972)
20. W3C: Named graphs (2004)
21. Russell, B.: Principles of Mathematics. Cambridge University Press, Cambridge (2011)
22. Ranta, A.: Structures grammaticales dans le français mathématique. Mathématiques, informatique et Sciences Humaines 138, 139, 5–56, 5–36 (1997)
23. Slagle, J.R.: Automatic theorem proving with built-in theories including equality, partial ordering, and sets. J. ACM 19, 120–135 (1972)
24. Schulz, S.: E - a brainiac theorem prover (2002)
25. Ramachandran, D., Reagan, P., Goolsbey, K.: First-orderized researchcyc: Expressivity and efficiency in a common-sense ontology. Papers from the AAAI Workshop on Contexts and Ontologies: Theory, Practice and Applications (2005)
26. Bringert, B., Angelov, K., Ranta, A.: Grammatical framework web service. In: EACL (Demos), 9–12 (2009)
27. Schwitter, R., Ljungberg, A., Hood, D.: ECOLE — a look-ahead editor for a controlled language. In: Proceedings of EAMT-CLAW 2003, pp. 141–150 (2003)
28. Lüttich, K.: Development of Structured Ontologies in CASL. PhD thesis, University of Bremen, PhD thesis (2007)
29. Bateman, J.A.: Enabling technology for multilingual natural language generation: the kpml development environment. Nat. Lang. Eng. 3(1), 15–55 (1997)
30. Baumgartner, P., Suchanek, F.M.: Automated reasoning support for sumo/kif (2005); (manuscript, Max-Planck Institute for Computer Science)
31. Wernhard, C.: System Description: KRHyper. Fachberichte Informatik 14-2003 (2003)
32. Fiedler, A.: Natural Language Proof Explanation. In: Hutter, D., Stephan, W. (eds.) Mechanizing Mathematical Reasoning. LNCS (LNAI), vol. 2605, pp. 342–363. Springer, Heidelberg (2005)

Controlling Ambiguities in Legislative Language

Alexandra Bünzli and Stefan Höfler

Institute of Computational Linguistics, University of Zurich
{buenzli,hoefler}@cl.uzh.ch

Abstract. Legislative language exhibits some characteristics typical of languages of administration that are particularly prone to eliciting ambiguities. However, ambiguity is generally undesirable in legislative texts and can pose problems for the interpretation and application of codified law. In this paper, we demonstrate how methods of controlled natural languages can be applied to prevent ambiguities in legislative texts. We investigate what types of ambiguities are frequent in legislative language and therefore important to control, and we examine which ambiguities are already controlled by existing drafting guidelines. For those not covered by the guidelines, we propose additional control mechanisms. Wherever possible, the devised mechanisms reflect existing conventions and frequency distributions and exploit domain-specific means to make ambiguities explicit.

1 Introduction

In many respects, legislative drafting constitutes an obvious field of application for controlled natural languages. Legislative texts are produced in a well-defined work process and go through several editorial cycles. Government agencies have devised drafting guidelines meant to ensure the quality of the texts, especially with regard to clarity and readability. To a certain extent, legislative language is therefore already controlled. In this paper, we focus on Swiss legislative texts written in German. Although most countries and legal systems face similar problems when drafting legislation, the conventions and work processes differ substantially from country to country. As a country with a long tradition in legislative drafting, Switzerland has established a well-defined work cycle in which every federal legislative text is edited by a specialized institution: the Federal Chancellery's Central Language Services [15]. This process aims at ensuring consistency between the different language versions of a law, as each of the language versions is considered equally authentic and legally binding. This has led to relatively detailed instructions and well-written legislative texts [11].

However, existing drafting guidelines [26,25,6,5,21,8] offer little or no advice on how ambiguity must be controlled – beyond stating that, whenever possible, it is to be avoided altogether. But due to some of its peculiarities, legislative language is particularly prone to exhibit certain types of ambiguity. Like most languages of administration, legislative language uses a style of writing using complex nominals, which elicits a range of ambiguous constructions. Long and

M. Rosner and N.E. Fuchs (Eds.): CNL 2010, LNAI 7175, pp. 21–42, 2012.

complex noun phrases that are complemented with attributive adjectives or participles, genitive attributes or prepositional phrases generate a high number of attachment ambiguities. The frequent use of light verb constructions, which normally combine a verb with a prepositional phrase, further increases this number. Moreover, the use of plurals instead of gender-specific singular forms, which is also typical of contemporary legislative writing, is a prolific source of semantic ambiguities.

Ambiguous constructions can pose a problem for the interpretation and application of a legislative text. In the light of this fact, it is particularly unfortunate that ambiguities are not always easy to detect for human editors. Since humans unconsciously activate contextual knowledge when reading a text, they easily overlook ambiguities and secondary interpretations that could become relevant in a specific case. But ambiguities that go unnoticed in the drafting process can later cause uncertainties with regard to the correct application of a statute or regulation.[1]

In this paper, we show that the methods of controlled natural language can provide means to tackle this problem. We first present the specific approach we have developed for this particular task (section 2). We then give a brief overview of the types of ambiguities that are already controlled by existing drafting guidelines and of those that are frequent in legislative language but still lack control (section 3). We then illustrate how these latter types of ambiguity – specifically attachment and plural ambiguity – can be controlled by applying the proposed methods (section 4). We conclude with a brief summary of our approach (section 5).

2 Approach

Controlled natural languages are artificially designed subsets of natural languages. Occasionally, a distinction is made between human-oriented controlled natural languages and computer-oriented controlled natural languages [18,30,20,22]. Human-oriented controlled natural languages aim to improve the readability and translatability of technical documents [22,35,2,14]; computer-oriented controlled natural language are meant to serve as interfaces to some sort of formal logical representation [9,28,7,33].

One main task of both classes of controlled natural languages is the reduction of natural language ambiguity. While human-oriented controlled natural languages usually focus on lexical ambiguities, simple sentence constructions and pragmatic issues, computer-oriented controlled natural languages aim at completely unambiguous constructions, which can be automatically translated into a formal representation. They approach this task by either allowing only specific constructions (e.g. prohibiting the use of ambiguous constructions) or by

[1] For an example of how attachment ambiguity can cause a legal dispute see the Appellate Court of Illinois, Regency Commercial Assocs., LLC v. Lopax, Inc., 2007 Ill. App. LEXIS 476 (Ill. App. Ct. May 4, 2007).

assigning them a default interpretation and providing paraphrases for the other readings.

We have adopted these methods to reduce ambiguity in legislative texts. While some research has looked into the applicability of computer-oriented controlled natural languages to the legal domain [19,10], the work presented in this paper focuses on a human-oriented perspective. We therefore do not assign a formal representation. Our aim is rather to systematically refine existing drafting guidelines by developing specific additional rules that deal with as yet uncontrolled types of ambiguity. Our approach builds on [12] and [3], who propose to control ambiguities by exposing the author or editor of a text to explicit paraphrases for the individual readings of an ambiguous construction.

We propose a three-step procedure:

1. *Conventional wording* (**C**)
   In a first step, drafters phrase the text they want to write within the boundaries defined by a set of construction rules. Such rules define which constructions are allowed and which are prohibited. They may also prescribe that a specific ambiguous constructions is only to be used in one particular sense – thus taking the role of what is usually called an interpretation rule in the context of controlled natural language.

   The requirement of usability demands that the rules we develop closely resemble conventional legislative language: drafters will only be able and willing to apply a controlled version of legislative language if it is not significantly different from what they have been used to. Its rules thus have to be designed to mimic (i) the pragmatics of the text domain, (ii) historically accrued conventions, iii) frequency distributions present in legislative texts, as well as (iv) standards defined in existing drafting guidelines.

   The problem with the requirement of conventionality is that users not familiar with the rules and standards applied will still find many constructions ambiguous – and likewise, drafters may still overlook such instances of ambiguity or use ambiguous constructions in a sense other than the one suggested by the rules.
2. *Explicit paraphrase* (**E**)
   To tackle this problem, we define paraphrases that can be constructed deterministically (either by hand or automatically) from the ambiguous constructions of the conventional form. These paraphrases make the interpretation the constructions would obtain according to the rules explicit. Drafters can use this explicit form of the same text to verify if the interpretation they intended complies with the usage suggested by the standard. They are thus pointed to ambiguities they may have overlooked otherwise.

   As they serve the purpose of visualization only, the explicit paraphrases do not have to resemble conventional legislative language (or even sound natural). Where it is not possible to make a specific reading explicit by using the means of natural language only, they resort to non-linguistic means (e.g. brackets) to achieve the task.

3. *Recommended wording* (**R**)
   The recommended version proposes a wording which is as explicit as possible while still sounding natural. Like the explicit version, it can be constructed deterministically from the conventional form. As the name suggests, it is a recommendation and represents the wording the drafter should usually choose. Depending on the constructions used, the recommended version may be identical to the original conventional wording or to the explicit paraphrase.

   However, we let the drafter decide which version he adopts for the final document: the recommended, the conventional or even the explicit version if it does not use any non-linguistic means. Often, ambiguous constructions are automatically disambiguated through contextual and world knowledge and it is not necessary to use the most explicit version: as they choose from the different versions, drafters can adapt to the specific situation, taking into consideration the clearness and readability of the sentence at hand and the text as a whole.

The following example will serve as an illustration of our approach:

(1) **C**: Die Kantone können Fachhochschulen einrichten. Sie werden selbstständig geleitet.
'The cantons may establish technical universities. They are governed autonomously.'

E: [Die Kantone]$_1$ können Fachhochschulen einrichten. [Sie]$_1$ werden selbstständig geleitet.
'[The cantons]$_1$ may establish technical universities. [They]$_1$ are governed autonomously.'

R: Die Kantone können Fachhochschulen einrichten. Die Kantone werden selbstständig geleitet.
'The cantons can establish technical universities. The cantons are governed autonomously.'

We have adopted this example from a drafting guideline of the canton of Zurich [21]. The drafting guideline states that sentences can only be introduced with a pronoun if the pronoun refers to the subject of the immediately preceding sentence. We have incorporated this as a drafting rule. The explicit version (**E**) makes the interpretation explicit that the drafted conventional version **C** would obtain if it complied to the drafting rule: the pronoun *sie* ('they') would refer to *die Kantone* ('the cantons'). It thus allows the drafter to verify if this is the intended interpretation. The recommended version (**R**) does not use a pronoun. This is due to the fact that this specific rule does not prevent ambiguity *per se*: although the drafting rule theoretically prevents ambiguous readings, for a reader not familiar with the rule, the sentence remains ambiguous in situations where, as it is the case in example (2), another suitable referent exists. In such cases, the drafter should either choose the recommended version or – realizing this interpretation is not what he intended to say – rephrase the passage and make *Fachhochschulen* ('technical universities') the subject of the second sentence.

In example (1) the explicit version used non-linguistic means such as indexes and brackets to visualize the interpretation. Example (2)[2] shows a case where the interpretation can be made explicit without such non-linguistic means:

(2) **C**: Die Parteivertreter und -vertreterinnen haben sich durch eine Vollmacht auszuweisen.
'The party representatives have to identify themselves with a letter of attorney.'
E: Jeder Parteivertreter und jede Parteivertreterin hat sich durch eine Vollmacht auszuweisen.
'Each party representative has to identify himself with a letter of attorney.'
R: Die Parteivertreter und -vertreterinnen haben sich durch eine Vollmacht auszuweisen.
'The party representatives have to identify themselves with a letter of attorney.'

Before we further explicate this approach, we shortly discuss in what way legislative language is already controlled through drafting guidelines and which constructions are not controlled yet although they are frequent in legislative language and potentially pose problems. In section 4 of the paper, we will then discuss a number of real-case examples, the issues that arise in them and the design decisions one has to make.

3 Status Quo in Legislative Drafting

In natural language, ambiguities exist on various levels: there is lexical, syntactic and semantic ambiguity as well as ambiguity on the pragmatic and discourse level. Governments have designed drafting guidelines that aim at ensuring the comprehensibility of legislative texts and thus, among other things, try to control some of these ambiguities.

These guidelines are – as the name indicates – mere recommendations and have no absolute force. The control is therefore on a quite abstract level and there is always a chance that an ambiguous structure could slip through the editing process and be then constituted in the law.[3] There is no systematic or technical process to ensure compliance with the drafting guidelines.

Not every type of ambiguity gets the same amount of attention in the existing drafting guidelines for legislative texts. In the remainder of this section, we will thus give a brief overview of (i) the types of ambiguity that existing Swiss drafting guidelines [26,25,6,21] deal with and (ii) those that are particularly frequent in legislative texts but as yet lack control.

[2] Example from Art. 40 Abs. 2 BGG.

[3] It is not to be forgotten that legislative texts are written for humans and a lot of ambiguities can be resolved through external knowledge such as situational knowledge or world knowledge.

3.1 Ambiguities Controlled by Existing Drafting Guidelines

Lexical ambiguity is probably the type of ambiguity that legal experts are most aware of. As in every technical language, the use of exact terminology is essential; it is therefore not surprising that lexical ambiguity is the most controlled type of ambiguity in the legal domain. Vast terminological databases exist, which are especially important for interlingual and international translations and communications. The same technical terms are often used quite differently by the different governments, which makes a semantically correct translation very hard and terminological control paramount. There is no controlled vocabulary in the strict sense – the domain is too broad – but the use of specific words is regulated, and existing drafting guidelines demand that terms have to be used consistently and that, in order to avoid confusion, new or unclear terms have to be defined in the regulation itself.

Syntactic ambiguity is not controlled explicitly. There is no real awareness of the peculiarities of syntactic ambiguity among legal experts. However, certain drafting guidelines indirectly control syntactic ambiguity: they state e.g. that sentences should be short and concise and default sentence patterns should be preferred. The last point is especially important as German has a relatively free word order and agreement information is not always sufficient to unambiguously identify the correct structure, as the following example from the drafting guidelines of the canton of Zurich [21] demonstrates:

(3) Die Bewilligung erteilt das Amt.
Interpretation 1: 'The permission grants the department.'
Interpretation 2: 'The department grants the permission.'

In example (3), it is not clear which noun phrase constitutes the subject and which the object. Syntactically, both interpretations are valid; semantically, the second interpretation is clearly to be preferred. The sentence would be a lot easier to understand if it followed the default sentence pattern *subject* ▷ *verb* ▷ *objects* (*Das Amt erteilt die Bewilligung*) and thus actively support the correct interpretation.

Another rule in existing guidelines demands that nominalizations should be avoided. This rule is mainly designed to improve readability, but it also prevents thematic ambiguities, which can arise from nominalizations of transitive verbs that are accompanied by a genitive attribute. A classical example is *die Untersuchung der Behörde* ('the inspection of the agency'), which does not specify if the agency inspects or if it is itself the object of the inspection.

Other drafting guidelines suggest that complex sentences are formalised as enumerations, a tool which is very frequently used in Swiss legislative language. As complex sentences are more likely to contain attachment ambiguities, this rule can also contribute to a reduction of ambiguity in a text.

Like syntactic ambiguity, *semantic ambiguity* is not controlled explicitly. But there are some isolated rules which deal with semantic problems such as (i) pronoun resolution and (ii) conjunctive vs. disjunctive enumerations. An existing rule to control pronoun resolution has already been discussed in section 2

above. The problem posed by ambiguities relating to conjunctive vs. disjunctive readings of enumerations can be illustrated with the following example from a regulations on weapons and ammunition:[4]

(4) Messer gelten als Waffen, wenn sie:

a. einen einhändig bedienbaren Spring- oder anderen automatischen Auslösemechanismus aufweisen;
b. geöffnet insgesamt mehr als 12 cm lang sind; und
c. eine Klinge haben, die mehr als 5 cm lang ist.

'Knives count as weapons if they

a. are equipped with a switchblade mechanism or any other automatic trigger that can be operated with one hand;
b. are at least 12 cm long when opened; and
c. have a blade that is at least 5 cm long.'

Enumerations can be interpreted as conjunctions or as disjunctions. Legal experts are very much aware of this kind of ambiguity. Existing drafting rules state that, if the intended reading is not clear from the context, the conjunctive or disjunctive nature of an enumeration has to be made explicit: by inserting *und* ('and') for a conjunctive reading or *oder* ('or') for an disjunctive reading after the second last item. In example (4), *und* is thus required, as the enumeration could otherwise be misinterpreted as a disjunction and consequently be applied to more types of knives than actually intended.

3.2 Ambiguities Not Controlled by Existing Guidelines

While existing guidelines do control ambiguities to a certain extent, there are at least two types of ambiguity that are not covered even though they are prevalent in legislative texts and can give rise to severe misinterpretations: attachment ambiguity and plural ambiguity.

Attachment ambiguities frequently arise when noun phrases in object positions are followed by prepositional phrases, or when complex noun phrase coordinations are accompanied by an attribute (a preceding adjective or participle, a postpositional prepositional phrase or a relative clause). In the latter case, it is often unclear if the attribute modifies the whole coordinated structure or only the nearest element [16]. Example (5) shows how such an attachment ambiguity can lead to a legal dispute.

(5) Seller will not after the date of this agreement sell, lease or permit to be occupied any real estate which Seller owns, manages or otherwise controls within one mile of the Land for the purpose of constructing, or having conducted thereon, any fast food [...] restaurant or restaurant facility whose principal food product is chicken on the bone, boneless chicken or chicken sandwiches.

[4] Art. 7 Abs. 1, SR 514.541, Verordnung über Waffen, Waffenzubehör und Munition, Stand am 12. Dezember 2008.

The above passage of a contract was the cause for a dispute which the Illinois appellate court had to decide.[5] The problem was caused by the uncertain attachment of *fast food*.[6] Does it only modify *restaurant* or *restaurant facility* as well? Is a restaurant that is not a fast food restaurant, but serves primarily chicken, permitted? It would be if *fast food* modifies both *restaurant* or *restaurant facility*, but it would not, if it only modifies *restaurant*. This ambiguity resulted in a two year dispute and illustrates how important the controlling of attachment ambiguities in legislative texts can be.

The frequency of *plural ambiguities* has especially increased with the introduction of gender neutral formulations. To facilitate grammatical agreement with other constituents of a sentence, existing drafting guidelines suggest that gender-neutral plural forms are used instead of coordinations of gender-marked singular forms. The following (slightly adapted) example from the drafting guidelines of the canton of Zurich [21] illustrates this point:

(6) Der Lehrer oder die Lehrerin sorgt für die zweckdienliche Einrichtung seines oder ihres Schulzimmers. Er oder sie wird angehört, bevor die Schulpflege bauliche Massnahmen beschliesst.
'The teacher takes care of the appropriate equipment of his or her class room. He or she will be heard before the School Board decides on constructional measures.'

If the subject of the first sentence is transformed into a plural, the anaphoric reference at the beginning of the second sentence becomes less cumbersome:

(7) Die Lehrerinnen und Lehrer sorgen für die zweckdienliche Einrichtung ihrer Schulzimmer. Sie werden angehört, bevor die Schulpflege bauliche Massnahmen beschliesst.
'The teachers take care of the appropriate equipment of their class rooms. They will be heard before the School Board decides on constructional measures.'

The problem is that by transforming the subject into plural, a new instance of ambiguity is created, which yields two additional interpretations: it is now unclear if, before any construction measure is taken, all the teachers have to be heard as a group, or if every teacher has to be heard individually, or if only the teacher of the class room concerned has to be heard.

Due to their relative frequency and their potential to cause misinterpretations, attachment ambiguity and plural ambiguity clearly pose a problem for legislative texts.

[5] Case: Appellate Court of Illinois, Regency Commercial Assocs., LLC v. Lopax, Inc., 2007 Ill. App. LEXIS 476 (Ill. App. Ct. May 4, 2007), retrieved from a legal blog: http://www.adamsdrafting.com/2007/05/15/illinois-syntactic-ambiguity/ at March 10, 2011.

[6] There is another attachment ambiguity in the sentence: the relative sentence at the end of the passage could modify only *restaurant facility* or both *restaurant* and *restaurant facility*, but this was not questioned, neither by the involved parties nor by the court.

4 Proposed Rules

In the following sections, we will demonstrate how the methods introduced in section 2 can be applied to control attachment ambiguity and plural ambiguity.

4.1 Controlling Attachment Ambiguity

As already mentioned above, one of the most common causes for attachment ambiguity are prepositional phrases modifying a verb or a noun phrase coordination. A study on Italian and English legislative texts carried out by Venturi [34] shows that prepositional phrases are particularly frequent in legislative texts. Similarly, Nussbaumer [17] points out that Swiss legal language is characterized (among other things) by a relatively high frequency of prepositional phrases. He speculates that this characteristic emerges from a conflict between the need to be brief and the need to provide all important information. Therefore, the attachment of prepositional phrases deserve special attention in the drafting process.

We will explain our approach to controlling the attachment ambiguity they can cause with the following example from the Federal Supreme Court Act:[7]

(8) Das Bundesgericht deckt seinen Bedarf an Gütern und Dienstleistungen im Bereich der Logistik selbständig.
'The Federal Supreme Court covers its need for goods and services in the sector of logistics autonomously.'

The example shows a short sentence with two prepositional phrases. While the prepositional phrase *an Gütern und Dienstleistungen* depends on *Bedarf* and thus has to follow right after it, the attachment of the prepositional phrase *im Bereich der Logistik* constitutes a typical case of attachment ambiguity [10]. There are four possible antecedents to which it could theoretically attach: (a) *deckt*, (b) *Güter und Dienstleistungen*, (c) *Dienstleistungen* and (d) *Bedarf*. The respective four readings of the sentence are shown below, with the prepositional phrase put in italics and the antecedent underlined:

(9) a. Das Bundesgericht deckt seinen Bedarf an Gütern und Dienstleistungen *im Bereich der Logistik* selbständig.
b. Das Bundesgericht deckt seinen Bedarf an Güter und Dienstleistungen *im Bereich der Logistik* selbständig.
c. Das Bundesgericht deckt seinen Bedarf an Gütern und Dienstleistungen *im Bereich der Logistik* selbständig.
d. Das Bundesgericht deckt seinen Bedarf an Gütern und Dienstleistungen *im Bereich der Logistik* selbständig.

To get an idea of how these ambiguities are best controlled, we investigated (i) under what conditions the attachment of the prepositional phrase would *not* be ambiguous and (ii) how prepositional phrases that modify coordinated structures (or components thereof) are prevalently used in legislative texts.

[7] Art. 25a Abs. 2, Bundesgerichtsgesetz (BGG) of June 17, 2005 (status January 1, 2011).

With regard to (i), we found that the attachment of the prepositional phrase is highly ambiguous as long as the prepositional phrase follows a noun phrase; if the prepositional phrase directly follows the verb, only one attachment is possible. In order to study (ii), we syntactically annotated a small corpus consisting of two texts: the Federal Supreme Court Act (*Bundesgerichtsgesetz*, BGG) and the Ordinance of the University of Zurich.[8] The corpus consists of a total of 1,124 sentences (Federal Supreme Court Act: 776 sentences, Ordinance of the University of Zurich: 348 sentences). We specifically looked at coordinated noun phrases which are directly followed by a prepositional phrase. That is, we looked at constructions in which the prepositional phrase was either attached to a coordinated phrase or attached to the last element of a coordination. We found 52 attachments to the coordinated phrase and 39 attachments to the last element of the coordination. Of the 39 attachments to the last element of a coordination, 9 were formulated according to our Rule 2 (see below); another 9 were formulated according to Rule 3; in 6 cases, the noun selected the preposition (valency information); in another 7 cases, semantics favored this attachment clearly; in 6 cases, the attachment was undecidable; and in the remaining 2 cases, some special construction prohibited the attachment to the whole coordinated phrase. On the basis of these findings, we have constructed new drafting rules (Rules 1 to 5 below).

Our first rule is a general instruction on where to attach constituents:

Rule 1. Attach to the nearest constituent
A prepositional phrase should only be attached to the nearest possible antecedent. If that antecedent is a coordinated phrase, the prepositional phrase should only be used if it refers to the whole coordinated phrase rather than to its last conjunct.

Had sentence (8) been constructed according to this rule, reading (9-b) would be its correct interpretation. Example (10) illustrates the explicit paraphrase and the recommended wording that we defined for the rule.

(10) **C**: Das Bundesgericht deckt seinen Bedarf an Gütern und Dienstleistungen im Bereich der Logistik selbständig.
'The Federal Supreme Court supplies its need for goods and services in the sector of logistics autonomously.'
E: Das Bundesgericht deckt seinen Bedarf an [[Gütern und Dienstleistungen] im Bereich der Logistik] selbständig.
'The Federal Supreme Court supplies its need for [[goods and services] in the sector of logistics] autonomously.'
R: Das Bundesgericht deckt seinen Bedarf an Gütern und Dienstleistungen im Bereich der Logistik selbständig.
'The Federal Supreme Court supplies its need for goods and services in the sector of logistics autonomously.'

[8] State: May 2010.

The explicit paraphrase uses brackets to visualize the attachment of the prepositional phrase. This non-linguistic means appears to be the simplest and most comprehensible way to achieve the task. Alternatively, one could define an explicit version as shown in (11), where no non-linguistic means are used:

(11) **E**: Das Bundesgericht deckt seinen Bedarf an Gütern im Bereich der Logistik und Dienstleistungen im Bereich der Logistik selbständig.
'The Federal Supreme Court supplies its need for goods in the sector of logistics and services in the sector of logistics autonomously.'

This alternative paraphrase could actually be used in legislative texts, if one wanted to be as clear as possible. However, due to the repetitions, it sounds cumbersome and is not recommended for use. Since there is no straightforward way of making the conventional reading more explicit, the recommended wording shown in (10) is identical to the original text.

To express the other three interpretations, the sentence has to be rephrased. Reading (9-a), for instance, can be obtained by placing the prepositional phrase right after the verb:

(12) **C**: Das Bundesgericht deckt im Bereich der Logistik seinen Bedarf an Gütern und Dienstleistungen selbständig.
'The Federal Supreme Court covers, in the sector of logistics, its need for goods and services autonomously.'

E: Das Bundesgericht [deckt im Bereich der Logistik] seinen Bedarf an Gütern und Dienstleistungen selbständig.
'The Federal Supreme Court [covers, in the sector of logistics,] its need for goods and services autonomously.'

R: Das Bundesgericht deckt im Bereich der Logistik seinen Bedarf an Gütern und Dienstleistungen selbständig.
'The Federal Supreme Court covers, in the sector of logistics, its need for goods and services autonomously.'

Here, the recommended wording is the same as the conventional version because the conventional wording is both natural and unambiguous. The explicit paraphrase would thus actually not be needed. We provide a version with brackets anyway to visualize the attachment for verification purposes and to raise awareness of potential ambiguities in the drafter.

Reading (9-c) can be obtained by switching the order of the two conjuncts and placing the prepositional phrase right after the new first conjunct (*Dienstleistungen*):

(13) **C**: Das Bundesgericht deckt seinen Bedarf an Dienstleistungen im Bereich der Logistik und Gütern selbständig.
'The Federal Supreme Court covers its need for services in the sector of logistics and (for) goods autonomously.'

E: Das Bundesgericht deckt seinen Bedarf an [Dienstleistungen im Bereich der Logistik] und Gütern selbständig.
'The Federal Supreme Court covers its need for [services in the sector of logistics] and (for) goods autonomously.'

R: Das Bundesgericht deckt seinen Bedarf an Dienstleistungen im Bereich der Logistik sowie an Gütern selbständig.
'The Federal Supreme Court covers its need for services in the sector of logistics as well as for goods autonomously.'

As before, the conventional wording is not ambiguous[9]; the explicit version simply serves the purpose of verification. But in contrast to the previous examples, the recommended version differs from the conventional wording: on the one hand, it suggests that the preposition *an* is repeated with the second conjunct; on the other hand, the conjunction *sowie* ('as well as') replaces *und* ('and') to make the structure of the sentence clearer.

The recommended version indicates that reading (9-c) can also be obtained by exploiting another convention that already exists in legislative language: the binding provided by the conjunction *sowie* ('as well as') is weaker than that of the conjunction *und* ('and'). We have constructed an actual drafting rule on the basis of this implicit convention:[10]

Rule 2. Use *sowie* and *oder aber* as barriers
The conjunctions *sowie* and *oder aber* can be used instead of *und* and *oder* respectively to introduce a barrier into a coordinated phrase in order to prevent attachment to the elements on the other side of the barrier.

Example (14) shows how this rule can be applied to express reading (9-c); it is now not necessary anymore to switch the order of the conjuncts:

(14) **C**: Das Bundesgericht deckt seinen Bedarf an Gütern sowie Dienstleistungen im Bereich der Logistik selbständig.
'The Federal Supreme Court covers its need for goods as well as services in the sector of logistics autonomously.'

E: Das Bundesgericht deckt seinen Bedarf an Gütern sowie [Dienstleistungen im Bereich der Logistik] selbständig.
'The Federal Supreme Court covers its need for goods as well as [services in the sector of logistics] autonomously.'

R: Das Bundesgericht deckt seinen Bedarf an Gütern sowie an Dienstleistungen im Bereich der Logistik selbständig.
'The Federal Supreme Court covers its need for goods as well as for services in the sector of logistics autonomously.'

Example (14) expresses the same meaning as (13), but the order in which the conjuncts are arranged make it sound more natural. This is due to the fact that, in natural language, longer elements tend to be placed at the end of a list.

[9] In the German version, *Logistik* and *Gütern* cannot be coordinated due to case agreement. However, the English version is ambiguous: it is unclear if 'goods' is coordinated with 'services' or 'logistics'.

[10] Our drafting rule was also discussed at a meeting of the German section of the Federal Chancellery's Central Language Services [4].

Again, the recommended wording suggests that the preposition *an* is repeated. Like the use of *sowie*, the repetition of shared components is a common technique to introduce a barrier into a coordinated phrase in order to prevent attachment to the elements on the other side of the barrier. We have cast a new drafting rule that reflects this phenomenon:

> **Rule 3. Use repetition as a barrier**
> The explicit repetition of shared components can be used to introduce a barrier into a coordinated phrase in order to prevent attachment to the elements on the other side of the barrier.

This drafting rule thus offers yet another way to express reading (9-c):

(15) **C**: Das Bundesgericht deckt seinen Bedarf an Gütern und seinen Bedarf an Dienstleistungen im Bereich der Logistik selbständig.
'The Federal Supreme Court covers its need for goods and its need for services in the sector of logistics autonomously.'

E: Das Bundesgericht deckt seinen Bedarf an Gütern und seinen Bedarf an [Dienstleistungen im Bereich der Logistik] selbständig.
'The Federal Supreme Court covers its need for goods and its need for [services in the sector of logistics] autonomously.'

R: Das Bundesgericht deckt seinen Bedarf an Gütern sowie seinen Bedarf an Dienstleistungen im Bereich der Logistik selbständig.
'The Federal Supreme Court covers its need for goods as well as its need for services in the sector of logistics autonomously.'

To be as explicit as possible, the recommended wording suggests that the repetition of shared components is combined with the use of *sowie* instead of *und*.

The last reading, (9-d), is not so easy to generate within the proposed rules. There is no position at which the prepositional phrase *im Bereich der Logistik* would unequivocally be attached to *Bedarf*; the only possible one, the one immediately after *Bedarf*, is already occupied by the prepositional phrase *an Gütern und Dienstleistungen*. Thus, a more substantial rephrasing is needed. Reading (9-d) can, for instance, be obtained by transforming the prepositional phrase in question into an attributive structure that precedes the phrase it modifies:

(16) **C**: Das Bundesgericht deckt seinen im Bereich der Logistik vorhandenen Bedarf an Gütern und Dienstleistungen selbständig.
'The Federal Supreme Court covers its logistics-related need for goods and services autonomously.'

E: Das Bundesgericht deckt seinen [im Bereich der Logistik vorhandenen Bedarf] an [Gütern und Dienstleistungen] selbständig.
'The Federal Supreme Court covers its [logistics-related need] for [goods and services] autonomously.'

R: Das Bundesgericht deckt seinen im Bereich der Logistik vorhandenen Bedarf an Gütern und Dienstleistungen selbständig.
'The Federal Supreme Court covers its logistics-related need for goods and services autonomously.'

This rephrasing strategy has led to another drafting rule:

Rule 4. Rephrase multiple attachments I
If two prepositional phrases should be attached to the same noun phrase, one has to be rephrased as an adjectival or participial attribute.

In the discussed example (8), only two prepositional phrases where involved. What to do if more than two occur in the same sentence? Even in the short example we had problems to achieve reading (9-d), where both prepositional phrases had to attach to the same constituent. With more prepositional phrases occurring, the intended attachments often cannot be satisfactorily resolved through basic means such as shifting of the prepositional phrase next to the modified constituent.[11] A more thorough paraphrasing step is needed. Take, for instance, the following sentence from the Swiss Federal Supreme Court Act:[12]

(17) In Fünferbesetzung entscheiden sie ferner über Beschwerden gegen referendumspflichtige kantonale Erlasse und gegen kantonale Entscheide über die Zulässigkeit einer Initiative oder das Erfordernis eines Referendums.

'In a composition of five, they further decide on appeals against cantonal decrees that are subject to referendum and against cantonal decisions on the admissibility of an initiative or the necessity of a referendum.'

The nesting of multiple prepositional phrases in this sentence gives rise to a range of interdependent attachment ambiguities. The main problem is that *entscheiden über*, *entscheiden gegen* and *entscheiden + noun phrase* are all valid constructions and every coordinated item can theoretically be attached to the verb. On the other hand, they can also be attached to *Beschwerden*. The attachment of the last noun phrase (*das Erfordernis eines Referendums*) is particularly unclear: is it coordinated with *Zulässigkeit einer Initiative* and therefore attached to *Entscheide (über)*, or is it coordinated with the long prepositional phrase *über Beschwerden . . . einer Initiative* and thus attached to the verb? Such a sentence is not only hard to parse but also difficult to understand: many combinations of attachment can only be ruled out if appropriate context knowledge is accessible.

[11] Interpretation rule 1 does, for example, not cover constructions where the prepositional phrase itself is a coordinated structure: the conjuncts could either (i) both separately modify the governing constituent or (ii) modify the governing constituent as a whole. Additionally, with multiple nested coordinations, it must be decided which items are mutually coordinated: 'x and (y or z)' vs. '(x and y) or z'.

[12] Art. 20 Abs. 3 Bundesgerichtsgesetz.

Rule 5. Rephrase multiple attachments II
Use an enumeration (1) if there are four or more elements to a coordination or (2) if the elements of the coordination are long, e.g. consist of noun phrases with complex attributes, or (3) if the attachment of prepositional phrases according to rules 1-4 results in a cumbersome wording.

Example (18) illustrates what the sentence looks like if it is rephrased according to the rules we introduced. We do not show the recommended version as it uses the same wording as the conventional version.

(18) **C**: In Fünferbesetzung entscheiden sie ferner über Beschwerden gegen:
a. referendumspflichtige kantonale Erlasse;
b. kantonale Entscheide über die Zulässigkeit einer Initiative;
c. kantonale Entscheide über das Erfordernis eines Referendums.
'In a composition of five, they further decide on appeals against:
a. cantonal decrees that are subject to referendum;
b. cantonal decisions on the admissibility of an initiative;
c. cantonal decisions on the necessity of a referendum.'

E: In Fünferbesetzung entscheiden sie ferner über Beschwerden gegen:
a. referendumspflichtige kantonale Erlasse;
b. kantonale Entscheide [über die Zulässigkeit einer Initiative];
c. kantonale Entscheide [über das Erfordernis eines Referendums].
'In a composition of five, they further decide on appeals against:
a. cantonal decrees that are subject to referendum;
b. cantonal decisions [on the admissibility of an initiative];
c. cantonal decisions [on the necessity of a referendum].'

As the example shows, explicit enumerations remove a large number of attachment ambiguities from sentences with complex, nested coordination structures and thus make them easier to read and understand.

4.2 Controlling Plural Ambiguities

Authors often find it hard to spot plural ambiguities contained in their texts; yet this particular type of semantic ambiguity is not only prevalent in legislative writing [1] but can indeed lead to confusions with regard to the meaning of a passage, as the following example illustrates:[13]

[13] This genuine example is taken from a draft of the Swiss Regulation on Immigration and Visa Granting (Art. 6 Abs. 3 Bst. e VEV).

(19) [Von der Visumpflicht sind ausgenommen:]

e. Inhaberinnen und Inhaber eines [...] Sonderpasses, der von den in Absatz 2 genannten Staaten ausgestellt wurde;

'[Exempt from the visa requirement are:]

e. owners of a special passport that was issued by the states mentioned in paragraph 2;'

Due to plural ambiguity, the correct interpretation of the noun phrase *eines Sonderpasses, der von den [...] genannten Staaten ausgestellt wurde* ('a special passport that was issued by the states mentioned [...]') is uncertain; it can have at least the following meanings:

(20) a. 'a special passport that was issued by one of the states mentioned' (singular reading)

b. 'a special passport that was issued by all of the states mentioned together' (collective plural reading)

c. 'a special passport that was issued by each of the states mentioned individually' (distributive plural reading)

Reading (20-a) does not interpret the noun phrase *die genannten Staaten* ('the states mentioned') as a plural at all: it rather assumes that the plural form of that noun phrase is merely a projection from *Inhaberinnen und Inhaber* ('owners'), in whose scope it appears. Readings (20-b) and (20-c), in contrast, represent the classical distributive and collective plural interpretations, as described e.g. in [27]. In legislative texts, this classical plural ambiguity seems seldom of great consequence: normally, world knowledge prevents a wrong interpretation, or the distinction between a collective and a distributive reading is not relevant in the first place and can thus be left underspecified. The distinction between a singular and a plural interpretation, on the other hand, can prove to be critical for the correct application of a statute or regulation.

Existing drafting guidelines such as [21] consequently suggest that reading (20-a) should be expressed by a singular rather than a plural. We have incorporated this guideline:

Rule 6. Use singular forms for singular objects
A singular form should be used to refer to a singular object, even if it occurs within the scope of a plural object.
Instead of an indefinite plural noun phrase, use an indefinite singular noun phrase; instead of a definite plural noun phrase, use *eine/r/s der* and that plural noun phrase.

The present example would thus have to be rephrased by replacing the plural *die genannten Staaten* ('the states mentioned') with *einer der genannten Staaten* ('any/one of the states mentioned') if reading (20-a) was intended:[14]

(21) **C**: Inhaberinnen und Inhaber eines Sonderpasses, der von einem der in Absatz 2 genannten Staaten ausgestellt wurde
'owners of a special passport that was issued by any of the states mentioned in paragraph 2'

E: jede Inhaberin und jeder Inhaber eines Sonderpasses, der von einem der in Absatz 2 genannten Staaten ausgestellt wurde
'every owner of a special passport that was issued by any of the states mentioned in paragraph 2'

R: Inhaberinnen und Inhaber eines Sonderpasses, der von einem der in Absatz 2 genannten Staaten ausgestellt wurde
'owners of a special passport that was issued by any of the states mentioned in paragraph 2'

Note that the recommended version is identical to the drafted conventional version. The explicit paraphrase only has to make the interpretation of the subject explicit (see below); the noun phrase in question is unambiguous now.[15]

The method controlled natural languages typically choose to control the ambiguity of 'real' plural noun phrases is to use markers such as *je* ('each') and *gemeinsam* ('together') to distinguish between distributive and collective readings [27]. However, as these markers do not modify the noun phrases in question but rather the verb, the derivation of explicit paraphrases and recommended wordings would not be trivial. We have thus chosen to take a different approach by constructing drafting rules that are based on (i) existing frequency distributions and (ii) conventions already present in legislative language.

Our analysis of several legislative texts has shown that distributive interpretations of plural noun phrases clearly occur more often than collective readings. We have thus constructed a drafting rule that reflects this fact:

Rule 7. Use plurals distributively
Plurals should only be used in the distributive sense.

Example (19) illustrates how the usage requested by this rule is made explicit and what wording is recommended for it:

[14] It is indeed the intended reading as showed by the final and published version of the Swiss Regulation on Immigration and Visa Granting (Art. 6 Abs. 3 Bst. e VEV). The unclear passage was corrected by the Central Language Services during their reviewing process.

[15] Theoretically, there still is a scope ambiguity: do the owners of a special passport each have their own passport or is there one specific passport that they all share? In the context of legislative texts, the interpretation where the existential quantified phrase gets wide scope without specifically being marked is very improbable and is a mere theoretical possibility.

(22) **C**: Inhaberinnen und Inhaber eines Sonderpasses, der von den in Absatz 2 genannten Staaten ausgestellt wurde
'owners of a special passport that was issued by the states mentioned in paragraph 2'

E: jede Inhaberin und jeder Inhaber eines Sonderpasses, der von jedem der in Absatz 2 genannten Staaten ausgestellt wurde
'every owner of a special passport that was issued by each of the states mentioned in paragraph 2'

R: Inhaberinnen und Inhaber eines Sonderpasses, der von jedem der in Absatz 2 genannten Staaten ausgestellt wurde
'owners of a special passport that was issued by each of the states mentioned in paragraph 2'

The distributive reading is made explicit by inserting *jeder der* ('each of the') in front of the definite plural noun phrase. This paraphrase sounds more or less natural and can be used in situations where it is essential to be precise and no general world-knowledge prevents unwanted interpretations. It is therefore also used in the recommended version.

Note that the subject of the sentence, *Inhaberinnen und Inhaber* ('owners'), is a plural noun phrase too and could thus also produce plural ambiguity. In the paraphrase, the distributive usage recommended by the drafting rule is made explicit by the insertion of *jeder* ('every') in front of it.[16]

To express the collective reading (20-b), we have adopted a convention that already exists in legislative language: the use of abstract singular terms instead of plurals. Legislative texts frequently coin and employ collective singular nouns such as *das Gericht* ('the court'), *die Erbengemeinschaft* ('the community of heirs') or *die Inhaberschaft* ('the ownership'). We have thus defined the following drafting rule:

Rule 8. Use singular terms for collective readings
To express collective plural readings, abstract singular terms should be used.

Example (23) illustrates how this rule can be applied. The collective plural reading of sample sentence (19) is achieved by the singular term *die Staatengemeinschaft* ('the community of states'):

(23) **C**: Inhaberinnen und Inhaber eines Sonderpasses, der von der in Absatz 2 genannten Staatengemeinschaft ausgestellt wurde
'owners of a special passport that was issued by the community of the states mentioned in paragraph 2'

[16] We have argued elsewhere that, in legislative texts, indefinite noun phrases in vorfeld position can be considered to be implicitly universally quantified [10]. Legislative texts state rules that apply to a certain group or set of objects. In Swiss German-language legislative texts, this subject matter of a norm is usually mentioned in the vorfeld position and must be interpreted in a definitional generic sense.

E: jede Inhaberin und jeder Inhaber eines Sonderpasses, der von der in Absatz 2 genannten Staatengemeinschaft ausgestellt wurde
'every owner of a special passport that was issued by the community of the states mentioned in paragraph 2'

R: Inhaberinnen und Inhaber eines Sonderpasses, der von der in Absatz 2 genannten Staatengemeinschaft ausgestellt wurde
'owners of a special passport that was issued by the community of the states mentioned in paragraph 2'

Because of the use of a singular term, the noun phrase in question is not ambiguous anymore. The explicit paraphrase thus only needs to make the interpretation of the subject of the sentence explicit, and the recommended wording is identical to the original conventional formulation.

5 Related Work

Ambiguity in law has been studied by several researchers, for the Anglo-American legal system for example by Lawrence Solan [31,32] and Sanford Schane [23,24]. Government agencies and organizations such as the Plain English or Plain Language initiative[17] have addressed the problem in a fashion that is broadly similar to the aforementioned guidelines [26,25,6,5,21,8]. However, these drafting manuals typically formulate general rules – for example, that the active voice should be used or sentences should be short. They do not cover rules that require deeper linguistic insight, although this would be desirable: Carl Vogel [36] presents a study of statutory drafting in Ireland in which he argues that it is particularly with regard to issues of natural language ambiguity that linguistics (and formal semanticists) can make a pivotal contribution to legislative drafting.

Regarding controlled languages, there have only been very few attempts to design controlled versions of German: "Siemens Dokumentationsdeutsch" [14] and "Controlled German" [13]. All these attempts were concerned with technical language, and, as far as we know, remained research prototypes. No effort has been made to transfer these attempts to legal language, but some of the rules could also be applied to legal texts.

Every authoring tool for controlled natural languages has to make the actual interpretation explicit and doing this by paraphrasing ambiguous constructions is not a new idea. It has been done in several approaches, for example for ACE [12], PENG [29] and IBM's Easy English [3]. However, the mentioned approaches rely on a complete grammar and generate the paraphrases automatically. In contrast, our rules are guidelines for humans. They are intended to help the drafter choose a sensible phrasing and to make legislative texts more consistent by establishing new interpretation rules.[18] Establishing interpretation rules for specific constructions

[17] `http://www.plainenglish.co.uk`, `http://www.plainlanguage.gov`

[18] Some of our rules discussed in this paper have already had an impact on the drafting process: there has been a thorough discussion about the proper use of "sowie" vs. "und" [4].

is indeed quite common in the legal domain: interpretation principles are accepted as a means to support deciding ambiguous or vague cases [36,24].

6 Conclusion

In this paper, we have shown how the methods of controlled natural language can be applied to reduce ambiguity in legislative texts. We have introduced a three-layer approach, in which we specify drafting rules for specific ambiguous constructions. Sentences that are formulated according to these rules can be transformed deterministically into pre-defined explicit paraphrases and recommended wordings. At the moment, this transformation has to be done by hand. However, the availability of deterministic mechanisms as proposed in this paper constitutes a precondition for a future automatic processing. The proposed drafting rules are designed to reflect conventions and frequency distributions that already exist in legislative language and to exploit guidelines that have already been issued by various government agencies. We have argued that two types of ambiguity are in need of such additional control: attachment ambiguity and plural ambiguity. Not only are these insufficiently covered by existing drafting guidelines but they are also particularly frequent in legislative texts and prone to causing problems for the correct interpretation of a statute or regulation.

A thorough application of drafting rules like the ones proposed in this paper has several benefits: it helps drafters become aware of ambiguities hidden in their texts, it helps them formulate sentences that are easier to understand and finally, it can lead to a greater standardization of legislative language, especially with regard to the phenomenon of ambiguity. Such a standardization can facilitate the interpretation of legislative texts but it can also support their translation into other languages – an aspect which is particularly relevant for multilingual countries like Switzerland or for international organizations like the European Union.

References

1. Adams, K.A., Kaye, A.S.: Revisiting the ambiguity of “And” and “Or” in legal drafting. St. John’s Law Review 80(4) (2006)
2. ASD: ASD Simplified Technical English: Specifications ASD-STE100. AeroSpace and Defence Industries Association of Europe, Simplified Technical English Maintenance Group (ASD STEMG) (2005)
3. Bernth, A.: EasyEnglish: A tool for improving document quality. In: Proceedings of the Fifth Conference on Applied Natural Language Processing, pp. 159–165. Association for Computational Linguistics, Morriston (1997)
4. Bratschi, R.: “und” vs. “sowie”. Redaktionsbeispiel vom 19 (August 2010) (unpublished)
5. Caussignac, G., Eberhard, C., Häusler, P., Kettiger, D., Pulitano, D., Schneider, R.: Rechtsetzungsrichtlinien des Kantons Bern, Modul 3: Rechtsetzungstechnische Richtlinien (RTR). Justiz-, Gemeinde- und Kirchendirektion und Staatskanzlei des Kantons Bern, Bern (2000)
6. Caussignac, G., Eberhard, C., Häusler, P., Kettiger, D., Pulitano, D., Schneider, R.: Rechtsetzungsrichtlinien des Kantons Bern, Modul 4: Sprache. Justiz-, Gemeinde- und Kirchendirektion und Staatskanzlei des Kantons Bern, Bern (2000)

7. Clark, P., Harrison, P., Jenkins, T., Thompson, J., Wojcik, R.: Acquiring and using world knowledge using a restricted subset of English. In: FLAIRS 2005, pp. 506–511 (2005)
8. Europäische Kommission, Luxemburg, Amt für amtliche Veröffentlichungen der Europäischen Gemeinschaften: Gemeinsamer Leitfaden des Europäischen Parlaments, des Rates und der Kommission für Personen, die in den Gemeinschaftsorganen an der Abfassung von Rechtstexten mitwirken (2003), http://eur-lex.europa.eu/de/techleg/index.htm
9. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web 2008. LNCS, vol. 5224, pp. 104–124. Springer, Heidelberg (2008)
10. Hoefler, S., Bünzli, A.: Controlling the language of statutes and regulations for semantic processing. In: Proceedings of the LREC 2010 Workshop on Semantic Processing of Legal Texts (SPLeT 2010), Valletta, Malta, pp. 8–15 (2010)
11. Iluk, J.: Die Verständlichkeit der deutschen, österreichischen, schweizerischen und polnischen Verfassung, Versuch einer komparatistischen Analyse. In: Eichhoff-Cyrus, K.M., Antos, G. (eds.) Verständlichkeit als Bürgerrecht? Die Rechts- und Verwaltungssprache in der öffentlichen Diskussion, pp. 136–154. Dudenverlag, Mannheim (2008)
12. Kaljurand, K.: Paraphrasing controlled English texts. In: Fuchs, N.E. (ed.) Pre-Proceedings of the Workshop on Controlled Natural Language (CNL 2009). CEUR Workshop Proceedings, vol. 448, CEUR-WS (April 2009)
13. Lehrndorfer, A.: Kontrolliertes Deutsch. Linguistische und sprachpsychologische Leitlinien für eine (maschinell) kontrollierte Sprache in der Technischen Dokumentation. No. 415 in Tübinger Beiträge zur Linguistik, Gunter Narr Verlag, Tübingen (1996)
14. Lehrndorfer, A., Schachtl, S.: Controlled Siemens Documentary German and TopTrans. Technical Communicators Forum 3 (1998)
15. Lötscher, A.: Multilingual law drafting in Switzerland. In: Grewendorf, G., Rathert, M. (eds.) Formal Linguistics and Law. Trends in Linguistics. Studies and Monographs, pp. 371–400. Mouton de Gruyter, Berlin (2009)
16. Nussbaumer, M.: Zwischen Rechtsgrundsätzen und Formularsammlung: Gesetze brauchen (gute) Vagheit zum Atmen. In: Bhatia, V.K., Engberg, J., Gotti, M., Helier, D. (eds.) Vagueness in Normative Texts, Linguistic Insights. Studies in Language and Communication, vol. 23, pp. 49–71. Peter Lang, Bern (2005)
17. Nussbaumer, M.: Rhetorisch-stilistische Eigenschaften der Sprache des Rechtswesens. In: Fix, U., Gardt, A., Knape, J. (eds.) Rhetorik und Stilistik / Rhetoric and Stylistics. Ein Internationales Handbuch Historischer und Systematischer Forschung / An International Handbook of Historical and Systematic Research, Handbücher zur Sprach- und Kommunikationswissenschaft / Handbooks of Linguistics and Communication Science / [HSK] 31/2, ch. 128, vol. 2 (Halbband), pp. 2132–2150. Mouton de Gruyter (2009)
18. O'Brien, S.: Controlling controlled English: An analysis of several controlled language rule sets. In: EAMT-CLAW-2003, pp. 105–114, Controlled language translation (2003)
19. Pace, G.J., Rosner, M.: A Controlled Language for the Specification of Contracts. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 226–245. Springer, Heidelberg (2010)
20. Pool, J.: Can controlled languages scale to the web? In: CLAW 2006 at AMTA 2006: 5th International Workshop on Controlled Language Applications (2006)

21. Regierungsrat des Kantons Zürich: Richtlinien der Rechtsetzung (2005)
22. Reuther, U.: Two in one - can it work? Readability and translatability by means of controlled language. In: Proceedings of EAMT-CLAW (2003)
23. Schane, S.: Ambiguity and misunderstanding in the law. T. Jefferson L. Rev. 25, 167–649 (2002)
24. Schane, S.A.: Language and the law. Continuum International Publishing Group (2006)
25. Schweizerische Bundeskanzlei, in Zusammenarbeit mit der Zürcher Hochschule für Angewandte Wissenschaften: Geschlechtergerechte Sprache. Leitfaden zum geschlechtergerechten Formulieren im Deutschen, 2 edn. (2009)
26. Schweizerisches Bundesamt für Justiz, Bern: Gesetzgebungsleitfaden: Leitfaden für die Ausarbeitung von Erlassen des Bundes, 3 edn. (2007)
27. Schwertel, U.: Controlling plural ambiguities in Attempto Controlled English. In: Proceedings of the 3rd International Workshop on Controlled Language Applications, Seattle, Washington (2000)
28. Schwitter, R., Tilbrook, M.: Let's talk in description logic via controlled natural language. In: Proceedings of the 3rd International Workshop on Logic and Engineering of Natural Language Semantics, Tokyo, pp. 193–207 (2006)
29. Schwitter, R., Ljungberg, A., Hood, D.: ECOLE — a look-ahead editor for a controlled language. In: Proceedings of EAMT-CLAW 2003, pp. 141–150 (2003)
30. Schwitter, R., Tilbrook, M.: Annotating websites with machine-processable information in controlled natural language. In: Proceedings of the Second Australasian Workshop on Advances in Ontologies, AOW 2006, vol. 72, pp. 75–84. Australian Computer Society, Inc., Darlinghurst (2006)
31. Solan, L.M.: Linguistic principles as the rule of law. In: Pupier, P., Woehrling, J. (eds.) Language and Law: Proceedings of the First Conference of the International Institute of Comparative Linguistic Law. Wilson & Lafleur, Montreal (1989)
32. Solan, L.M.: Vagueness and ambiguity in legal interpretation. In: Bhatia, V.K., Engberg, J., Gotti, M., Helier, D. (eds.) Vagueness in Normative Texts, Linguistic Insights. Studies in language and Communication, vol. 23, pp. 73–96. Peter Lang, Bern (2005)
33. Sowa, J.F.: Common logic controlled English, draft, March 15 (2007), http://www.jfsowa.com/clce/clce07.htm
34. Venturi, G.: Parsing legal texts. A contrastive study with a view to knowledge management applications. In: LREC 2008 – W9 Workshop on Semantic Processing of Legal Texts (2008)
35. Verbeke, C.: Caterpillar Fundamental English. A basic approach for multination technical communication in an industry basic approach for multination technical communication in an industry. Training and Development Journal 27(2), 36–40 (1973)
36. Vogel, C.: Law matters, syntax matters and semantics matters. In: Grewendorf, G., Rathert, M. (eds.) Formal Linguistics and Law, Trends in Linguistics. Studies and Monographs, vol. 212, pp. 25–54. Mouton de Gruyter, Berlin (2009)

Interpreting Plurals in the Naproche CNL

Marcos Cramer and Bernhard Schröder

University of Bonn and University of Duisburg-Essen
cramer@math.uni-bonn.de, bernhard.schroeder@uni-due.de
http://www.naproche.net

Abstract. The Naproche CNL is a controlled natural language for mathematical texts. A recent addition to the Naproche CNL are plural statements. We discuss the collective-distributive ambiguity in the context of mathematical language, as well as pairwise interpretations of collective plurals. Additionally, we present a special scope ambiguity conjunctions give rise to. Finally, we describe an innovative plural interpretation algorithm implemented in Naproche for disambiguating plurals in DRT and giving them the interpretation that would normally be preferred in a mathematical context.

Keywords: Naproche, CNL, plurals, DRT, distributive reading, collective reading.

1 Introduction

The Naproche CNL [2] is a controlled natural language for mathematical texts, i.e. a controlled subset of the semi-formal language of mathematics (SFLM) as used in mathematical journals and textbooks. The Naproche system translates Naproche CNL texts first into Proof Representation Structures (PRSs, [2]), an adapted version of Discourse Representation Structures, which are further translated into lists of first-order formulae which are used for checking the logical correctness of a Naproche text using automated theorem provers.

The two main applications that we have in mind for Naproche are to make formal mathematics more readable to the average mathematician, and to use it as a tool that supports undergraduate students in writing formally correct proofs and thus get used to (a subset of) SFML.

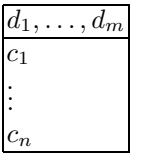
A recent addition to the Naproche CNL are plural statements. By this we mean not only statements involving nouns in the plural (e.g. "numbers") and verbs conjugated in plural forms (e.g. "are"), but also conjunctive coordinations of noun phrases (e.g. "$x + y$ and $x \cdot y$ are even"). We discuss two kinds of ambiguities that originate from plural statements: the ambiguity between collective and distributive readings of plurals, and a special scope ambiguity conjunctions give rise to. Both ambiguities are resolved by an innovative *plural interpretation algorithm* that is geared towards the use of plurals in mathematical texts, and described in detail in this paper. Plural definite noun phrases (e.g. "the real numbers") are not yet implemented in Naproche and are left out of the discussion in this paper.

M. Rosner and N.E. Fuchs (Eds.): CNL 2010, LNAI 7175, pp. 43–52, 2012.

2 Proof Representation Structures

Proof Representation Structures (*PRSs*) are Discourse Representation Structures, which are enriched in such a way as to represent the distinguishing characteristics of the mathematical language. For the purpose of this paper, we present a simplified definition of PRSs:

A PRS is a pair consisting of a list of discourse referents and an ordered list of conditions,[1] usually depicted as a box, similarly to a DRS:



Just as in the case of DRSs, PRSs and PRS conditions are defined recursively: Let A, B be PRSs and $d, d_1, \ldots, d_n$ discourse referents. Then

- for any n-ary predicate p (e.g. expressed by adjectives and noun phrases in predicative use and verbs in SFLM), $p(d_1, \ldots, d_n)$ is a PRS condition.
- A mathematical formula is a PRS condition.
- $\neg A$ is a PRS condition, representing a negation.
- $B \Rightarrow A$ is a PRS condition, representing an assumption (B) and the set of claims made inside the scope of this assumption (A).
- *static*(A) is a PRS condition.

Accessibility in PRSs is defined analogously to accessibility in DRSs: Thus discourse referents introduced in conditions of the form $\neg A$ or $B \Rightarrow A$ are not accessible from outside these conditions. We have introduced an additional condition of the form *static*(A), which allows us to represent existential claims with a static rather than a dynamic existential quantification: Thus discourse referent introduced in a condition of the form *static*(A) are also not accessible from outside this condition.

3 Collective vs. Distributive Readings of Plurals

The following sentence is ambiguous:[2]

(1) Three men lifted a piano.

It can mean either that three men lifted a piano together (in a single lifting act), or that there were three lifting acts, each of which involved a different man lifting a piano. The first is called the *collective* reading, the second the

[1] The use of ordered lists rather than sets in the definition of PRSs was motivated in [2].

[2] A comprehensive overview over plural readings is given by [6].

distributive reading.[3] The ambiguity arises because the agent of a lifting event can either be a collection of individuals or a single individual.

In SFLM, both the collective and the distributive reading exist:

(2) 12 and 25 are coprime.

(3) 2 and 3 are prime numbers.

Instead of (2), one could also say "12 is coprime to 25." So the adjective "coprime" can be used in two grammatically distinct ways, but in both cases refers to the same mathematical binary relation: either it is (predicatively or attributively) attached to a plural NP that gets a collective reading, or it has as a complement a prepositional phrase with "to". When used in the first way, we call "coprime" a *collective adjective*, when used in the second way, a *transitive adjective*. We say that the two logical arguments of "coprime" can be *grouped* into one collective linguistic argument, a plural NP with a collective reading. In general, mathematical adjectives expressing a symmetric binary relation have these two uses (cf. "parallel", "equivalent", "distinct", "disjoint"; in the case of "distinct" and "disjoint", the preposition used for the transitive case is "from" rather than "to"). Other cases of grouped arguments are "x and y commute" (cf. "x commutes with y") and "x connects y and z" (cf. "x connects y to z"). "x is between y and z" is an example of an expression with a grouped argument for which there is no corresponding expression without grouped arguments.

Since "prime number" expresses a unary relation, it is not possible to group two of its logical arguments into a single linguistic argument; this explains why (3) can't have a collective reading of the sort that (2) has. Which expressions can have grouped arguments is coded into the lexicon of the Naproche CNL.

An ambiguity like that of (1) can only arise when an expression (here the verb "to lift") has a linguistic argument that can be either a collectively interpreted plural NP or a singular NP (and can hence also be a distributively interpreted plural NP). Such expressions are extremely rare in SFLM. One example that we are aware of is the adjective "inconsistent":

(4) φ and ψ are inconsistent.

(4) can be mean either that the set of formulae $\{\varphi, \psi\}$ is an inconsistent set of formulae, or that φ is inconsistent and ψ is inconsistent. This ambiguity is avoided in Naproche by not marking "inconsistent" as an expression with grouped arguments in our lexicon, so that (4) only has the distributive reading; the collective reading can only be expressed with explicit set notation in Naproche.

4 Scope Ambiguity

Another kind of ambiguity of special interest for our treatment of plurals and noun phrase conjunctions is a scope ambiguity that arises in certain sentences containing a noun phrase conjunction and a quantifier:

[3] We ignore cumulative readings here, because they play a negligible role in the mathematical contexts we have in mind.

(5) A and B contain some prime.

(5) can mean either that A contains a prime and B contains a (possibly different) prime, or that there is a prime that is contained in both A and B. In the first case we say that the scope of the noun phrase conjunction "A and B" contains the quantifier "some", whereas in the second case we say that the scope of "some" contains the noun phrase conjunction. We call the first reading the *wide-conjunction-scope* reading and the second the *narrow-conjunction-scope* reading.

Sometimes certain considerations of reference or variable range force one of the two readings, as in (6) and (7).

(6) x and y are integers such that some odd prime number divides $x + y$.

(7) x and y are prime numbers p such that some odd prime number q divides $p + 1$.[4]

(6) only has a narrow-conjunction-scope reading, because the existentially introduced entity is linked via a predicate ("divides") to a term ("$x + y$") that refers to the coordinated noun phrases individually. (7) on the other hand only has a wide-conjunction-scope reading, because the variable p must range over the values of both x and y, and q depends on p.

In general, there is, like in common language use, a strong tendency in SFLM texts to resolve scope ambiguities by giving wider scope to a quantifier that is introduced earlier in a sentence than to a quantifier introduced later in the sentence. This is a principle that we have already long ago adopted into Naproche in order to avoid scope ambiguities in the Naproche CNL. With the addition of coordinated NPs, we extended this principle to their scopes, with the exception of the cases like (6) where another reading is forced by certain syntactical considerations. Section 6 contains an account of how cases like (6) are identified.

5 Pairwise Interpretations of Collective Plurals

In SFLM texts, one often sees sentences like (8) and (9), which are interpreted in a pairwise way as in (10) and (11):

(8) 7, 12 and 25 are coprime.

(9) All lines in A are parallel.

(10) $\mathit{coprime}(7, 12) \wedge \mathit{coprime}(12, 25) \wedge \mathit{coprime}(7, 25)$

[4] Given that this example is made up, one might ask whether it really occurs in SFLM texts that a plural noun followed by a variable is predicatively linked to a conjunction of terms as in this example. One real example that we found comes from page 4 of [1]: "Notice that 13, 37, 61, ..., are primes p such that $p^3 + 2$ and $p^3 + 1$ are squarefree."

(11) $\forall x, y \in A\ (x \neq y \rightarrow \mathit{parallel}(x, y))$[5]

Sometimes, especially in connection with the negative collective adjectives "distinct" and "disjoint", this interpretation is reinforced through the use of the word "pairwise", in order to ensure that one applies the predicate to all pairs of objects collectively referred to by the plural NP. But given that this pairwise interpretation is at any rate the standard interpretation of such sentences even in the absence of the adverb "pairwise", we decided not to require the use of the word "pairwise" in the Naproche CNL.

The Naproche CNL allows only this pairwise interpretation for a plural NP that is used as a grouped argument of such a collective adjective. (12) is a sentence where another reading (14) might naturally be preferred to the pairwise interpretation (13) that Naproche assigns to it:

(12) Some numbers in A and B are coprime.

(13) $\exists n, m\ (\mathit{number}(n) \wedge n \in A \wedge n \in B \wedge \mathit{number}(m) \wedge m \in A \wedge m \in B \wedge \mathit{coprime}(n, m))$

(14) $\exists n, m\ (\mathit{number}(n) \wedge n \in A \wedge \mathit{number}(m) \wedge m \in B \wedge \mathit{coprime}(n, m))$

However, it seems to us that such sentences hardly appear in real mathematical texts.

6 The Plural Interpretation Algorithm

In the Naproche system, the PRS construction algorithm for the representation of single sentences has been added to the standard threading algorithm for DRS construction (see [4]), and is implemented in Prolog. The algorithm can cope with plurals, plural ambiguity resolution and pairwise interpretations as explained in the previous sections. We illustrate how the algorithm treats plurals by considering the following example sentence:

(15) x and y are distinct primes p such that $2p+1$ is a square number and some odd prime divides $x + y$.

This example has only one natural reading, and illustrates all the natural disambiguation methods mentioned in the previous sections: The plural construction "x and y" is modified by one predicate ("distinct") that needs to be interpreted collectively and by one predicate ("prime") that needs to be interpreted distributively. One of the existential NPs in the such-that clause ("a square number") has to be given a narrow scope, while the other ("some odd prime") has to be

[5] The distinctness condition here can be ignored in the case of reflexive relations like "parallel", but is certainly needed for non-reflexive relations like "coprime" or "disjoint".

given a wide scope. The algorithm specifies a formal procedure to attain this natural reading.

The algorithm works by first producing a preliminary representation (Fig.1):

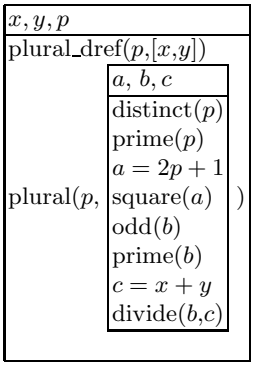


Fig. 1. Preliminary PRS

Here the NP conjunction gets a plural discourse referent (p in Fig. 1), which is linked to the discourse referents of the conjuncts by a *plural_dref condition*. We give the NP conjunction wide scope over all quantifiers introduced later, and all assertions made in the scope of the plural NP are inserted in a special *plural sub-PRS*. The *plural-dref* and *plural* conditions used in such preliminary PRSs are book-keeping devices and not part of the official PRS language.[6]

The goal of the algorithm is to eliminate the plural discourse referents in favour of the singular discourse referents they subordinate. This has to be done separately for the distributively and collectively interpreted parts. The distributive interpretations opens a scopus, in which there may occur dependent variables. The algorithm consists of five steps, which can be summarized as follows: For each plural referent:

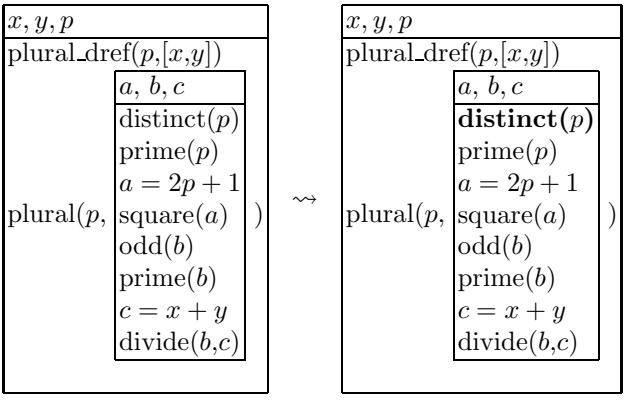
1. Mark the collective uses of the plural referent.
2. Mark the distributive uses of the plural referent and dependent variables.
3. Separate the scopus of distributive uses of the plural referent from the rest.
4. Replace collective variable occurrences.
5. Replace distributive variable occurrences.

Now we describe each of the steps more formally:

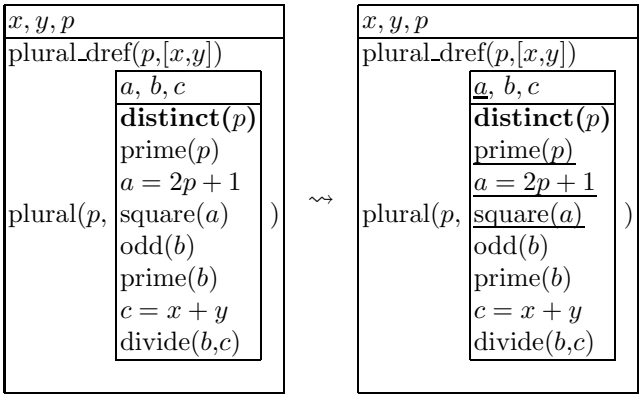
1. Marking the Collective Uses of the Plural Referent: In the plural sub-PRS, we mark every PRS condition which consists of a predicate that has the

[6] Alternatively, one may consider these conditions as extensions of the PRS language, in which case the semantics of the *plural* condition has to be an underspecified semantics in the sense of [3], which represents the different scopal interpretations of the plural NP.

plural discourse referent as grouped argument ("distinct(p)" in the example PRS, marked by boldface). That the plural discourse referent is a grouped argument is derived from the fact that the number of arguments, with which the predicate appears in the plural sub-PRS, is one less than its logical number of arguments fixed in the lexicon, and from the fact that the lexicon specifies the possibility of grouping two of its arguments into one.

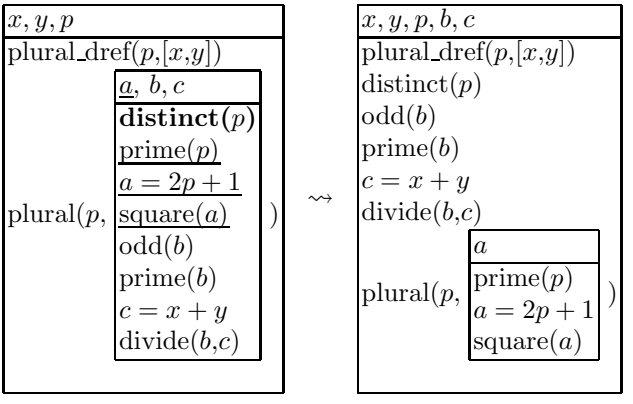


2. Marking the Distributive Uses of the Plural Referent and Dependent Variables: In the plural sub-PRS, we recursively mark (in the figure by underlining) all PRS conditions that were not marked in step 1 and contain the plural discourse referent or a marked discourse referent, and all discourse referents contained in a PRS condition marked in this way, until no more conditions and discourse referents can be marked by this process:

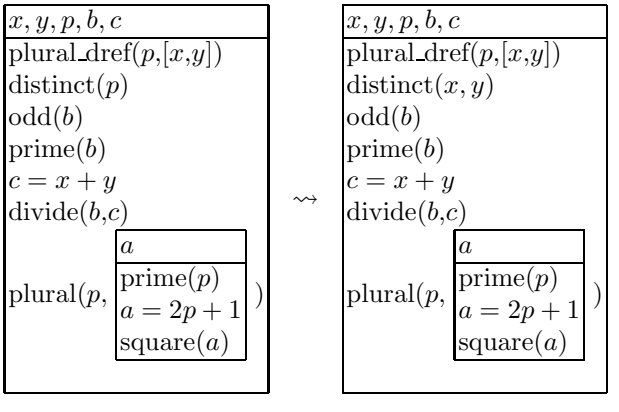


3. Separating the Scopus of Distributive Uses of the Plural Referent from the Rest: All discourse referents and PRS conditions in the plural

sub-PRS not marked in step 2 get pulled out of the plural sub-PRS and inserted into its super-PRS:[7]

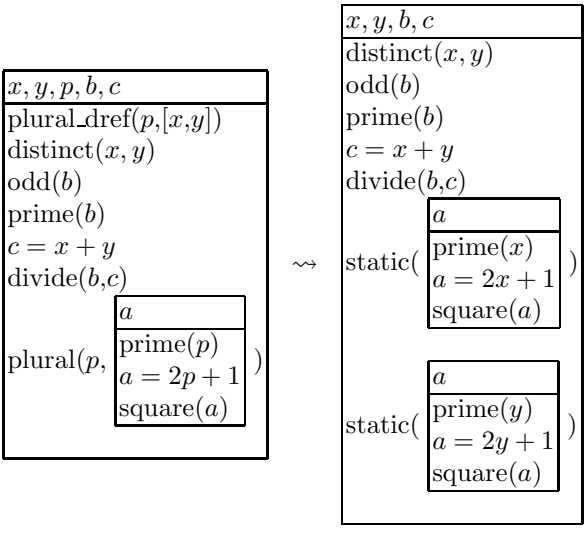


4: Replacing Collective Variable Occurrences: For every PRS condition $p(d)$ with grouped argument d, and every pair d_1, d_2 of distinct discourse referents linked to d via a plural_dref condition, we create a PRS condition of the form $p(d_1, d_2)$ and remove the original PRS condition $p(d)$ (in our example this amounts to replacing "distinct(p)" by "distinct(x, y)"):



5. Replacing Distributive Variable Occurrences: For every discourse referent d linked to the plural discourse referent p, we make a static copy of the plural sub-PRS in which every instance of p is replaced by d, removing the original plural sub-PRS:

[7] Since this step moves discourse referents and conditions around, one might wonder whether it can cause formally bound variables to become free. This, however, is impeded by the recursive procedure in step 2: If a certain discourse referent stays in the plural sub-PRS, no condition containing this discourse referent can be pulled out of the plural sub-PRS.



The final PRS corresponds to the natural reading of sentence (15) that we described at the beginning of this section.

7 Related and Future Work

The syntax of Attempto Controlled English (ACE) allows plurals, which are interpreted in ACE in an unambiguous way [8]. The disambiguation used by ACE is very distinct from Naproche's: while Naproche gives preference to distributive and wide-conjunction-scope readings, ACE allows only collective and narrow-conjunction-scope readings, unless the word "each" is used. This difference is due to the fact that for Naproche we focused on the interpretations common in SFLM, whereas ACE took the English language as a whole into account. Our focus on mathematical language also made it important for us to treat "x and y are coprime" and "x is coprime to y" as logically equivalent, which ACE does not do.

ForTheL, the controlled natural language of the System for Automated Deduction (SAD), a project with similar goals to Naproche, already included the two uses of words like "parallel" and "to commute" and produced the same representation no matter in which way they were used [7].

At the moment, Naproche does not yet allow anaphoric pronouns like "it" and "they". When Naproche is extended to allow them, some rules specifying how to control the many ways in which an anaphoric antecedent for "they" can be chosen (see [5]) will have to be specified and implemented, again with special attention to existing usage in SFLM.

8 Conclusion

We have implemented a plural intepretation algorithm that can handle a number of constructs related to plurals in a way that seems desirable for a mathematical

CNL: While a distributive reading of plurals is preferred, a collective reading is chosen for predicates with grouped arguments and the pairwise interpretation of predicates with grouped arguments is chosen when feasible. Additionally, the scope ambiguity that noun phrase conjunctions give rise to is disambiguated with respect to the syntactic-semantic context.

References

1. Cohen, G.L.: Derived Sequences. Journal of Integer Sequences 6 (2003)
2. Cramer, M., Fisseni, B., Koepke, P., Kühlwein, D., Schröder, B., Veldman, J.: The Naproche Project Controlled Natural Language Proof Checking of Mathematical Texts. In: Fuchs, N.E. (ed.) CNL 2009 Workshop. LNCS (LNAI), vol. 5972, pp. 170–186. Springer, Heidelberg (2010)
3. Egg, M.: Semantic underspecification. In: Maienborn, C., von Heusinger, K., Portner, P. (eds.) Semantics. HSK, vol. 33. De Gruyter Mouton (2011)
4. Johnson, M., Klein, E.: Discourse, anaphora and parsing. In: Proceedings of the 11th Coference on Computational linguistics (1986)
5. Kamp, H., Reyle, U.: From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Language. Kluwer Academic Publisher (1993)
6. Link, G.: Plural. In: von Stechow, A., Wunderlich, D. (eds.) Semantics. HSK, vol. 6. de Gruyter (1991)
7. Paskevich, A.: The syntax and semantics of the ForTheL language (2007)
8. Schwertel, U.: Plural Semantics for Natural Language Understanding – A Computational Proof-Theoretic Approach. PhD thesis. University of Zurich (2005)

Engineering a Controlled Natural Language into Semantic MediaWiki

Pradeep Dantuluri, Brian Davis, Pierre Ludwick, and Siegfried Handschuh

Digital Enterprise Research Institute, National University of Ireland, Galway
{pradeep.varma,brian.davis,pierre.ludwick,siegfried.handschuh}@deri.org

Abstract. The Semantic Web is yet to gain mainstream recognition. In part this is caused by the relative complexity of the various semantic web formalisms, which act as a major barrier of entry to naive web users. In addition, in order for the Semantic Web to become a reality, we need semantic metadata. While controlled natural language research has sought to address these challenges, in the context of user friendly ontology authoring for domain experts, there has been little focus on how to adapt controlled languages for novice social web users. The paper describes an approach to using controlled languages for fact creation and management as opposed to ontology authoring, focusing on the domain of meeting minutes. For demonstration purposes, we developed a plug-in to the Semantic MediaWiki, which adds a controlled language editor extension. This editor aids the user while authoring or annotating in a controlled language in a user friendly manner. Controlled content is sent to a parsing service which generates semantic metadata from the sentences which are subsequently displayed and stored in the Semantic MediaWiki. The semantic metadata generated by the parser is grounded against a project documents ontology. The controlled language modeled covers a wide variety of sentences and topics used in the context of a meeting minute. Finally this paper provides a architectural overview of the annotation system.

1 Introduction

The Semantic web[1] aims to simplify the process of building knowledge-based applications by enabling a web of inter-operable and machine-readable data. This is done by formalizing the descriptions of the structure and semantics of the data available on the web. However creating and exposing semantic data is a task that requires thorough knowledge of various technologies. The relative complexity of the various semantic web formalisms, act as a major barrier of entry to naive web users. A solution to this is to create technologies which enable the average internet user to annotate and embed data in his/her own textual resources. While controlled natural language research has sought to address these challenges, in the context of user friendly ontology authoring for domain experts,

[1] http://www.w3.org/2001/sw/

M. Rosner and N.E. Fuchs (Eds.): CNL 2010, LNAI 7175, pp. 53–72, 2012.

there has been little focus on how to adapt CNLs for novice social web users for fact creation and semantic annotation.

The paper describes an approach using controlled languages for fact creation and management as opposed to ontology authoring. We explore the possibility of using controlled natural languages (hereby referred to as CNL) as an interface to semantic web applications, specifically targeting the domain of project documents like meeting minutes, status reports, etc. The major goal is to enable novice users to author and annotate text documents using a controlled language. Furthermore, these documents can be parsed to extract the implicit knowledge contained, due to the enforcement of a fixed grammar and vocabulary. The authors have previously used this approach to build an annotation tool along with prototypes of the grammar and ontologies for the meeting minutes domain [1]. Additionally, the controlled language allows users to create facts in two ways, using the controlled language text and using declarative annotations with the controlled language text, we call CNL snippets. Both of these methods are explained in further detail later. CNL snippets lie in the formality continuum as mentioned in [2], where formal constraints are relaxed in favour of higher user uptake by a reduction in knowledge capture.

CLANN (Controlled Language for ANNotation)[2] builds on the experiences gained from the previous work, by incorporating redesigned versions of the grammar and the domain ontology. CLANN is designed to be an end-to-end semantic web application complete with a domain ontology, a persistent layer based on RDF[3] and a user interface for editing and authoring documents. The domain was expanded to include all the documents in a project specific setting (for example, meeting minutes, status reports, etc). For demonstration purposes, we developed a plug-in to the Semantic MediaWiki, which adds a controlled language editor as an extension. This editor helps the user author or annotate in controlled language in a user friendly manner. Controlled content is sent to a parsing service which generates RDF metadata from the sentences which are subsequently displayed and stored in the Semantic MediaWiki. The RDF generated by the parser is grounded against a project documents ontology. The controlled language grammar, modeled using Link Grammar [3], covers a wide variety of sentences and topics used in the context of a meeting minute.

The main contributions of the paper include

- Modeling an ontology for the domain of project documents .
- Design and implementation of the CLANN grammar using link grammars.
- Design and implementation of the CLANN Editor as a plugin to the Semantic Mediawiki.

This paper is structured as follows. Section 2 introduces the CLANN software and describes its various components : the CLANN editor, CLANN grammar and the PDO ontology. Section 3 showcases the implementation details of the

[2] In this document *CLANN* refers to the annotation platform as well as controlled grammar.

[3] Resource Description Framework - `http://www.w3.org/RDF/`

various components of the CLANN system. Section 4 discusses related work in the fields of Controlled languages and Semantic Controlled.

2 CLANN System

The CLANN system is composed of a CLANN editor and a stand-alone server. The Clann editor, a custom-built web-based editor, allows for easy editing and correction of controlled language text, using auto-suggestions and sentence error corrections . The CLANN editor communicates with the server for parsing the sentences and retrieving additional error information for incorrectly parsed sentences. The user is allowed to iterate over several parses until he gets the sentence correct. A successfully parsed sentence generates valuable information as RDF triples, which are sent back to the editor.

2.1 CLANN Grammar

The CLANN grammar is designed to facilitate knowledge capture from everyday, repetitive, domain-specific texts. To better explore the applicability, two independent prototypes of the grammar were developed [1] focusing alternatively on usability and expressivity. This work has eventually led to the CLANN grammar, which is essentially a merge between the former two grammars, incorporating most of the advantages, albeit a few changes. The grammar is designed with a focus on both usability and expressivity. Each sentence adheres to one of the syntactic rules and uses a lenient vocabulary. This domain vocabulary was derived by corpus analysis using Word Smith tools [4] on a corpus based on three years of meeting minutes from the Nepomuk project. This ensured that most of the sentences resembled normal English sentences. Examples of such syntactic constructs are given in Table 1.

Table 1. Excerpt of CLANN grammar with examples

Sentence Pattern	Example & Parsed pattern

`<NP><VP><NP>(<PP>+)`	*Ambrosia to submit her PhD Proposal by Friday.*
	`(Ambrosia <NP>) (to submit <VP>)`
	`(her PhD Proposal <NP>) (by (Friday <NP>)<PP>).`

	Mark attended CNL2010 in Sicily during the last week.
	`(Mark <NP>) (attended<VP>) (CNL2010 <NP>)`
	`(in (Sicily <NP>) <PP>)(during (the last week`
	`<NP>)<PP>).`

The CLANN grammar differs from the conventional notion of CNL, whereby the entire document is written in CNL, rather it allows the user to add snippets of

[4] http://www.lexically.net/wordsmith/version5/index.html

CNL text, enclosed in "[ ]", to the document or associate them to a particular text in the document. These snippets act as annotations to the text preceding the snippet. They should adhere to a *Verb-Object* syntax, where the subject is either specified in the snippet or taken from the free text. This approach was inspired by the CLOnE[5] Language [4]. Examples of CLANN snippets are described in Table 2.

Table 2. CLANN Snippets with examples

Sentence Pattern	Example
`<text>[is a <Class>].`	`Dirk[is a Person].` Creates an object of the class Person with label *Dirk*.
`<text>[same as <Instance>].`	`Brian[same as Brian Davis]` Annotates the text *Brian* with the instance *Brian Davis*.
`<text>[<property> <object>].`	`Dirk[toComplete "PhD Proposal"]` Creates a triple which links the instances of *Dirk* and *PhD Proposal* with the property *toComplete*.

The CLANN grammar, thus, makes use of **two** different modes to encode knowledge into the text. They are the (i)*CNL snippets* (controlled language snippets) and (ii)*controlled english*. The user can also mix both modes thus enabling the user to bootstrap the vocabulary of the CNL in the second mode using CNL snippets. Each of these are explained in further detail below.

2.1.1 CNL Snippets

Controlled Language snippets are used to explicitly add annotation to words in the text. The text maybe uncontrolled or controlled. Snippets of text (enclosed in "[ ]") can be appended to words in the document to explicitly add annotations to the document. These snippets should adhere to a *-verb-object* syntax, where the subject is text preceding the snippet. These snippets merely append additional information within the sentence, but are not considered to be part of the controlled language sentence. An example of the snippets is shown below. Let us consider the sentence below

```
Brian went to Dublin for the weekend.
```

More information about "Brian" and "Dublin" can be added by using CL Snippets.

```
Brian[same as Brian Davis] went to Dublin[is a City] for the weekend.
```

[5] CLOnE - Controlled Language for Ontology Editing

Further new resources and links between existing resources can be defined. Assuming *livesIn* is a property between Person and City pre-defined in the domain ontology, we can express the following.

```
[City is a subclass of Place].
[Brian livesIn Dublin].
Brian[same as Brian Davis] went to Dublin[is a City] for the weekend.
```

The reader should note that any CNL Snippets ([...]) which are written are hidden in the final published wiki page[6].

2.1.2 Controlled English

The controlled english, developed during the process, forms the core of the CLANN grammar. The structure of sentences of the controlled english are restricted by a simple set of design principles described below.

- Every sentence should be *declarative* and in *active voice*.
- Every verb used should be either an *infinitive* (to - verb : to denote tasks) or *simple past tense* (to denote reports)
- Every sentence should follow the *Subject-Verb-Object* pattern.
- Only *prepositional clauses* (prep NP : "by next week", etc) can be used to append nouns.
- Include *"snippets"* inline to add extra annotations
  - *Pradeep[is a Student]...*

2.2 CLANN Editor

The CLANN editor is designed from ground up to be an extendable, customizable and stand-alone controlled language editor. This allowed the editor to be easily integrated into the Semantic MediaWiki platform as a plugin, thereby inheriting the RDF support of the platform. The plugin adds an extra button to the edit page of the Semantic MediaWiki, which, when clicked, opens the Clann Editor in a separate dialog box. RDF data generated by the editor is stored natively in the Semantic Mediawiki too. A screenshot of the interface integrated into the Semantic Mediawiki is depicted in Figure 1.

As evident from the screenshot, users would be able to use the medaiwiki in the usual way , creating and editing pages using the standard wiki markup[7]. The screen shot shows the edit mode of a page describing meeting minutes . Additionally the screenshot also shows a pop-up of the CLANN Editor interface, which is used to add controlled language text. When a user wants to add controlled language text , he click on the CNL button, which pops up the CLANN

[6] The above guidelines are only easy-to-read restrictions of the grammar. For a more complete specification of the grammar please refer to http://smile.deri.ie/projects/clann

[7] Mediawiki uses a very popular syntax for page formatting. More information available at http://en.wikipedia.org/wiki/Help:Wiki_markup

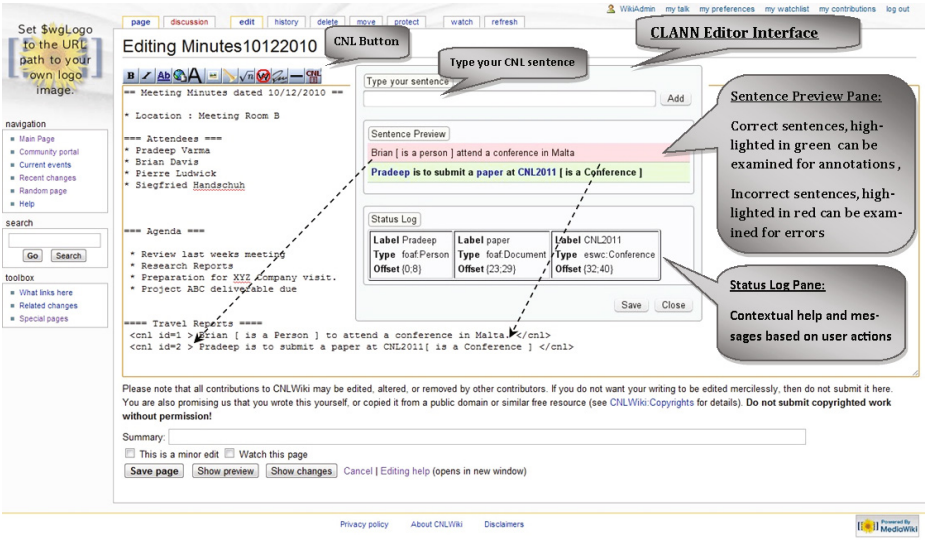


Fig. 1. Screenshot of the CLANN Editor Interface

editor in a separate frame. The user would be able to use this editor to add his controlled text, which would appear within the wiki page within special tags : `<cnl> ... </cnl>` . On saving the page, the CLANN plug-in for the Semantic Mediawiki, picks up the text within the `<cnl>` tags and sends them to a server which returns the appropriate RDF data for the sentences. This RDF data is stored in the local RDF store and used to create additional pages in the Semantic Mediawiki to reflect the concepts described in the data. Let us consider the following example .

```
Pradeep is to submit a paper at CNL2011[is a Workshop].
```

The above sentence should be able to return the following RDF data.

```
:Sentence4567 rdf:type pdo:ActionItem;
  pdo:mentions :Person9821, :Document5427, :Workshop8672;
  pdo:hasText "Pradeep is to submit a paper at CNL2011[is a Workshop]".

:Person9821 rdf:type foaf:Person;
            skos:altLabel "Pradeep",

:Document5427 rdf:type pdo:Document;
           skos:altLabel "paper",

:Workshop8672 rdf:type bibo:Workshop;
            skos:altLabel "CNL2011".

:Document5427 dc:contributor :Person9821;
```

This is a turtle[8] representation of the data and parts of it are removed for brevity. The sentence is recognized as an action item and an appropriate instance is created. Additionally, *Pradeep*, *paper* and *CNL2011* are recognized as a *Person*, *Document* and *Conference* respectively and linked to the *ActionItem* instance. Additionally, the verb *to submit* is mapped to the property *dc:contributor*, linking the person and document instances. As you may notice several ontologies are used to describe the concepts, namely, foaf[9] , bibo[10], skos[11], dc[12] and pdo[13].

Semantic Mediawiki encodes the structure of the wiki, its various pages and links between them as RDF data. This is done by interpreting the pages as instances and links between them as properties. The CLANN plugin uses the RDF data generated from the above sentence to create new pages for all the new instances as well as the appropriate links between the pages to reflect the relations between the instances in the data. Thus, new pages are created for *:Sentence4567*, *Pradeep*, *paper* and *CNL2011* and they are linked appropriately. Hence, the original Wiki Page becomes semantically annotated. Storing this valuable data as wiki pages and RDF data would now allow the user to use all the capabilities provided by the Semantic Mediwiki to search, aggregate, visualize and export this data.

2.3 PDO Ontology

The domain of meeting minutes and status reports was used to engineer an ontology for the purpose of knowledge management. The initial CLANN prototype was bootstrapped using the the Nepomuk [5] ontologies[14] and later extended by MEMO(Meeting Minutes Ontology). However, the MEMO ontology was only used as a proof-of-concept implementation of the domain. Later, this was completely redesigned and a new ontology PDO (Project Document ontology) was developed in accordance with proper ontology design principles, specifically the METHONTOLOGY approach outlined by [6]. The PDO ontology, described using RDFS[15] and OWL-DL[16], models the inherent structure and concepts of various documents in a project-specific setting, like meeting minutes, status reports etc. A graphical representation of the ontology is shown in Figure 2. *Document* is the central class which is subclassed by *Minutes* and *StatusReport*. *Artefact* is the main place-holder class for various artefacts contained in a document, like

[8] Turtle is a human-readable serialization of RDF, more information can be found at `http://www.w3.org/TeamSubmission/turtle/`

[9] Friend of a Friend ontology, used to describe persons and their profiles

[10] Bibliography Ontology

[11] Simple Knowledge Organization System

[12] Dublin Core Terms ontology

[13] Project Documents Ontology

[14] `http://www.semanticdesktop.org/ontologies/`

[15] RDFSchema `http://www.w3.org/TR/rdf-schema`

[16] OWL (Web Ontology Language) has three flavours, OWL Lite, OWL DL and OWL Full. `http://www.w3.org/TR/owl-features/`

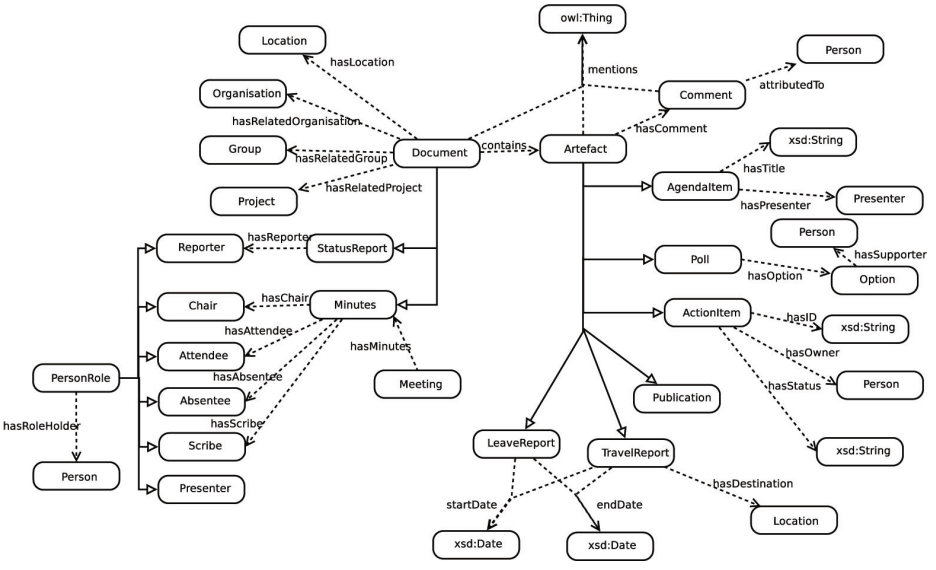


Fig. 2. Overview of the PDO ontology

AgendaItem, *Poll*, *ActionItem*, *TravelReport*, etc. A partial instantiation of the ontology is described in Turtle[17] syntax in the Figure 3. For a complete specification of the PDO ontology please refer to `http://ontologies.smile.deri.ie/pdo#`.

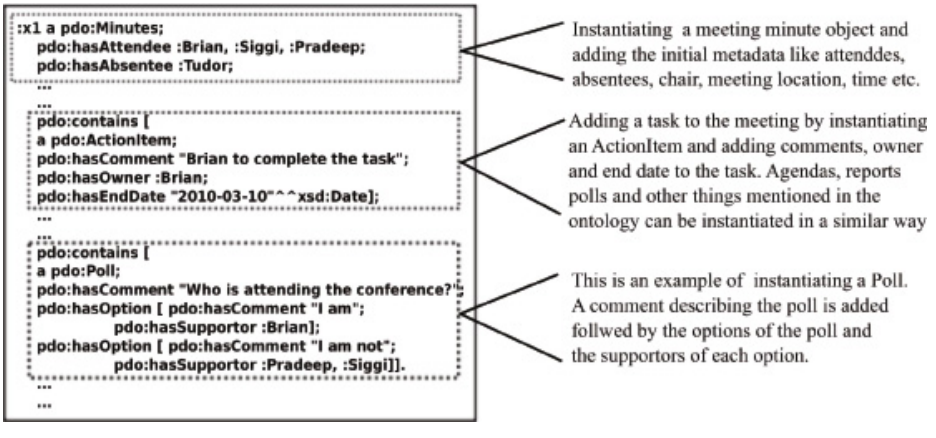


Fig. 3. Using the PDO Ontology

[17] Turtle is an easy, human-readable serialization of RDF. `http://www.w3.org/TeamSubmission/turtle/`

The scope of this ontology was limited to modeling the discourse structure of various project documents like meeting minutes, status reports, final reports, deliverables, etc. The content of these documents is not modeled, in order to make the ontology very flexible and interoperable. Care was taken to ensure that other domain ontologies can be easily integrated. So, for instance, a meeting minute note might talk about anything from software projects to movie reviews but still be modeled by the ontology, while using the respective domain ontologies of software projects and movies. CLANN uses several external ontologies, which describe related domains. The current list of ontologies used are mentioned in Table 3. As the software evolves, more domain ontologies would be added to this list.

Table 3. External Ontologies

Prefix	Description
foaf	Friend of a Friend Ontology describing people and profiles. URL : `http://xmlns.com/foaf/0.1/`
event	Event Ontology describing events. URL : `http://purl.org/NET/c4dm/event.owl#`
bibo	Bibliography Ontology describing bibliographic information. URL : `http://purl.org/ontology/bibo/`
swrc	Semantic Web for Research Communities, describes academic conferences URL : `<http://swrc.ontoware.org/ontology#`
skos	Simple Knowledge Organization System . URL : `http://www.w3.org/2004/02/skos/core#`
time	Time ontology describing people and profiles. URL : `http://www.w3.org/2006/time#`
geo	Geospatial vocabulary describing spatially located things. URL : `http://www.w3.org/2003/01/geo/wgs84_pos#`
doap	Description of a Project Ontology, describes sofware projects. URL : `http://usefulinc.com/ns/doap`

3 Implementation

3.1 CLANN Grammar

3.1.1 Implementation of the Grammar

The previous prototypes of the grammar were developed using JAPE(Java Annotation Patterns Engine) rules in GATE. JAPE provides finite state transduction over annotations based on regular expressions[7] .[18] The ease of writing rules in Jape coupled with the extensive support of the GATE platform immensely helped in developing and testing rapid prototypes of the grammar. However adding support for auto-completion, based on Jape rules proved to be much harder. So we decided to explore other grammar formalisms which would take

[18] Elaborating on JAPE is out of scope of this paper. For more information on the subject the reader is referred to
`http://gate.ac.uk/sale/tao/splitch8.html#chap:jape`

advantage of the restricted vocabulary of the grammar to produce efficient parses along with support for auto-completion.

Various grammar formalisms have been used over the years for understanding natural language. Phrase structure grammars (PSG), the most widely used formalism, model the inherent structure of the sentences of a language by breaking it into different phrases. They belong to the class of generative grammars and are composed of a set of productions or rules which break-up the sentences into meaningful phrases. Dependency grammars(DG), however, concentrate on the links between words without paying attention to the word order. Structure of a sentence is not broken down into phrases, but determined by adding relations between a head word and its dependent words. There have been many variations of grammar formalisms, each based on either PSGs or DGs. The next few sections describe one such variation of the dependency grammar, the Link grammar, and justifies its selection.

3.1.2 Link Grammar

Link grammars, introduced by [3], are a variation of dependency grammars. Similar to DGs, the link grammars use relations between words to generate a structure for a sentence. However, unlike DGs, the links also encode information about directionality and distance. Moreover, they do not enforce a head-dependent relationship like the DGs.

[3] defines link grammar as follows:

> *A sequence of words is a sentence of the language if there is a way to draw links between words in such a way that*
> - *the linking requirements of all the words are satisfied,*
> - *the links do not cross, and*
> - *the words form a connected graph*

The linking requirements of each word are specified as a dictionary, which forms the basis of the link grammar. Each entry in the dictionary consists of a word or a group of words belonging to the same grammatical category, appended on the right-hand-side with its linking requirements. The linking requirements are a series of connectors joined by the logical operators *&* and *or*. Each connector denotes the type and direction of the link. It is a label followed by *+/-* . A *+* denotes a link to the right and - denotes a link to the left. For illustration purposes, an example of a sentence parsed using a very simple link grammar is provided in Figure 4, and an explanation of the same is provided below.

The *D+* connector on the word *the* denotes that *the* is expecting a *D* link to its right. So It can connect to any word which has a *D-* connector, which, in this case, is either *boy* or *apple*. The word *ate* has an *&* operand on *S-* and *O+*. This means, for the word *ate* to be part of a valid sentence, it should connect to both an *S* connector to its left and an *O* connector to its right. The case for the nouns *boy* and *apple* is more interesting. They have two expressions joined by the *or* operand. On closer observation, the first one, *(A- & D- & S+)*, models the behavior of a subject noun and the second one, *(D- & O-)*, models that of an object noun. The reader should also note that the order of the connectors is

words	linking requirements
the	D+
small	A+
ate	S- & O+
boy apple	(A- & D- & S+) or (D- & O-)

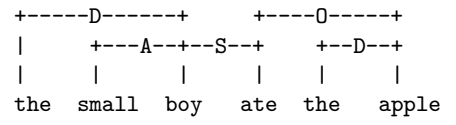


Fig. 4. A sample Link grammar and parse structure

also valuable. The expression *(A- & D- & S+)* also declares the order of linking. So an *A* link should be made to a word closer than the *D* link. This is illustrated in the parse structure shown in Figure 4.

3.1.3 Why Link Grammar?

The main design principles for the CLANN grammar are ease of use and the ability to extend the ontology. However, the development of the grammar posed different challenges.

One major priority was to extract RDF triples from the sentences. This works very well with the link grammar parse, because the the triples can directly be extracted by mapping the links. In the example shown in Figure 4, the triple *boy ate apple*, can be easily extracted from the left and right links of the word *ate*. This is not the case with phrase structure grammars, where extracting dependencies requires detailed analysis of the tree structure.

Another major priority was to develop an intelligent editor on top of the grammar, which supports auto-suggestion and sentence-completion. An intuitive editor which assists the user while writing the CNL sentences, is extremely beneficial to the user in that it can help her to quickly learn the restrictions of the grammar. This requires an ability to predict text and check the grammatical correctness of partial sentences. The pre-existing dictionaries of the link grammar provide valuable information about all the words of the language, which can be exploited for the purpose.

3.2 Architecture

Although, the current CLANN system is developed for a very specific domain, i.e, meeting minutes, the idea was to make the architecture flexible enough to allow adapting the system to new domains. Every domain needs a specific grammar and a set of ontologies tailored for that domain. Hence, a modular approach is followed while reducing the coupling between the various modules to a minimum. The architecture of the system is depicted in the Figure 5.

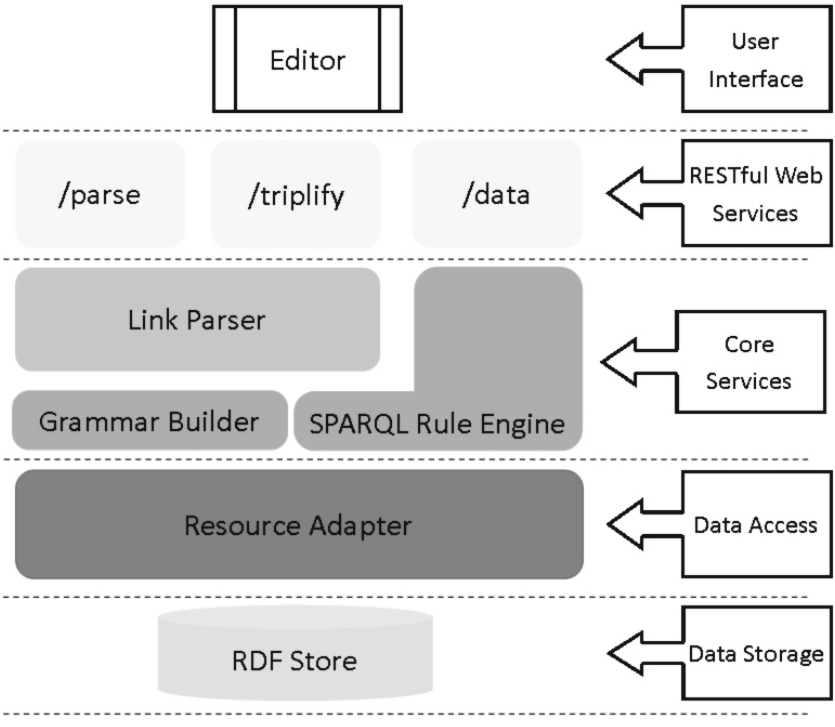


Fig. 5. CLANN Architecture

The architecture follows a layered approach where each layer interacts with the layers above and below it. This provides flexibility to the whole design, making it easy to change parts of it as required. The User Interface Layer consists of the CLANN Editor interface which was implemented as a web-based editor pluggable into Semantic Mediawiki. This Editor accesses the core business logic through a set of RESTFul Web services. RESTful web services allow for stateless processing of the sentences. The Core services include a Link Parser module, A SPARQL Rule Engine and Grammar Builder Module.

The Link Parser module is responsible for generating an intermediate representation, MLINK, of the parse result for a given sentence as input. MLINK (Meta-Link) is an RDF vocabulary developed to model the grammatical constructs of a controlled language parse. An MLINK representation of sentence similar to the one illustrated in Section 2.2 is provided below.

```
Pradeep to submit a paper[is a Document].
```

The above sentence should be able to return the following MLINK data as an intermediate representation.

```
:Sentence4567 :hasRootNode [rdf:type :TextNode;
      :hasText submit;
      :hasSubType Verb;
      :hasInfinitive [ rdf:type :TextNode;
          :hasText to];
      :hasObject [rdf:type :TextNode;
          :hasText paper;
          :hasSubType Noun;
          :hasDeterminer [ rdf:type :TextNode;
              :hasText a;
              :hasSubType Article]];
          :hasSnippet [ rdf:type :ClassSnippet;
              :containsClass foaf:Document]
      :hasSubject [ rdf:type :TextNode;
          :hasText Pradeep;
          :hasSubType Person;
          ].
```

This representation is, essentially, an RDF representation of the Link parse structure for the sentence mentioned above. This allows for flexibility with the kind of grammar formalisms used. We currently use the Link grammar formalism, to represent our grammar rules. However a different parser can be applied if the parse structure of the new formalism is mapped to MLINK. This removes the dependency of CLANN on Link Grammar.

4 Related Work

A plethora of tools exist for the manual or (semi-)automatic semantic annotation of free text. To our knowledge, however, very little research exists involving the application of CNL to semantic annotation. Our related work focuses on controlled natural languages for knowledge creation but for a thorough survey of manual and (semi-)automatic semantic annotation tools and platforms, we refer the reader to [8]. In addition, the idea of *latent annotation*, blurring the lines between authoring and annotation, has its origins in [9] as part of the CREAM(CREAting Metadata) framework for (semi-automatic)semantic annotation. However, the implementation is simplistic and implies dragging an RDF Label for a given concept from the ontology viewer and essentially pasting the label into the document.

"Controlled Natural Languages (CL)s are subsets of natural language whose grammars and dictionaries have been restricted in order to reduce or eliminate both ambiguity and complexity" [10]. They have also found favour in large multinational corporations, usually within the context of machine translation and machine-aided translation of user documentation [10,11].

The application of CNLs for ontology authoring and instance population is an active research area[12]. *Attempto Controlled English*[19] (ACE) [13], is a popular CNL for ontology authoring. It is a subset of standard English designed for

[19] http://www.ifi.unizh.ch/attempto/

knowledge representation and technical specifications, and is constrained to be unambiguously machine-readable DRS - Discourse Representation Structure, a form of first-order logic. It can also be re-targeted to other formal languages[14]. The Attempto Parsing Engine (APE) consists principally of a definite clause grammar, augmented with features and inheritance and is written in Prolog [15]. ACE OWL, a sub language of ACE, proposes a means of writing formal, simultaneously human- and machine-readable summaries of scientific papers [16,17].

ACEView is a plugin for the Protégé editor[20] [18]. It empowers Protégé with additional interfaces based on the ACE CNL in order to create, browse and edit an ontology. The user can also query the ontology using ACE questions to access newly asserted facts from the knowledge base. A recent development with respect to ACE is the translation of a complete collection of pediatric guideline recommendations into ACE[19].

The Rabbit CNL is a another well known implementation[20]. It is similar to CLOnE in its implementation but is much more powerful with respect to grammar expressiveness and ontology authoring capabilities. Rabbit was developed by the national mapping agency of Great Britain - Ordnance Survey. Rabbit can be converted in OWL[21] to provide natural language support for ontology authoring. OWL development is not the primary objective of Rabbit and not all Rabbit expressions can be mapped into OWL. It is primarily a vehicle for capturing, representing and communicating knowledge in a form that is easily understood by domain experts.

In [21], the authors undertake a paraphrase-based evaluation to assess whether domain experts without any ontology authoring development can author and understand declaration and axiom sentences in Rabbit. The experiment included 21 participants from the ordnance survey domain and a Rabbit language expert. The participants were given a text that describes a fictional world and were asked to make knowledge statements which were then compared to equivalent statements created by the Rabbit expert. The sentences produced by non-experts were analyzed for correctness (with regard to the knowledge captured) by independent experts and were compared to those produced by the Rabbit expert. Interestingly, on average 51% of the sentences generated at least one error. Furthermore, the most common error was the omission of the quantifier "every" at the beginning of a sentence. This observation was of statistical significance. Other user errors included: confusing instances with subclass declarations, a tendency to omit intensional information as well difficulties modeling knowledge under the open world assumption. The work of [20] and [21] is important in the context of CNL evaluation in that we see the advent of the paraphrase-based approach to evaluating CNL's.

Other work involves integrating Rabbit(as well as support for ACE) into Semantic Media Wiki[22], the purpose of which is user friendly collaborative

[20] http://protege.stanford.edu/

[21] http://www.w3.org/TR/owl-features/

[22] More information about Semantic MediaWiki can be found at http://semantic-mediawiki.org/wiki/Help:Introduction_to_Semantic_MediaWiki

ontology authoring using multiple CNLs and template based language generation capabilities[22].

With respect to evaluation frameworks, Kuhn [23] describes an evaluation framework for CNLs based on *Ontographs*. Ontographs are a graphical notation to enable tool independent and reliable evaluation of the human understanding of a given knowledge representation language. The author categorises CNLs evaluations into (1)task-based, whereby users are provided with a specific task to complete and (2) paraphrase-based which are are concerned with testing the understandability of the CNL. Ontographs serve as a common basis for testing and comparing the understandability of two different formal languages and facilitate the design of tool-independent and reliable experiments. The author claims that Ontographs are simple and intuitive. They are useful for representing simple logical forms but they do not cater for functions and are restricted to unary and binary predicates. In short, Ontographs serve to test the relative understanding of the core logic for two different formal languages.

A recent addition to the CNL field is GF - Grammatical Framework[24] and [25], which is an implementation framework which the authors claim can cope with a variety of CNLs as well as boost of the development of new ones. In their paper, the authors reverse engineer ACE for GF in order to demonstrate how portable CNLs are to the GF framework as well as how CNLs can be targeted to other natural languages. In this case ACE is ported from English to five other natural languages. In short, the core advantage of GF is its multilingualism in that its primary task is domain specific knowledge based Machine Translation (MT) of controlled natural languages. GF follows the functional programming paradigm and began as an experimental system in 1998 at XEROX Europe[23] [26]. The GF framework uses a logical framework based on Martin Loef's type theory[27] for building semantic models of languages. It adds a syntax formalism to the logical framework which defines realizations of formal meanings as concrete linguistic expressions. The semantic model is called the *abstract syntax* while the syntactic realization functionality is called *concrete syntax*. A substantial amount of linguistic competence and domain expertise is needed to define a concrete syntax for a given source/target language. Consequently the authors developed a collection of GF resource libraries to provide a language engineering solution to this issue. The GF libraries now contain a collection of wide coverage grammars for over 15 natural languages. One could view GF as a general framework for developing and extending controlled languages, similar to NLP architectures such as GATE.

The most closely related technologies our work specifically the CNL snippets are semantic wikis, which have become a somewhat popular way of adding semantics to user generated wiki pages. The term semantic wiki often implies either ontology authoring or the semantic annotation of wiki content. A traditional wiki creates links between pages without defining the kind of linkage between pages. Semantic Media Wiki[28] allows a user to define the links semantically, thereby adding meaning to links between pages. Each concept or an instance has

[23] http://www.xrce.xerox.com/

a page in Semantic Media Wiki(SMW), and each outgoing link from this page is annotated with well-defined properties as links.

With respect to user evaluation,[29] describe observations regarding SMW usage. They state first and foremost that the 'majority of users will neglect annotation as it does not bear immediate benefit". This is understandable given any annotation context, whereby the benefits of annotation are not recognized until the semantic search stage. In addition, they argue that "without conclusive studies on the usage of wikis in general, any prediction on the effect of introducing semantics in the (wikipedia) environment lacks justifications". In [29], the authors base their wiki usage experiments on `ontoworld.org`, which is itself maintained by the authors. The site's function is to collect information about semantic web researchers, events and projects. The authors record 930 registered users, the majority of which have contributed little to the total recorded 37,880 edits. The semantic knowledge base of `ontoworld.org`, at the time, counted 17,562 property annotations for 808 property edits. The majority of the properties have a page in the wiki, while 50% are of type `Page` and are used to annotate hyper-links. With respect to the usage of properties, they noted that 5% of the properties accounted for over 74% of the annotations, whereas the least used 80% of the properties accounted for less than 9% of all semantic statements. The authors state that these results have very similar power-law distributions to those of Wikipedia's categories[29]. In conclusion, they argue that SMW features are at least equivalent to Media WiKi functionality, however as the authors themselves state "one cannot conclude whether or not the requested (annotation)functions are actually considered useful for a given purpose" and that additional research is needed to obtain definitive results. While the research is very important in that it records observations with respect to SMW usage over a large user populations, one cannot conclude any specific user satisfaction rating with respect to the users and further more the user group is arguably extremely biased to that of the semantic web community.

Recent work by Pfisterer et al [30] reports better results, but it is in the context of the interface extensions to SMW by AIFB[24] in collaboration with Ontoprise GmbH[25]. . The interface enhancements include: (1) a factbox which summarises all facts, linked to a given article, (2) a semantic query interface with strong auto-completion features, (3) an ontology browsing interface and (4) a Semantic Tool-Bar, which seeks to ease the semantic annotation process, but a classical a posteriori fashion and not at the editing/authoring stage. The Semantic Tool-Bar, which is enabled by software derived from the HALO[26] project, allows users to add/change annotations, whereby the changes are written directly to the wiki source text. They conducted two evaluations, whereby the shared scenario is the creation of a scientific Semantic Wikipedia. They recruited seven test subject

[24] Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB) `http://www.aifb.kit.edu`

[25] `http://www.ontoprise.de/`

[26] `http://www.projecthalo.com/`

matter experts consisting of two experts in physics, three in chemistry and two in biology. The experts had little or no familiarity with semantic wikis and had never edited a Wiki article before the evaluation. They participated in the design and development of the enhanced SMW over a period of seven months. With respect to both evaluations, they are more aligned to usability testing rather than being able to make and statistically significant inferences about the general target user population regarding the usability of their Wiki. A SUS[31] questionnaire was administered to the group, once before and once after user feedback had been integrated into the enhanced Wiki. The user satisfaction was low prior to the enhancements and high, upon re-administration of the questionnaire. The sample size, at seven, was too small to make any statistically significant claims. More importantly, as the authors note, there are flaws in the evaluation in that a portion of first questionnaire group intersected with the second questionnaire group. This portion had been exposed to the enhanced SMW for a few months, so the high user satisfaction observed is inaccurate.

With respect to the second of the two aforementioned user evaluations, forty-two students from an introductory human-computer interaction class served as the sample user population. Each subject, after being provided introductory material, were asked to annotate a random wiki page in the enhanced SMW. In addition, after completing this task they were asked to formulate a number of queries to the SMW. This was followed by a SUS usabily questionnaire. The resulting user satisfaction score was below the SUS baseline at 54.8%. In addition, on average each student created 4.2 annotations and only 50.3% were fully correct. Errors were caused primarily by unrecognized characters or date formats, which are rectifiable. The speed of the systems reaction accounted for a large amount of negative feedback. However, performance limitations in speed were undoubtably caused by over 20 users editing the SMW at the same time. This would invariably have had an impact on user satisfaction scores. Nevertheless, no statistical tests are performed on the SUS results and no inferences are made about the general target population. Despite the weak empirical results, the work presented in [30] is very important in that in represents a shift towards proper user evaluation and user centered design within the the semantic wiki community. The authors themselves acknowledge that there is still a need to provide more "concrete examples" with respect to application of user centered design to semantic wikis.

Other flavours of semantic wikis include IkeWiki[27] [32] and KiWi[28]. One could argue that Semantic Wikis are only usable by the Semantic Web community and retain a significant formal barrier to a casual user or even an IT professional in the industry. ACEWiki[33] attempts to circumvent this using the CNL ACE in combination with a predictive editor as an interface to a Semantic Wiki. However the task here is collaborative ontology authoring and not annotation, whereby we seeks to provide wiki content with a semantic backbone.

[27] http://ikewiki.salzburgresearch.at/
[28] http://www.kiwi-project.eu/

5 Conclusion

In conclusion, we have set out to explore the possibility of using controlled languages as interfaces to semantic web applications. We decided to narrow down the domain to meeting minutes and status reports, and designed an ontology representing the domain. We have developed the CLANN system for knowledge acquisition along with the CLANN grammar, the PDO ontology,a smart CNL text editor for novice users and packaging the CLANN module as an extension to the Semantic MediaWiki. Future work will involve performing a user-based evaluation on our work against other semantic wikis, to judge the feasibility of this approach.

Acknowledgments. The work presented in this paper has been funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Líon-2).

References

1. Davis, B., Dantuluri, P., Dragan, L., Handschuh, S., Cunningham, H.: On Designing Controlled Natural Languages for Semantic Annotation. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 187–205. Springer, Heidelberg (2010)
2. Kaufmann, E.: Talking to the semantic web: natural language query interfaces for casual end-users. PhD thesis, Universität Zürich (2009)
3. Sleator, D.D.K., Sleator, C.F.D., Temperley, D.: Parsing english with a link grammar. In: Third International Workshop on Parsing Technologies (1991)
4. Funk, A., Tablan, V., Bontcheva, K., Cunningham, H., Davis, B., Handschuh, S.: CLOnE: Controlled Language for Ontology Editing. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 142–155. Springer, Heidelberg (2007)
5. Decker, S.: The social semantic desktop: Next generation collaboration infrastructure. Information Services and Use 26(2), 139–144 (2006)
6. Fernández-López, M., Gómez-Pérez, A., Juristo, N.: Methontology: from ontological art towards ontological engineering. In: Proceedings of the AAAI 1997 Spring Symposium, Stanford, USA, pp. 33–40 (March 1997)
7. Cunningham, H., Maynard, D., Tablan, V.: JAPE: a Java Annotation Patterns Engine, 2 edn., Research Memorandum CS–00–10, Department of Computer Science, University of Sheffield (November 2000)
8. Uren, V.S., Cimiano, P., Iria, J., Handschuh, S., Vargas-Vera, M., Motta, E., Ciravegna, F.: Semantic annotation for knowledge management: Requirements and a survey of the state of the art. J. Web Sem. 4(1), 14–28 (2006)
9. Handschuh, S., Staab, S.: Authoring and annotation of web pages in cream. In: WWW, pp. 462–473 (2002)
10. Schwitter, R.: Controlled natural languages. Technical report, Centre for Language Technology, Macquarie University (June 2007)
11. Adriaens, G., Schreurs, D.: From COGRAM to ALCOGRAM: Toward a controlled English grammar checker. In: Conference on Computational Linguistics (COLING 1992), Nantes, France, pp. 595–601 (1992)

12. Smart, P.R.: Controlled natural languages and the semantic web. Technical report, School of Electronics and Computer Science, University of Southampton (2008) (unpublished)
13. Fuchs, N., Schwitter, R.: Attempto Controlled English (ACE). In: CLAW 1996: Proceedings of the First International Workshop on Controlled Language Applications, Leuven, Belgium (1996)
14. Fuchs, N.E., Kaljurand, K., Kuhn, T., Schneider, G., Royer, L., Schröder, M.: Attempto Controlled English and the semantic web. Deliverable I2D7, REWERSE Project (April 2006)
15. Hoefler, S.: The syntax of Attempto Controlled English: An abstract grammar for ACE 4.0. Technical Report ifi-2004.03, Department of Informatics, University of Zurich (2004)
16. Kaljurand, K., Fuchs, N.E.: Bidirectional Mapping Between OWL DL and Attempto Controlled English. In: Alferes, J.J., Bailey, J., May, W., Schwertel, U. (eds.) PPSWR 2006. LNCS, vol. 4187, pp. 179–189. Springer, Heidelberg (2006)
17. Kuhn, T.: Attempto Controlled English as ontology language. In: Bry, F., Schwertel, U. (eds.) REWERSE Annual Meeting 2006 (March 2006)
18. Kaljurand, K.: ACE View — an ontology and rule editor based on Attempto Controlled English. In: 5th OWL Experiences and Directions Workshop (OWLED 2008), Karlsruhe, Germany, October 26-27, 12 pages (2008)
19. Shiffman, R.N., Michel, G., Krauthammer, M., Fuchs, N.E., Kaljurand, K., Kuhn, T.: Writing Clinical Practice Guidelines in Controlled Natural Language. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 265–280. Springer, Heidelberg (2010)
20. Hart, G., Johnson, M., Dolbear, C.: Rabbit: Developing a Control Natural Language for Authoring Ontologies. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 348–360. Springer, Heidelberg (2008)
21. Engelbrecht, P., Hart, G., Dolbear, C.: Talking Rabbit: A User Evaluation of Sentence Production. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 56–64. Springer, Heidelberg (2010)
22. Bao, J., Smart, P., Braines, D., Shadbolt, N.: A controlled natural language interface for semantic media wiki using the rabbit language. In: Workshop on Controlled Natural Language (CNL 2009) (March 2009)
23. Kuhn, T.: An Evaluation Framework for Controlled Natural Languages. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 1–20. Springer, Heidelberg (2010)
24. Angelov, K., Ranta, A.: Implementing Controlled Languages in GF. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 82–101. Springer, Heidelberg (2010)
25. Ranta, A.: Grammatical Framework: A Type-Theoretical Grammar Formalism. Journal of Functional Programming 14(02), 145–189 (2004)
26. Dymetman, M., Lux, V., Ranta, A.: Xml and multilingual document authoring: convergent trends. In: Proceedings of the 18th Conference on Computational Linguistics, vol. 1, pp. 243–249. Association for Computational Linguistics, Morristown (2000)
27. Nordstrom, B., Petersson, K., Smith, J.M.: Programming in Martin-Löf's Type Theory: An Introduction. Oxford University Press, USA (1990)
28. Krötzsch, M., Vrandečić, D., Völkel, M.: Semantic MediaWiki. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 935–942. Springer, Heidelberg (2006)
29. Krötzsch, M., Vrandečić, D., Völkel, M., Haller, H., Studer, R.: Semantic Wikipedia. Journal of Web Semantics 5(4), 251–261 (2007)

30. Pfisterer, F., Nitsche, M., Jameson, A., Barbu, C.: User-Centered Design and Evaluation of Interface Enhancements to the Semantic MediaWik. In: Proceedings of Semantic Web User Interaction at CHI 2008: Exploring HCI Challenges. CEUR Workshop Proceedings (2008)
31. Brooke, J.: SUS: a "quick and dirty" usability scale. In: Jordan, P., Thomas, B., Weerdmeester, B., McClelland, A. (eds.) Usability Evaluation in Industry. Taylor and Francis, London (1996)
32. Schaffert, S.: Ikewiki: A semantic wiki for collaborative knowledge management. In: 1st International Workshop on Semantic Technologies in Collaborative Applications (STICA 2006) (2006)
33. Kuhn, T.: AceWiki: Collaborative Ontology Management in Controlled Natural Language. In: Proceedings of the 3rd Semantic Wiki Workshop. CEUR Workshop Proceedings (2008)

First-Order Reasoning for Attempto Controlled English

Norbert E. Fuchs

Department of Informatics & Institute of Computational Linguistics
University of Zurich
fuchs@ifi.uzh.ch
http://attempto.ifi.uzh.ch/

Abstract. RACE is a first-order reasoner for Attempto Controlled English (ACE) that can show the (in-) consistency of a set of ACE axioms, prove ACE theorems from ACE axioms and answer ACE queries from ACE axioms. In each case RACE gives a proof justification in ACE and full English. This paper is a system description of RACE sketching its structure, its implementation, its operation and its user interface. The power and the limitations of RACE are demonstrated and discussed by concrete examples.

Keywords: controlled natural language, Attempto Controlled English, ACE, first-order reasoning, RACE.

1 Introduction

Attempto Controlled English (ACE)[1] is a logic-based knowledge representation language that uses the syntax of a subset of English. ACE allows domain specialist to represent formal knowledge in familiar English without having to resort to visibly formal languages. Working with ACE is supported by a number of tools[2], foremost by the ACE Parsing Engine (APE)[3] that translates an ACE text into a discourse representation structure (DRS) that is expressed in a variant of the standard language of first-order logic.

The DRS can be translated into other logic languages, for instance into the standard and the clausal forms of first-order logic, into the TPTP[4] notation, and – with some syntactic restrictions – into the semantic web languages OWL and SWRL[5].

Once you have knowledge expressed as an ACE text you may want to reason with this knowledge, for instance to show its consistency. The large variety of logical representations of an ACE text allows you to use existing reasoning tools for this purpose. The translation of an ACE text into the TPTP notation gives you access to all reasoning tools provided by the TPTP system. In fact, the TPTP web-interface already

[1] http://attempto.ifi.uzh.ch/
[2] http://attempto.ifi.uzh.ch/site/tools/
[3] web-interface of APE: http://attempto.ifi.uzh.ch/ape/
[4] http://www.cs.miami.edu/~tptp/
[5] http://www.w3.org/TR/owl2-overview/, http://www.w3.org/Submission/SWRL/

M. Rosner and N.E. Fuchs (Eds.): CNL 2010, LNAI 7175, pp. 73–94, 2012.

accepts input in ACE. The translation of an ACE text into OWL and SWRL makes all theorem provers for these languages available. This approach is taken by Kaljurand's ontology editor ACE View[6] and by Kuhn's semantic wiki AceWiki[7] both of which also provide a back translation of the reasoning results into ACE.

This paper describes the reasoner RACE that allows users to show the (in-) consistency of an ACE text, to deduce one ACE text from another one, and to answer ACE queries from an ACE text.

RACE stands in the long tradition that investigates the relation between natural language, reasoning and logic – a tradition that started in ancient Greece, continued during the European middle ages, and thrived especially vigorously after the formalisation of logic and the invention of the digital computer. Note, however, that this paper is a system description of RACE. Thus the relation between natural language and logic will be alluded to occasionally, but will not be discussed in depth. Also you will not learn much about the state of the art of natural language processing and of theorem proving. Nor will this paper introduce you to the language Attempto Controlled English beyond what is needed to demonstrate features of RACE. To learn more about ACE refer to the relevant documentation[8].

The rest of this paper is organised as follows. In section 2, I will summarise general features of RACE. Section 3 very briefly shows how one works with RACE via its web-client. Section 4 gives a rather detailed look under the hood of RACE. In section 5 I investigate how RACE handles some typical proofs and how RACE processes specific ACE constructs. Section 6 discusses questions like decidability, termination, looping and efficiency. Section 7 concludes with a discussion of RACE's strengths and limitations, and with a preview of further research.

2 General Features of RACE

RACE has the following general features:

- RACE offers consistency checking, textual entailment and query answering of ACE texts.
- Conforming to the goal of the Attempto project – provide formal methods clad in (controlled) English – RACE does not presuppose any knowledge of formal logic or theorem proving, does not require users to understand RACE's workings, nor does it require users to control the reasoning process.
- All input is in ACE, all output is in ACE and full English.
- Consistency checking: For inconsistent ACE axioms RACE will list all minimal subsets of the axioms that lead to inconsistency.
- Textual entailment and query answering: If the ACE axioms entail the ACE theorems, respectively ACE queries, RACE will list all minimal subsets of the axioms that entail the theorems, respectively queries.

---

[6] http://attempto.ifi.uzh.ch/aceview/

[7] http://attempto.ifi.uzh.ch/acewiki/, https://launchpad.net/acewiki/

[8] http://attempto.ifi.uzh.ch/site/docs/

- Textual entailment and query answering: If the ACE axioms do not entail the ACE theorems, respectively ACE queries, RACE will list all ACE words and ACE language constructs of the theorems, respectively queries, that could not be entailed.
- Textual entailment and query answering: If the ACE axioms are inconsistent, RACE outputs a warning message, and reports the results found so far.
- Non-termination: ACE is a superset of a fragment of English that [5] proved to be undecidable. For undecidable problems RACE terminates with a time-out.
- RACE covers the first-order subset of ACE, that is all ACE constructs with the exception of imperative sentences, negation-as-failure and the modal operators *may* and *should*. Currently RACE does not yet cover arithmetic, formulas and operations on lists, sets and strings.

3 Working with RACE

RACE is implemented as a Prolog program and can be accessed remotely via its web-client[9] or via its web-service[10]. Both offer interfaces for consistency checking, textual entailment and query answering. For convenience we will use screen-shots of the web-client to give a first impression how to work with RACE.

The consistency checking of figure 1 shows that the minimal subset {*Every man is a human., John is a man., John is not a human.*} of the axioms leads to inconsistency.

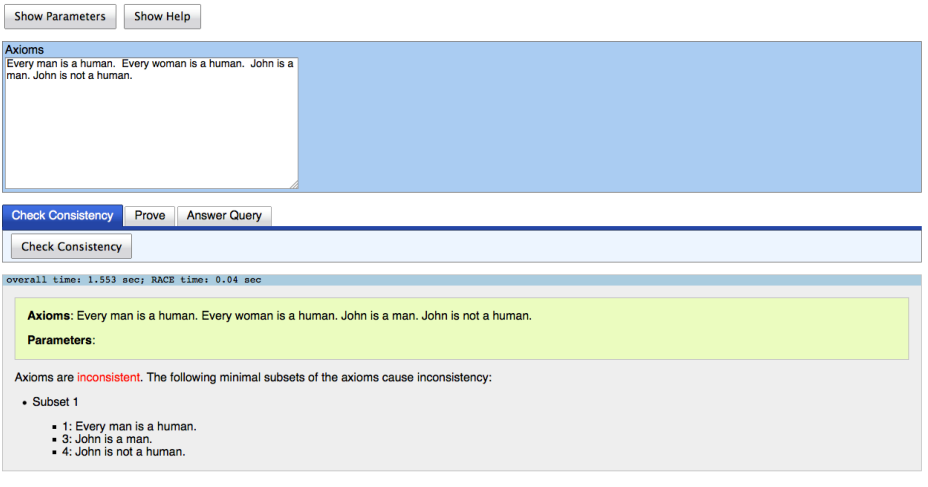


Fig. 1. Consistency checking

In the following I will restrict the screen-shots to the result window since it also contains the input. Figure 2 shows that the theorem *There is a human.* is entailed by two minimal subsets of the axioms, namely {*Every man is a human.*, *John is a man.*} and {*Every woman is a human.*, *Mary is a woman.*}.

[9] http://attempto.ifi.uzh.ch/race/

[10] http://attempto.ifi.uzh.ch/ws/race/racews.perl

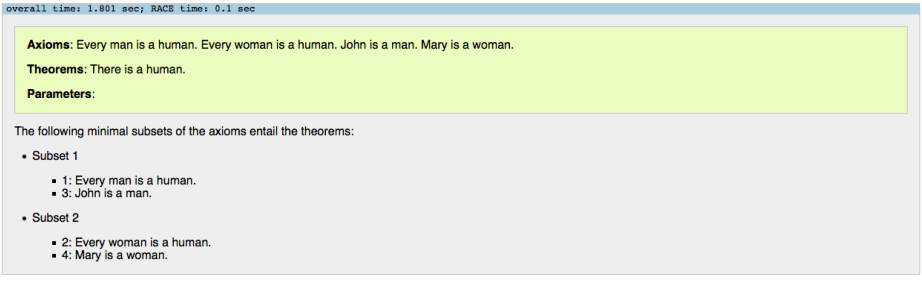


Fig. 2. Textual entailment

Figure 3 contains a case of query answering. The results show that the query *Who tires how?* can be answered from the axioms {*If John waits then he tires easily.*, *John waits.*} Also, the query word *who* is substituted by the proper name *John,* and the query word *how* by the positive form of the adverb *easily.* Note that the substitution refers to *how/when/where* since these three query words are treated as equivalent.

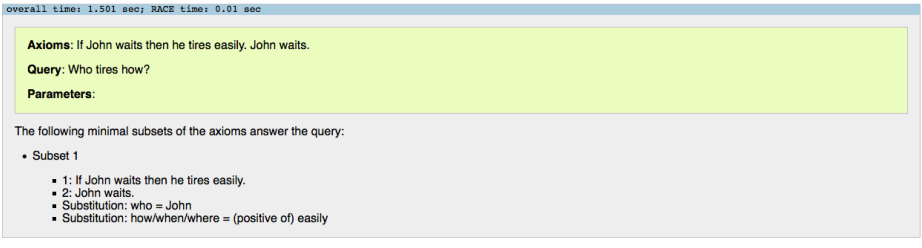


Fig. 3. Query answering

As we have seen, RACE answers for succeeding proofs the question "Why?" by listing the axioms that are inconsistent or that are needed to prove a theorem or answer a query. For failing proofs RACE answers the question "Why Not?" by listing the ACE words and ACE constructs of the theorem or query that could not be proved. Figure 4 shows an example.

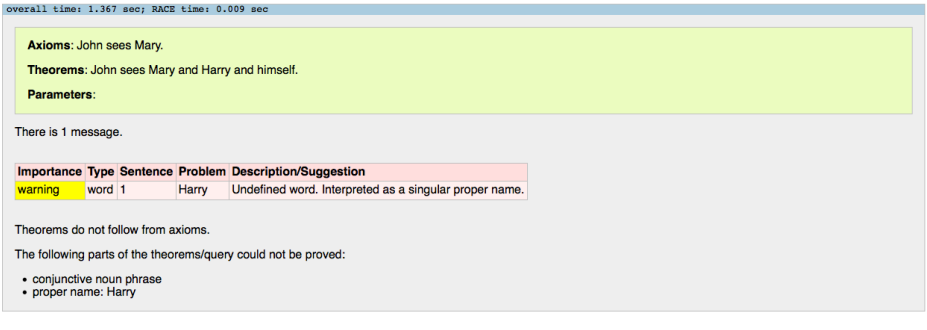


Fig. 4. "Why Not?" answer for a failing proof

The proof failed, and RACE list the proper name *Harry* and the ACE construct *noun phrase conjunction* as not provable. Since the proper name *Harry* was not found in the ACE lexicon, a warning is generated that *Harry* was automatically interpreted as proper name.

4 A Look under the Hood of RACE

Satchmo. RACE is implemented as a Prolog program on the base of the model generator Satchmo [1]. Satchmo was chosen since it is available as a small Prolog program (see figure 5) that lent itself to be locally modified and extended to provide RACE's functionality. As will be seen in the next sections, RACE is not simply a wrapper of Satchmo since the modifications applied concern also core functionality of Satchmo.

```
satisfiable :-
  setof(Clause, violated_instance(Clause), Clauses),
  !,
  satisfy_all(Clauses),
  satisfiable.
satisfiable.

violated_instance((B ---> H)) :-
  (B ---> H),
  B,
  \+ H.

satisfy_all([]).
satisfy_all([(_B ---> H) | RestClauses]) :-
  H,
  !,
  satisfy_all(RestClauses).

satisfy_all([(_B ---> H) | RestClauses]) :-
        satisfy(H),
  satisfy_all(RestClauses).

satisfy((A;B)) :-
  !,
  (satisfy(A) ; satisfy(B)).

satisfy(Atom) :-
  \+ Atom = fail,
  assume(Atom).

assume(Atom) :-
  asserta(Atom).

assume(Atom) :-
  retract(Atom),
  !,
  fail.
```

Fig. 5. Satchmo model generator [1]

Satchmo works with first-order clauses of the form *Body* ---> *Head* where *Body* is *true* or a conjunction of logical atoms, and *Head* is *fail* or a disjunction of logical atoms. There is not explicit negation, instead one uses an implication to *fail*.

Satchmo executes the clauses by forward-reasoning. Once the *Body* of a clause can be proved from the Prolog database, then its *Head* is asserted to the Prolog data base if not already there. If the *Head* of a clause is *fail* then Satchmo backtracks, retracting previously asserted atoms from the Prolog database.

Satchmo generates a minimal finite Herbrand model of the clauses – if finite models exist [2]. If the model of the clauses is infinite then Satchmo loops. If the clauses are unsatisfiable, Satchmo just fails. Satchmo is correct for unsatisfiablity if the clauses are range-restricted, i.e. all variables of *Head* occur already in *Body* [1]. Satchmo is complete for unsatisfiability if it is used level-saturated, i.e. clauses are tried bottom-up, level by level [1].

Satchmo is a very efficient program. Its efficiency can be enhanced in various ways [1], most effectively when first-order clauses are replaced by Prolog clauses that are directly executed without incurring the overhead of forward-reasoning.

From Satchmo to RACE. The modifications and extension applied to Satchmo in order to achieve RACE's functionality resulted in a program that is two orders of magnitude larger than Satchmo. All modifications and extensions of RACE take into account the preservation of Satchmo's theoretical attributes – correctness, completeness, minimal model generation – though this has not yet been proven.

Here are the similarities and the main differences between Satchmo and RACE.

- Like Satchmo RACE works with first–order clauses.
- For satisfiable clauses both Satchmo and RACE generate minimal finite Herbrand models – if they exists.
- For infinite models Satchmo loops, while RACE stops with a time-out.
- For unsatisfiable clauses Satchmo stops immediately when it detects unsatisfiability, while RACE finds all minimal unsatisfiable subsets of the clauses.
- While Satchmo works on clauses found in the Prolog data base and stores the model of the clauses as Prolog facts, RACE translates ACE axioms, theorems, and questions into clauses and outputs its results in ACE and full English.
- While Satchmo just succeeds or fails indicating that the clauses are satisfiable or not, RACE generates for each sucessful proof a report showing which minimal subsets of the ACE axioms are inconsistent, respectively entail the ACE theorems or queries. For a failing proof RACE will list those ACE text fragments of the theorems or queries that could not be proved.

In light of RACE's additional functionality and increased size it is obvious that it cannot be as efficient as Satchmo. Concerning RACE's efficiency see section 6.

Structure and Operation of RACE. To best understand the structure and the operation of RACE we will consider a concrete example.

Given the ACE text T
There is a cat. Every cat is an animal.
answer the ACE query Q
Is a cat an animal?

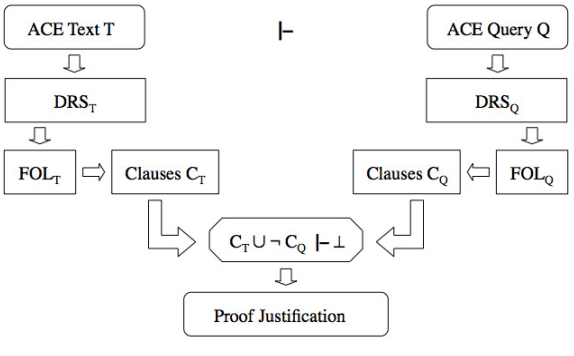


Fig. 6. RACE answers an ACE query Q from an ACE text T

The query answering is summarised in Figure 6. In a first step both the ACE text T and the ACE query Q are submitted to the Attempto Parsing Engine APE that translates them into the Discourse Representation Structures DRS_T, respectively DRS_Q.

```
DRS_T:
[A]
 object(A,cat,countable,na,eq,1)-1/4
    [B]
    object(B,cat,countable,na,eq,1)-2/2
    =>
    [C,D]
    object(C,animal,countable,na,eq,1)-2/5
    predicate(D,be,B,C)-2/3

DRS_Q:
[]
    QUESTION
    [A,B,C]
    object(A,cat,countable,na,eq,1)-1/3
    object(B,animal,countable,na,eq,1)-1/5
    predicate(C,be,A,B)-1/1
```

These are pretty-printed versions of the DRSs internally represented as Prolog terms. The language of the DRSs – described in Fuchs et al. [7] – uses a compact, reified variant of the standard language of first-order logic. Countable noun phrases like *a cat* are represented as `object(Ref,Noun,countable,na,Operator,Count)` where `object/6` is a predefined predicate, `Ref` is a quantified variable called discourse referent, `Noun` stands for the represented noun, `countable` classifies the noun as a countable common noun, `na` is a place-holder otherwise used for measurement nouns, and `Operator` and `Count` express the cardinality. Transitive predicates like the transitive form of *to be* are represented as `predicate(Ref,Verb,Ref1,Ref2)` where `predicate/3` is a predefined predicate, `Ref` is a discourse referent, `Verb` stands for the represented transitive verb, and `Ref1` and `Ref2` are the discourse referents of two `object/6` definitions. The structure `S/T` attached to each logical atom indicates that the atom was derived from token `T` in ACE sentence `S`. The sentence indices `S` are later used by RACE to label individual axioms, theorems and queries.

In a next step the two DRSs are individually translated into the standard language of first-order logic, and we get FOL_T and FOL_Q.

FOL_T:

```
exists(A,&(object(World,A,cat,countable,na,geq,1)-axiom(1),forall(B,
=>(object(World,B,cat,countable,na,geq,1)-axiom(2),exists(C,exists(D,
&(object(World,C,animal,countable,na,geq,1)-axiom(2),
predicate(World,D,be,B,C)-axiom(2)))))))
```

FOL_Q:

```
exists(A,exists(B,exists(C,&(object(World,A,cat,countable,na,geq,1)-
theorem(1),&(object(World,B,animal,countable,na,geq,1)-theorem(1),
predicate(World,C,be,A,B)-theorem(1))))))
```

During the translation each logical atom received an additional argument `World` that stands for a possible world needed to process modality within first-order logic.

Operators `eq` were replaced by `geq` to express the meaning of, for instance, *a cat* as *at least one cat*. Also the sentence indices of the logical atoms have been wrapped by `axiom/1`, respectively `theorem/1` to distinguish axioms from theorems.

In a final step the formula `exists(World, FOL_T & ¬ FOL_Q)` is translated into clauses using a variant of the standard clausification procedure. The general form of the clauses used by RACE is

```
satchmo_clause(Body, Head, Index)
```

where – as in Satchmo – *Body* is *true* or a conjunction of logical atoms and *Head* is *fail* or a disjunction of logical atoms. Negation is expressed as implication to *fail*. *Body* implies *Head. Index* is a list containing the term *axiom(N)* or *theorem(N)* where *N* is the sentence index carried over from the DRS.

The three clauses derived from `exists(World, FOL_T & ¬ FOL_Q)` are

```
satchmo_clause(true, object(sk1, sk2, cat, countable, na, geq, 1), [axiom(1)])
satchmo_clause(object(sk1, D, cat, countable, na, EQ, Count), (object(sk1, sk3(D), animal, countable, na, EQ, Count), predicate(sk1, sk4(D), be_NP, D, sk3(D))), [axiom(2)])
satchmo_clause((object(sk1, A, cat, countable, na, geq, 1), object(sk1, B, animal, countable, na, geq, 1), predicate(sk1, C, be_NP, A, B)), fail, [theorem(1)])
```

During clausification further transformations were performed. The argument `World` – being existentially quantified – was skolemised. The arguments `be` were replaced by `be_NP` – the reason for which is explained in section 5. The operator and count arguments of the second clause – derived from *Every cat is an animal.* – were replaced by the variables `EQ`, respectively `Count`. This allows RACE to flexibly match the clause body to all values of operators and counts. Furthermore, clauses are checked for range-restriction, i.e. all variables in the head of a clause must already occur in its body.

Once we have the clauses we can call RACE's main predicate

```
satchmo(Clauses, Models, Inconsistencies)
```

where `Clauses` is a list of clauses, `Models` is a list of models of the clauses, and `Inconsistencies` is a list of list of indices of clauses that led to inconsistency.

The predicate `satchmo/3` executes clauses bottom-up by forward-reasoning. If the *Body* of a clause `satchmo_clause(Body, Head, Index)` is *true* or can be proved from the Prolog data base then *Head* is asserted to the Prolog database together with a list that contains the element of *Index* plus the sentence indices of all clauses that were used to prove *Body*. This amounts to building a proof-tree labelled with lists of indices. Range restriction ensures that all atoms added to the Prolog data base are ground. If *Head* is a disjunction – meaning that the proof-tree branches – then the asserted index list also contains an identification of the respective disjunctive branch. If *Head* is *false* then `satchmo/3` backtracks and removes atoms added previously to the Prolog data base. This also indicates that a branch of the proof-tree is closed, and the indices of the leaf of the closed branch are stored. Forward-reasoning ends when no clause can be executed. The result is either a set of ground atoms in the Prolog data base that constitute minimal

models of `Clauses` or an empty Prolog data base plus a set of closed branches. In the second case `satchmo/3` checks whether all branches of the proof-tree are closed and then collects in `Inconsistencies` the indices of the leaves of all closed branches. Thus for terminating proofs we have the following outcomes:

- `Clauses` are consistent: `Inconsistencies = []`
- `Clauses` are inconsistent: `Models = []`, `Inconsistencies ≠ []`

Tracing a Proof. Here is an annotated proof of our example displaying the assertions and retractions mentioned above. The argument `Clauses` of `satchmo/3` is the list of clauses derived before.

```
call: satchmo(Clauses, Models, Inconsistencies)

% satchmo_clause(true, object(sk1, sk2, cat, countable, na, geq, 1),
[axiom(1)]) can fire, yielding ...
Asserting Head and Indices:
object(sk1,sk2,cat,countable,na,geq,1),[axiom(1)]

% satchmo_clause(object(sk1, D, cat, countable, na, EQ, Count),
(object(sk1, sk3(D), animal, countable, na, EQ, Count), predicate(sk1,
sk4(D), be_NP, D, sk3(D))), [axiom(2)]) can fire, yielding ...
Asserting Head and Indices:
object(sk1,sk3(sk2),animal,countable,na,geq,1),[axiom(1),axiom(2)]
Asserting Head and Indices:
predicate(sk1,sk4(sk2),be_NP,sk2,sk3(sk2)),[axiom(1),axiom(2)]

% satchmo_clause((object(sk1, A, cat, countable, na, geq, 1),
object(sk1, B, animal, countable, na, geq, 1), predicate(sk1, C, be_NP,
A, B)), fail, [theorem(1)]) can fire, yielding ...
Asserting: closed_branch([axiom(1),axiom(2),theorem(1)])
Retracting Head and Indices:
predicate(sk1,sk4(sk2),be_NP,sk2,sk3(sk2)),[axiom(1),axiom(2)]
Retracting Head and Indices:
object(sk1,sk3(sk2),animal,countable,na,geq,1),[axiom(1),axiom(2)]
Retracting Head and Indices:
object(sk1,sk2,cat,countable,na,geq,1),[axiom(1)]

exit: satchmo(Clauses, [], [[axiom(1), axiom(2), theorem(1)]])
```

Since `Models=[]` and `Inconsistencies=[[axiom(1), axiom(2), theorem(1)]]` `satchmo/3` proved that the clauses derived from the two axioms and the theorem are inconsistent, meaning that the ACE axioms *There is a cat. Every cat is an animal.* answer the ACE query *Is a cat an animal?*.

Note that if the query could not have been answered, then the asserted logical atoms `Head` would not have been retracted, and would have constituted a minimal model of `Clauses`, respectively the ACE axioms.

Interface Predicates. RACE – for example its web-client – does not call `satchmo/3` directly. The translation of ACE texts into clauses, the call of `satchmo/3` and the back translation of `Inconsistencies` into the respective ACE texts are performed by three

interface predicates that take care of the peculiarities of consistency checking, theorem proving and query answering. These predicates are

- *check_consistency(Axioms, Parameters, Messages, RunTime, AxiomsUsed)* that checks the consistency of a set of ACE *Axioms* and returns a list of lists *AxiomsUsed* of all minimal inconsistent subsets of the *Axioms*.
- *prove(Axioms, Theorems, Parameters, Messages, RunTime, AxiomsUsed, WhyNot)* proves a set of ACE *Theorems* from a set of ACE *Axioms* and returns a list of lists *AxiomsUsed* that contains all minimal subsets of the *Axioms* needed to prove the *Theorems*.
- *answer_query(Axioms, Queries, Parameters, Messages, RunTime, AxiomsUsed, WhyNot)* answers a set of ACE *Queries* from a set of ACE *Axioms* and return a list of lists *AxiomsUsed* that contains all minimal subsets of the *Axioms* needed to answer the *Queries*. For *wh*-questions *AxiomsUsed* contains also the substitutions of the query words.

RACE's interface predicates have some common arguments. The argument *Parameters* is a list containing RACE's parameters. Currently, there is but one parameter *raw* used for testing. The argument *Messages* contains warning and error messages produced during a run of RACE. If there are error messages then RACE reports that the proof failed. The argument *RunTime* returns the overall run time in milliseconds. If *Axioms* cannot be proved or *Queries* cannot be answered then the argument *WhyNot* contains a list of ACE text fragments of the *Theorems* or *Queries* that RACE was unable to prove on the basis of the *Axioms*. For demonstration purposes here is the call of *answer_query/7* for our example.

```
answer_query('There is a cat. Every cat is an animal.','Is a cat an
animal?', [], Messages, RunTime, AxiomsUsed, WhyNot).
```

```
Messages = []
RunTime = 0
AxiomsUsed = [proof(['1: There is a cat.', '2: Every cat is an
animal.'], [])]
WhyNot = []
```

Since the identification of the minimal subsets of inconsistent axioms, or the identification of minimal subsets of axioms needed to entail a theorem or a query depends essentially on the threading and processing of sentence indices one could say that RACE uses a form of labelled deduction [3].

Auxiliary Axioms. RACE uses about 50 auxiliary axioms as meaning postulates for plurals and singulars, to handle generalised quantifiers, to interpret the copula *be* that in ACE is underspecified, for summation, for the substitution of query words, and for other purposes. Auxiliary axioms are implemented as Prolog predicates that access elements of the DRS representation of ACE texts. Being implemented in Prolog, the auxiliary axioms do not participate in the eager forward-reasoning of the clauses, but – called only on demand – are executed by lazy backward-reasoning. Auxiliary predicates are also labelled, and the RACE parameter *raw* – introduced for testing purposes – enables the output of the auxiliary axioms used together with the ACE axioms. Individual auxiliary axioms will be presented in the next section.

5 All Things Considered

In this section I will investigate in greater detail how RACE handles some typical proofs and how RACE processes specific language constructs of ACE. I will use RACE's interface predicates instead of the more space-consuming screen-shots of the web-client. Irrelevant argument substitutions will be suppressed.

Plurals. As Schwertel discusses in great detail in her thesis [8], English plural nouns are extremely complex. To eliminate some of this complexity, ACE offers only collective and distributive plurals. The sentence

```
Three men lift two pianos.
```

introducing the collective plurals *three men* and *two pianos* describes a complex lifting event involving a group of three men and a group of two pianos. The lifting event is underspecified concerning the number of individual lifting actions and the number of men and pianos involved in each lifting action. Compare this to the sentence

```
Each of three men lifts each of two pianos.
```

involving the two distributive plurals *each of three men* and *each of two pianos*. The lifting relation consists of six lifting actions each involving one of the three men and one of the two pianos. One could say that the lifting relation is fully specified as far as the number of lifting actions and the number of participants are concerned.

Deductions from Collective Plurals. Collective plurals like *three men* are ambiguous between the reading *at least 3 men* and *exactly three men* (= *at least three men and at most three men*). This ambiguity also affects the deductions that can be made from collective plurals. Given the attempted deductions

```
(1) Three men lift a piano. |?- A man lifts a piano.

(2) Three men see a piano. |?- A man sees a piano.
```

humans – on the basis of their world knowledge – would probably consider deduction (1) as incorrect and deduction (2) as correct.

The reasoner RACE does not have any world knowledge and thus needs to take recourse to syntactic means to enable/disable deductions from collective plurals. By default RACE interprets collective plurals like *three men* as *at least 3 men*. (Note that ACE also provides the explicit construct *at least 3 men*.) The alternative "exactly" reading is explicitly expressed as *exactly three men*. To enforce the expected deductive behaviour we can reformulate the deductions (1) and (2) as

```
(1') Exactly three men lift a piano. |/- Exactly one man lift a piano.

(2') Three men see a piano. |- A man sees a piano.
     (= At least three men see a piano.|- At least one man see a piano.)
```

Note that the determiners *a* and *one* are treated as synonyms, that is *a man* and *one man* get the same logical representation.

To perform deduction (2') RACE needs a meaning postulate relating the noun phrases *at least three men* and *at least one man*. To formulate this meaning postulate we rely on RACE's logical representations of noun phrases. As introduced in section 4, nouns are represented by the predefined predicate *object/7*, concretely

- *at least three men* as *object(World, A, man, countable, na, geq, 3)*
- *at least one man* as *object(World, B, man, countable, na, geq, 1)*

Thus *at least three men* and *at least one man* get the same representation with appropriate cardinalities. This allows us to state the relation between singulars and collective plurals of all countable nouns as the first-order formula

```
∀ A, Noun, N, M (object(World, A, Noun, countable, na, geq, N) ∧ N ≥ M
→
object(World, A, Noun, countable, na, geq, M))
```

where *Noun* stands for any noun, and N and M are the respective cardinalities. RACE expresses this formula as the auxiliary Prolog axiom `cd5`

```
prolog_axiom(object(World,A,Noun,countable,na,geq,M), Block,
[prolog_axiom(cd5)|Indices]) :-
 nonvar(M),
 exists_asserted_atom(object(World,A,Noun,countable,na,geq,N), Indices),
 N>=1,
 M<N.
```

The logical atom `object(World, A, Noun, countable, na, geq, M)` can be proved and gets the index `[prolog_axiom(cd5)|Indices]` if the logical atom `object(World, A, Noun, countable, na, geq, N)` with the index `Indices` can be found in the Prolog data base, and the conditions `N>=1` and `M<N` are fulfilled. The argument `Block` is used to block unwanted combinations of auxiliary axioms.

Calling the RACE interface predicate *prove/7* with the parameter *raw* set we get the expected results for deductions (1') and (2'). While (1') fails

```
prove('Exactly three men lift a piano.','Exactly one man lifts a
piano.', [raw], Messages, RunTime, AxiomsUsed, WhyNot).

WhyNot = ['countable common noun: (at least 1) man']
```

(2') succeeds with the help of auxiliary Prolog axiom `cd5` as shown in the proof trace

```
prove('Three men see a piano.', 'A man sees a piano.', [raw], Messages,
RunTime, AxiomsUsed, WhyNot).

Clauses=[satchmo_clause(true, (object(sk1, sk2, man, countable, na, geq,
3), object(sk1, sk3, piano, countable, na, geq, 1), predicate(sk1, sk4,
see, sk2, sk3)), [axiom(1)]), satchmo_clause((object(sk1, A, man,
countable, na, geq, 1), object(sk1, B, piano, countable, na, geq, 1),
predicate(sk1, C, see, A, B)), fail, [theorem(1)])]

call: satchmo(Clauses, Models, Inconsistencies)
```

```
Asserting Head and Indices:
object(sk1,sk2,man,countable,na,geq,3),[axiom(1)]
Asserting Head and Indices:
object(sk1,sk3,piano,countable,na,geq,1),[axiom(1)]
Asserting Head and Indices: predicate(sk1,sk4,see,sk2,sk3),[axiom(1)]
Auxiliary Prolog axiom cd5: Body object(sk1,sk2,man,countable,na,geq,1)
is proved from asserted atom object(sk1,sk2,man,countable,na,geq,3)
Asserting: closed_branch([axiom(1),prolog_axiom(cd5),theorem(1)])
Retracting Head and Indices: predicate(sk1,sk4,see,sk2,sk3),[axiom(1)]
Retracting Head and Indices:
object(sk1,sk3,piano,countable,na,geq,1),[axiom(1)]
Retracting Head and Indices:
object(sk1,sk2,man,countable,na,geq,3),[axiom(1)]
exit: satchmo(Clauses, [], [[axiom(1), prolog_axiom(cd5), theorem(1)]])
AxiomsUsed = [proof(['1: Three men see a piano.'], ['Prolog Axiom cd5:
at least M objects |- at least 1, 2, ..., M-1 objects'])]
```

Deductions from Distributive Plurals. As we have seen distributive plurals are fully specified concerning the structure of the relation between the participants. Thus deductions to distributive plurals with smaller cardinalities and to the singular are always allowed. Again meaning postulates, i.e. auxiliary Prolog axioms, based on the representation of distributive plurals are needed to enable the deductions. As a demonstration an example using distributive plurals in four different syntactic roles.

```
prove('Each of 6 men gives each of 5 students each of 4 books on each of
3 mornings.', 'Each of 5 men gives each of 4 students each of 3 books on
1 morning.', [raw], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: Each of 6 men gives each of 5 students each of
4 books on each of 3 mornings.'], ['Prolog Axiom cd0: enable deductions
from distributive plurals', 'Prolog Axiom cd1: each of M objects |- each
of M-1, ..., each of 2 objects, 1 object']), proof(['1: Each of 6 men
gives each of 5 students each of 4 books on each of 3 mornings.'],
['Prolog Axiom cd1: each of M objects |- each of M-1, ..., each of 2
objects, 1 object'])]
```

There are two results involving the auxiliary axioms `cd0` and `cd1`.

Deductions Involving Collective and Distributive Plurals. Deductions from collective to distributive plurals are not possible, while deductions from distributive to collective plurals are allowed. Here are two examples demonstrating this behaviour. The first example

```
prove('3 women wait.', 'Each of 3 women waits.', [raw], Messages,
RunTime, AxiomsUsed, WhyNot).
```

correctly fails. The second example shows the other direction

```
prove('Each of 3 women waits.', '3 women wait.', [raw], Messages,
RunTime, AxiomsUsed, WhyNot).
```

```
AxiomsUsed = [proof(['1: Each of 3 women waits.'], ['Prolog Axiom cd0:
enable deductions from distributive plurals'])],
```

This proof succeeds with the help of the auxiliary Prolog axiom `cd0`.

Deductions from Conjunctive Plurals. Collective and distributive plurals can also be formed by conjunctions of noun phrases leading to so called conjunctive plurals. Since conjuncts can again be plurals we may end up with rather complex plural structures.

The first example with a collective conjunctive plural correctly fails – though in full English the deduction is valid.

```
prove('John and Mary wait.', 'Mary waits.', [raw], Messages, RunTime,
AxiomsUsed, WhyNot).
```

Replacing the collective conjunctive plural by a distributive one the proof succeeds.

```
prove('Each of John and Mary waits.', 'Mary waits.', [raw], Messages,
RunTime, AxiomsUsed, WhyNot).
```

```
AxiomsUsed = [proof(['1: Each of John and Mary waits.'], ['Prolog Axiom
cd0: enable deductions from distributive plurals'])],
```

Finally an example with a complex conjunctive plural.

```
prove('Each of 2 men and 3 women waits.', 'A man waits.', [raw],
Messages, RunTime, AxiomsUsed, WhyNot).
```

```
AxiomsUsed = [proof(['1: Each of 2 men and 3 women waits.'], ['Prolog
Axiom cd0: enable deductions from distributive plurals', 'Prolog Axiom
cd1: each of M objects |- each of M-1, ..., each of 2 objects, 1
object'])],
```

Deductions from Generalised Quantifiers. RACE allows deductions involving generalised quantifiers for both collective and distributive plurals if and only if the mathematical relations between the generalised quantifiers are correct. But before we come to this there is an additional aspect of generalised quantifiers to be taken into account. ACE provides five generalised quantifiers that – according to their deductive behaviour – can be split into three groups.

- monotone increasing: *at least N*, *more than N*
- monotone decreasing: *at most N*, *less than N*
- non-monotone: *exactly N*

A generalised quantifier *Q* is called monotone increasing if for all sets *Smaller* and *Larger* with *Smaller* $\subseteq$ *Larger* we have that *Q(Smaller)* entails *Q(Larger)*. Here is an example with the monotone increasing generalised quantifier *at least* where *Smaller* = *{3 tall men that John sees}* and *Larger= {3 men that John sees}*.

```
prove('John sees at least 3 tall men.', 'John sees at least 3 men.', [],
Messages, RunTime, AxiomsUsed, WhyNot).
```

```
AxiomsUsed = [proof(['1: John sees at least 3 tall men.'], [])]
```

A generalised quantifier *Q* is called monotone decreasing if for all sets *Smaller* and *Larger* with *Smaller* $\subseteq$ *Larger* we have that *Q(Larger)* entails *Q(Smaller)*. Here is an example with the monotone decreasing generalised quantifier *at most* where *Smaller = {3 tall men that John sees}* and *Larger= {3 men that John sees}*.

```
prove('John sees at most 3 men.', 'John sees at most 3 tall men.', [],
Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: John sees at most 3 men.'], [])]
```

While RACE achieves the monotone increasing effect directly, the monotone decreasing effect is created by interpreting *at most N* as *not more than N* and *less than N* as *not at least N* – producing a deductive behaviour similar to the determiner *no*.

The generalised quantifier *exactly N* is neither monotone increasing nor monotone decreasing. RACE interprets *exactly N* as *at least N & at most N*, i.e. as *at least N & not more than N*. This explains why the generalised quantifier *exactly N* allows us only to deduce a set from itself. While the following deduction succeeds

```
prove('John sees exactly 3 men.', 'John sees exactly 3 men.', [],
Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: John sees exactly 3 men.'], [])]
```

this one fails

```
prove('John sees exactly 3 men.', 'John sees exactly 2 men.', [],

WhyNot = ['countable common noun: (at least 2) man'].
```

Finally, one example demonstrating the mathematical properties of generalised quantifiers.

```
prove('At least 3 boys run fast.', 'More than 2 boys run. A boy runs
fast.', [raw], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: At least 3 boys run fast.'], ['Prolog Axiom
cd5: at least M objects |- at least 1, 2, ..., M-1 objects', 'Prolog
Axiom cd6: at least M objects |- more than 1, 2, ..., M-1 objects'])]
```

Summation. RACE performs summations on individuals found in an inheritance tree. Here is a simple example showing that *2 male cats* and *3 tricolor cats* are *5 animals*.

```
prove('There are at least 2 male cats. There are at least 3 tricolor
cats. Every tricolor cat is a female cat. Every male cat is a cat. Every
female cat is a cat. Every cat is an animal. No male cat is a female
cat.', 'There are at least 5 animals.', [raw], Messages, Time,
AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: There are at least 2 male cats.', '2: There are
at least 3 tricolor cats.', '3: Every tricolor cat is a female cat.',
'4: Every male cat is a cat.', '5: Every female cat is a cat.', '6:
Every cat is an animal.', '7: No male cat is a female cat.'], ['Prolog
Axiom agg1: aggregation', 'Prolog Axiom c2: Identity of attributive
adjectives.'])]
```

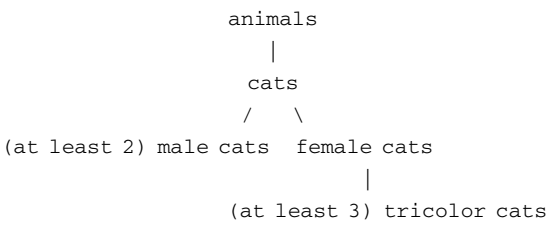


Fig. 7. Inheritance tree of cats and animal

The proof uses the auxiliary Prolog axiom `agg1` that implements the following algorithm. First, `agg1` extracts the inheritance tree (figure 7) from the axioms checking that the classes involved – *male cats* and *female cats* – are disjoint. Then `agg1` adds the cardinalities of the leaf nodes. The runtime of this algorithm depends on the size of the inheritance tree – for a tree with the height H and the width W the worst-case runtime is $O(H^2 \times W^2)$ – and not on the cardinalities of the leaf nodes. If the relevant classes are not disjoint then `agg1` fails.

Modality. ACE provides modal constructs for *possibility*, *necessity* and *sentence subordination* represented in the DRS language [7] by the three modal operators *diamond* <>, *box* [] and *colon* :.

Modal logic can be mapped to first-order predicate logic via the so-called *standard translation*[11] that adds to each logical atom an argument that stands for a *possible world*. Bos [4] extended the standard translation to cover also sentence subordination.

As mentioned in section 4, the possible world argument is added to each logical atom during the translation of DRSs into FOL, meaning that RACE uses possible world semantics even in the absence of modal constructs. This does not seem to have any adverse effect on performance. Possible worlds are connected by a binary accessibility relation that RACE assumes to be *reflexive*, *symmetric* and *transitive*, i.e. an *equivalence relation*. The accessibility relation is expressed as three auxiliary Prolog axioms.

Here are three examples two of which simulate standard axioms of modal logic.

modality axiom: If A then it is possible that A.

```
prove('John waits patiently in the hall.', 'John can wait.', [raw],
Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: John waits patiently in the hall.'], ['Prolog
Axiom pw1: Accessibility relation is reflexive.'])]
```

modality axiom: If A is necessary then it is not possible that not A.

```
prove('John must wait.', 'It is not possible that John cannot wait.',
[raw], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: John must wait.'], ['Prolog Axiom pw1:
Accessibility relation is reflexive.']), proof(['1: John must wait.'],
['Prolog Axiom pw1: Accessibility relation is reflexive.', 'Prolog Axiom
pw2: Accessibility relation is symmetric.'])]
```

[11] `https://secure.wikimedia.org/wikipedia/en/wiki/Standard_translation`

sentence subordination

```
prove('John sincerely promises to wait for Mary.', 'John promises to
wait.', [raw], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: John sincerely promises to wait for Mary.'],
[])]
```

Copula. In ACE the copula *to be* is used to express

- identity, for instance *John is Harry.*
- class membership, for instance *Every cat is an animal.*
- predication, for instance *The cat is black.* or *John is in the garden.*

To reason correctly, RACE needs to distinguish the various uses of the copula and to support them by appropriate meaning postulates. During the translation into clauses step RACE replaces the verb *be* by variants that express the respective use. Concretely by

- *be_MOD* if the copula *be* is modified by adverbs or prepositional phrases
- *be_ID* if the copula *be* links two proper names or a proper name and *something/somebody*
- *be_NP* if the copula *be* links a proper name or a noun phrase to a noun phrase
- *be_ADJ* if the copula *be* links a proper name or a noun phrase to an adjective

For each variant of the copula RACE provides respective meaning postulates in the form of auxiliary Prolog axioms.

The variant *be_NP* can serve as a concrete example. Since *be_NP* is meant to express class membership, it needs to have the attributes of a partial order, i.e. the auxiliary axioms must ensure that *be_NP* behaves as a *reflexive*, *antisymmetric* and *transitive* relation. While reflexivity is trivial, here are examples showing the antisymmetry

```
prove('Every man is a male. Every male is a man.', 'Every man is a
man.', [raw], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: Every man is a male.', '2: Every male is a
man.'], ['Prolog Axiom c1c: Identity of countable objects.']), ...]
```

and the transitivity

```
prove('Every man is a person. Every person is a human.', 'Every man is a
human.', [raw], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: Every man is a person.', '2: Every person is a
human.'], ['Prolog Axiom c1c: Identity of countable objects.'])]
```

Obviously, if the need arose it would also be possible to interpret *be_NP* as an equivalence relation by defining appropriate auxiliary Prolog axioms.

Variations of Query Answering. RACE allows for three different types of queries: *yes/no* queries ask for the existence or non-existence of a specific situation, *wh*-queries (*who*, *whose*, *what*, *which*) ask for noun phrases, and *wh*-queries (*how*, *when*, *where*) ask for adverbs and prepositional phrases. Successful answers to *wh*-queries not only report the axioms used during the proof, but also the substitutions of the query words. For this purpose RACE uses yet another set of auxiliary Prolog predicates.

Here are examples for each of the query types.

```
answer_query('A man sleeps. A woman does not sleep.', 'Is there somebody
who does not sleep?', [], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['2: A woman does not sleep.'], [])]


answer_query('John sees a red book that lies on a table.', 'Who sees
which book?', [raw], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: John sees a red book that lies on a table.',
'Substitution: who = John', 'Substitution: which = (at least 1) book,
(positive of) red'], ['Prolog Axiom w11: If there are countable objects
then the question "which" can be answered.', 'Prolog Axiom w2: If there
are named objects then the question "who" can be answered.'])]


answer_query('Mary sleeps silently. John sleeps with a heavy dream.',
'How does somebody sleep?', [], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: Mary sleeps silently.', 'Substitution:
how/when/where = (positive of) silently'], []), proof(['2: John sleeps
with a heavy dream.', 'Substitution: how/when/where = (with) (at least
1) dream, (positive of) heavy'], [])]


answer_query('Mary sleeps silently. John sleeps in a bedroom.', 'Where
does somebody sleep?', [], Messages, RunTime, AxiomsUsed, WhyNot).

Messages = [message(warning, 'query answering', ''-'', 'The query word
"where" was interpreted as the query word "how".', '')]

AxiomsUsed = [proof(['1: Mary sleeps silently.', 'Substitution:
how/when/where = (positive of) silently'], []), proof(['2: John sleeps
in a bedroom.', 'Substitution: how/when/where = (in) (at least 1)
bedroom'], [])]
```

Note that the last proof generated a warning message. Since ACE does not use thematic roles – that would allow RACE to distinguish location, time, manner, instrument etc. – the *where*- and *when*-queries are interpreted as the less specific *how*-query. As the example shows this can lead to possibly surprising answers.

6 Decidability, Termination, Efficiency, Looping and All That

Decidability of ACE. Pratt-Hartman and Third [5] investigated several fragments of natural language with respect to their complexity and decidability and found that the fragment containing singular quantified nouns, predicative adjectives, copula with/without negation, relative clauses, transitive verbs and reflexive/non-reflexive pronouns as anaphors resolved by co-indexing is undecidable.

This means that ACE is undecidable since already its first-order subset is larger than the above fragment. However, ACE contains some decidable – though less expressive – subsets, for instance the subset defined by its translation into OWL.

Termination of RACE. Since ACE is not decidable, RACE would not terminate for axioms with an infinite model. To prevent this, RACE is equipped with a time-out whose limit depends on the size of the problem. Figure 8 shows a set of axioms – originally devised by Pratt-Hartman and Third [5] and translated into ACE by Mandl [6] – that would generate an infinite model, if not stopped by RACE after 10 s.

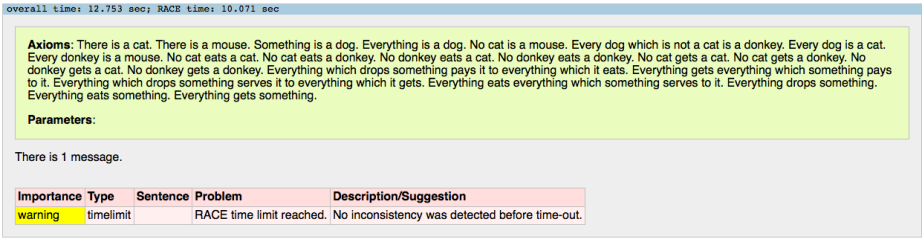


Fig. 8. Termination in spite of an infinite model

Efficiency of RACE. If RACE terminates then its run-time can mainly be attributed to the forward-reasoning of clauses, i.e. to unifying logical atoms of clause bodies with logical atoms previously asserted to the Prolog data base. If N is the number of clauses, B is the average number of body atoms and A is the number of atoms asserted to the Prolog data base, then in the worst-case $O(N \times A^B)$ unifications are required. To reduce this number, RACE uses the following means:

- The DRS representation of ACE texts is very compact reducing the number of clauses and the number of body atoms.
- RACE represents auxiliary axioms as Prolog predicates instead of clauses thus reducing the number of clauses, and furthermore replacing eager forward-reasoning by lazy backward-reasoning.
- RACE reduces the number of clauses that at any moment participate in forward-reasoning.
    - RACE blocks after the first round of forward-reasoning the clauses with the body *true* since these clauses are only used once.
    - Before a clause is selected for execution RACE checks whether at least one logical atom of the body of the clause was asserted to the

Prolog database in the preceding round; if not then the clause can certainly not be executed.

- To minimise the number of atoms asserted to the Prolog data base RACE ensures that no logical atom is asserted that would subsume an atom already in the database.
- All remaining unifications profit from Prolog's indexing.

RACE's efficiency and scaling-up behaviour have not yet been systematically investigated. On a MacBook Pro most examples presented in this paper use less than 10 ms, while Lewis Carroll's *Grocer Puzzle*

Given
Every honest and industrious person is healthy. No grocer is healthy. Every industrious grocer is honest. Every cyclist is industrious. Every unhealthy cyclist is dishonest. No healthy person is unhealthy. No honest person is dishonest. Every grocer is a person. Every cyclist is a person.
show that
No grocer is a cyclist.

needs 40 ms. As figure 9 shows the *Grocer Puzzle* executed via the web-client needs an order of magnitude more time because the RACE server runs on an older machine. Note that RACE reports one proof based on five of the original nine axioms.

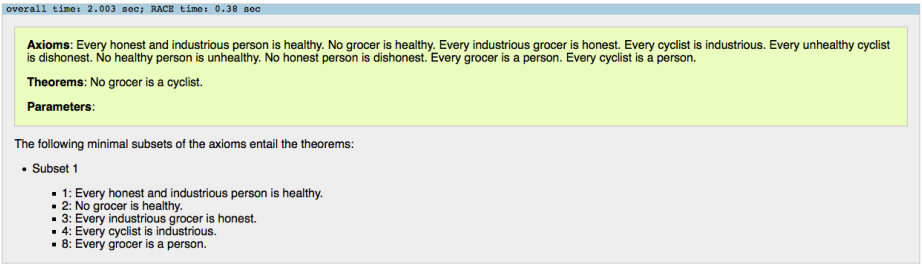


Fig. 9. Lewis Carroll's Grocer Puzzle

Loop Prevention. Forward-reasoning systems are prone to loops. Given the clauses

```
satchmo_clause(true, A, ...)
satchmo_clause(A, B, ...)
satchmo_clause(B, A, ...)
```

with the circular definitions *A*, *A => B, B => A*, RACE would loop infinitely producing A, B, A, B, ... if it did not contain an effective way of loop prevention. This loop prevention is the effect of functionality introduced for other purposes. RACE uses a subsumption check to prevent asserting the same logical atom twice to the Prolog data base, and selects a clause only if the body of the clause contains a logical atom asserted to the Prolog database in the preceding round of forward-reasoning. Thus in our example RACE prevents the second occurrence of *A* to be asserted to the

database with the result that nothing is asserted and that no clause can be selected in the next round. Figure 10 is an example showing that RACE can reason correctly even in the presence of circular definitions.

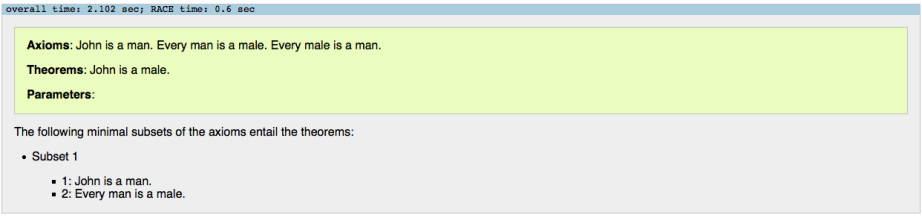


Fig. 10. Deduction in the presence of loops

Inconsistent Axioms. From inconsistent axioms it is possible to deduce anything. If RACE detects that the axioms are inconsistent then it outputs the generated results together with a warning message.

... and All That. By default RACE uses the

- Unique name assumption, i.e. different proper names – *Robert*, *Bob*, ... – refer to different entities unless explicitly stated otherwise, for instance by *Robert is Bob*.
- Open world assumption, i.e. missing knowledge of the truth of an ACE sentence *A*, respectively a failed proof of *A*, is not interpreted as ¬ *A*.

7 Conclusions

I presented the reasoner RACE that can show the (in-) consistency of a set of ACE axioms, prove ACE theorems from ACE axioms and answer ACE queries from ACE axioms. In each case RACE gives a proof justification in ACE and full English. Most deductions are supported by auxiliary Prolog axioms that provide meaning postulates and other functionality.

The next version of RACE will cover the still missing language features of ACE, namely arithmetic, formulas, sets, lists and strings. Future extensions of RACE will focus on executable specifications and temporal reasoning. This will also involve larger examples possibly using external knowledge sources.

Furthermore, an open issue will have to be investigated that is due to RACE's flexibility. Obviously, much of the reasoning power of RACE stems from the more than 50 auxiliary Prolog axioms. The flexibility and power of Prolog allows us to craft auxiliary axioms to virtually deduce anything. Thus what we actually can deduce in a given situation raises not only the question of correctness and completeness, but also the question of need and requirement, and consequently the question of the availability of adequate auxiliary axioms.

Here is a case that illustrates this problem. Given that different proper names refer to different entities unless explicitly stated otherwise, RACE could easily be extended by an auxiliary axiom to perform the following two deductions.

Robert is a man. Bob is a man. |– *There are two men.*

Robert is a man. Bob is a man. Robert is Bob. |– *There is one man.*

This would introduce non-monotonic reasoning that the current version of RACE does not support. Future research will decide whether non-monotonicity is needed.

Here is another case. While English provides query words for most constituents of a sentence, it does not provide a query word for the verb. To get information on the verb you have to formulate non-specific questions, for instance *What does ... do?*. Given the sentence *John stands at the window and looks into the garden.* the question *What does John do?* can be answered by a number of complete sentences that cover at least part of the state of affairs. To equip RACE with this functionality we need additional auxiliary axioms, and furthermore a standardisation of the questions that can legitimately be asked. Again, future research will show whether this extension of RACE is useful.

Acknowledgements. I very much appreciate the constructive comments and helpful suggestions of the three reviewers of a previous version of this paper. Thanks also go to my colleagues Kaarel Kaljurand, Tobias Kuhn, Ian Pratt-Hartmann, Aarne Ranta and Rolf Schwitter for their valuable feedback.

References

1. Manthey, R., Bry, F.: SATCHMO: A Theorem Prover Implemented in Prolog. In: Lusk, E., Overbeek, R. (eds.) CADE 1988. LNCS, vol. 310, pp. 415–434. Springer, Heidelberg (1988)
2. Bry, F., Yahya, A.: Positive Unit Hyperresolution Tableaux and Their Application to Minimal Model Generation. Journal of Automated Reasoning 25, 35–82 (2000)
3. Basin, D., D'Agostino, M., Gabbay, D.M., Matthews, S., Viganò, L. (eds.): Labelled Deduction. Applied Logic Series, vol. 17. Springer (2000)
4. Bos, J.: Computational Semantics in Discourse: Underspecification, Resolution, and Inference. Journal of Logic, Language and Information 13, 139–157 (2004)
5. Pratt-Hartmann, I., Third, A.: More fragments of language. Notre Dame Journal of Formal Logic 47(2), 151–177 (2006)
6. Mandl, M.: Zur Komplexität des Erfüllbarkeitsproblems von Sprachfragmenten, Hauptseminar Kontrollierte Sprache CIS, SS 2010. Universität München (2010)
7. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Discourse Representation Structures for ACE 6.6, Technical Report ifi-2010.0010, Department of Informatics, University of Zurich (2010)
8. Schwertel, U.: Plural Semantics for Natural Language Understanding — A Computational Proof-Theoretic Approach, PhD thesis, University of Zurich (2004)

Codeco: A Practical Notation for Controlled English Grammars in Predictive Editors*

Tobias Kuhn[1,2]

[1] Department of Informatics & Institute of Computational Linguistics,
University of Zurich, Switzerland
[2] Department of Intelligent Computer Systems,
University of Malta
kuhntobias@gmail.com
http://www.tkuhn.ch

Abstract. This paper introduces a new grammar notation, called Codeco, designed for controlled natural language (CNL) and predictive editors. Existing grammar frameworks that target either formal or natural languages do not work out particularly well for CNL, especially if they are to be used in predictive editors and if anaphoric references should be resolved in a deterministic way. It is not trivial to build predictive editors that can precisely determine which anaphoric references are possible at a certain position. This paper shows how such complex structures can be represented in Codeco, a novel grammar notation for CNL. Two different parsers have been implemented (one in Prolog and another one in Java) and a large subset of Attempto Controlled English (ACE) has been represented in Codeco. The results show that Codeco is practical, adequate and efficient.

1 Introduction

Controlled natural languages (CNL) have been proposed to make knowledge representation systems more user-friendly [25,15]. The scope of this work is restricted to controlled natural languages with a direct and deterministic mapping to some kind of formal logic, like Attempto Controlled English (ACE) [8].

One of the main problems of CNL is that it can be very difficult for users to write statements that comply with the restrictions of CNL. Three approaches have been proposed so far to solve this problem: error messages [3], language generation [24], and predictive editors [26,15]. Each of the approaches has its own advantages and drawbacks. The work presented here follows the predictive editor approach, where the editor shows all possible continuations of a partial sentence. However, implementing lookahead features (i.e. retrieving the possible continuations for a given partial sentence on the basis of a given grammar) is not

* The work presented here was funded by the research grant (Forschungskredit) programs 2006 and 2008 of the University of Zurich.

M. Rosner and N.E. Fuchs (Eds.): CNL 2010, LNAI 7175, pp. 95–114, 2012.

a trivial task, especially if the grammar describes complex nonlocal structures like anaphoric references. Furthermore, good tool integration is crucial for CNL, no matter which approach is followed. For this reason, it is desirable to have a simple grammar notation that is fully declarative and can easily be implemented in different kinds of programming languages.

In the remainder of this paper, the relevant background is discussed and the problems we are facing are explained (Sect. 2). Then, the Codeco grammar notation is introduced, which is designed to solve the identified problems (Sect. 3). Next, two implementations of Codeco are presented (Sect. 4) and a specific grammar written in this notation is introduced (Sect. 5). Finally, several evaluation results of the approach are shown (Sect. 6) and the conclusions are drawn (Sect. 7).

This paper is built from a chapter of the author's doctoral thesis [17], which gives more information about the details of the approach and about the concrete application in specific tools. The Codeco notation was outlined in the extended abstract of this full paper [16].

2 Background

Below, a number of requirements for controlled English grammars are introduced. Then, different kinds of existing grammar frameworks are discussed: frameworks for natural languages, parser generators, and definite clause grammars (DCG). Finally, related work is introduced in the form of the Grammatical Framework.

2.1 Requirements for Controlled English Grammars

What follows is 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 followed and that the language supports complex phenomena like anaphoric references.

Concreteness. Concreteness is an obvious requirement. Due to their practical and computer-oriented nature, CNL grammars should be concrete, i.e. fully formalized to be read and interpreted by computer programs.

Declarativeness. As a second requirement, CNL grammars should be declarative, i.e. defined in a way that does not depend on a concrete algorithm or implementation. Declarative grammars can be completely separated from the parser that processes them. This makes it easy for such grammars to be used by other programs, to replace the parser, or to have different parsers for the same language. Declarative grammars are easy to change and reuse and can be shared easily between different parties using the same CNL.

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.

Anaphoric References and Scopes. CNLs that support anaphoric references raise special requirements. For such languages, 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. 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. Every sentence that contains an expression that can only be interpreted in a referential way but cannot be resolved must be considered syntactically incorrect. In other words, CNL grammars must be able to represent the fact that only resolvable anaphoric references are allowed.

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. While scopes in natural language 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 CNL 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.

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 besides the parser that need to read and process the grammar, e.g. editors [26], paraphrasers [12] and verbalizers[1]. Furthermore, more than one parser might be necessary for practical reasons. For example, a simple top-down parser may be the best option for simple grammars when used for parsing large texts

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

in batch mode and for doing regression tests (e.g. through language generation). On the other hand, a chart parser is better suited for complex grammars and for providing lookahead capabilities.

2.2 Natural Language Grammar Frameworks

A large number of grammar frameworks exist to process natural languages. Some of the most popular ones are *Head-Driven Phrase Structure Grammars* (HPSG) [23], *Lexical-Functional Grammars* [13], and *Tree-Adjoining Grammars* [11]. More of them are discussed by Cole et al. [5]. 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.

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 *language descriptions*: they describe existing phenomena. CNL grammars, in contrast, are *language definitions*: they define something new.

Obviously, grammars for natural languages and those for CNLs differ in complexity. Natural languages are very complex and so must be the grammars that thoroughly describe such languages. CNLs are typically much simpler and abandon natural structures that are difficult to process.

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 [4] or CLOnE [9].

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, 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 paper focuses) remove ambiguity by their design, which typically makes underspecification and heuristics unnecessary.

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. [19]). The following example should 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 clearly identified in the text. In order to address the problem of anaphoric references, natural language grammar frameworks like HPSG establish "binding theories" [2,23] 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 and does not deal with deterministic resolution of them.

2.3 Parser Generators

A number of systems exist that are aimed at the definition and parsing of formal languages (e.g. programming languages). In the simplest case, such grammars are written in Backus-Naur Form [21,14]. Examples of more sophisticated grammar formalisms for formal languages — called *parser generators* — include Yacc [10] and GNU bison[2]. Some CNLs like Formalized-English [20] are defined in such parser generator notations.

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 by using plain Backus-Naur style grammars. However, such grammars are very limited and even very simple CNLs cannot be defined appropriately. 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.

When discussing lookahead capabilities of parser generators, it has to be noted that the term *lookahead* has a different meaning in the context of parser generators: *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, a simple kind of lookahead (in our sense of the word) is available in many source code editors in the form of code completion features.

2.4 Definite Clause Grammars

Definite clause grammars (DCG) [22], finally, are a simple but powerful way to define grammars for natural and formal languages and are mostly written in

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

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 large overhead is necessary to simulate backtracking and unification.

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 DCG grammars. The following two exemplary grammar rules show how antecedents and anaphors could be defined:

```
np(Agr, Ante-[Agr|Ante]) --> determiner(Agr), noun(Agr).
np(Agr, Ante-Ante) --> ana_pron(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. 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, as illustrated by the following example:

```
s(Ante1-Ante3) --> np(Agr, Ante1-Ante2), vp(Agr, Ante2-Ante3).
```

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 neither convenient nor elegant.

Different DCG extensions have been proposed in order to describe natural language in a more appropriate way. Assumption Grammars [6], for example, are motivated by natural language phenomena that are hard to express otherwise, like free word order. Simple anaphoric references can be represented in a very clean way, but there are problems with more complex anaphor types like irreflexive pronouns.

A further problem with the DCG approach concerns lookahead features. In principle, it is possible to provide lookahead features with standard Prolog DCGs, as shown in previous work conducted with Rolf Schwitter [18]. However, this approach is not very efficient and can become impractical for complex grammars and long sentences.

2.5 Related Work

The Grammatical Framework (GF) [1] is the only existing grammar framework (to the author's knowledge) that has a specific focus on CNL. Apart from the fact that it has no particular support for describing the resolvability of anaphoric references on the syntactic level, GF fulfills the requirements introduced above. With extensions (perhaps inspired by Codeco), it could become a suitable grammar notation for the specific problem described in this paper.

3 The Codeco Notation

On the basis of the aforementioned requirements, a grammar notation called *Codeco*, which stands for "*co*ncrete and *de*clarative grammar notation for *co*ntrolled natural languages", is introduced here.

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. As I will show, it works 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.

Below, 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.

3.1 Simple Categories and Grammar Rules

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 on):

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

Terminal categories are represented in square brackets:

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

In order to provide a clean interface between grammar and lexicon, Codeco has a special notation for pre-terminal categories, being marked with an underline:

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

Pre-terminal categories are conceptually somewhere between non-terminal and terminal categories, in the sense that they can be expanded but 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 represented with a plain arrow, for instance:

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

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

In order to support context-sensitivity, non-terminal and pre-terminal categories can be augmented with flat feature structures. They are shown here using the colon operator ":" with the name of the feature to the left and its value to the right. Values can be variables, which are displayed as boxes:

$$vp\left(\begin{matrix}\text{num: }\boxed{\text{Num}}\\ \text{neg: }\boxed{\text{Neg}}\end{matrix}\right) \;\overset{\vdots}{\longrightarrow}\; v\left(\begin{matrix}\text{num: }\boxed{\text{Num}}\\ \text{neg: }\boxed{\text{Neg}}\\ \text{type: tr}\end{matrix}\right)\; np\Big(\text{case: acc}\Big)$$

$$v\left(\begin{matrix}\text{neg: }+\\ \text{type: }\boxed{\text{Type}}\end{matrix}\right) \;\overset{\vdots}{\longrightarrow}\; [\,\text{does not}\,]\;\; verb\Big(\text{type: }\boxed{\text{Type}}\Big)$$

$$np\Big(\text{noun: }\boxed{\text{Noun}}\Big) \;\overset{\vdots}{\longrightarrow}\; [\,\text{a}\,]\;\; \underline{noun}\Big(\text{text: }\boxed{\text{Noun}}\Big)$$

An important restriction is that feature values *cannot* be feature structures themselves. This means that feature structures in Codeco are always flat. This restriction has practical reasons. It should keep Codeco simple and easy to implement, but it can easily be dropped in theory.

3.2 Normal Forward and Backward References

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

In principle, it is easy to support sentences like

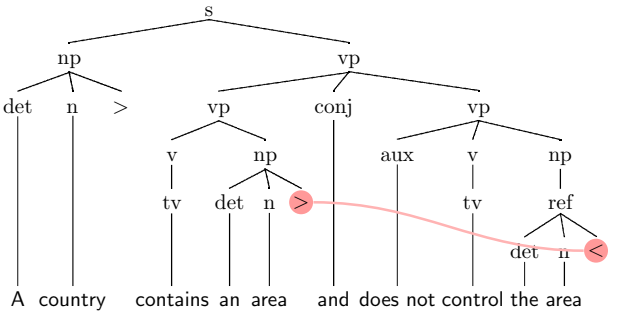
A country contains an area that is not controlled by the country.

where "the country" is a resolvable anaphoric reference. 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.

where "the country" is not resolvable. This can be acceptable, but in many situations there are good reasons to disallow such non-resolvable references.

In Codeco, the use of anaphoric references can be restricted to positions where they can be resolved. This is done with the help of the special categories ">" and "<", which describe 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, i.e. “>” stands for antecedents. “<” represents a *backward reference* and refers back to the closest 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:

$$np \overset{\vdots}{\longrightarrow} [\,\text{a}\,] \quad \underline{noun}\left(\text{text: } \boxed{\text{Noun}}\right) \; > \begin{pmatrix} \text{type: noun} \\ \text{noun: } \boxed{\text{Noun}} \end{pmatrix}$$

$$ref \overset{\vdots}{\longrightarrow} [\,\text{the}\,] \quad \underline{noun}\left(\text{text: } \boxed{\text{Noun}}\right) \; < \begin{pmatrix} \text{type: noun} \\ \text{noun: } \boxed{\text{Noun}} \end{pmatrix}$$

The forward reference of the first rule establishes an antecedent to which later backward references can refer. 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 to the left in the syntax tree.

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

Altogether, these special categories provide a very simple way to establish nonlocal 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. We need more reference types, but first accessibility constraints have to be discussed.

3.3 Scopes and Accessibility

As already pointed out, anaphoric references are affected by scopes. References are resolvable only to positions in the previous text that are accessible, i.e. that are not inside closed scopes. An example 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:

$$quant\Big(\text{exist: }-\Big) \xrightarrow{\vdots} \; // \quad [\,\text{every}\,]$$

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 "$\xrightarrow{\sim}$", for instance:

$$vp\Big(\text{num: }\boxed{\text{Num}}\Big) \xrightarrow{\sim} v\begin{pmatrix}\text{neg: }+\\ \text{num: }\boxed{\text{Num}}\\ \text{type: tr}\end{pmatrix} np\Big(\text{case: acc}\Big)$$

This rule states that any scope that is opened by the direct or indirect children of "*v*" and "*np*" is closed at the end of "*np*". If no scopes have been opened, scope-closing rules simply behave like normal rules.

In contrast to most other approaches, scopes are defined in a way that is completely independent from the semantic representation.

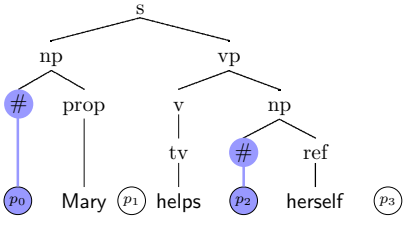
3.4 Position Operators

With the introduced Codeco elements, anaphoric definite noun phrases like "the area" can be restricted to positions where they are resolvable. However, it is not possible so far 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, we cannot distinguish 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 takes a variable and assigns it an identifier that represents the respective position in the text. The following picture visualizes how position operators work:



With the use of position operators, reflexive pronouns can be defined in a way that excludes unresolvable pronouns, i.e. excludes pronouns that do not match with the subject of the given verb phrase:

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

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

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

3.5 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:

* 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 CNL, however, the overriding of variables can be confusing to the readers. ACE, for example, does not allow variables to be overridden.

Such restrictions cannot be defined with 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:

$$newvar \overset{:}{\longrightarrow} \underline{var}\left(\text{text: } \boxed{\text{V}}\right) \not< \begin{pmatrix}\text{type: var}\\ \text{var: } \boxed{\text{V}}\end{pmatrix} > \begin{pmatrix}\text{type: var}\\ \text{var: } \boxed{\text{V}}\end{pmatrix}$$

The special category "$\not<$" succeeds only if there is no accessible forward reference that unifies with the given feature structure.

3.6 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:

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

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.

Complex backward references are a powerful construct, with which anaphoric references can be restricted in a very general way. The following example — which is rather artificial and would probably not be very useful in practice — illustrates the general nature of complex backward references: 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 with the following rule:

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

Complex backward references that have exactly one positive feature structure and no negative ones are equivalent to normal backward references.

3.7 Strong Forward References

Finally, one last 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 "$\gg$" can be used, which introduces a *strong forward reference*:

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

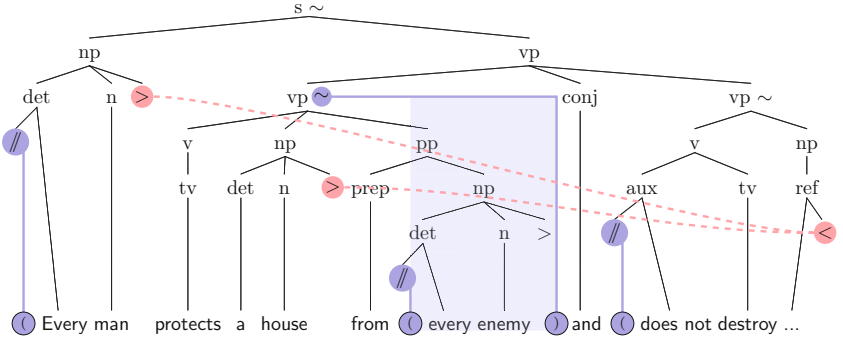
Strong forward references are always accessible even if they are within closed scopes. Apart from that, they behave like normal forward references.

3.8 Principles of Reference Resolution

The resolution of references in Codeco 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.

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 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 could look as follows:

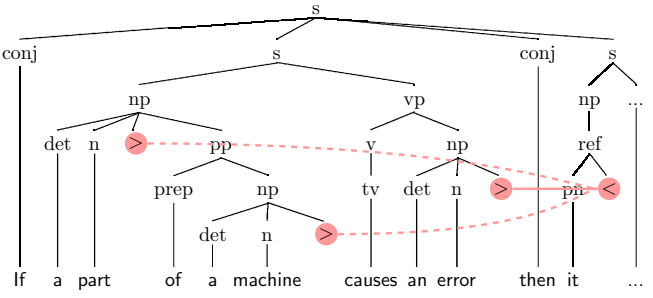


All nodes that represent the head of a scope-closing grammar rule are marked with "~". The positions in the text where scopes open and close are marked by parentheses. In this example, three scopes have been opened but only the second one (the one in front of "every enemy") has been closed (after "enemy"). The shaded 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 shown as dashed lines. Thus, the partial sentence can be continued by the anaphoric references "the man" or "the house" (or equivalently "himself" or "it", respectively) but not by the reference "the enemy".

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. Or more precisely, when traversing the syntax tree starting from the backward reference and going back in a right-to-left, depth-first manner, the first matching and accessible 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 "error". According to the principle of proximity, the closest antecedent is taken, i.e. "error".

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 to the left of the reference are considered for its resolution, whereas variable bindings that would be entailed by a part of the syntax tree to the right are not considered.

The following example illustrates why the principle of left-dependence is important:

$$ref \xrightarrow{\vdots} [\,\text{the}\,] \; < \begin{pmatrix} \text{type: noun} \\ \text{noun: } \boxed{\text{N}} \end{pmatrix} \; \underline{noun}\left(\text{text: } \boxed{\text{N}}\right)$$

$$ref \xrightarrow{\vdots} [\,\text{the}\,] \; \underline{noun}\left(\text{text: } \boxed{\text{N}}\right) \; < \begin{pmatrix} \text{type: noun} \\ \text{noun: } \boxed{\text{N}} \end{pmatrix}$$

These are two versions of the same grammar rule. The only difference is that the backward reference and the pre-terminal category "*noun*" are switched. The first version is not a very sensible one: the backward reference is resolved without considering how the variable "N" is bound by the category "*noun*". The second version is much better: the resolution of the reference takes into account which noun has been read from the input text.

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

3.9 Restriction on Backward References

In order to provide proper and efficient lookahead algorithms that can handle backward references, their usage must be restricted: Backward references 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 rule body and they must not follow a non-terminal category.

However, the algorithms that process Codeco also work for grammars that do not follow this restriction. Only the lookahead feature would not work as expected.

4 Implementations

The Codeco notation currently has two implementations: one that transforms it into Prolog DCG grammars to be executed as Prolog programs, and another one that executes Codeco grammars in Java with a chart parsing algorithm.

The Prolog DCG implementation is primarily intended for running parsing and generation tasks in batch mode. As we will see, it is very fast, but it lacks lookahead features. Chart parsers are a good choice for such lookahead features, and this is the approach of the second implementation: Codeco grammars are executed in Java following the Earley chart parsing algorithm [7]. The Java implementation is slower than the one in Prolog (see below), but it has lookahead support (i.e. it can list all possible next tokens for partial sentences) and enables Java applications to run Codeco grammars without Prolog.

Both implementations can not only be used to parse but also to automatically generate all syntactically correct sentences up to a certain sentence length. The details of these implementations, including the lookahead algorithm and the necessary extensions for the Earley algorithm, can be found in the author's doctoral thesis [17].

The web applications AceWiki [15] and the ACE Editor[3] both use the Codeco notation. The Prolog implementation is used for regression testing, and the Java implementation is used by their predictive editors for the lookahead features.

5 ACE Codeco Grammar

The introduced Codeco notation has been used to describe a large subset of ACE, to be called *ACE Codeco*. This grammar consists of 164 grammar rules[4] and is used by the ACE Editor mentioned above.

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 probably the broadest unambiguous subset of English that has ever been defined in a concrete and fully declarative way and that includes complex issues like anaphoric references.

[3] http://attempto.ifi.uzh.ch/webapps/aceeditor/
[4] See [17] for the complete grammar.

When sentences are generated for evaluation purposes 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 can only use a subset of the grammar for such evaluations. Such an evaluation subset has been defined for ACE Codeco, using a minimal lexicon and only 97 of the 164 grammar rules. These 97 grammar rules are chosen in a way that reduces combinatorial explosion but retains the complexity of the language.

6 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. On the other hand, it can be taken as a test case to evaluate the Codeco notation and the two implementations thereof.

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, the evaluation subset of ACE Codeco generates 2'250'869 sentences, which are all distinct. Thus, at least a large subset of ACE Codeco is unambiguous for at least relatively short sentences.

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.

Equivalence Check of the Implementations. Since both implementations support language generation, we can 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 below), the former cannot generate as long sentences as the latter within reasonable time. The resulting set of sentences generated by the Java implementation is 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.

Table 1. These are the results of a performance test of the two implementations of Codeco. As a comparison, the performance of the existing ACE parser (APE) is shown for the parsing task.

task	grammar	implementation	time in seconds *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

Performance Tests of the Implementations. Finally, the performance of the two implementations can be evaluated and compared. Both can be used for parsing and for generation, and thus the runtimes in these two disciplines can be compared.

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 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 was 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. Table 1 shows the results of these performance tests.

The generation of the resulting 14'240 sentences only requires about 41 seconds in the case of the Prolog DCG implementation. This means that less than 3 milliseconds are 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.

The fact that the Java implementation requires considerably more time than the Prolog DCG is not surprising. 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 grammar rules: they have to be implemented on a higher level. The same holds for variable unifications, which come for free with Prolog but have to be implemented on a higher level 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.

7 Conclusions

In summary, the Codeco notation allows us to define controlled natural languages in a convenient and adequate way, including complex nonlocal phenomena like anaphoric references. 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, which are important for predictive editors. Furthermore, Codeco enables automatic grammar testing, e.g. by exhaustive language generation, which can be considered very important for the development of reliable practical applications. Altogether, Codeco embodies a more engineering focused approach to CNL.

Codeco is a proposal for a general CNL grammar notation. It cannot be excluded at this point that extensions become necessary to express the syntax of other CNLs, but Codeco has been shown to work very well for a large subset of ACE, which is one of the most advanced CNLs to date.

References

1. Angelov, K., Ranta, A.: Implementing Controlled Languages in GF. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 82–101. Springer, Heidelberg (2010)
2. Chomsky, N.: On binding. Linguistic Inquiry 11(1), 1–46 (1980)
3. Clark, P., Chaw, S.-Y., Barker, K., Chaudhri, V., Harrison, P., Fan, J., John, B., Porter, B., Spaulding, A., Thompson, J., Yeh, P.: Capturing and answering questions posed to a knowledge-based system. In: K-CAP 2007: Proceedings of the 4th International Conference on Knowledge Capture, pp. 63–70. ACM (2007)
4. Clark, P., Harrison, P., Jenkins, T., Thompson, J., Wojcik, R.H.: Acquiring and using world knowledge using a restricted subset of English. In: Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2005), pp. 506–511. AAAI Press (2005)

5. Cole, R., Mariani, J., Uszkoreit, H., Varile, G.B., Zaenen, A., Zampolli, A., Zue, V. (eds.): Survey of the State of the Art in Human Language Technology. Cambridge University Press (1997)
6. Dahl, V., Tarau, P., Li, R.: Assumption grammars for processing natural language. In: Naish, L. (ed.) Proceedings of the Fourteenth International Conference on Logic Programming, pp. 256–270. MIT Press (1997)
7. Earley, J.: An efficient context-free parsing algorithm. Communications of the ACM 13(2), 94–102 (1970)
8. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web. LNCS, vol. 5224, pp. 104–124. Springer, Heidelberg (2008)
9. Funk, A., Tablan, V., Bontcheva, K., Cunningham, H., Davis, B., Handschuh, S.: CLOnE: Controlled Language for Ontology Editing. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 142–155. Springer, Heidelberg (2007)
10. Johnson, S.C.: Yacc: Yet another compiler-compiler. Computer Science Technical Report 32, Bell Laboratories, Murray Hill, NJ, USA (July 1975)
11. Joshi, A.K., Levy, L.S., Takahashi, M.: Tree adjunct grammars. Journal of Computer and System Sciences 10(1), 136–163 (1975)
12. Kaljurand, K.: Paraphrasing controlled English texts. In: Pre-Proceedings of the Workshop on Controlled Natural Language (CNL 2009). CEUR Workshop Proceedings, vol. 448, CEUR-WS (April 2009)
13. Kaplan, R.M., Bresnan, J.: Lexical-functional grammar: A formal system for grammatical representation. In: Bresnan, J. (ed.) The Mental Representation of Grammatical Relations, pp. 173–281. MIT Press (1982)
14. Knuth, D.E.: Backus normal form vs. backus naur form. Communications of the ACM 7(12), 735–736 (1964)
15. Kuhn, T.: How controlled English can improve semantic wikis. In: Proceedings of the Forth Semantic Wiki Workshop (SemWiki 2009). CEUR Workshop Proceedings, vol. 464, CEUR-WS (2009)
16. Kuhn, T.: Codeco: A grammar notation for controlled natural language in predictive editors. In: Pre-Proceedings of the Second Workshop on Controlled Natural Languages (CNL 2010). CEUR Workshop Proceedings, vol. 622, CEUR-WS (2010)
17. Kuhn, T.: Controlled English for Knowledge Representation. PhD thesis, Faculty of Economics, Business Administration and Information Technology of the University of Zurich (2010)
18. Kuhn, T., Schwitter, R.: Writing support for controlled natural languages. In: Proceedings of the Australasian Language Technology Association Workshop 2008, pp. 46–54 (December 2008)
19. Lappin, S., Leass, H.J.: An algorithm for pronominal anaphora resolution. Computational Linguistics 20(4), 535–561 (1994)
20. Martin, P.: Knowledge Representation in CGLF, CGIF, KIF, Frame-CG and Formalized-English. In: Priss, U., Corbett, D.R., Angelova, G. (eds.) ICCS 2002. LNCS (LNAI), vol. 2393, pp. 77–91. Springer, Heidelberg (2002)
21. Naur, P., Backus, J.W., Bauer, F.L., Green, J., Katz, C., McCarthy, J., Perils, A.J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J.H., van Wijngaarden, A., Woodger, M.: Revised report on the algorithmic language ALGOL 60. Communications of the ACM 6(1), 1–17 (1963)

22. Pereira, F., Warren, D.H.D.: Definite clause grammars for language analysis. In: Readings in Natural Language Processing, pp. 101–124. Morgan Kaufmann Publishers (1986)
23. Pollard, C., Sag, I.: Head-Driven Phrase Structure Grammar. Studies in Contemporary Linguistics. Chicago University Press (1994)
24. Power, R., Stevens, R., Scott, D., Rector, A.: Editing OWL through generated CNL. In: Pre-Proceedings of the Workshop on Controlled Natural Language (CNL 2009). CEUR Workshop Proceedings, vol. 448, CEUR-WS (April 2009)
25. Schwitter, R., Kaljurand, K., Cregan, A., Dolbear, C., Hart, G.: A comparison of three controlled natural languages for OWL 1.1. In: Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Directions. CEUR Workshop Proceedings, vol. 496, CEUR-WS (2008)
26. Schwitter, R., Ljungberg, A., Hood, D.: 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-CLAW 2003), Ireland, pp. 141–150. Dublin City University (2003)

Controlled Language for Everyday Use: The MOLTO Phrasebook

Aarne Ranta, Ramona Enache, and Grégoire Détrez

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract. Controlled languages are usually targeted for technical domains and designed to be unambiguous. This paper presents a controlled language whose domain is touristic phrases, aimed to be usable by anyone without prior training. Despite its informal nature, the language of phrases has a firm notion of semantics, defining the correctness of translations. However, this semantics is formulated in terms of context and situation rather than by logical formulas. Moreover, the language is often ambiguous, and the translation may depend on resolving the ambiguity. This paper shows how to formalize a semantics for tourist phrases and implement it in 15 languages, how to deal with the ambiguities, and how to make the system available for layman users on the web and on mobile phones. While a useful application as such, the Phrasebook also paves the way for an extended notion of controlled language, and the techniques are aimed to be general enough to support many such extensions.

Keywords: controlled language, Grammatical Framework, multilingual grammar, tourist phrasebook, mobile translation application.

1 Introduction

Controlled languages are typically designed for use on technical domains. Their users are experts such as aircraft engineers [1], medical doctors [2], and topographers [3]. The language is typically a natural-language image of a formal system, such as predicate logic in [4] or OWL (Web Ontology Language) [5] in [6]. The purpose of these controlled languages is to support knowledge representation, reasoning, and mechanical checking of correctness; the main point of using a natural language fragment rather than a formalism is to have a notation that is readable without special training. When there is no underlying formalism, as in [1], the purpose is to eliminate the ambiguity, vagueness, and unclarity of uncontrolled natural language.

However, the notion of controlled language can be given a wider interpretation: it can be just *any* fragment of natural language specified with a formal set of rules. Actually it can be seen as the technological counterpart of Wittgenstein's philosophical notion of **language games** [7], which are systems of rules specifying how language is used for performing different tasks. In the tradition represented by Wittgenstein and his followers, language games are the very essence of language: they should *not* be seen as mere fragments of an underlying total system, but as the building blocks that actually constitute the thing called language. Very little can be rigorously said about natural

M. Rosner and N.E. Fuchs (Eds.): CNL 2010, LNAI 7175, pp. 115–136, 2012.

language as a whole, whereas these limited fragments are units that (at least in many cases) permit a formal description and—consequently—a computer implementation.

When we start looking at language from the language game point of view, we suddenly begin to see "formal systems" everywhere. One of the most basic ones is the social game of greetings and politeness phrases. For instance, when I ask for something, I attach the word *please*. When you hand it over to me, you say *here we are*, to which I should say *thank you*, and you can conclude by replying *you're welcome*. These four phrases get their precise meanings in the context of this game. Actually, each of them could be used in some other context and mean something different. This is seen clearly when we look at their translations. Here is a simple dialogue in three languages:

English	**Swedish**	**German**
A beer please.	*En öl tack.*	*Ein Bier bitte.*
Here we are.	*Var så god.*	*Bitte.*
Thank you.	*Tack.*	*Danke.*
You're welcome.	*Var så god.*	*Bitte.*

English makes most distinctions here, by using a different phrase for each of the four moves of the game. Swedish uses *tack* for both asking and thanking. German uses different phrases for these two, but the word *bitte* (literally, "I request") is otherwise used for everything! Nevertheless, there's no problem in translating Swedish and German phrases to English, *as soon as we know what move they express in the language game*.

Of course, it is just a coincidence that English has unambiguous phrases for all language game moves here. English, and all other languages, are full of ambiguities, if we look only at the syntax without context. This is not just a feature of everyday language but even of mathematics, as convincingly shown in [8]. But the ambiguities are almost always easily resolved by looking at the context of use.

An ambiguity specific to English is generated from the word *you*. It has two translations in Swedish (the familiar singular *du*, the plural or formal singular *ni*), three in German (the familiar singular *du*, the familiar plural *ihr*, and the formal *Sie*), and up to eight in languages like Spanish (singular/plural, familiar/formal, masculine/feminine). For instance, the English phrase *are you German* has eight translations in Spanish. The translation is determined by the context of use—basically, by the addressee.

The "language game" of social phrases is not only a philosophical experiment, but also a lucrative business. Phrasebooks like Berlitz and Lonely Planet are still sold in millions of copies, although electronic phrasebooks running on mobile phones are taking more and more of the market share. A typical electronic phrasebook is just a digital version of the printed book: a collection of phrases that can be looked up either by typing search strings or by browsing in hierarchic menus. A particularly smart example is the Chinese iPhone application YoChina[1], which puts each phrase into a context and also shows a set of responses from which the interlocutor can choose.

Even the most sophisticated commercial phrasebooks are still just collections of **canned phrases**: fixed strings, which, even though there might be thousands of them, don't cover all possible combinations of the concepts involved. A different approach can

[1] http://www.yocoy.com

be taken by using **machine translation**; thus Google Translate[2] is available as a mobile phone application that actually translates each individual phrase separately. While this is the most natural and powerful approach to the problem, it still has open issues. The first issue is quality: even though Google Translate often does a good job, it can just as easily produce something totally wrong, and this can lead to embarrassing situations if used in a social context of communication (mostly resolved by a good laugh, of course). In particular, Google Translate is based on a generic, statistical language model which cannot make distinctions like the ones needed for the different uses of German *bitte*. The second issue with Google Translate as used by a traveller far away from home is the cost of mobile data transfer. It may just be too expensive to use the service.

In this paper, we will introduce a controlled language translator approach to tourist phrasebooks. We will show a **formal semantic model**, which unambiguously specifies an infinite class of phrases. Then we will show how the semantic model is translated to phrases in 15 natural languages. The translations are reversible, which means that the phrasebook can both generate natural language from the formal semantics and interpret it in the formal semantics. The combination of generation and interpretation is translation; our phrasebook is able to translate equally well with all of the 14*15=210 language pairs. The translator runs as an off-line application on Android mobile phones and can be downloaded free of charge from Android Market[3]. The phrasebook is also available as a web application[4].

Figure 1 (left) shows the web interface to the phrasebook. The user has constructed the English sentence *how far is the Russian restaurant*, which the system has translated to the other 14 languages. The construction is carried out by a **predictive parser** [9], which predicts the set of possible next words at each point. The input can be made by typing text (in the white slot on the right) or by clicking at one of the rectangles showing a word. The possible continuations here are *?* (to terminate the phrase), *by* (as in *by tram*), and *from* (as in *from the hotel*). As soon as there is enough input to translate, the translations are shown. Figure 1 (right) shows the Android mobile application. For size reasons, the application shows only one target language at a time. As a bonus, it has speech synthesizer output for some languages.

Touristic phrases are a rich domain, and one could easily spend a lifetime on building, refining, and extending an electronic phrasebook. What we want to show in this paper is a technology that gives maximal support to this work. The technology is based on GF (Grammatical Framework, [10]), which is a grammar formalism designed for supporting multilingual grammars of controlled languages. In addition to a programming language, GF provides RGL (Resource Grammar Library, [11]), which encapsulates the low-level linguistic knowledge of morphology and syntax that is needed when building high-quality translation systems.

In addition to the grammar engineering tools, GF has a set of tools supporting run-time applications. These include libraries for web servers and clients [12] and, most importantly for the current purpose, a Java-based run-time system for Android phones.

[2] http://translate.google.com

[3] https://market.android.com/details?id=org.grammaticalframework.android.apps.phrasedroid

[4] http://www.grammaticalframework.org/demos/phrasebook/

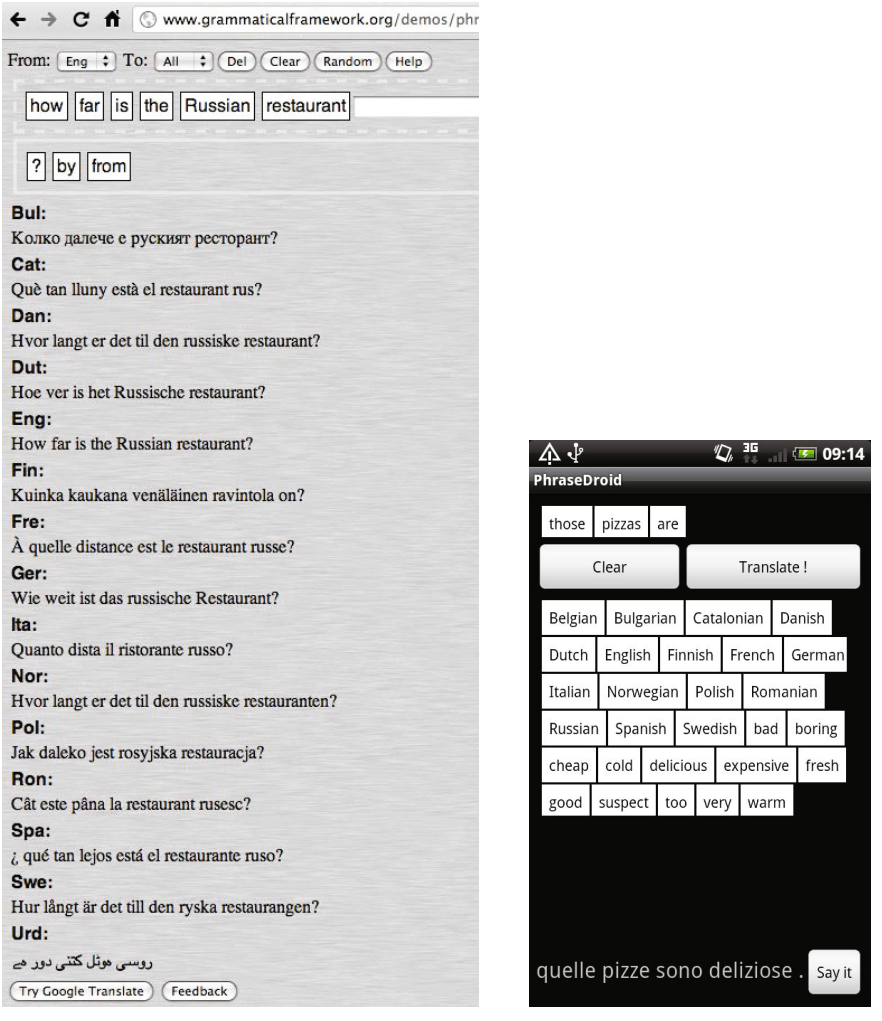


Fig. 1. The MOLTO Phrasebook as a web application (left) and as an Android mobile application (right)

Thus, at the same time as the phrasebook is a practical help for tourists, it is a showcase for a powerful general technology. This technology is being developed in the European MOLTO project [5]). In addition to using GF, MOLTO explores ways to use statistical translation models to help the construction and improve the coverage of grammar-based systems. The MOLTO Phrasebook is a first experiment of this: some of the languages involved were implemented by programmers not knowing the language at all, but using a statistical model to bootstrap the grammar and a native-speaker

[5] MultilingualOn-LineTranslation, http://www.molto-project.eu

informant to evaluate it. This was developed into a general method that will be usable for any further project of building multilingual controlled language systems.

The structure of the paper is as follows: Section 2 specifies the coverage of the MOLTO phrasebook by giving an overview of its semantic model. Section 3 shows examples of how the different languages are implemented by using GF and RGL. Section 4 shows how ambiguities are displayed to users by means of disambiguation grammars. Section 5 introduces the method of example-based grammar writing using statistical models and human informants. Section 6 explains the Java run-time system of GF and the architecture of the mobile Android application. Section 7 presents some results from evaluation, and Section 8 concludes.

2 The Semantic Model

In GF, a semantic model is called an **abstract syntax**. It is defined by giving a set of **categories** (keyword `cat`) and a set of **functions** (keyword `fun`), which together define the notion of **well-typed trees**. For instance, the phrases in the beer-ordering dialogue above can be given the following abstract syntax:

```
cat
  Phrase ; Item
fun
  GivePlease    : Item -> Phrase
  HereWeAre     : Phrase
  ThankYou      : Phrase
  YouAreWelcome : Phrase
  ABeer         : Item
```

The model could be made more precise by specifying that these phrases must appear in a certain order to constitute a valid dialogue. But for the purposes of a phrasebook, it is enough to specify uniquely each type of phrase by giving it a function name. All functions in this simple model are actually **constants**, i.e. they take no arguments—except `GivePlease`, which takes an `Item` as its argument.

The linguistic realizations of the semantic model are specified by a **concrete syntax**, which tells how trees formed in abstract syntax are **linearized** into strings in different languages. We will return to the details of linearization in Section 3; just to give an example, the following linearization rules (`lin`) could be given for German:

```
lin
  GivePlease item = item ++ "bitte"
  HereWeAre       = "bitte"
  ThankYou        = "Danke"
  YouAreWelcome   = "bitte"
  ABeer           = "ein Bier"
```

All linearization rules in GF can be also used for **parsing**, that is, converting strings to trees. This tiny example clearly shows that parsing can be **ambiguous**, that is, return more than one tree. The everyday counterpart of parsing ambiguity is shown by the

situation where someone asks: "What is *bitte* in English?" The correct answer is that it depends on context: it may mean *please* or *here we are* or *you are welcome*.

In the full MOLTO Phrasebook, none of the 15 languages is unambiguous. What we need is an abstract syntax that formalizes all possible distinctions, so that each abstract syntax tree has a unique linearization in every language. Now, capturing all relevant distinctions in 15 languages might sound like a hopeless task, but in fact the semantic model scaled up quite well when the grammar was extended language by language. After a careful initial design (with awareness of what typically happens in languages), almost no changes were needed in the abstract syntax when new languages were added.

Printed phrasebooks have canned, static phrases, whereas a digital grammar-based phrasebook has rules for forming phrases from smaller expressions. The MOLTO Phrasebook has 42 categories and 290 functions. Of the functions, 130 take arguments and 160 are constants, which means that they are either lexical items or canned phrases. What is a lexical item in one language can be a multiword phrase in another language, as shown for instance by *bitte* vs. *here we are*. The number of phrases is infinite because of recursion, but on the reasonable level of tree depth 3, the Phrasebook has 484,938 abstract syntax trees of phrases.

The full code of the phrasebook, with some documentation, can be found on-line[6]. We will here show a sample of the coverage, and then focus on a few interesting problems created by some of the constructions. Table 1 gives some of the categories, and Table 2 some of the combination functions.

For a detailed sample, let us focus on the category `Action`, and the ways of asking persons for information about themselves and what they do. The complete `Phrase` corresponding to the question

Are you Swedish?

has the tree (in GF's LISP-like notation)

```
PQuestion (QProp (PropAction
  (ACitizen YouFamMale (CitiNat Swedish))))
```

This tree is formed by the functions

```
PQuestion  : Question -> Phrase
QProp      : Proposition -> Question
PropAction : Action -> Proposition
ACitizen   : Person -> Citizenship -> Action
YouFamMale : Person
CitiNat    : Nationality -> Citizenship
Swedish    : Nationality
```

But thinking in terms of reliable translations, there are many more trees, resulting from the semantic ambiguity of English *you*. Of these, the Phrasebook deals with dimensions

[6] http://www.grammaticalframework.org/examples/phrasebook/doc-phrasebook.html

Table 1. Some of the 42 categories of the Phrasebook

category	**explanation**	**example**
Phrase	complete phrase, unit of translation	*Where are you?*
Greeting	idiomatic greeting	*hello*
Sentence	declarative sentence	*I am in the bar*
Question	question, either yes/no or wh	*where are you*
Proposition	can be used as sentence or question	*this pizza is good*
Object	object of wanting, ordering, etc	*two pizzas and a beer*
Item	a single entity	*this pizza*
Kind	a type of an item	*pizza*
Quality	qualification of an item	*very good*
Place	location	*the bar*
PlaceKind	type of location	*bar*
Person	agent wanting or doing something	*you*
Action	proposition about a Person	*you are here*
Nationality	complex of language, property, country	*Swedish, Sweden*
Language	language (can be without nationality)	*Flemish*
Citizenship	property (can be without language)	*Belgian*
Country	country (can be without language)	*Belgium*
Currency	currency	*Swedish crown*
Number	number expression in words	*two hundred and five*
Price	price (number + currency)	*sixty-five dollars*

of gender and politeness; plural *you* is not covered by the current version (mostly because it is not so frequently needed). Thus *you* corresponds to four constants of type `Person`,

```
YouFamMale, YouFamFemale, YouPolMale, YouPolFemale
```

Varying this constant in the above tree gives four French linearizations:

YouFamMale: *Est-ce que tu es suédois ?*
YouFamFemale: *Est-ce que tu es suédoise ?*
YouPolMale: *Est-ce que vous êtes suédois ?*
YouPolFemale: *Est-ce que vous êtes suédoise ?*

Although German also has gender, it makes no difference in this example. Thus we obtain

YouFamMale, YouFamFemale: *Bist du schwedisch?*
YouPolMale, YouPolFemale: *Sind Sie schwedisch?*

One challenge in the Phrasebook is to communicate the ambiguities to the end user: when she types in *Are you Swedish?*, she should get a list of the alternatives in the desired target language, with explanations that enable her to decide which alternative to choose in her situation of use. We will return to this question in Section 4.

As `Action` is a subcategory of `Proposition`, it can be used for both questions and assertions, both positive and negated. Thus the functions involved in the question

Table 2. Some of the 130 combination rules of the Phrasebook

arguments	**value**	**examples**
Number, Kind	Object	*five pizzas*
Quality, Kind	Kind	*Italian pizza*
Kind	Item	*this pizza*, *the pizzas*
PlaceKind	Place	*the bar*, *a bar*
Proposition	Sentence	*the bar is open*, *the bar isn't open*
Proposition	Question	*is the bar open*
Action	Proposition	*I speak Polish*
Person, Object	Action	*you have beer*, *you have no beer*
Person, Citizenship	Action	*you are German*
Person, Place	Action	*you are in the bar*
Person, Sentence	Action	*you know that I am in the bar*
Person, Person	Action	*you know my wife*
Person, Question	Action	*you know how far the bar is*
Person, Number	Action	*I am seventy years old*
Person, Number	Action	*I have six children*
Person, Name	Action	*my name is Bond*
Person	Action	*I am hungry*
Person, Item	Action	*I like this pizza*
Person, Country	Action	*I live in Sweden*
Person, Language	Action	*I speak Polish*
Person, Currency	Action	*I have Swedish crowns*
Person, Object	Action	*I want two apples*
Person, Place	Action	*I want to go to the hospital*
Person	Question	*how old are you*
Item	Question	*how much does the pizza cost*
Item, Price	Proposition	*the pizza costs five euros*
Place	Proposition	*the museum is open*
Place, Date	Proposition	*the museum is open today*
Place, Day	Proposition	*the museum is open on Mondays*
Place, Date	Greeting	*see you in the bar on Monday*
Person	Person	*my wife, your husband*
Number, Currency	Proposition	*five euros*
Place	Question	*how far is the zoo*
Place, Place	Question	*how far is the centre from the hotel*
Transport, Place	Question	*which bus goes to the hotel*

can be reused for sentences like *I am not Swedish*, which has two French translations. In general, the design of the abstract syntax follows two principles, which can be explained via geometrical metaphors:

- **Convexity:** For any two phrases contained, also all phrases "between" them (i.e. combining their concepts in different ways) are contained. This principle guarantees that the users can easily learn what to expect from the phrasebook, and their expectations will be fulfilled.
- **Orthogonality:** Phrases are built from the least number of independent components.

While convexity is a great help for the user of the phrasebook, orthogonality helps the developer by giving her the minimum of concepts to implement for each language. A user who knows that the Phrasebook contains the property *Swedish* and the country *France* will, by convexity, expect it also to contain the property *French* and the country *Sweden*. The category `Nationality` is used to guarantee this, as it collects triples of language, property, and country. These triples can often be formed by a systematic word formation mechanism (e.g. *Swedish, Swedish, Sweden*), which helps the developer.

As a downside, abstract concepts like `Nationality` may be more complex to implement than more specific concepts like `Language`, `Citizenship`, and `Country`.[7] Often there is no regular word formation mechanism, and there are countries and languages that do not fit into the "national state" concept. For instance, the languages spoken in Belgium are Flemish and French. Thus in the Phrasebook, *Belgium* is a country without a lexically associated language, whereas *Flemish* is a language without a lexically associated country.

The set of combination rules in the Phrasebook is quite useful as it is, but the set of lexical items is still small and a little random. Therefore an obvious next step in developing the Phrasebook is to add words for drinks, food, nationalities, places, and so on. Keeping all this in synchrony for 15 simultaneous languages is not trivial.

3 Concrete Syntax

Constant phrases, such as *thank you* and *please*, are easy to define for all languages. Combination rules are more tricky: even in the small fragment covered by the Phrasebook, linguistic problems such as inflection, agreement, and word order arise, and require expertise in the grammar of each of the target languages. Fortunately for the Phrasebook, this expertise was readily available in the GF Resource Grammar Library (RGL). This is of course not just a lucky coincidence—it is more proper to say that the Phrasebook was built as a showcase of GF in general and of the RGL in particular.

A concrete syntax has two components. One is linearization rules (`lin`) as shown above, telling how abstract trees are mapped into strings. The other one is the **linearization types** of categories (`lincat`). These types are linguistic categories such as sentences, noun phrases, and adjectives. In the `lin` rule examples so far, only one linearization type was used: the type `Str` of strings. But this is usually not enough. For instance, to account for all combinations of a German noun, we need the type

```
{s : Number => Case => Str ; g : Gender}
```

that is, a record with a string depending on number and case (the component `s`), and a gender (component `g`). In other languages, nouns can have other linearization types, and the features number, case and gender can get other values than in German. But in RGL, all this complexity is defined internally, and the user only needs to know that the type `CN` covers common nouns in all RGL languages.[8]

[7] The terminological choice between "Nationality" and "Citizenship" is of course arbitrary, and only an implementation detail not visible to the end user.

[8] See `http://grammaticalframework.org/lib/doc/synopsis.html` for RGL categories and functions.

To give a sample of linearization types used in the Phrasebook, let us consider the categories needed in the example *Are you Swedish?*:

category	**linearization type**	**explanation**
Phrase	`Text`	text
Question	`QS`	question
Proposition	`Cl`	clause
Person	`NP`	noun phrase
Action	`Cl`	clause
Citizenship	`A`	adjective
Nationality	`{l : NP ; p : A ; c : NP}`	NP, adjective, NP

All these types are standard linguistic categories of RGL, except the one of `Nationality`, which uses a record consisting of a language noun phrase `l`, a property adjective `p`, and a country noun phrase `C`. This record is, so to say, the linguistic representation of the complex concept of a nationality, thus representing a **lexical family**.

The linearization rules are specified by RGL functions, most of which have the name `mk`*C* for the value category *C*. Thus we have

```
PQuestion q = mkText q
QProp p = mkQS (mkQCl p)
PropAction a = a
ACitizen p c = mkCl p c
YouFamMale = youSg_Pron
CitiNat n = n.p
Swedish = mkNationality "Sweden" "Swedish"
```

The last rule uses the operation `mkNationality`, which takes a string for a noun and for an adjective, to form the country name from the noun (*Sweden*) whereas both the property and the language use the adjective (*Swedish*). This is the only English-specific rule in this set. Other languages have different ways of defining this lexical family. Finnish, for instance, uses the country name as the language name, just spelled with a small initial.

Another example of a lexical family is types of locations. They are defined

```
PlaceKind = {name : CN ; at : Prep ; to : Prep}
```

Thus places have associated prepositions, used for expressing location and direction. For instance, in English we have *in the bar*, *at the station* for the location and *to* expressing the direction for both. In Finnish, prepositions are expressed by cases, so that "bar" uses so-called internal cases (*baarissa* "in the bar" inessive, *baariin* "to the bar" illative) whereas "station" uses external cases (*asemalla* "at the station" adessive, *asemalle* "to the station" allative). Sometimes even more fine-grained distinctions are needed; for instance, in Swedish "to the toilet" is expressed as *på toaletten* in phrases relating to the function ("I want to go to the toilet"), whereas phrases expressing pure direction say *till toaletten*.

The prepositions are thus stored in the record as lexical properties of the places; they are idiomatic in each language and highly unpredictable. GF provides ways to express

them on a reasonably high level, so that just the minimal information need be given in the lexicon: thus in Finnish, we just need the noun and an identifier `ssa` or `lla` which is conventionally used for indicating the type of local case:

```
Bar = mkPlace (mkN "baari") ssa
Station = mkPlace (mkN "asema") lla
```

To determine this little piece of information—the proper case or preposition for each location—is linguistic knowledge that turned out to be possessed only by native speakers, who made several corrections to the initial grammars.

As the RGL has a common API for the syntax functions of the 18 languages included, combination rules in application grammars can in principle be expressed by code that is common to all languages. This is technically implemented by the use of **functors** ([10], chapter 5), and it is the technique used, for instance, in the GF implementation of Attempto Controlled English [13]. The use of a functor means that the languages use the same syntactic structures to express the meanings. For instance, all languages in the Phrasebook use an equivalent of *you know that I am in the bar* to express this proposition. However, the Phrasebook domain is particularly rich of idioms that the languages express by different syntactic means. This was a challenge we expected, and one of the reasons why the Phrasebook was an interesting case study for multilingual translation in the first place. Thus, of the 130 combination rules, only 96 (74%) are implemented by a functor; usually the percentage is close to 100.

Some typical examples of non-functorial expressions are the following:

- *I am fifty years old*: French *j'ai cinquante ans* ("I have fifty years").
- *my name is Bond*: German *ich heisse Bond* ("I have-name Bond"), French *je m'appelle Bond* ("I call myself Bond").
- *I am hungry*: French *j'ai faim* ("I have hunger"), Finnish *minun on nälkä* ("of-me is hunger").
- *I like this pizza*: Italian *questa pizza mi piace* ("this pizza pleases me").
- *I am married*: Finnish *olen naimisissa* ("I am in-marriage", with a special adverbial).
- *how old are you*: French *combien d'ans as-tu* ("how many years do you have").
- *how far is the station*: French *à quelle distance est la gare* ("at what distance is the station"), Italian *quanto dista la stazione* ("how much does the station distance", with a special verb).

Most of these variations are clustered in systematic ways, so that for instance all Romance languages use the same structure and all Germanic languages (except perhaps English) another structure. The construct *how* with an adjective or adverb does not exist in Romance languages, and is hence not even a part of the RGL API.

4 Ambiguity and Disambiguation

The abstract syntax is by definition unambiguous. Therefore the main way in which a grammar developer can analyse the ambiguity of a string is by inspecting the abstract

syntax trees. But this device is of course not appropriate for a tourist phrasebook: it would be awkward and often useless to show the syntax trees to the user.

Fortunately, the technique of multilingual grammars provides a straightforward, declarative way to display ambiguities: one can write for each language a special concrete syntax, which is like the original grammar except that it eliminates its ambiguities by using alternative (although less idiomatic and often longer) expressions—a **disambiguation grammar**. For example, the original English grammar linearizes each of the four abstract variants of *you* as just *you*, but the disambiguation grammar attaches an explanation in parentheses: *you (familiar,male)*, *you(polite,female)*, etc. This idea is inspired by the notion of **feedback texts** of the WYSIWYM system [14].

The implementation of a disambiguation grammar can be written on top of the base grammar by using **restricted inheritance**: it inherits everything from the base grammar except those rules that need disambiguation. Those rules can then be replaced by other rules. The following module is a complete code for a disambiguation grammar for the phrasebook dealing with the four *you*'s. The unambiguous variants are formed from *you* by attaching an adverbial to it.

```
concrete DisambPhrasebookEng of Phrasebook = PhrasebookEng -
  [YouFamMale, YouFamFemale, YouPolMale, YouPolFemale]
  ** open SyntaxEng, ParadigmsEng in {
lin
  YouFamMale   = mkNP you_NP (mkAdv "(familiar,male)") ;
  YouFamFemale = mkNP you_NP (mkAdv "(familiar,female)") ;
  YouPolMale   = mkNP you_NP (mkAdv "(polite,male)") ;
  YouPolFemale = mkNP you_NP (mkAdv "(polite,female)") ;
}
```

In the full Phrasebook, the number of ambiguous constructs is between 10 and 20 for each language. An ambiguity shared by all languages is the notion of the currency *crown*, as used for the currency of different Scandinavian countries. In the normal usage, one says *crown* rather than e.g. *Danish crown*, if it is clear from the context that one is speaking about Danish crowns. The implementation of this does not use the disambiguation grammar, because both expressions make sense in the base grammar as well. Thus the base grammar defines crowns by using the **variants** construct of GF (expressed by |):

```
DanishCrown =
  mkCN (mkA "Danish") (mkN "crown") | mkCN (mkN "crown")
SwedishCrown =
  mkCN (mkA "Swedish") (mkN "crown") | mkCN (mkN "crown")
```

and similarly in all languages.

Since the abstract syntax encodes all interpretations that are relevant in any of the languages, it can lead to **spurious ambiguities** when applied to any particular language pairs. For instance, the familiar *you* is *sinä* and the polite *you* is *Te* in Finnish, without the gender distinction involved anywhere in the sentence. Hence, when translating from English to Finnish, only two alternatives should be displayed. The same thing may

happen in Italian, where the masculine and feminine forms of some adjectives are the same. Thus *are you Swedish* has only two translations (*sei/è svedese*) even though *are you Italian* has four (*sei/è italiano/italiana*).

In the Phrasebook, the user should of course not see spurious ambiguities but only relevant ones. This is guaranteed by the following modification of GF's translation algorithm, which otherwise shows as many translation strings as there are parse results. Each translation is equipped by the set of those disambiguating expressions that give rise to it. The translation algorithm is as follows:

1. parse the source sentence to obtain trees $t_1, \ldots, t_n$
2. for each target language L_i:
   (a) for each tree t_j: linearize t_j in L_i
   (b) group trees with the same linearization s_k into the pair $< s_k, \{t \mid t^* = s_k\} >$
   (c) return each s_k together with the linearizations of the associated trees in the disambiguation grammar of the target language

Here is an example of the algorithm at work:

English input:
- *Are you Swedish?*

French output:
- *Est-ce que tu es suédois ?* (Are you (Familiar,Male) Swedish?)
- *Est-ce que tu es suédoise ?* (Are you (Familiar,Female) Swedish?)
- *Est-ce que vous êtes suédois ?* (Are you (Polite,Male) Swedish?)
- *Est-ce que vous êtes suédoise ?* (Are you (Polite,Female) Swedish?)

Italian output:
- *Sei svedese?* (Are you (Familiar,Male) Swedish? / Are you (Familiar,Female) Swedish?)
- *È svedese?* (Are you (Polite,Male) Swedish? / Are you (Polite,Female) Swedish?)

As a further optimization, the algorithm could compress the alternatives (Familiar,Male) and (Familiar,Female) to just (Familiar). This would be helped by a disambiguation grammar that has more structure than just the unanalysed strings in parentheses. One could also achieve this by some hand-written code in the phrasebook application; however, this would be against the purpose of developing the Phrasebook as a show-case for a general technology.

5 Example-Based Grammar Writing

In previous projects, the typical author of a GF concrete syntax is fluent in the target language and has GF skills which are directly proportional to the complexity of the abstract syntax to implement. However, when dealing with 15 languages and a reasonably rich semantic interlingua, the task of finding such people is a difficult one. When adding the time constraints yielded by the MOLTO deadlines and the time needed to improve a native speaker's GF skills or a GF programmer's knowledge of a language that she had little to no skill in before, the task seemed to be a mission impossible. This was the case

for German, Dutch, Danish, Norwegian and Polish. As a solution to this, we devised the example-based grammar learning system, that is meant to automate a significant part of the grammar writing process and ease grammar development. The two main usages of the system are, first, to reduce the amount of GF programming necessary in developing a concrete grammar, and, secondly and more importantly, to make the extraction of certain features of a language automatic for grammar development.

In the last years, the GF community has constantly increased and so has the number of languages in the resource library and the number of application grammars using them. The writer of a concrete application grammar is typically different from the writer of the resource grammar for the same language, has less GF skills and is most likely unaware of the almost 300 constructors that the resource grammars implement for building various syntactical constructions [11]. In order to hide this detail, an API is provided so that the domain grammar writer only needs to know the GF categories and look up how they can be built from each other.

For example, the sentence *my name is John* is parsed to the following abstract syntax tree:

```
PredVP (DetCN (DetQuant (PossPron i_Pron) NumSg)
  (UseN name_N)) (UseComp (CompNP (UsePN john_PN)))
```

If we use the API constructors, the abstract syntax tree is simpler and more intuitive, as the structure is flatter and each function has an easily memorable name:

```
mkCl (mkNP (mkQuant i_Pron) name_N) (mkNP john_PN)
```

The example-based grammar learning system aims to make one step more in this direction and reduce the need for using even the API functions. The key idea is based on parsing, followed by compilation to API. It provides considerable benefits, especially for idiomatic grammars such as the Phrasebook, where the abstract syntax trees are considerably different.

For example, when asking for a person's name in English the question *what is her name* has the syntax trees shown above. On the other hand, in French the question would be translated to *je m'appelle John* (literally, "I call myself John"), which is parsed to:

```
PredVP (UsePron i_Pron)
  (ComplSlash (SlashV2 appeler_V2) (UsePN john_PN))
```

and corresponds to the following API abstract tree:

```
mkCl i_NP appeler_V2 (mkNP john_PN)
```

By replacing `i_NP` and `john_PN` with variables, this tree can be used as the linearization of a two-place predicate:

```
lin HasName x y = mkCl x appeler_V2 (mkNP y)
```

Figure 2 shows the algorithm for example-based grammar writing. It shows the construction steps of the concrete syntax of the Phrasebook grammar for the language *X*,

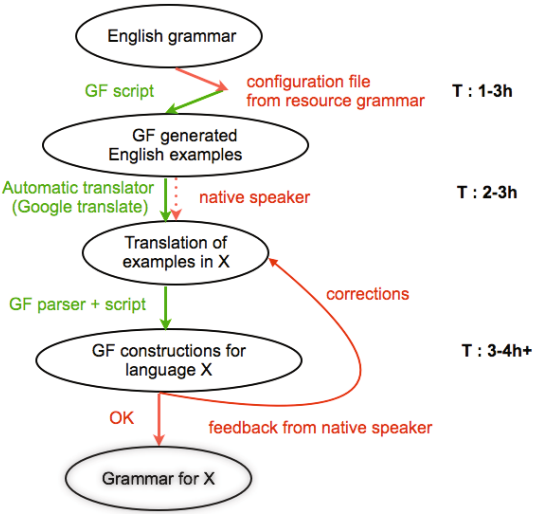


Fig. 2. The example-based grammar learning schema

where the developer has basic or no skills in the language. In our experiment *X* was one of Danish, Dutch, German, Norwegian, and Polish. The arrows represent the main steps of the process, whereas the circles represent the initial and final results after each step of the process. For every step, the estimated time is given. This is variable and greatly influenced by the features of the target language and the semantic complexity of the phrases and would only hold for the Phrasebook grammar.

Initial Resources

- English Phrasebook
- resource grammar for X
- script for generating the inflection forms of words and the corresponding linearizations of the lexical entries from the Phrasebook in the language X. For example, in the case of the nationalities, since we are interested in the names of countries, languages and citizenship of people and places, we would generate constructions like "I am English. I come from England. I speak English. I go to an English restaurant" and from the results of the translation we will infer the right form of each feature. In English, in most cases there is an ambiguity between the name of the language and the citizenship of people and places, but in other languages all three could have completely different forms. This is why it is important to make the context clear in the examples, so that the translation will be more likely to succeed. The correct design of the test of examples, is language dependent and assumes analysis of the resource grammar, also. For example, in some languages we need only the singular and the plural form of a noun in order to build its GF representation, whereas in other languages such as German, in the worst case we would need 6 forms which need to be rendered properly from the examples.

– script for generating random test cases that cover all the constructions from the grammar. It is based on the current state of the abstract syntax and it generates for each abstract function some random parameters and shows the linearization of the construction in both English and language *X*, along with the abstract syntax tree that was generated.

Example-Based Concrete Grammar Learning Algorithm

– Step 1: **Analysis of the target grammar and lexicon acquisition**
The first step assumes an analysis of the resource grammar and extracts the information needed by the functions that build new lexical entries. A model is built so that the proper forms of the word can be rendered, and additional information, such as gender, can be inferred. The script applies these rules to each entry that we want to translate into the target language, and one obtains a set of constructions.
– Step 2: **Generation of examples in the target language**
The generated constructions are given to an external translator tool (Google translate) or to a native speaker for translation. One needs the configuration file even if the translator is human, because formal knowledge of grammar is not assumed.
– Step 3: **Parsing and decoding the examples with GF**
The translations into the target language are further more processed in order to build the linearizations of the categories first, decoding the information received. Furthermore, having the words in the lexicon, one can parse the translations of functions with the GF parser and generalize from that.
– Step 4: **Evaluation and correction of the resulting grammar**
The resulting grammar is tested with the aid of the testing script that generates constructions covering all the functions and categories from the grammar, along with some other constructions that proved to be problematic in some language. A native speaker evaluates the results and if corrections are needed, the algorithm runs again with the new examples. The examples validated by the native informant are kept for regression testing of the future results. The algorithm is repeated as long as corrections are needed.

It is worth noting that the time needed for preparing the configuration files for a grammar will not be repeated, since the files are available for future usage. The time for the second step can be saved if automatic tools, like Google translate are used. This is only possible in languages with large corpora available. Good results were obtained for German and Dutch with Google translate, but for languages like Polish, which are both complex and lack enough resources, the results are discouraging. If the statistical oracle works well, the only step where the presence of a human translator is needed is the evaluation and feedback step. An average of 4 hours per round and 2 rounds were needed for the languages for which we performed the experiment. The final results are comparable to a grammar developed by a native speaker GF programmer.

However, one can already remark that the success of this method also depends highly on the lexicon acquisition, which we perform in the first step. What is more is that the lexicon is language-dependent, and is not alignable. Also, without previous knowledge of all the languages, one cannot foresee what words we would need to use, and since

they are not used in all languages, it wouldn't make sense to have them all in a multilingual aligned lexicon. For the moment, this task was solved by either guessing the correct part-of-speech based on a similar concrete grammar already developed(for example Danish and Norwegian were bootstrapped from Swedish) or by having the lexicon built and POS-tagged with the aid of native informants.

Among the 5 languages considered, a concrete Phrasebook grammar was successfully built for Danish, Dutch, German and Norwegian, whereas for Polish, it was not possible to get through the first and most difficult step—target grammar analysis and lexicon acquisition, because of the complex morphology of the language and the lack of available resources. In the end the concrete grammar was developed by the writer of the resource grammar.

The experiment involved 7 programmers with basic or advanced GF skills that wrote 10 resource grammars, whereas for the 4 languages mentioned before, the example-based algorithm was used. The approximate development total time is 1 person month for the whole Phrasebook, or 1.5 days per language on the average.

Based on this case study, we roughly estimated the effort used in constructing the necessary sources for each new language and compiled Table 3.

Table 3. Development effort for the Phrasebook

Language	Fluency	GF skills	Inf. dev.	Inf. testing	Ext. tools	RGL edits	Effort
Bulgarian	###	###	-	-	-	#	##
Catalan	###	###	-	-	-	#	#
Danish	-	###	+	+	+	##	##
Dutch	-	###	+	+	+	#	##
English	##	###	-	+	-	-	#
Finnish	###	###	-	-	-	#	##
French	##	###	-	+	-	#	#
German	#	###	+	+	+	##	###
Italian	###	#	-	-	-	##	##
Norwegian	#	###	+	+	+	#	##
Polish	###	###	+	+	+	#	##
Romanian	###	###	-	-	+	###	###
Spanish	##	#	-	-	-	-	##
Swedish	##	###	-	+	-	-	##

Explanation of the Scores

- Grammarian's language skills:
  - – : no skills
  - # : basic skills(general knowledge of the grammar)
  - ## : medium skills(fluent)
  - ### : advanced skills(native speaker)
- Grammarian's GF skills
  - — : no skills
  - # : basic skills(simple GF exercises)

  - ## : medium skills(more comprehensive GF exercises)
  - ### : advanced skills(resource grammar writer/substantial contributor)
- Informant needed for development/Informant needed for testing
  - —: no
  - + : yes
- Changes on the resource grammars
  - —: no changes
  - # : 1-3 minor changes
  - ## : 4-10 minor changes, 1-3 medium changes
  - ### : >10 changes of any kind
- Overall effort
  - # : less than 8h/person
  - ## : 8-24h/person
  - ### : >24h/person

This experiment is significant because it is a showcase for the ongoing work on example-based concrete grammar learning technology which will make GF grammar writing easier in terms of adding more languages and developing larger grammars, but also because it represents an analysis on the possible interaction of GF with other available translation tools, which will ease the work of both beginners and advanced users of the technology.

6 The Mobile Application

If one wants to build a tool for a controlled language for everyday usage, it seems logical for this tool to be as unobtrusive as possible. Moreover, since our language is targeting tourists, we have to take into account a particular setting where people, when going on vacation, may not have access to a computer and access to Internet can be very limited due to low coverage or prohibitive costs. This are the criteria that we tried to meet when building PhraseDroid, an application that works offline, on smartphone devices and with a simple user interface. Moreover, we wanted to do this by creating a technology that is as general as possible, and in fact applies to any multilingual GF grammar.

PhraseDroid is an Android application, that can be used on handheld devices running the Android operating system. Figure 1 shows a screen shot of the application in its current state. As you can see, the application is using the same “magnet interface” as the web application. This permits the user to compose a sentence while staying in the coverage of the grammar. Moreover, this kind of interaction works well on devices with touch screens because the magnets are large enough to be able to be selected with fingers.

What is more is that the Android platform provides a high-quality speech synthesis for several of the languages covered by the grammar, which can be plugged into our application. This gives clear benefits compared to a traditional (paper) phrasebook.

As mentioned in Section 1, there are more and more phrasebook applications developed for smartphones nowadays. They can be divided in two main categories:

1. The finite phrasebook. Those are usually made of a list of sentences translated in one, or more, foreign languages. Those phrasebooks are lacking from the point of view of expressivity since it isn't possible to change a sentence as needed, even if a very similar sentence is covered by the phrasebook.
2. The application providing machine translation through an on-line service. The Google Translate application (and the various applications that are just front-ends to it) is the best example in this category. This kind of applications can obviously be used while traveling, but they require the possibility to connect to the Internet, which is not guaranteed when one is travelling abroad due to technological or economical reasons. In addition, unlike in our application, the translation engine is not tailored toward tourist usage but is generic, which can lead so sub-optimal translation in many cases.

In contrast, our application, once installed, works off-line and features a grammar design specifically for tourist translations. And since the user inputs the sentence to be translated herself, it allows a great deal of variation and fine-tuned translation for a given situation.

The application is based on a Java interpreter for GF's binary grammar format, called PGF [15]. Therefore the application is very modular: adapting the application to a different controlled language requires little more than dropping a new pgf file in the right folder. This means that one can in no time create a translation application for another controlled language given that a GF grammar for this language has been written.

Thus an important part of the process of creating the application was to write a library in Java that provides the functions needed in the application. Its usage is not limited to Android phones, but it can also be plugged in into any Java program, whether on a desktop computer, or a web browser plug-in. In the current state, the Java library supports (predictive) parsing, linearization, and random generation of well-typed trees. This is less than the features available in the full GF interpreter, written in Haskell, but it is sufficient for machine translation and lots of other uses.

7 Evaluation

7.1 Translation Quality

This is the first criterion of evaluation. It was first assessed by the systematic use of native speaker testers, and later by comments collected from more random users of the web demo and the Android application. The goal has been what might be called *perfect quality*, in terms of meaning-preservation, grammaticality, idiomaticity, and fluency. Hence all errors found in earlier versions were corrected immediately. With the first "official version" (the one also running on Android), few direct errors have been found, but there are some inadequacies that appear in reports:

– Some sentences permitted by the abstract syntax are semantically anomalous, e.g. *is there an airplane to the toilet*. This could be fixed by using a more strict type system; however, we consider this to be less important as long as the translations are correct.

– The choice of prepositions is not always fine-grained enough; for instance the distinction between *gå till toiletten* and *gå på toaletten* (both “go to the toilet”) in Swedish is not handled (cf. Section 4).
– The usage of nationality adjectives for persons is not always optimal, but nouns should be introduced in the lexical family. Thus *I am a Finn* would be better than *I am Finnish*, with corresponding variations in many languages.

7.2 Coverage

There is no end of conceivable extensions if we want to cover everything that a tourist might want to say. The syntactic combination rules are sufficient for many situations, but they should definitely be extended with more vocabulary. For instance,

– drinks, food, currencies, countries
– time expressions like *half past eight*
– free-string input for names of places and persons

7.3 Engineering Effort

One of the main goals of the MOLTO project is to improve the productivity of GF-based translation systems "by an order of magnitude". This means that the development time of translation systems should be shortened to 10% of the original. The development time for the Phrasebook was two working days per language on average. If this is the baseline to be compared with at the end of the project (in 2013), then a new language should be possible to add in a couple of hours. Some of this improvement can be possible to reach by a better use of example-based grammar writing.

However, some parts of the grammar may be inherently difficult, due to idiomatic structures. Another way to interpret the productivity improvement would then be in terms of the concepts covered. If the first Phrasebook built in two days covers hundreds of concepts, a realistic goal could be to cover thousands of concepts in the same time. To this end, methods of automatic lexicon extraction are being developed, with ontologies, terminologies, and statistical translation models as sources.

7.4 Usability

The size of the run-time PGF grammar is 500 kB. It runs smoothly on both web applications and mobile phones. For mobiles, a substantial optimization effort was needed, but it was made on the level of GF and will therefore benefit all future applications.

The web application provides the input method of typing strings, which the mobile doesn’t have. This will certainly become an issue when the Phrasebook is extended to contain thousands of concepts. It will also become an issue how to navigate in the large space of words to find exactly the phrase one wants to use. A hierarchical approach similar to syntax editors [16] will probably be introduced as a useful complement to string-based input.

The mobile application has some usability issues reported by users, which will have to be addressed in future releases.

8 Conclusion

We have explained a controlled language approach to a multilingual tourist phrasebook, covering 15 languages. While intending to build a useful application for travellers, we have also seen it as an experiment to extend the notion of controlled language and scale it up in various respects:

- extending the notion of semantics from logic to "language games" (Section 2)
- porting a controlled language from one language to many (Section 3)
- coping with ambiguity, rather than banning it (Section 4)
- making it easier to implement controlled languages, in terms of both effort and skill (Section 5)
- building applications for laymen rather than specialists, and making them run on light devices (Section 6)

Our conclusion is that there is a lot of potential in controlled languages to become more useful in everyday life, the multilinguality aspect being at least as interesting for laymen as the traditional reasoning aspect is.

Acknowledgements. The MOLTO Phrasebook is a collaborative project. In addition to the authors of this paper, it has involved Krasimir Angelov, Olga Caprotti, Thomas Hallgren, Inari Listenmaa, Jordi Saludes, Adam Slaski, and Shafqat Virk as programmers and Richard Bubel, Rise Eilert, Karin Keijzer, Michał Pałka, Willard Rafnsson, and Nick Smallbone as testers and informants. The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n:o FP7-ICT-247914.

References

1. The Boeing Company: Boeing Simplified English Checker. (2001), http://www.boeing.com/assocproducts/sechecker/
2. Shiffman, R.N., Michel, G., Krauthammer, M., Fuchs, N.E., Kaljurand, K., Kuhn, T.: Writing Clinical Practice Guidelines in Controlled Natural Language. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 265–280. Springer, Heidelberg (2010)
3. Hart, G., Johnson, M., Dolbear, C.: Rabbit: Developing a Control Natural Language for Authoring Ontologies. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 348–360. Springer, Heidelberg (2008)
4. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web 2008. LNCS, vol. 5224, pp. 104–124. Springer, Heidelberg (2008)
5. Dean, M., Schreiber, G.: OWL Web Ontology Language Reference (2004), http://www.w3.org/TR/owl-ref/
6. Gruzitis, N., Barzdins, G.: Towards a More Natural Multilingual Controlled Language Interface to OWL. In: 9th International Conference on Computational Semantics (IWCS), pp. 335–339 (2011), http://www.aclweb.org/anthology/W/W11/W11-0138.pdf
7. Wittgenstein, L.: Philosophical Investigations. Basil Blackwell, Oxford (1953)

8. Ganesalingam, M.: The Language of Mathematics. PhD thesis, Department of Computer Science, University of Cambridge (2010), http://people.pwf.cam.ac.uk/mg262/
9. Angelov, K.: Incremental Parsing with Parallel Multiple Context-Free Grammars. In: Proceedings of EACL 2009, Athens (2009)
10. Ranta, A.: Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford (2011) ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth)
11. Ranta, A.: The GF Resource Grammar Library. Linguistics in Language Technology 2 (2009), http://elanguage.net/journals/index.php/lilt/article/viewFile/214/158
12. Bringert, B., Angelov, K., Ranta, A.: Grammatical Framework Web Service. In: System demo, Proceedings of EACL 2009, Athens (2009)
13. Angelov, K., Ranta, A.: Implementing Controlled Languages in GF. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 82–101. Springer, Heidelberg (2010)
14. Power, R., Scott, D.: Multilingual authoring using feedback texts. In: COLING-ACL 1998, Montreal, Canada (1998)
15. Angelov, K., Caprotti, O., Enache, R., Hallgren, T., Listenmaa, I., Ranta, A., Saludes, J., Slaski, A.: D10.2 molto web service, first version (D10.2) (June 2010)
16. Khegai, J., Nordström, B., Ranta, A.: Multilingual Syntax Editing in GF. In: Gelbukh, A. (ed.) CICLing 2003. LNCS, vol. 2588, pp. 453–464. Springer, Heidelberg (2003), http://www.cs.chalmers.se/~aarne/articles/mexico.ps.gz

Controlled Natural Language in a Game for Legal Assistance

John J. Camilleri, Gordon J. Pace, and Michael Rosner

University of Malta

Abstract. This paper addresses the design of an automated legal assistant capable of performing a logical analysis of legal documents and using natural language as a medium of communication with a human client. We focus on the interplay between *natural language* in which the legal document is expressed and the *formal logic* used for reasoning about it — ideally approached using a controlled natural language (CNL) together with an appropriately chosen logic for analysis and reasoning. In translating from CNL to logic, information about the CNL structure is lost. For example, the CNL might contain legal clause numbers, whilst the logic might not. This can lead to problems when for example the reasoning system discovers an inconsistency in the contract and needs to explain its whereabouts to the client. Below we discuss the issues affecting the choice of logic, arguing in favour of keeping certain structural information during formal analysis of legal documents to be able to refer to that structure when interacting with the user.

We present a framework in which to experiment and seek solutions to these issues. Having identified a sufficiently restricted domain of application we also report on the development of a CNL to interact with a variant of the game Nomic — a game based on the notion of contract specification and amendment — and argue how this game provides an ideal platform to explore the use of structure information in the domain of legal analysis.

1 Introduction

In practice, the notion of legal assistance is extremely broad, encompassing a range of possible activities involving a legal expert, who offers the assistance, and a client who receives it. These activities will vary according to the type of legal expert in question (e.g. a barrister; a notary; a solicitor), the client (e.g. another lawyer; the victim of a crime; a multi-national company), and the nature of the law in question (criminal; civil; international). A barrister, for example, being a lawyer qualified to present cases in court, must be familiar with court procedures and in a position to offer advice with respect to those procedures. The kind of advice would be different for the victim of a crime, than for the lawyer in charge of the office who is engaging him to work on a case. In the first case, it might be important to warn the victim about tricky questions that might be asked by the defense lawyer, whilst in the second, the cost of the case, its duration, and its chances of success might be more to the point.

In this paper we present our long-range aim towards the creation of an artificial legal assistant that is capable of performing a useful legalistic task on behalf of a human client, and of being able to carry out that task using natural language as a medium of

M. Rosner and N.E. Fuchs (Eds.): CNL 2010, LNAI 7175, pp. 137–153, 2012.

communication. We have had to make a whole series of strong assumptions in order to develop a working model. These centre around (i) the choice of a suitable subarea within which the notion of assistance can be developed, (ii) a formal model of legalistic reasoning which will underpin the whole enterprise, and (iii), an appropriate setting in which to investigate the practical issues by building a testable artifact. The major challenge lies in the interaction of the natural language descriptions and the formal reasoning about legal tests. While much work has been done in both fields, a legal assistant needs to be positioned in between — on one hand processing natural language legal texts and explaining the results of analysis in natural language, while on the other hand performing formal analyses for issues such as contract inconsistencies and calculating consequences of a course of action. In order to explain consequences with respect to the original text, it is crucial to keep track of its structure.

In section 2 we briefly survey the natural language issues, concentrating on legal contracts. Two particular aspects of legal assistance are developed in this section. One concerns the notion of assistance itself; the other concerns the medium in which contracts are expressed, and with respect to which assistance is offered. For the purpose of putting an upper bound on the complexity of the assistance problem, we have developed a CNL called BanaL, discussed in section 3.3, capable of expressing certain key contract-related concepts. Section 3 relates these issues to the controlled, interactive setting in which we have chosen to carry out initial experiments. The control is enforced in terms of a CNL that is used for communication, and the interactivity is controlled by embedding the interaction within a simple rule-based internet-based game called BanaNomic. This environment has been designed to raise the level of access to a live but non-specialist audience. We believe that such settings not only lower the barrier to participation in a general sense, but provide a good setting in which to carry out much-needed evaluation activities. We then turn our focus to the challenge of dealing with layout and structural information in section 4 — a little-studied phenomenon which plays a crucial role in making written contracts understandable. We thus briefly survey the work that has been carried out from a linguistic perspective on layout and outline a way of dealing with this issue from a semantic point of view.

The aim of the paper is to highlight the challenges in formal reasoning about legal texts through a CNL interface, and set out a framework in which to experiment with and seek solutions to these issues, as stated in the conclusion appearing in section 5.

2 Artificial Legal Assistance

Clearly, in order to develop the area of automated legal assistance, we need to further narrow our view and focus our attention upon an area of which offers a set of non-trivial problems and use-cases for us to study in detail, and which is simple enough for a set of computationally plausible methods to be brought to bear. In an earlier paper we took a first step in this direction by focusing on assistance with respect to contracts (Pace and Rosner [PR09]). The reasons for focusing on contracts are first that contracts are less complex than the law in general, and second that they are are *legal documents* agreed between *parties* which regulate the future *behaviours* of those

parties[1]. A key characteristic is that they make good candidates for formalisation: attempts have been made at giving a formal analysis to all the italicised words in the present paragraph, i.e.

- *Legal documents* are finitely expressed as texts in natural language. Computational linguistics and natural language processing are devoted to the problem of developing computational machinery to handle just these kinds of text.
- *Parties*, whether simple or complex, animate or inanimate, be represented as logical individuals, to which variables over certain domains can be bound. In this way we have a kind of minimal requirement for further analysis. As individuals, we can given them any properties we please. We can elaborate relationships between such individuals, or between individuals and other kinds of entity that we are prepared to talk about, such as goods and services. We can model intentions in terms of goals that are "possessed" by such individuals – and hence we can involve individuals in different kinds of intentional behaviour or action.
- *Behaviours* along with actions have long been an object of study in Artificial Intelligence (cf. Pat Hayes [Hay71]) where typically, they have been conceptualised as state-to-state functions, or as "events" à la Davidson in his much-cited essay 'The Logical Form of Action Sentences [Dav67].

2.1 Legal Assistance with Contracts

So, how can we elaborate legal assistance with respect to contracts? Daskalopulu and Sergot [DS97] distinguish between two main kinds of activity associated with contracts: contract *formation* and contract *performance*. During contract formation, the parties specify what they want the contract to express, ensure that it is accurately expressed, and finally arrive at an agreement of some kind with respect to a representation of the contract, which is normally a document of some kind. Contract performance, on the other hand, takes place after the contract is in place. Consequently we might envisage two kinds of tool that respectively address these two aspects.

- *Contract formation tools*, which are designed to support the process of producing a document that is correct both in the syntactic sense, with respect to form, and in the semantic sense of being true to the intentions of the parties. Typically, we can imagine an incremental approach in which the final contract emerges out of a dialogue with the client involving a series of successive approximations to the final contract.
- *Contact performance tools*, on the other hand, provide advice about the current state of execution of the contract offering suitable reminders to the parties whenever non-compliance is detected, possibly suggesting remedial action if this has been explicitly foreseen in some part of the contract. The BanaNomic game described in section 3.1 below is an example of a tool that falls into this category.

[1] Contracts are defined at http://legal-dictionary.thefreedictionary.com/contract as *"an agreement with specific terms between two or more persons or entities in which there is a promise to do something in return for a valuable benefit known as a consideration"*.

Central to both kinds of tool is the ability to provide *explanations* to the client about specific contract-related issues.

At the contract formation stage, it is fundamental that all parties to a contract understand the terms included in a contract as well as the rights and responsibilities to which they bind themselves. Thus the client might need different kinds of explanations e.g. concerning

- *terminology:*, Given the complexity of legal terminology, a glossary capability could provide for the explanation of technical terms - with highlighting of instances of those terms in the emerging contract.
- *understandability:*. A contract is understandable if and only if each of its parts are understandable. Therefore if the user does not understand the contract, there must be at least one part that the user does not understand. Therefore a coarse strategy for this kind of explanation would be to (i) identify the part or parts that that are not understood and (ii) explain the content of the part. Of course the kind of explanation would depend on the nature of the part (e.g. a clause; a section; a sentence; a phrase) and the nature of the client's lack of understanding (e.g. incomprehension versus misunderstanding versus disagreement).
- *consistency:* i.e concerning the consistency with respect to possible actions that a party might allow. A party might wish to know whether a contract permits, obliges or forbids a certain action from being executed or a state of affairs from coming into existence. The party might claim that the contract is impossible to satisfy, in which case the expert should demonstrate circumstances that satisfy it (or rephrase the contract if the client turns out to be right!).

At the contract performance stage, the two main activities are *monitoring* and *communication*.

- Monitoring involves keeping track of all obligations and prohibitions that the contract imposes on the parties. The heart of this activity revolves around the detection or measurement of concrete observables, which can be formulated in terms of either *actions* or *states*.

  In the action-based formulation, obligations and prohibitions are phrases in terms of actions that must be carried out - like paying a certain amount or delivering a certain document. In the state-based formulation, it is the presence or absence of properties of states that must be maintained, e.g. the a party shall maintain a positive bank balance. The choice between actions and states depends on the domain being dealt with. In the second case, for example, we are probably more interested in the state of having a negative balance than in the exact action which causes the account to become negative.

  In addition to keeping track, the monitoring process has to take special actions when things go wrong. Many contracts specify contrary-to-duty obligations which are imposed when a party fails to carry out an obligation. In such cases we would expect the monitoring process to keep track not only of the failed obligations, but of the newly imposed ones (this issue is discussed further in Pace and Rosner [PR09]). In the example cited, a negative balance could trigger an obligation to pay interest at an exhorbitant rate.

– Communication is necessary in order to inform the client about the dynamically changing state of current obligations. The bank balance goes negative, so the client must be informed that a new obligation to pay interest has arisen. Several issues surround such an act of communication, not the least of which is the timing. In the case cited, the timing would coincide with the moment at which the change of state took place – in other words *after* the action that broke the existing obligation. In other cases it might be possible or even necessary to issue a warning *before* the action takes place in order to avoid the imposition of penalties. Another issue concerns explanation. Suppose the client wants a justification or simply further elaboration of the newly imposed obligation. Then the tool would need to provide an explanation, referring back to relevant part of the contract.

The above remarks suggest that for the two kinds of assistance we have considered, contract formation and contract monitoring, explanation enters into the picture. One of the main claims of this paper is that successful explanation requires knowledge of structure and of the way in which the structure relates to the underlying semantics that defines the legal concepts involved. These considerations are further elaborated in the next section.

3 CNLs, Contracts and Games

The use of CNLs to enable processing of and formal reasoning about statements in a particular domain is an established approach [Sch08, BSBS09]. By constraining the language, together with the structural complexity of the grammar, one obtains a means to make statements about the underlying domain, without moving too far off from a natural language description, so that it remains understandable to native language speakers. At the same time it is easy to translate into an suitable logic.

In identifying an appropriate domain-specific CNL, one faces two primary challenges — that of identifying the basic underlying concepts in the domain, and secondly that of selecting an appropriate and sufficiently rich grammar through which to combine these basic concepts. Going further, and enabling formal reasoning and manipulation of statements made in the CNL is further hindered by the fact that typically, relating the basic concepts together requires much tedious and error-prone work. Some work has been carried out on CNLs for contracts [PR09]. The primary concepts of interest here are deontic ones, namely obligations, permissions and prohibitions on actions or states. In practice, using a CNL for contracts requires not only semantics of the language operators, but also the underlying implicit concepts that the contract mentions.

Once the CNL has been defined, one also needs to define a setting in which the effectiveness of a CNL for contracts can be investigated empirically. In this paper we investigate the use of a CNL to describe game rules. In particular, to enable interesting cases, and the need for consistency checking of the rules, we apply the technique to implement a variant of the game of Nomic — a game in which changing the rules is part of the game itself.

3.1 Nomic and BanaNomic

Nomic is a game of self-amendment [Sub90] — starting with an initial rule set, each player takes their turn changing the game's rules through a system of rule proposals and player voting. What makes Nomic so particular is that *everything* is theoretically up for amendment during the game, including the voting system itself and what players need to do to win. Despite the popularity of the game, only one Nomic variant could be found which uses automated rule processing. The game is encoded and played directly in Perl [PB05], and circumvents the contract specification and processing by identifying the contract with the Perl program governing the voting process — what the program accepts (or rejects) is considered to be the semantics of the program. Encoding the full game of Nomic with natural language contracts is particularly challenging, since it combines challenges in natural language analysis and formal reasoning about contracts. In this section, we show the challenges in combining formal reasoning with controlled natural language reasoning in the game, without the use of (i) layout information; and (ii) macro definitions. We then argue why we believe that the use of these will enhance the game and its interface.

The major challenges in Nomic playing using natural language contracts are twofold: (i) formulating a language in which the contract clauses are expressed — rich enough to be able to reason about notions such as permission and obligation; and (ii) the contracts frequently refer to statements about the real world which require a strong semantic framework ('Players wearing glasses cannot propose amendments to clauses labelled by a prime number'). The former problem we have addressed by developing a CNL called BanaL, which we discuss more concretely in the next section, and the latter was circumvented by reducing the domain of the game to a simpler setting.

BanaNomic is a more concrete version of Nomic, in which players represent monkeys living in a tree, fighting to pick bananas and defend their stash. The constitution corresponds to the rules of the jungle — and can refer to the state of affairs (e.g. how many bananas a player owns) and actions possible in this limited setting (e.g. climbing up the tree). The rules cannot be violated, but the monkeys are allowed to add and remove rules at will. During each turn, the players may carry out actions and modify the constitution. The game is governed by banana-time, thus enabling constitution clauses to refer to time.

3.2 A Deontic Contract Logic and Language for BanaNomic

Typically, most logics allow reasoning about a state of affairs — studying the consequences of what predicates hold or otherwise in a particular situation. In computer science, one finds various extensions of this notion to deal with the need for interaction with the actual state of the system. For instance, in runtime verification [HR01], one typically identifies properties using an appropriate logic, together with actions to be triggered upon violation of these properties. Reasoning about these extended systems presents an extended challenge due to the feedback between the properties and actions applied to the system behaviour. Similarly, legal reasoning deals with consequences of violations with respect to the state and actions of an entity, thus requiring this additional layer of cognition. In contrast with runtime monitoring, however, most

of the consequences of violations are new (or modified) legal statements. For instance, the consequence of violating the obligation to pay one's subscription leads to a permission on the service provider to cancel the service and a further obligation to pay the due amount with additional interest. For millennia, effective reasoning about such contracts and legal texts has proved to be a major challenge for philosophers; more recently computer scientists have also become involved with the issues.

Moral and normative notions such as obligation, permission and intention, have been studied as far back as the time of Aristotle. The first formal analysis can be attributed von Wright [Wri51] in 1951 (although some authors identify Mally's work in 1926 to be a precursor of this [Mal26]). This family of logics, called deontic logics, allow for reasoning at least about the notions of permission, obligation and prohibition. However, it turns out that even restricting the logic to these notions exposes various challenges and choices [McN06]. One is the inclusion of constructs to deal with contract violation, such as the concepts of *contrary-to-duty obligations* (CTD) and *contrary-to-duty prohibitions* (CTP). CTDs state the consequences of not respecting an obligation while CTPs similarly deal with prohibitions that might be violated. In both cases, one specifies the resulting clauses which are to be fulfilled as reparations in case of violation.

Two main approaches taken in the literature turn on whether the deontic notions are attached to actions or to states. Is one to be prohibited from moving the opponent's pieces (action-based) or is one to be prohibited from having more than 16 pieces on the board at the same time (state-based)? Although both approaches and their combination have been shown to be useful in practice for different domains, in our game setting, we adopt an action-based approach to avoid having to talk about causality of who brought about which parts of the state, and is thus responsible for the consequences.

The contract grammar used for BanaNomic is based on the deontic logic used in [PR09]. The deontic logic is based on three fundamental deontic modal operators: obligation $\mathbb{O}$, permission $\mathbb{P}$ and prohibition $\mathbb{F}$, and is action-based, in that all the deontic operators act on action expressions. Furthermore, all actions are tagged by their subject and object (if relevant) e.g. the action *throwBanana* takes both the name of the monkey throwing the banana, and the monkey at whom it is being thrown — $throwBanana(Michael, Gordon)$. These basic actions can be combined together using sequential composition (;) and choice (+) to obtain action expressions such as:

$$(throwBanana(Michael, Gordon); eatBanana(Gordon)) + eatBanana(Michael)$$

To enable quantification over actors, rather than introducing explicit quantifiers, we borrow the notation used for polymorphic type place-holders from type systems, and enable quantification by using a name placeholder * e.g.: $\mathbb{F}(throwBanana(*, John))$ would be the statement saying that everyone is forbidden from throwing a banana at John. Ideally, in such a logic we would allow for different variable names, allowing us to unify different actors in a statement, but for the sake of a more direct mapping to and from the CNL, we assume that the instances of * all refer to different variables.

The contract is modelled as a function from the natural numbers to clauses, and is interpreted as the conjunction of all clauses. The clauses can be (i) deontic statements over action expressions; (ii) temporal operators — $\Box[b, e]C$ says that from time b to time e clause C will always be enforced and $\Diamond[b, e]C$ says that at some time between time b and e, clause C will hold; (iii) choice operators — $C_1 + C_2$ says that one of C_1 and C_2 must hold and $C_1 \twoheadleftarrow q \twoheadrightarrow C_2$ checks whether query q holds (queries are boolean

Table 1. Example of the rules enacted during a four-turn sequence of BanaNomic, showing rule clauses in formal notation along with their natural language linearisations. Other turn actions are omitted.

Player	Rule enacted
1. George	$\mathbb{F}(pickBanana(Paul))$ Paul is forbidden to pick a banana
2. Paul	$\Diamond\,[0,9]\,\mathbb{O}(throwBanana(\phi, George))$ At some point before time 9 every player is obliged to throw a banana at George
3. George	$\mathbb{F}(abolish(Paul,*)) \triangleleft \mathbb{P}(enact(George,*,*)) \triangleright Ok$ If George is permitted to enact a rule then Paul is forbidden to abolish any rule
4. Paul	$\Box\,[0,\infty]\,\mathbb{F}(enact(George,*,*))$ At all times George is forbidden to enact a rule

expressions over the state of the game — how many bananas each player has, the height in the tree where each player can be found, etc) and enacts C_1 or C_2 accordingly; (iv) a consequence operator $C_1 \triangleleft DE \triangleright C_2$ which checks for the existence of deontic clause DE and enacts clause C_1 or C_2 accordingly; and (v) the conjunction of two clauses $C_1 \wedge C_2$. The syntax of the logic is as follows:

$$
\begin{aligned}
ActionExp &::= Action \mid ActionExp; ActionExp \mid ActionExp + ActionExp \\
DeonticExp &::= \mathbb{O}(ActionExp) \mid \mathbb{F}(ActionExp) \mid \mathbb{P}(ActionExp) \\
Clause &::= Ok \mid Fail \mid DeonticExp \mid Clause \wedge Clause \mid Clause + Clause \\
&\mid\ Clause \lhd\!\lhd Query \rhd\!\rhd Clause \mid Clause \triangleleft DeonticExp \triangleright Clause \\
&\mid\ \Box\,[Time, Time]\,Clause \mid \Diamond\,[Time, Time]\,Clause
\end{aligned}
$$

Two of the basic actions which can be used in action expressions are *enact* and *abolish*, which refer to a particular player enacting or abolishing an existing clause. When used in conjunction with the deontic operators, one can express clauses about power e.g. $\mathbb{F}(enact(John,*,*))$ says that John is not allowed to enact any clause anywhere in the contract. Using the logic defined above, a few example contracts and their natural language readings are given in table 1.

The clauses are given an operational semantics, defining a relation $C \xrightarrow{a}_{\sigma} C'$ which says that when in game state is σ and upon action a, contract C evolves to C' advancing forward in time. For example, $\Box\,[0,7]\,\mathbb{O}(a) \xrightarrow{a}_{\sigma} \Box\,[0,6]\,\mathbb{O}(a)$. The main advantage of adopting this approach is the also the reason it is frequently adopted for monitoring of systems: we can progressively consume actions coming from the players, updating the active game contract as required. The semantics of a number of the operators are given below to illustrate how the rules are defined.

Basic clauses: The base clauses *Ok* and *Fail* denote the trivial cases of the clause which can never be violated and the clause which will always lead to a violation. The rules for these clauses are rather straightforward:

$$\frac{}{Ok \xrightarrow{a}_{\sigma} Ok} \qquad \frac{}{Fail \xrightarrow{a}_{\sigma} Fail}$$

Obligation: If we are obliged to match an action expression e, then upon receiving action a, we have three possible situations: (i) a matches the expression e, in which case the obligation can be discharged; (ii) after consuming a, one is still obliged to match action expression e' (for example receiving a when obliged to perform $a; b + a; c + d; e$ will result in an obligation to perform $b + c$); and (iii) a cannot match action expression e, resulting in a violation:

$$\frac{}{\mathbb{O}(e) \xrightarrow{a}_{\sigma} Ok}\ e \xrightarrow{a} \checkmark \qquad \frac{}{\mathbb{O}(e) \xrightarrow{a}_{\sigma} \mathbb{O}(e')}\ e \xrightarrow{a} e' \qquad \frac{}{\mathbb{O}(a) \xrightarrow{a}_{\sigma} Fail}\ e \xrightarrow{a} \times$$

Conjunction: For the case of clause conjunction, we have special cases for the different cases when either conjunct reduces to *Ok* or *Fail*:

$$\frac{C_1 \xrightarrow{a}_{\sigma} Ok \qquad C_2 \xrightarrow{a}_{\sigma} C_2'}{C_1 \wedge C_2 \xrightarrow{a}_{\sigma} C_2'} \qquad \frac{C_1 \xrightarrow{a}_{\sigma} Fail}{C_1 \wedge C_2 \xrightarrow{a}_{\sigma} Fail}$$

$$\frac{C_1 \xrightarrow{a}_{\sigma} C_1' \qquad C_2 \xrightarrow{a}_{\sigma} Ok}{C_1 \wedge C_2 \xrightarrow{a}_{\sigma} C_1'} \qquad \frac{C_1 \xrightarrow{a}_{\sigma} C_1' \qquad C_2 \xrightarrow{a}_{\sigma} C_2'}{C_1 \wedge C_2 \xrightarrow{a}_{\sigma} C_1' \wedge C_2'}\ C_1',\ C_2' \notin \{Ok,\ Fail\}$$

Queries: The rules for queries use the game state σ to choose which branch to follow:

$$\frac{}{C_1 \triangleleft q \triangleright C_2 \xrightarrow{a}_{\sigma} C_1}\ \sigma(q) \qquad \frac{}{C_1 \triangleleft q \triangleright C_2 \xrightarrow{b}_{\sigma} C_2}\ \neg\sigma(q)$$

The semantics of a contract made up of indexed clauses is then simply the lifting of these semantics over the clause locations.

3.3 BanaL, a CNL for BanaNomic User Input

We have developed BanaL — a CNL for BanaNomic, designed as an application-specific method of natural language representation based on the syntax of the logic. This has the effect of making the conversion from contract logic to natural language and back (*linearisation* and *analysis*, respectively) very simple and deterministic. The Grammatical Framework (GF) [Ran04] was adopted for the guided-input methods it facilitates (see below), and its support for sophisticated forms of language generation — thus future-proofing the design so that subsequent versions of BanaL could easily be extended to include much more intelligent natural language realisation choices.

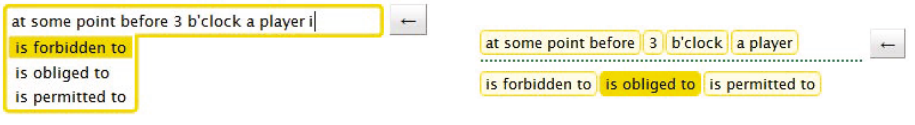
GF is a specialised functional language for defining grammars, having separate abstract/concrete syntax rules, a strong type system, and inherent support for multilinguality. GF grammars are declarative in nature, with a primary focus on the linearisation of syntax trees. By writing an abstract GF grammar and defining *how* it should be expressed in one or more natural languages (concrete grammars), GF is able to derive both a generator *and* a parser for each of those languages [Ran04].

Given the declarative nature of GF grammars, the abstract syntax of BanaNomic could very easily be implemented on the basis of its formal logic. For example, the abstract GF equivalent for the definition of the *Clause* category would be as follows:

```
cat
  DeonticExp ; Time ; Clause ;
fun
  C_Deontic     : DeonticExp -> Clause ;
  C_Always      : Time -> Time -> Clause -> Clause ;
  C_Conditional : DeonticExp -> Clause -> Clause -> Clause ;
  ...
```

For the design of the concrete grammar, each of the functions from the abstract syntax is given a template-like linearisation. While suitable for many cases, certain constructs required a more subtle approach in order to produce phrases which still sound natural. Nested phrases were particularly problematic to express unambiguously ('Paul is allowed to pick a banana and climb the tree or climb down the tree'), and the use of pronouns was avoided altogether.

A major part of GF is its partial evaluation algorithm (or *incremental parser*), which gives rise to interesting guided-input possibilities. By presenting the user with a list of possible words which may come next in a partial sentence, they are able to construct grammatical sentences in an auto-complete fashion. This is highly useful as it ensures that only syntactically-correct phrases are entered first-time round, and will avoid user frustration of trying to construct parseable sentences in free-text. The guided input methods developed for BanaNomic are based on the drop-down suggestions (figure 1a) and the "fridge magnets" (figure 1b) — developed by the GF team. These input methods are of particular interest to the area of CNLs, as they help avoid the problem of users having to know what is grammatical in a particular CNL.



(a) Drop-down suggestions (b) Fridge magnets

Fig. 1. Guided input methods used in BanaNomic

3.4 Discussion

As should be evident from this section, the combination of CNL and formal reasoning in this setting is crucial, yet extremely challenging. From the implementation of the game, it is evident that the major issue is one of giving feedback and explanations to the users. While parsing of the CNL input and executing the formal semantics of the logic can be done using standard techniques, explaining the consequences of a rule, and how the rules change through the advancement of time is particularly challenging due to the flat nature of the contract language and logic. Allowing the CNL rules to be tagged with layout information is necessary to allow players to deal with longer, and thus more intricate, rules. However, explanations must then be adapted to maintain and refer to this structure as required. Another issue which hampered strategies in the game was the low-level nature of the logic constructs. Advanced players would benefit from the ability to

define macros to be able to build rules of higher complexity, without a proportionate increase in their length. As in the case of layout information, it is however crucial that the use of these macros in the rules is to be appropriately handled in user explanations. It is in this manner that we believe that the use of the techniques proposed in this paper will enable more complex strategies to be more tractable for human players.

4 Explanation, Structure and Layout

In the development of Nomic, we kept a simple contract layout, encoded as an enumerated list of clauses. Cross referencing was limited to references to locations of clauses, and due to the nature of the game, the clauses themselves were typically kept simple by the players. However, in real-world contracts, layout and cross-references play a much more important role. Also, certain contract structures recur multiple times, and would ideally be defined as syntactic sugar above the basic logic. Both these layers of non-functional information do not change the behaviour of the contract — whether a conjunction is enumerated or not, and when syntactic sugar is unrolled should not change the semantics of the contract. However, it changes the way natural language would be generated from the clauses in order to interact with the user. In this section we look at the challenges this poses, both from a CNL and logic perspective.

4.1 Layout and Presentation

An important aspect of contracts already alluded to is that they have a characteristic structure and layout which explanations must exploit in order to be successful. The dimension of layout arises because written text appears on a page and must therefore have visual characteristics which include not only structuring devices such as paragraph division and indentation but (e.g. in presentations) other structures such as bullet points. Of particular interest are ordered lists, whose internal structure and outward appearance are close the clause and section structure which characterises the presentation of many legal documents.

The question is where to look for further details of that structure. Legal textbooks provide a high-level overview. Under English Law, for example, we know that every contract includes four essential components: an offer, an acceptance, a consideration (the requirement of reciprocal obligations on the parties to a contract) and an intention to create legal relations between the parties. The lower level structure bottoms out in terms of sections and clauses, which, as we have said, closely resemble ordered lists.

Contracts display what might be called a *macro structure* and a *micro structure*. The macro structure deals with the main sections etc. and depends on the kind of contract under consideration. Vella [Vel10] for instance, studied several hundred property sale contracts under Maltese law and found that they always comprised a number of *sections* including

- a description of the parties;
- a description of the property;
- a description of the financial arrangements;
- the signatures of the parties;
- a date.

Apart from the fact that each of these sections played a distinct legal role (and can be related to the essential high-level components just mentioned), they were each characterised by the use of particular phrases, and also certain layout conventions including the use of certain keywords, phrases, fonts and capitalisation.

Clearly, different kinds of contract will have different macro structures, and these to a large extent could be successfully represented in the form of document templates, style sheets, etc. These can be very concrete (e.g. word template files) or else more abstract structures which have the advantage of being platform independent. There is a movement towards the formal representation of legal documents using XML[2].

The micro structure, on the other hand, concerns a level of presentation corresponding roughly to a collection of legal clauses. This transcends sentence structure, since a single clause be realised by two simple sentences in preference one complex sentence. It includes the order in which sentence forms occur, since this can reflect the order in which actions might have to be carried out. For example, under Maltese Law, you are obliged to pay all outstanding fines *before* applying for a road licence. It has to be in that specific order else the application will fail.

The micro structure also addresses punctuation. One of the problems with punctuation is that nobody can agree on the range of phenomena it includes. Is it just the use of certain punctuation symbols, or does it also include, say, list markers, indentation, text styles, fonts, etc? This is not just a theoretical discussion, since the way we answer this question will determine whether it can properly be referred to in explanations.

Opinions differ as to the importance of punctuation marks in legal texts, where traditionally, they are used sparsely. Nunberg [Nun90] argued strongly against a "transcriptional" view (Crystal and Davy [CD69]) according to which punctuation has an essentially secondary role. Instead, he claims that punctuation properly belongs to a distinct level of linguistic structure that is peculiar to the written language - *text structure* - that no less important that traditional sentence structure and exists alongside it. Nunberg introduces what we feel is an important distinction between this abstract text structure and the concrete graphical devices used to express it.

A practical application based on Nunberg's ideas is offered by Power-et-al [PSBA03], which describes a system, ICONOCLAST, for the generation of text structures of the kind mentioned from a meaning representation that includes not just propositional information, but the *rhetorical function* of that information.

The input to ICONOCLAST takes the form of a tree whose leaves are propositions, and whose nodes indicate the rhetorical relation between them. The latter is based on Mann and Thompson's Rhetorical Structure Theory (RST) [Man88], an attempt to provide a formal and computational basis for the description of rhetorical function. A simple example of such a structure is

concession(`ban(fda,elixir),approve(fda,exlixirplus)`)

where **concession** is the rhetorical relation holding between the propositions `ban(fda,elixir)` (the FDA[3] banned the drug Elixir) and `approve(fda, elixirplus)` (the FDA approved the drug Elixirplus).

[2] Legal XHTML - see `http://www.hypergrove.com/legalxhtml.org`

[3] Federal Drug Association

The process of transforming such a rhetorical-semantic message into a document involves not only realising the basic propositions, but solving a series of realisation constraints holding between nodes of rhetorical tree and text-structures. Several solutions are possible and are in fact generated by ICONOCLAST:

- The FDA approves ElixirPlus, although it bans Elixir.
- Although the FDA bans Elixir, it approves ElixirPlus.
- The FDA bans Elixir, but it approves ElixirPlus.
- The FDA bans Elixir; but it approves ElixirPlus.
- The FDA bans Elixir. But it approves ElixirPlus.

The main claim underlying Power *et. al.*'s work is that to handle the phenomena present in the above sentences it is necessary to distinguish a third level of structure – rhetorical structure. This level is over and above the abstract and concrete levels of text structure postulated by Nunberg.

Finally, no discussion of the issue of layout could be complete without mentioning the ambitious research agenda of Bateman and colleagues [BKKR01]. Bateman's work addresses "the desirability of combining text, layout, graphics, diagrams, 'punctuation' and typesetting" for the most effective presentation of information. The scope is thus wider than Power-*et. al.*'s, but the general methodology is similar insofar as in both cases, the message to be conveyed is expressed using a rhetorical structure tree. The main difference is that whilst Power is concerned with transforming this into a text-structure, Bateman's concerns a wider-ranging *layout structure* whose nodes correspond to blocks that will be realised as regions on the printed page. Blocks *may* be realised by text-structures, but may also be expressed with different kinds of graphics such as images. Furthermore, blocks can participate in graphic relations, such as proximity, similarity of style, etc.

Power and Bateman have effectively both added a semantic basis in the form of the underlying layer of RST. It is the RST structure which defines *what* any potential realisation is supposed to express. The two approaches differ in the range of realisation phenomena considered. Power only considers text structures, whilst Bateman includes in addition other kinds of layout device.

4.2 Syntactic Sugar and Semantics

Well-formed contracts abiding by the syntactic rules of the domain may have multiple semantic interpretations depending on their application. For instance, to print out a contract, layout information plays a crucial role while the distinction between an obligation and a prohibition is limited to a superficial change in the symbol used. On the other hand, to monitor behaviour with respect to a contract, we require a semantic interpretation in which the distinction between obligation and prohibition is a major one, while layout information can be discarded without any loss of information. This approach of having multiple semantic interpretations of the same syntax is a well-known one in other domains, such as embedded languages [Hud96]. For instance, in [CSS03], circuit descriptions have multiple interpretations, depending on how they are intended to be used. To print out a circuit, layout information is sufficient, while for the simulation or model checking of circuits, an interpretation which discards layout information, but

uses the semantics of logic gates and latches is required. For other analysis, such as signal delay propagation (due to the gates and wire length) both types of information must be retained. The solution usually adopted, is to give independent semantics for each of these applications.

In the case of contracts, the main challenge comes with the need for explanation. Clearly, the original contract one starts off with, fully annotated with layout information, may use the annotations for explanations. However, as already discussed, contract performance requires the keeping track of obligations, prohibitions and permissions which might appear over time as the contract evolves. For instance, consider a clause which states that after three consecutive occurrences of action *bad-password*, the user is obliged to answer a *captcha*[4]. Explaining the state of the contract after two consecutive bad passwords, one would have to say that if one more wrong password is given, the user will have the obligation to answer a captcha. If this clause occurs within an extensive contract, giving this explanation, without giving a reference to the location of the clause, or presenting the 'evolved' clause as part of the contract as a whole will not be of much help to the person reading the explanation. Similarly, consider a conflict analysis procedure, which given a contract returns whether the contract may lead to a conflicting state e.g. having an action obliged and prohibited at the same time. Such potential conflicts are typically explained by giving a trace of actions which leads to the problem, and the conflict itself. However, in the case of extensive contracts, simply telling the user 'After the actions $\langle a,\ b,\ c\rangle$, action d is both obliged and prohibited', is not of much use, since which parts of the contract give rise to these clauses may not be obvious. A better explanation would include information as to which parts of the contract led these clauses, or better still, allow the visualisation of how the contract evolves as the trace is traversed, finally highlighting the parts which have led to a conflict.

A related issue with human readability of contracts is that of syntactic sugar. Although the core logic one reasons in may just have a handful of constructs, one would typically have various compound constructs used in the contract, and which are defined in terms of the underlying basic ones. For instance, in a contract logic which allows for sequential composition, one may define the repetition of a contract n times by unrolling it using sequential composition. As in the case of layout information, one would thus lose all such high-level constructs at the reasoning level thus hampering the explanation given back to the user when the need arises.

Both layout information and macro-definitions typically have no consequences on whether a trace leads to a violation of a contract, or whether a contract contains a conflict. Due to this, semantics of violation and conflict-analysis typically apply Occam's razor, ignoring all such information. The dilemma is that, although the layout information is irrelevant to answer questions such as *'Does a conflict exist?',* it is relevant in *explaining* the result. One would like to be able to ensure that (i) the semantics extended to handle layout and macro-definition context information does not change the trace (and hence conflict) interpretation of a contract; and (ii) the logic retains as much of this information as possible, in order to aid the explanation process.

[4] A captcha is a transformed visual representation of text, intended to identify non-human access to a resource. They are frequently used to avoid posting of spam and repeated password attempts from automated scripts.

Adding layout information within the context-free structure of the logic involves having an additional constructor $tag_t(C)$, which tags contract C with tag $t \in Tag$.

$$Clause ::= tag_{\text{Tag}}(Clause) \mid Ok \mid Fail \mid \ldots$$

Thus, as a simple example, we can add information about enumerated clauses using tags in a formula of the form: $tag_{lst}(tag_{lst:1}(\mathbb{O}(a)) \wedge tag_{lst:2}(\mathbb{F}(b)))$. Tags can be given a semantics to enable the transformation of such a clause into an enumerated list. Note that, even if one applies some analysis which commutes the conjunction operator to transform the clause into $tag_{lst}(tag_{lst:2}(\mathbb{F}(b)) \wedge tag_{lst:1}(\mathbb{O}(a)))$, the originally intended description can still be achieved.

Although these tags may have a layout semantics of their own, the addition of these tags will not change the normative semantics of the contract — which can be handled by the following rule:

$$\frac{C \xrightarrow{a}_{\sigma} C'}{tag_t(C) \xrightarrow{a}_{\sigma} tag_t(C)}$$

For instance, upon receiving an action c, one can show that the enumerated contract we saw earlier, behaves as follows:

$$tag_{lst}(tag_{lst:1}(\mathbb{O}(a)) \wedge tag_{lst:2}(\mathbb{F}(b))) \xrightarrow{c} tag_{lst}(tag_{lst:1}(Fail) \wedge tag_{lst:2}(Ok))$$

With the tags included one can still provide information regarding which of the sub-clauses failed due to action c. On its own, however, this rule is not sufficient to handle tags as one would expect. The problem is that by keeping the tag, other rules may be inhibited from firing. For example, the expression we obtain after action c cannot be reduced to $tag_{lst}(Fail)$, as one would it expect it to. The only solution is such cases is to discard a tag within any syntactic context α by performing an internal action τ:

$$\frac{}{\alpha(tag_t(C)) \xrightarrow{\tau}_{\sigma} \alpha(C)}\ deadlock_{\sigma}(\alpha(C))$$

The side-condition $deadlock(\alpha(C))$ is used delay untagging until no other external actions are possible[5].

In this manner, we can extend the syntax and semantics of a logic in such a manner to retain as much information as possible to be used for explanations. By refining and generalising the approach proposed here, one can retain much information, without the need for redefining the semantics from scratch. If the semantics of contracts remains unchanged under the addition of tags, one can perform all analysis in the untagged logic, and use the tagged inference rules only once a scenario or counter-example is discovered, which is to be explained in a controlled natural language. This approach can be ideal in a setting where the transitions system semantics of the logic with no annotations is automatically extended to maintain the necessary information for explanations.

A related problem is that of the use of syntactic sugar or macros which is also ideally kept for explanations. For instance, one can define a macro $soon(C)$ which allows contract C to be satisfied either now or in 5 time units: $soon(C) \stackrel{df}{=} C + \Box[5,5]C$. As with tags, one can add new syntax to the logic and allow for keeping the macros intact. However, more caution needs to be put in to avoid unsound inferences from being made.

[5] The predicate $deadlock_{\sigma}(C)$ can be defined as $\neg\exists a, C' \cdot C \xrightarrow{a}_{\sigma} C'$.

5 Conclusions

One of the primary challenges we have found when supporting reasoning through the use of a CNL is the domain to which the language is applied — controlling the structure of the sub-language ensures that a mapping to and from the operators of the formal underlying representation is possible. On the other hand, if the domain of the basic terms is not carefully controlled, the reasoning one can perform is strictly limited. In this paper we have investigated the use of a controlled domain of application for BanaL, a CNL to specify contract clauses as input to the game BanaNomic, in which the basic actions and state queries are limited. The CNL has been used as a front end input to a web-based version of BanaNomic, with players taking turns to change the constitution and take actions — as regulated by the current contract.

The tractability of legal documents from a human perspective much depends on the way in which the legal clauses are organised, and the use of definitions to avoid lengthy identical descriptions in different contexts. Collapsing even a fraction of a legal agreement into one long paragraph of text, transforms a document from one which can be followed by a human expert to one which is unintelligible. However, since this structure is void of legal meaning, formal reasoning typically discards this information. The main challenge is to maintain this structure in such a manner as to enable feedback and explanation of the outcome of formal reasoning back in a natural language setting. We are currently looking into how these notions can be incorporated into BanaNomic so as to enrich the game, by enabling more complex rule sets which are still tractable to human players.

The conclusion here is that assistance comes in two flavours, so to speak. Assistance with the strictly logical properties requires an underlying semantic model. However, when assistance takes the form of *explanation* it is invaluable to be able to refer to identifiable parts of the document structure. Here layout is crucial.

References

[BKKR01] Bateman, J., Kleinz, J., Kamps, T., Reichenberger, K.: Towards Constructive Text, Diagram, and Layout Generation for Information Presentation. Computational Linguistics 27, 409–449 (2001)

[BSBS09] Bao, J., Smart, P.R., Braines, D., Shadbolt, N.R.: A Controlled Natural Language Interface for Semantic Media Wiki Using the Rabbit Language. In: CNL (2009)

[CD69] Crystal, D., Davy, D.: Investigating English style. Studies in the History and Theory of Linguistics. Indiana University Press (1969)

[CSS03] Claessen, K., Sheeran, M., Singh, S.: Functional Hardware Description in Lava. In: The Fun of Programming, Cornerstones of Computing, pp. 151–176. Palgrave (2003)

[Dav67] Davidson, D.: The Logical Form of Action Sentences. In: Rescher, N. (ed.) The Logic of Decision and Action. University of Pittsburgh Press (1967)

[DS97] Daskalopulu, A., Sergot, M.: The Representation of Legal Contracts. AI and Society 11, 6–17 (1997)

[Hay71] Hayes, P.J.: A Logic of Actions. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence 6, pp. 495–520. Edinburgh University Press (1971)

[HR01] Havelund, K., Rosu, G.: Monitoring Programs Using Rewriting. In: Proceedings of the 16th IEEE International Conference on Automated Software Engineering, ASE 2001, pp. 135–143. IEEE Computer Society, Washington, DC (2001)

[Hud96] Hudak, P.: Building domain-specific embedded languages. ACM Computing Surveys 28, 196 (1996)

[Mal26] Mally, E.: Grundgesetze des Sollens. Elemente fer Logik des Willens. Leuschner & Lubensky, Graz (1926)

[Man88] Mann, W.C., Thompson, S.A.: Rhetorical Structure Theory: Toward a functional theory of text organization. Text 8(3), 243–281 (1988)

[McN06] McNamara, P.: Deontic Logic. In: Gabbay, D.M., Woods, J. (eds.) Handbook of the History of Logic, vol. 7, pp. 197–289. North-Holland Publishing (2006)

[Nun90] Nunberg, G.: The Linguistics of Punctuation (Center for the Study of Language and Information - Lecture Notes) (August 1990)

[PB05] Phair, M.E., Bliss, A.: PerlNomic: Rule Making and Enforcement in Digital Shared Spaces. In: Online Deliberation 2005 / DIAC 2005, Stanford, CA, USA (2005)

[PR09] Pace, G.J., Rosner, M.: A Controlled Language for the Specification of Contracts. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 226–245. Springer, Heidelberg (2010) ISBN: 978-3-642-14417-2

[PSBA03] Power, R., Scott, D., Bouayad-Agha, N.: Document Structure. Computational Linguistics 29, 211–260 (2003)

[Ran04] Ranta, A.: Grammatical Framework: A Type-Theoretical Grammar Formalism. Journal of Functional Programming 14(02), 145–189 (2004)

[Sch08] Schwitter, R.: A Controlled Natural Language for the Semantic Web. Journal of Intelligent Systems 17(1-3), 125–141 (2008)

[Sub90] Suber, P.: Nomic: A Game of Self-Amendment. In: The Paradox of Self-Amendment. Peter Lang Publishing (1990)

[Vel10] Vella, G.: Automatic Summarisation of Legal Documents. Master's thesis, Univerisity of Malta, Dept Intelligent Computer Systems, University of Malta, Msida MSD2080, Malta (2010)

[Wri51] Von Wright, G.H.: Deontic Logic. Mind 60, 1–15 (1951)

Working with Events and States in PENG Light

Rolf Schwitter

Centre for Language Technology
Macquarie University
Sydney 2109 NSW, Australia
Rolf.Schwitter@mq.edu.au

Abstract. In this paper I discuss how the controlled natural language PENG Light can be modified so that it can serve as a high-level interface language to the Event Calculus. The Event Calculus is a narrative-based formal language for reasoning about events, their effects and timepoints, and can be used for various reasoning tasks where a representation of time is important. Using a scenario from a dynamic domain, I show what kind of modifications are necessary on the level of the controlled natural language to specify the background knowledge that is required to deal with direct and indirect effects of events and with continuous change in that domain. I discuss how the output of the controlled natural language processor of PENG Light that distinguishes between events and states can be aligned with the input language of the Event Calculus, and then be used for automated reasoning. Finally, I show how the Event Calculus can be used to support the question answering process and then evaluate its reasoning capabilities using a number of benchmark questions stated in controlled natural language.

1 Introduction

Over the last decade, a number of controlled natural languages have been developed that can serve as high-level knowledge representation and specification languages for various application domains [27]. These controlled natural languages look seemingly informal since they are subsets of natural languages and have the advantage over full natural languages that they can often be translated unambiguously into a formal target language. Controlled natural languages are particularly attractive because they can balance most disadvantages that natural languages and formal languages have when they are used as specification languages: in particular, they are easy to read and understand by subject matter experts [14], and they have the same formal properties as their formal target languages [4].

With the exception of Computer Processable English [2] that uses the Knowledge Machine system [1] that implements a version of the Situation Calculus [16] to perform temporal projection, not a lot of attention has been paid so far by the research community to the use of controlled natural languages for dynamic domains. The description of dynamic domains usually requires a lot of complex background knowledge in order to predict what happens at a given point in time,

M. Rosner and N.E. Fuchs (Eds.): CNL 2010, LNAI 7175, pp. 154–174, 2012.

to fill in the semantic gaps if some information is missing, and to reason backwards from effects to possible causes – if necessary. What seems to be simple for humans at first glance, turns out to be surprisingly hard for a machine.

In order to process a sequence of events in a dynamic domain and to answer questions about the effects of these events at various points in time, I use a logic programming implementation of the Event Calculus [10,28,30] as starting point and show how the controlled natural language PENG Light [37] that distinguishes between verbs that denote events and states can be modified and interfaced with the Event Calculus. These modifications are necessary so that it becomes possible to express the relevant background knowledge about events and their direct and indirect effects as well as the knowledge that is required to deal with continuous change. As we will see, the Event Calculus is a very flexible formal language for encoding reasoning about action and change, and it can be used for making inferences about events, their effects and timepoints using various forms of reasoning, in particular deduction and abduction, but also induction (see [19] or [29] for an introduction).

There exist a number of other formalisms (e.g., the Situation Calculus [16], the Fluent Calculus [32], and Temporal Action Logics [3]) to reason about actions and about how these actions change the state of the world. In particular, the differences between the Event Calculus and the Situation Calculus have been investigated in a number of comparative studies [12,13,23,35], and the development of action formalisms and the assessment of the correctness of these formalisms is still a very active research topic [33]. The Event Calculus is particularly interesting in our context because of its commitment to an explicit narrative description of events where information about conditions that can change over time can be derived immediately from the narrative discourse.

Instead of using the formal notation of the Event Calculus directly to describe a scenario and to specify the relevant background knowledge for this scenario, I modify the controlled natural language PENG Light in such a way that it can serve as a high-level interface language to the Event Calculus. That means instead of writing axioms in a formal notation that is potentially difficult to learn and understand for a subject matter expert, the author writes a specification in controlled natural language and is supported by an intelligent authoring tool that guides the writing process. The language processor of PENG Light resolves anaphoric references on the fly and generates look-ahead information that informs the author about the words and phrases that can follows the current input. The evolving specification is incrementally translated into discourse representation structures [7] and then into the input language of the Event Calculus where the formal representation can be used for deductive and abductive reasoning tasks, in particular for question answering.

The rest of this paper is structured as follows: In Section 2, I discuss the characteristics of the controlled natural language PENG Light, then I illustrate how the writing process of a specification is supported by a predictive authoring tool, and show how verbs are classified into event and state verbs. In Section 3, I introduce a short scenario written in PENG Light together with a number of

benchmark questions. Answering these benchmark questions requires additional background knowledge and a suitable inference mechanism. In Section 4, I introduce the Event Calculus as such an inference mechanism, present the language of the Simplified Event Calculus in detail, and discuss a number of modifications that are required to interface the formal output of the controlled natural language processor of PENG Light with the language of the Simplified Event Calculus. In Section 5, I show what kind of background knowledge is necessary in order to answer the benchmark questions and what kind of extensions are required on the level of the controlled natural language to express this background knowledge. In Section 6, I replace the Simplified Event Calculus that is based on the Horn clause subset of the Predicate Calculus augmented with negation-as-failure by an Event Calculus planner that is able to find explanations to questions and to deal with negative information. In Section 7, I summaries the advantages of the presented approach and conclude.

2 Controlled Natural Languages (CNLs)

Controlled natural languages are engineered subsets of full natural languages whose grammar and vocabulary have been restricted to reduce both ambiguity and complexity of full natural languages. Recently, a number of general-purpose controlled natural languages such as Attempto Controlled English (ACE) [4,5], Processable English (PENG) [25,37], and Boeing's Computer-Processable Language (CPL) [2,6] have been designed and used as high-level interface and specification languages for various knowledge acquisition and representation tasks [27].

Controlled natural languages can significantly improve the knowledge acquisition process and the understandability of a specification text compared to specifications written in formal languages [6,14], in particular if the writing process of these specifications is supported by an intelligent authoring tool [26,34] that provides a feedback mechanism and communicates the interpretation of the machine back to the author.

2.1 PENG Light

PENG Light is a controlled natural language designed for writing unambiguous and precise specification texts [37]. PENG Light covers a strict subset of standard English and is defined by a controlled grammar and a controlled lexicon. The controlled lexicon consists of domain-dependent content words, predefined function words, a number of predefined fixed phrases, and an open list of exclusion words. The author can access and inspect these words and phrases via an authoring tool during the writing process (see Section 2.3 for details).

Similar to ACE and CPL, simple PENG Light sentences have the following functional structure:

1. `subject + verb + [ complements ] + { adjuncts }`

Complements depend on the verb and are necessary constituents to establish a well-formed sentence. For example, an intransitive verb (2) does not take a complement; a transitive verb (3) takes one complement (a direct object); a ditransitive verb (4) takes two complements (an indirect object and a direct object); and a copular verb (5) links the subject with a subject complement:

2. An aircraft arrives.

3. The bus leaves the airport.

4. The booking clerk sells John a ticket.

5. John is the president of ALTA.

In contrast to complements, adjuncts are optional constituents, removing them leaves a grammatically well-formed sentence behind. In PENG Light, adjuncts are used to establish the circumstances under which the information expressed by a verb takes place. Adjuncts can be realised by adverbial phrases in the form of a prepositional phrase (6), an adverb (7), a sequence of prepositional phrases (8), or an adverb followed by one or more prepositional phrases (9):

6. John arrives with Flight AZ1777.

7. The bus stops abruptly.

8. John arrives at 10:10 with Flight AZ1777 at the airport of Palermo.

9. The bus stops abruptly at 11:50 in Alcamo.

While adverbial phrases in adjunct position always modify the verb in PENG Light, relative sentences always modify a noun, for example:

10. The bus that is at the terminal A leaves the airport at 10:30.

This brings us to complex sentences: complex sentences are built from simpler sentences in a principled way through coordination, subordination, quantification, and negation. The structure of these sentences is enforced by an authoring tool as we will see in more detail in Section 2.3.

2.2 The Language Processor of PENG Light

The language processor of PENG Light uses a unification-based phrase structure grammar and a chart parser. The chart parser processes the input sentences incrementally and generates a discourse representation structure [7] during the parsing process. This discourse representation structure is then translated into a first-order logic notation and is further processed by a model builder [37]. In this paper, I work first with a subset of PENG Light that can be translated into Horn clause logic [9], modify the controlled language and the processing of the

controlled language so that the output of the language processor can be aligned with the input language of the Simplified Event Calculus. Later I use a more expressive version of the Event Calculus that can deal with negative information.

In contrast to ACE and CPL, the processing of syntactic, semantic as well as pragmatic information is done in parallel in PENG Light. Apart from a discourse representation structure, the language processor generates look-ahead information as well as a paraphrase, and resolves anaphoric references during the parsing process. The look-ahead information instructs the author about the restrictions of the controlled natural language, and the paraphrase points out the interpretation of the machine. PENG Light allows only for restricted forms of anaphoric references, namely via definite noun phrases, proper names, and variables. In PENG Light, variables are normally introduced in apposition to a noun and can then be used later anaphorically and unambiguously instead of personal pronouns. The result of the anaphora resolution process is displayed as a paraphrase that the author can inspect.

2.3 Writing Support for PENG Light

Writing a specification in controlled natural language is not an easy task without the assistance of an intelligent authoring tool because the author would have to learn and remember the restrictions of the controlled language. For this reason, we support the writing process of a specification in PENG Light by a predictive authoring tool that guides the author and enforces the restrictions of the controlled language. That means the author of a PENG Light specification does not need to know the grammatical restrictions of the controlled language explicitly. The authoring tool clarifies these restrictions with the help of look-ahead information while the specification text is written.

In our case, the language processor communicates with a web-based authoring tool that displays look-ahead information for each syntactic category that can follow the current input. Figure 1 shows a simple web-based prototype of this authoring tool. The text field of the authoring tool accepts sentences and displays a new set of look-ahead categories whenever the author completes a word form. The author can either directly enter a word form that falls under one of these categories or click on the name of one of these categories and then select a word form from an ordered menu. The selected word form is then inserted directly into the text field. The current version of the authoring tool displays the emerging specification together with the corresponding paraphrase. The author can inspect the syntax tree, the discourse representation structure (DRS) as well as the Simplified Event Calculus (SEC) representation. It is important to note that there is no need to resort to this formal level of the representation since the entire specification can be developed on the level of the controlled natural language, and the author can ask questions about the specification in controlled natural language. If a domain-dependent content word is not available in the lexicon and is not in the list of exclusion words, then the author can add this word to the user lexicon using minimal linguistic knowledge as proposed in [26].

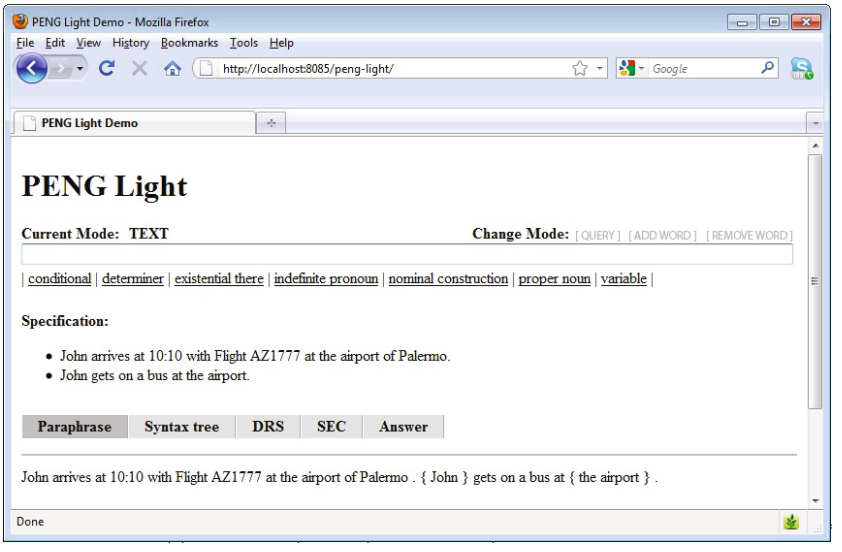


Fig. 1. Web-based Authoring Tool of PENG Light

2.4 Events and States in PENG Light

In contrast to the current version of ACE [5], PENG Light distinguishes on the semantic level between verbs that denote events and verbs that denote states. This distinction is based on a semantic theory [22] that argues for subatomic quantification over events and states in order to account for the behaviour of modifiers. In PENG Light, event verbs can be used to express what happens at a given point in time and state verbs can be used to express what holds from a given timepoint onwards. Each occurrence of a verb in PENG Light is implicitly or explicitly related to a timepoint, and the textual order establishes the temporal order of the associated event or state. For example, the following two sentences are temporally ordered, the first one (11) uses an event verb while the second one (12) uses a state verb:

11. John gets on a bus at 10:30.

12. John is in the bus.

One problem with this approach is that it does not pay attention to the effects of events. It does not tell us anything about how events are related to states and how states are updated when new information becomes available. We will discuss how we can fix this problem in Section 4.

In PENG Light, each verb is classified in the lexicon as an 'event verb' or a 'state verb'. Event verbs include intransitive, most transitive, and all ditransitive verbs; state verbs include intransitive verbs and some transitive verbs but also copula constructions; for example, the copular verb *be* plus an adjective as in (13) or the copular verb *be* plus a prepositional phrase as in (14):

13. The weather is bad.

14. John is at the airport of Palermo.

The classification of verbs into 'event verb' and 'state verb' is a basic one; there exist more detailed classifications, in particular event verbs are often further classified as activities (e.g., *run, drive*), accomplishments (e.g., *paint, build*), and achievements (e.g., *recognise, notice*) [36] depending on their interaction with aspectual and temporal modifiers. We will see that our basic classification is sufficient for our context, and it has the advantage that it can be carried out without a lot of additional linguistic knowledge. This is important because an author of a PENG Light specification should be able to add new verbs to the lexicon and classify them correctly. For this purpose, the author can use a simple test that makes it possible to distinguish between event verbs and state verbs [22]. This test uses a pseudo-cleft construction and distinguishes between things that are done (events) and things that are not done (states). The following pseudo-cleft construction is well-formed for event verbs (15) but not for state verbs (16):

15. What John does is to get on the bus. – event verb

16. *What John does is to have a ticket. – state verb

In PENG Light, thematic relations [8,22] are used to connect the events and states with other entities that are described in a sentence. These thematic relations are derived from mandatory and optional surface level constituents and from their position in the sentence. In the case of mandatory constituents, the thematic relations (plus information about events or states) are stored together with the verb in the lexicon. For example, the lexical entry for the verb form *gets on* of sentence (11) looks as follows in PENG Light:

17.
```
lexicon([cat:tv,
         wfm:[gets, on],
         syn:[vform:fin, agr:[pers:third, num:sg, case:nom],
              agr:[pers:_, num:_, case:acc]],
         sem:[event:E, arg:[ind:I1], arg:[ind:I2]],
         con:[theta(E, agent, I1), event(E, getting_on),
              theta(E, theme, I2)]]).
```

That means *gets on* is a transitive verb that is associated with two thematic relations: an `agent` and a `patient`. To complete the picture for sentence (11): the optional prepositional phrase *at 10:30* results in the additional thematic relation `time` that is derived from the preposition and the temporal expression. Sentence (11) is then translated via discourse representation structures into a set of facts:

18.
```
named(sk1, john).
theta(e1, agent, sk1).
```

```
event(e1, getting_on).
theta(e1, theme, sk2).
object(sk2, bus).
theta(e1, time, sk3).
timex(sk3, '10:30').
```

These facts consist of a small number of predefined predicates (e.g., `named/2`, `theta/3`, `event/2`, `object/2`, `timex/2`). The constant `e1` in (18) is a Skolem constant and stands for an event token of a certain type. The constants `sk1`, `sk2`, and `sk3` are Skolem constants that stand for different entities. The predicate `theta/3` connects the Skolem constant for the event `e1` via thematic relations (e.g., `agent`, `theme`, `time`) with the Skolem constants for the other entities that take part in this event.

3 A Scenario in PENG Light

Let us assume the following scenario consisting of a sequence of unfolding events that an agent can observe and monitor in controlled natural language while these events occur:

19. John arrives at 10:10 with Flight AZ1777 at the airport of Palermo.

20. There is a bus at the terminal A of the airport and John gets on that bus.

21. The bus leaves the airport at 11:30 and arrives at the port of Trapani at 13:05.

22. John gets off the bus at the port and stays in Trapani.

Let us further assume that we have some additional information about what might happen later depending on the weather conditions:

23. If the weather is good then John boards the hydroplane and travels to Marettimo Island.

24. If the weather is bad then John stays in Trapani and goes to the Albergo Maccotta.

As the scenario proceeds over time, we would like to be able to answer the following benchmark questions (25-30) expressed in controlled natural language:

25. When does John arrive at the airport?

26. Where is John now?

27. When does John get on the bus?

28. Where is John at 11:45?

29. How many kilometers is John away from the airport?
30. Why is John in Trapani at 15:00?

The first question (25) is easy to answer since the answer is part of the textual information, and we can look it up directly in the knowledge base. Finding an answer to the second question (26) is more difficult, since the answer depends on the timepoint of the question and additional background knowledge; the machine needs to know, for example, that John *is at* the airport after the arrival of the aircraft and before the departure of the bus, or that John *is in* the bus after boarding the bus. Answering the third question (27) in a precise way is not possible without recording the time when the actual event occurs, although a human can figure out the timespan relatively easily from the text. In order to find an answer to the fourth question (28), the machine needs to be able to look back in most cases in order to find out what happened before the current point in time. Answering the fifth question (29) is difficult since we need at least some knowledge about the average speed of the bus and need to be able to deal with continuous change. And finally, answering the sixth question (30) requires the machine to come up with an explanation taking incomplete information into account.

As these examples illustrate, answering the questions (26-30) automatically requires complex background knowledge about the effects that events initiate and terminate at a given point in time, information about timepoints when events occur, background knowledge about continuous change, and background knowledge about how to reason backwards from effects to causes. Apart from this background knowledge, we need a reasoning mechanism that can handle these phenomena. We show in the next section that a simplified version of the Event Calculus is a good starting point to solve these problems and to answer the benchmark questions (26-29); finally, an extension of the Event Calculus based on abduction (presented in Section 6) will provide us then with the machinery for processing *why*-questions such as (30).

4 The Event Calculus (EC)

The Event Calculus (EC) is a logic-based formalism for representing and reasoning about events, their effects and the timepoints for which these effects hold. The EC was originally introduced by Kowalski and Sergot [10] as a logic programming framework with a particular focus on database updates and narrative discourses [11]. Since then many alternative axiomatisations have been developed for the EC, and the EC has been used in many different application domains; for example, for workflow modelling, cognitive robotics, abductive reasoning, legal reasoning, mathematical modelling, and story understanding (see [18,20,21] for an overview).

4.1 The Simplified Event Calculus (SEC)

Over the time, it turned out that a simpler form of the original Event Calculus, known as the Simplified Event Calculus (SEC), can be used as a starting point

for many interesting applications [24,28]. The SEC can be extended in a systematic way so that it can deal with indirect effects, continuous change and other phenomena. The basic components of the SEC are events, fluents and timepoints. An event represents an action that may happen in the world at a given point in time. A fluent is a time-varying property that depends on an event, and a timepoint represents an instant of time on a time line. The language of the SEC allows us to specify knowledge about the effects of events on fluents, in particular knowledge about events that initiate and terminate fluents at a specific point in time. The SEC distinguishes for this purpose between domain-independent and domain-dependent axioms and is characterised by the following six predicates:

- `holds_at(F, T)` represents that the fluent `F` is true at the timepoint `T`.
- `happens(E, T)` represents that the event `E` occurs at the timepoint `T`.
- `initiates(E, F, T)` represents that the event `E` initiates the fluent `F` at the timepoint `T`.
- `terminates(E, F, T)` represents that the event `E` terminates the fluent `F` at the timepoint `T`.
- `T1 < T2` represents that the timepoint `T1` is before the timepoint `T2`.
- `initially(F)` represents that the fluent `F` holds from the timepoint zero.

Note that the variable `E` stands for an event token in these predicates, the variable `F` – in our case – for a list of complex terms, and the variables `T`, `T1` and `T2` for numbers. As we will see in the next section, we have to interface the language of the SEC with the output that the controlled language processor generates and want to do this in a direct way. For the time being, it is important to note that the variable `F` stands for a list of complex terms. Instead of representing a fluent as a simple function, we have to take special care of the particular notation that PENG Light uses for representing states including their thematic relations, and we use therefore a list as data structure. We then interpret the states in PENG Light as fluents that can be updated and can change over time. Additionally, we have to take special care of temporal expressions that occur in the standard 24 hour format in our scenario (e.g., `10:10`). We translated these expressions into POSIX time format (defined as the number of seconds elapsed since January 1, 1970 (UTC)).

4.2 The SEC at Work

The SEC relies on a number of domain-dependent axioms that monitor the effects of the events, a fixed number of domain-independent core axioms that process the domain-dependent axioms, and a few auxiliary axioms that interface the SEC with the knowledge base. In the simplest case, the domain-dependent axioms are expressed as positive and negative effect axioms. It is important to

note that fluents are subject to the commonsense law of inertia [28] which states that the truth value of a fluent persists unless the fluent is affected by an event. With the help of the two effect predicates `initiates/2` and `terminates/2`, we specify which fluents become true and which ones false when an event occurs. In order to demonstrate how these effect predicates are used in the SEC, we stick – for the time being – to the formal notation of the SEC and specify the following two clauses (31) and (32):

31.
```
initiates(E,
       [theta(sk(E, ...), experiencer, X),
        state(sk(E, ...), being),
        theta(sk(E, ...), location, Y)], T) :-
      object(X, person),
      theta(E, agent, X),
      event(E, arriving),
      theta(E, location, Y),
      object(Y, location),
      theta(E, time, Z),
      timex(Z, T).
```

32.
```
terminates(E,
       [theta(S, experiencer, X),
        state(S, being),
        theta(S, location, Y)], T) :-
      object(X, person),
      theta(E, agent, X),
      event(E, leaving),
      theta(E, theme, Y),
      object(Y, location),
      theta(E, time, Z),
      timex(Z, T).
```

The first clause (31) initiates a fluent; it uses the term `sk(E, ...)` in the head of the clause. This term is simply an abbreviation for a Skolem function applied to all quantified variables in the clause. This Skolem function serves as a unique identifier for the fluent. The second clause (32) terminates a fluent and uses a variable (`S`) instead of an identifier since this clause picks up an existing fluent and checks if there is an event (`E`) in the knowledge base that terminates that fluent. Let us assume that the first sentence (19) of our scenario here repeated as (33):

33. John arrives at 10:10 with Flight AZ1777 at the airport of Palermo.

has already been processed by the language processor of PENG Light and that the following facts are available in the knowledge base:

```
34. named(sk1, john).
    theta(e1, agent, sk1).
    event(e1, arriving).
    theta(e1, time, sk2).
    timex(sk2, 1304849400).
    theta(e1, instrument, sk3).
    named(sk3, az1777).
    theta(e1, location, sk4).
    object(sk4, airport).
    associated_with(sk4, sk5).
    named(sk5, palermo).
```

In order to trigger the effect axioms (31) and (32), we need additional rules that interface the predicates in the body of these clauses with the facts in the knowledge base. For example, in the case of the effect axiom (31) and the facts in (34), we need to allow for the inference that John is a person (35) and that every airport is a location (36):

```
35. object(X, person)   :- named(X, john).

36. object(X, location) :- object(X, airport).
```

Let us now introduce the **core** axioms of the SEC. The following two clauses (37) and (38) implement these core axioms and are used to process the domain-dependent clauses. The first clause (37):

```
37. holds_at(F, T2) :-
      happens(E, T1),
      T1 < T2,
      initiates(E, F, T1),
      \+ clipped(T1, F, T2).
```

says that a fluent `F` holds at timepoint `T2`, if an event `E` happens at the timepoint `T1`, and `T1` is before `T2`, and the event `E` initiates the fluent `F` after `T1`, if the fluent `F` has not been clipped between `T1` and `T2`. The second clause (38):

```
38. clipped(T1, F, T3) :-
      happens(E, T2),
      T1 < T2, T2 < T3,
      terminates(E, F, T2).
```

states that a fluent `F` is clipped between the timepoints `T1` and `T3`, if an event `E` happens at the timepoint `T2`, and `T2` is between `T1` and `T3`, and the event `E` terminates the fluent `F` at `T2`.

In order to interface the predicate `happens/2` that occurs in the core axioms with the information about events and their temporal modifiers in the knowledge base, we need an auxiliary clause of the following form:

```
39. happens(E, T) :-
      event(E, N),
      theta(E, time, Y),
      timex(Y, T).
```

It is often convenient to use an additional core axiom to specify the initial situation of a scenario with the help of the predicate `initially/1` in the body of the clause. The following clause (40) does this; it specifies that a fluent `F` holds at the timepoint `T`, if it held `initially` and has not been clipped between the timepoint `0` and `T`:

```
40. holds_at(F, T) :-
      initially(F),
      \+ clipped(0, F, T).
```

Also the predicate `initially/1` that occurs in the body of (40) needs to be interfaced with the information in the knowledge base (if some initial information is available there):

```
41. initially(
        [theta(S, R1, X),
         state(S, T),
         theta(S, R2, Y)]) :-
      theta(S, R1, X),
      state(S, T),
      theta(S, R2, Y),
      X \= Y.
```

Given the translation (34) of the first sentence of the scenario together with the effect axioms (31, 32), the core axioms (37, 38) of the SEC, and the auxiliary axioms (35, 36, 39), we can now investigate which fluents hold at a given point in time. Still working on the formal level, we can use Prolog's built-in predicate `findall/3` and the user-defined predicate `get_current_time/1` that returns the current timepoint for this purpose:

```
42. ?- get_current_time(T), findall(F, holds_at(F, T), FL).
   FL = [ theta(sk(e1, ...), experiencer, sk1),
          state(sk(e1, ...), being),
          theta(sk(e1, ...), location, sk4) ].
```

Of course, we can refine this query and other queries are possible at different timepoints as the scenario proceeds depending on the effect axioms. In the next section, we show how we can express the scenario and the required domain-dependent axioms of the SEC in controlled natural language. This has the advantage that a subject matter expert does not need to encode the relevant knowledge in a formal notation [4,31].

5 Specifying Background Knowledge in PENG Light

As we have seen in the last section, the SEC provides a general framework for reasoning about events, their effects and timepoints where the effects of events are described with the help of positive and negative effect axioms. We need a way to express these axioms unambiguously in PENG Light and be able to distinguish between positive and negative effects, indirect effects and to deal with knowledge about continuous change.

5.1 Specifying Axioms for Direct Effects in PENG Light

We can use direct effects axioms to predict the outcome of an event. Direct effect axioms can capture the effects that an event has on a fluent. They can take the conditions into account under which an event occurs and update the fluents that will be holding or will no longer be holding when the event has been performed. We use conditional sentences for this purpose and employ the continuous future tense in the consequence of these conditional sentences in order to make the status of the fluent explicit; for example, instead of (31) and (32) in SEC notation, we write these axioms as follows in PENG Light:

43. If a person arrives at a location then the person will be at that location.

44. If a person leaves a location then the person will no longer be at that location.

The two expressions *will be* and *will no longer be* in the consequence of these conditional sentences correspond to the underlying `initiates/2` and `terminates/2` predicates in the SEC notation. The positive effect axiom (43) and the negative effect axiom (44) are written in a way that abstracts away from a specific person and a specific location (as already illustrated in (31) and (32)). Therefore, we need additional terminological statements about the domain that will trigger the inference process. These terminological statements can be expressed directly in PENG Light and correspond to the two clauses (35) and (36) in the SEC notation:

45. Whoever is John is a person.

46. Every airport is a location.

Given this kind of background knowledge, we can now successfully answer the benchmark question (26), repeated here as (47):

47. Where is John now?

In PENG Light, questions are translated via discourse representation structures into Prolog queries that can then be answered with the help of the SEC axioms. The translation of question (47) results among other things in a `holds_at/2` predicate that takes a template for a fluent and the current timepoint (now = `1304851800`) as input:

```
48. ?- named(X, john),
       holds_at(
        [theta(S, experiencer, X),
         state(S, being),
         theta(S, location, Y)], 1304851800),
       (object(Y, O) ; named(Y, N)).
```

We can now also answer the benchmark question (27), but this is not directly dependent on these effect axioms but rather a side-effect of our decision to time-stamp all events that occur without an explicit temporal marker in the scenario.

5.2 Specifying Axioms for Indirect Effects in PENG Light

Let us consider the case where John gets on a bus and the bus leaves the airport. The second event has the indirect effect that John will no longer be at the airport. The problem of representing and reasoning about indirect effects of events is known as the ramification problem [15] that is strongly related to the frame problem [28]. Recall fluents normally obey the commonsense law of inertia in the EC. This law states that a fluent stays the same unless it is directly initiated or terminated by an event, but this is not the case if John is in the bus and the bus leaves the airport, since John will no longer be at the airport.

There exist a number of methods for dealing with the ramification problem [19]; one method is to explicitly release a fluent from the commonsense law of inertia when a specific event occurs, and to make the fluent subject to a state constraint, and later when another event occurs to restore this law again. This method would require the addition of a new **core** axiom to the SEC that releases a fluent from inertia and can be used to restore that fluent again.

Another method of dealing with the ramification problem is to represent indirect effects in the same way as direct effects, namely with the help of positive and negative effect axioms [19]. While the first method allows for a more abstract representation, the second method is simpler to understand for a subject matter expert, since it does not require the explicit release and restoration of fluents from the commonsense law of inertia.

We use the second method for specifying indirect effects in PENG Light, choose again a suitable level of abstraction for our context, and write:

49. If a person is at a location and a vehicle is at that location and the person gets on the vehicle then the person will be in the vehicle.

50. If a person is in a vehicle and the vehicle leaves a location then the person will no longer be at that location and the vehicle will no longer be at that location.

As these examples illustrate, we make sure that the person is wherever the vehicle is – as long as the person is in the vehicle. Of course, we need also additional assertional and terminological knowledge and have to specify, for example, that

the bus is at the airport and that every bus is a vehicle in order to relate the information expressed in the scenario with the effect axioms. Given this background knowledge, we can now successfully answer the benchmark question (28).

5.3 Specifying Axioms for Continuous Change in PENG Light

Let us now consider the case where the bus leaves the airport, and we want to find out how far away the bus is from the starting point. Answering this kind of questions requires us to deal with continuous change. If we know that a vehicle is travelling with a fixed velocity of, let's say 75 km/h, then we can calculate the distance from the starting point with the help of the following conditional sentence in PENG Light:

51. If a vehicle is travelling and the vehicle is X km away from a starting point at a timepoint T1 and [Y is X + (75 * (T2 - T1) / 3600)] then the vehicle will be Y km away from the starting point at the timepoint T2.

We can embed simple arithmetic expressions that basically follow Prolog syntax into the controlled natural language. These arithmetic expressions are parsed and maintained in a list in the discourse representation structure. The syntax of these arithmetic expressions is enforced by the look-ahead editor in a similar way as the syntax for the linguistic expressions. The left argument of the `is` operator in (51) has to be a variable and the right argument has to be a mathematical function. Arithmetic expressions can only occur in the conditions of conditional sentences and have to be embedded in square brackets. The variables on the right-hand side of such an arithmetic expression need to be instantiated before the expression can be evaluated. Note that the variables in the arithmetic expression interact with the variables in the conditional sentence; for example in (51), the variables *X* and *T1* are introduced in the text and then used in the arithmetic expression while the variables *Y* and *T2* are introduced in the arithmetic expression and then used in the consequence of the conditional sentence. The language processor then generates a paraphrase that shows all anaphoric relations that occur in this sentence in curly brackets:

52. If a vehicle is travelling and { the vehicle } is X km away from a starting point at a timepoint T1 and [Y is { X } + (75 * (T2 - { T1 }) / 3600)] then { the vehicle } will be { Y km } away from { the starting point } at { the timepoint T2 }.

The translation of (51) results in a new SEC predicate (`trajectory(F1, T1, F2, T2)`). This predicate extends the SEC and is used in the head of a clause, in a similar way as the effect predicates introduced in Section 4.2. It represents that if a discrete fluent `F1` has been initiated by an event at the timepoint `T1`, then the continuous fluent `F2` holds at the timepoint `T2`. The trajectory predicate depends on further conditions and a continuous function in the body of the clause as the translation of (51) illustrates:

```
53. trajectory(
      [theta(S1, experiencer, A),
       state(S1, being),
       theta(S1, predicate, travelling)], T1,
      [theta(sk(S2, ...), experiencer, A),
       state(sk(S2, ...), being),
       theta(sk(S2, ...), origin, B),
       theta(sk(S2, ...), direction, away),
       theta(sk(S2, ...), quantity, [eq, Y, km])], T2) :-
         object(A, vehicle),
         object(B, starting_point),
         holds_at(
           [theta(S2, experiencer, A),
            state(S2, being),
            theta(S2, origin, B),
            theta(S2, direction, away),
            theta(S2, quantity, [eq, X, km])], T1),
         Y is X + 75 * ( ( T2 - T1 ) / 3600 ).
```

In order to process the trajectory axiom (53), we need to add an additional domain-independent **core** axiom to the SEC. This **core** axiom deals with the trajectory of a continuously changing quantity and makes sure that the fluent `F1` is first initiated by an event `E` before it is processed by the trajectory predicate:

```
54. holds_at(F2, T2) :-
      happens(E, T1),
      T1 < T2,
      initiates(E, F1, T1),
      trajectory(F1, T1, F2, T2),
      \+ clipped(T1, F1, T2).
```

Additionally, we need to specify on the level of the controlled natural language the initial distance (55) and describe what happens when a discrete fluent is turned into a continuous one and vice versa (56-59). We do this with the help of the following statements in PENG Light:

55. The bus is 0 km away from the airport.

56. If a vehicle leaves a location then the vehicle will be travelling.

57. If a vehicle stops then the vehicle will no longer be travelling.

58. If a vehicle is X km away from a starting point and stops then the vehicle will be X km away from the starting point.

59. If a vehicle leaves a starting point then the vehicle will no longer be X km away from the starting point.

Given this kind of background knowledge (51, 55-59) written in controlled natural language, the new core axiom (54), and additional terminological axioms that relate an airport to a starting point and a starting point to a location, then we can successfully answer the benchmark question (29). Note that as soon as we state that the bus stops, the distance does not change anymore.

6 Finding Explanations via Abduction

As we have seen, the SEC can be used to solve prediction problems through deduction. Deductive reasoning is sound and makes explicit what is already implicitly present in the premises; it is the only form of allowed reasoning in fields where rigorous arguments are required.

Sometimes, however, we have to deal with incomplete information and make assumptions in order to reach a conclusion. This is in particular the case, if we want to get an explanation for an observation. In such a situation, we usually start from a set of observed facts (= observation) and use the given clauses (= theory) to derive the relevant conditions (= explanation) that can explain the observed facts. This form of reasoning is known as abductive reasoning [17]. Abductive reasoning is not sound, but it captures many aspects of human reasoning. It can be used to find possible explanations and has successfully been used to solve planning problems with the help of an abductive version of the Event Calculus [30].

Let us take again the initial scenario as starting point where we can observe that John stays in Trapani. Given this information and the following effect axiom (60), plus the triggering axiom (61):

60. If a person stays at a location then the person will be at that location.

61. If the sirocco wind blows then the weather is bad.

together with the necessary terminological information, we want to answer the *why*-question (30) – here repeated as (62):

62. Why is John in Trapani?

and abduce the preconditions of (61) – the sirocco wind blows – as a possible explanation. Note that this information has not been previously asserted as a set of facts to the knowledge base.

In order to abduce explanations, we take up an idea from Shanahan [30] and compile the core axioms of the SEC into meta-level clauses implementing an abductive planner in form of a meta-interpreter. The task of this meta-interpreter is to collect those predicates that cannot be proved from the object-level program. The following meta-level clause implements the core axiom (37) of the SEC and takes part in the generation of an explanation:

63.
```
abduce([holds_at(F, T2)|Goals1],
          Residue1, Residue5, NGoals1, NGoals4) :-
    abresolve(initiates(E, F, T1), Goals3, Residue1, Residue1),
    abresolve(happens(E, T1), Goals2, Residue1, Residue2),
    abresolve(before(T1, T2), [], Residue2, Residue3),
    append(Goals2, Goals3, Goals4),
    append(Goals4, Goals1, Goals5),
    add_neg([clipped(T1, F, T2)], NGoals1, NGoals2),
```

```
abduce_nafs(NGoals2, Residue3, Residue4, NGoals2, NGoals3),
abduce(Goals5, Residue4, Residue5, NGoals3, NGoals4).
```

The variables $\texttt{Residue}_x$ are used to collect the abducible predicates during the question answering process (= proof). There are two technical details that are important here. Firstly, special care needs to be taken of the negation as failure operator (\+) that occurs in the core axiom (37) and of the execution order of the subgoals. Since we are working here on the meta-level, negated goals ($\texttt{NGoals}_x$) need to be recorded and checked each time the residue is modified. Secondly, the subgoals (`initiates/3` and `happens/2`) on the right hand side of the clause are not executed in the same order as in (37); this is to prevent looping during abductive reasoning. The meta-interpreter can be extended to handle negative information so that we can also specify, for example, that John is not in Palermo. This results in two EC axioms that use the predicate `holds_at(neg(F), T)` with a `neg` function on the object-level which gives us additional expressiveness since these axioms are outside of the Horn clause fragment [30]. These predicates are then again compiled into a meta-level clauses, similar to (63), and are part of the abductive meta-interpreter.

7 Conclusions

For the descriptions of many dynamic problems that occur in the real world, an expressive specification language is required that can be used to describe knowledge about events and their effects in a way that is on the one hand easy to understand by a subject matter experts and on the other hand fully processable by a machine. In this paper, I worked towards such a specification language and showed how the controlled natural language PENG Light can be modified and used as a high-level interface language to the Event Calculus. The Event Calculus is a powerful logic-based formalism for reasoning about events, their effects and timepoints and can be used for deductive and abductive reasoning tasks, in particular for question answering. I showed with the help of a scenario written in PENG Light that there is in principle no need to express this scenario and the required background knowledge in a formal notation. Instead the entire specification can be written in controlled natural language and then be translated automatically via discourse representation structures into the input language of the Event Calculus. In particular, I showed how PENG Light can be used to state direct and indirect effect axioms, how to express axioms for continuous change, and how to answer a number of benchmark questions. To the best of my knowledge, PENG Light is the first controlled natural language that allows an author to write a specification text for the Event Calculus in a subset of English. There are many interesting phenomena that require further research like the processing of compound events, nondeterministic and concurrent events.

References

1. Clark, P., Porter, B.: KM – The Knowledge Machine 2.0: Users Manual. Department of Computer Science, University of Texas at Austin (2004)
2. Clark, P., Harrison, P., Jenkins, T., Thompson, J., Wojcik, R.H.: Acquiring and Using World Knowledge using a Restricted Subset of English. In: Russell, I., Markov, Z. (eds.) Proceedings FLAIRS 2005, pp. 506–511 (2005)
3. Doherty, P., Gustafsson, J., Karlsson, L., Kvarnström, J.: Temporal action logics (TAL): Language specification and tutorial. Linköping Electronic Articles in Computer and Information Science 3(15) (1998)
4. Fuchs, N.E., Schwertel, U., Schwitter, R.: Attempto Controlled English – Not Just Another Logic Specification Language. In: Flener, P. (ed.) LOPSTR 1998. LNCS, vol. 1559, pp. 1–20. Springer, Heidelberg (1999)
5. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web. LNCS, vol. 5224, pp. 104–124. Springer, Heidelberg (2008)
6. Gunning, D., Chaudhri, V.K., Clark, P.K., Barker, K., Chaw, S.-Y., Greaves, M., Grosof, B., Leung, A., McDonald, D.D., Mishra, S., Pacheco, J., Porter, B., Spaulding, A., Tecuci, D., Tien, J.: Project Halo Update – Progress Toward Digital Aristotle. AI Magazine 31(3), 33–58 (2010)
7. Kamp, H., Reyle, U.: From Discourse to Logic. Kluwer, Dordrecht (1993)
8. Kipper, K., Korhonen, A., Ryant, N., Palmer, M.: Extending VerbNet with Novel Verb Classes. In: Proceedings of LREC 2006, Genoa, Italy (May 2006)
9. Kowalski, R.: Logic for Problem Solving. Elsevier North Holland, New York (1979)
10. Kowalski, R., Sergot, M.: Logic-Based Calculus of Events. New Generation Computing 4, 67–95 (1986)
11. Kowalski, R.: Database Updates in the Event Calculus. Journal of Logic Programming 12, 121–146 (1992)
12. Kowalski, R., Sadri, F.: The Situation Calculus and Event Calculus Compared. In: Proceedings of ILPS, pp. 539–553 (1994)
13. Kowalski, R., Sadri, F.: Reconciling the Event Calculus with the Situation Calculus. Journal of Logic Programming, Special Issue: reasoning about action and change 31(1-3), 39–58 (1997)
14. Kuhn, T.: Controlled English for Knowledge Representation. Doctoral Thesis, University of Zurich (2010)
15. McCain, N., Turner, H.: A Causal Theory of Ramifications and Qualifications. In: Proceedings of 14th IJCAI, pp. 1978–1984 (1995)
16. McCarthy, J., Hayes, P.J.: Some Philosophical Problems from the Standpoint of Artificial Intelligence. In: Michie, D., Meltzer, B. (eds.) Machine Intelligence 4, pp. 463–502. Edinburgh University Press (1969)
17. McKaughan, D.J.: From Ugly Duckling to Swan: C. S. Peirce, Abduction, and the Pursuit of Scientific Theories. Transactions of the Charles S. Peirce Society: A Quarterly Journal in American Philosophy 44(3), 446–468 (2008)
18. Miller, R., Shanahan, M.: Some Alternative Formulations of the Event Calculus. In: Kakas, A.C., Sadri, F. (eds.) Computational Logic: Logic Programming and Beyond, Part II. LNCS (LNAI), vol. 2408, pp. 452–490. Springer, Heidelberg (2002)
19. Mueller, E.T.: Commonsense Reasoning. Morgan Kaufmann Publishers (2006)
20. Mueller, E.T.: Event Calculus. In: van Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Representation, ch. 17, pp. 671–708 (2008)

21. Mueller, E.T.: Automating Commonsense Reasoning Using the Event Calculus. Communications of the ACM 52(1), 113–117 (2009)
22. Parsons, T.: Events in the Semantics of English: A Study in Subatomic Semantics. Current Studies in Linguistics. MIT Press (1994)
23. Pinto, J., Reiter, R.: Temporal Reasoning in Logic Programming: A Case for the Situation Calculus. In: Proceedings of ICLP, pp. 203–221 (1993)
24. Sadri, F., Kowalski, B.: Variants of the Event Calculus. In: Proceedings of ICLP, pp. 67–81 (1995)
25. Schwitter, R.: English as a Formal Specification Language. In: Proceedings of DEXA 2002, Aix-en-Provence, France, September 2-6, pp. 228–232. NLIS (2002)
26. Schwitter, R., Ljungberg, A., Hood, D.: ECOLE – A Look-ahead Editor for a Controlled Language. In: Controlled Translation, Proceedings of EAMT-CLAW 2003, May 15-17, Dublin City University, Ireland, pp. 141–150 (2003)
27. Schwitter, R.: Controlled Natural Language for Knowledge Representation. In: Proceedings of COLING 2010, pp. 1113–1121 (2010)
28. Shanahan, M.P.: Solving the Frame Problem. A Mathematical Investigation fo the Common Sense Law of Inertia. MIT Press, Cambridge (1997)
29. Shanahan, M.P.: The Event Calculus Explained. In: Veloso, M.M., Wooldridge, M.J. (eds.) Artificial Intelligence Today. LNCS (LNAI), vol. 1600, pp. 409–430. Springer, Heidelberg (1999)
30. Shanahan, M.P.: An Abductive Event Calculus Planner. Journal of Logic Programming 44(1-3), 207–240 (2000)
31. Sowa, J.F.: Architectures for Intelligent Systems. IBM Systems Journal 41(3), 331–349 (2002)
32. Thielscher, M.: Introduction to the fluent calculus. Electronic Transactions on Artificial Intelligence 2(3-4), 179–192 (1998)
33. Thielscher, M.: A Unifying Action Calculus. Artificial Intelligence 175(1), 120–141 (2010)
34. Thompson, C.W., Pazandak, P., Tennant, H.: Talk to Your Semantic Web. IEEE Internet Computing 9(6), 75–78 (2005)
35. Van Belleghem, K., Denecker, M., De Schreye, D.: Combining Situation Calculus and Event Calculus. In: Proceedings of ICLP, pp. 83–97 (1995)
36. Vendler, Z.: Verbs and times. The Philosophical Review 66(2), 143–160 (1957)
37. White, C., Schwitter, R.: An Update on PENG Light. In: Pizzato, L., Schwitter, R. (eds.) Proceedings of ALTA 2009, Sydney, Australia, pp. 80–88 (2009)

Using CNL Techniques and Pattern Sentences to Involve Domain Experts in Modeling

Silvie Spreeuwenberg[1], Jeroen van Grondelle[2], Ronald Heller[2], and Gartjan Grijzen[2]

[1] LibRT, Amsterdam, The Netherlands
silvie@librt.com
[2] Be Informed, Apeldoorn, The Netherlands
{j.vangrondelle,r.heller,g.grijzen}@beinformed.com

Abstract. Involving domain experts in modeling is important when knowledge needs to be captured in a model and only domain experts can establish whether the models are correct. We have experienced that a natural language based representation of a model helps them to understand the semantics of a model and has advantages over a visual representation. Therefore a controlled natural language (CNL) is designed for our existing semantic reasoning tool Be Informed, which is based on conceptual graphs. The resulting CNL has a formal logical basis but the goal of the CNL representation is to improve readability for human readers. We report on the challenge to develop a CNL that 1) is easy and intuitively readable for domain experts with no background in formal logics, 2) can be easily generated from the formal representation and 3) can be easily adjusted for other natural languages and cultural preferences. The solution uses patterns to represent the CNL that map to the conceptual graph. The patterns are based on SBVR's RuleSpeak and can be easily adjusted for local differences.

Keywords: Controlled Natural Language, Business Rules, Specifications, Knowledge Representation, CNL Design and Evaluation, SBVR, RuleSpeak.

1 Need for Controlled Natural Languages in Modeling

The adoption of model driven technologies such as Enterprise Decision Management and Business Process Management is growing. As a result, involving business users in modeling is more important than ever. Their ability to capture business knowledge in models correctly is a key factor in the adoption of these technologies. The main challenge in involving business users in knowledge modeling is the fact that most business users are not trained in formal knowledge representation techniques. A formal, concise, visual representation can be quite intimidating to the uninitiated. Consequently, they will not be able to verify the accuracy of the model directly.

Typically, the business users involved in modeling are experts in their domain, acting in positions that are vital to the execution of their business, such as

M. Rosner and N.E. Fuchs (Eds.): CNL 2010, LNAI 7175, pp. 175–193, 2012.

for instance public sector organizations dealing with benefits and permits and financial institutions in areas such as insurance and pensions. They typically represent business owners, whose main concern is whether the services based on the modeling will help, both in daily operations and in reaching the organizations goals. They are typically highly educated, very experienced and respected in their field. Yet they have no training or experience in formal representation techniques. And often, they are somewhat technology averse. Whether the result of role perception, or earlier experiences in classical system development projects, it's this combination of skills and mindset towards modeling that poses challenges in involving business users at the right level.

The ultimate ambition in business user involvement is often having business users model their own business. This suggests maximum ownership for the business units and would allow for a single step implementation process for changing business policies: The policy owner can just alter the models used himself. Although designing and creating a sizable business ontology will remain out of reach in many cases, or at least will require extensive patterns to abstract away from the modeling, there are excellent examples of business users taking active ownership of models and maintaining and altering them with minimal involvement of information professionals.

Probably even more important, and within reach in almost all phases of business modeling, is involving business users in a number of roles that do not involve actual modeling.

1. Enable business users to review models for accuracy of representation. In this way, trained professionals perform the modeling work, but business users are able to review their work and provide specific feedback on modeling constructs that they do not agree with;
2. Enable dissemination and communication: Sharing the models among stakeholders, both in the modeling phase and later on, when the models are used to drive business processes and decision making;
3. Explaining individual cases that were treated based on the rules in the models. Often, decisions have to be explained in for instance rejection letters. If the decision is taken based on models, the explanation should be derived from those same models;
4. If business services and applications are executed based on models, users of these applications should be enabled to provide feedback from their operation perspective on how models could be improved to meet their needs.

Typically, graph oriented representations are not suitable for any of these tasks. Early in Be Informed's existence, experiments were performed to use natural language to facilitate business users in participating in modeling efforts.

A first triple-oriented textual representation was used to communicate a risk taxonomy to classify shipments of goods to insurance underwriters. Although the sentences produced were very basic and consisted of just the subject and object of a triple with a verb in between encoding the relation type, the underwriters immediately spotted constructs that appeared odd to them. This resulted in an improved recall rate of modeling errors.

This early success has motivated further research and the development of a pattern oriented approach. A next version of the text generator [5] was used at the Dutch Immigration Office [9] to validate candidate policy decisions for consistency before they are accepted. In workshops with business representatives and legal advisors, the policy is defined in the tool that also will be used to execute this policy. Both a visual graph oriented representation and the textual representation discussed in this paper were used. It is important to note that the parties involved here were unfamiliar with formal representation techniques and would normally express any policy in unrestricted, natural language. The expectation that the textual representation was preferred over the diagrams was confirmed by the participants. An interesting new observation was that the sentence should be a grammatically correct sentence: Even small errors, which did not seem to obfuscate the meaning of the sentence at all, made sentences less effective and less acceptable to their audience.

In parallel, some techniques were evaluated in the area of Domain Specific Languages. Originating in software development, they aim to create small, problem centric languages that meet a single problem domain well. Technologies, as for instance developed in the Eclipse Modeling Project [3], are available to develop editors with both visual and textual grammars. The textual grammars developed in this community often stay close to programming languages, with respect to their syntax and formatting conventions. As a consequence, these experiments suffered from the same weaknesses that the graphical notation does: Business users regard them as technical artifacts, and do not adopt them naturally.

This paper reports on the design and implementation of a CNL that helps Be Informed customers to actively participate in modeling knowledge.

2 Related Work

Controlled languages are often classified in one of two categories, as described for instance by Clark [2]: those that improve readability for human readers and those that enable reliable automatic semantic analysis of the language. The language that we designed has a basis in formal logic. But all too often languages in the second category do not read very naturally. The challenge for Be Informed was to design a language that can be easily generated from a conceptual graph and is natural for people to read and understand.

2.1 Syntax Based Approaches

Using CNLs to represent ontologies has been done before, for instance in Attempto Controlled English [11] and CLOnE [4]. They both use natural language generation (NLG) to create a textual representation and natural language processing (NLP) to roundtrip the ontology based on the changed text.

The textual syntax definition proposed in this paper is quite similar to the definition used in CLOnE. Our approach towards editing a model based on a natural language representation does not use NLP and has more in common

with Conceptual Authoring [16]. Editing is not performed by manipulating text but by performing editing operations at the concept level, with the text being updated to reflect concept-level changes.

2.2 Pattern Based Approaches

Our current mechanism is based completely on pattern sentences. Reiter wrote an interesting paper [17] on the differences, pros and cons of pattern oriented approaches vs NLG. We propose some future work on hybrid approaches in Section 6.

The use of pattern sentences positions our work in the tradition of NLG and template filling. A drawback of this approach is that we need pattern templates for each language as described by [10]. We propose future work (see Section 6) to research issues with contextual translation and to minimize the effort for pattern development. Compared to Terry Halpins work on (automated) verbalization of ORM models [7] [8] our work follows a similar approach but our patterns are not as generic. We create domain specific grammars that result in domain specific patterns that help domain experts familiar with the specific domain to express additional knowledge. In our example (see Section 3) we provide patterns for a product domain to add knowledge about different kinds of products and product discounts. The choice for domain specific patterns is based on the fact that they are easier for domain experts to understand and learn, but increases the work to develop patterns. Consequently in our future work section we focus on approaches to minimize the pattern development work effort.

2.3 Textual versus Graphical Representations

We started with a graphical representation of our ontologies and described our extension to a textual representation. This relates our work to a debate with a long history on when a textual model is preferable to a graphical model and vice versa. A discussion in [6] concludes that there is some evidence that text-based modeling constitutes a noteworthy alternative to graphical modeling because of its simple usage, scalability and easy development and reuse of tool support. Although this is a claim in favor of our choice for textual modeling, it does not align with our motivation. We are motivated by the fact that legal advisors and business people are used to information in textual form. We have experienced graphical representations as a barrier for them. The graphical models may still be useful for another audience to show the general picture. There are not many empirical studies comparing the graphical approach with the textual approach. Since our tool will support both approaches it may be used to support such studies.

3 Using Pattern Sentences to Verbalize Models

We want to bridge the gap between formal and natural approaches, so our challenge is to design a language that can be easily generated from a conceptual graph and still is natural and understandable for human users.

3.1 Separating Syntax from Semantics

Be Informed develops a software suite that is used by complex, knowledge intensive organizations to capture their business knowledge and run model driven services based on these knowledge models. Knowledge representation in Be Informed is based on conceptual graphs, containing concepts, relations between concepts and properties of both concepts and relations. To add semantics, the concepts, relations and properties are typed, using types from a meta model associated with the graph.

For the purpose of presenting and editing these models, they are visualized to users using a syntax that matches the semantic information in the meta model. These syntaxes can be both graphical and textual, as is visualized in Figure 1. For instance, the default visualization in Be Informed is a graphical visualization, based on a visual syntax that maps iconography, line styles and colors to meta model types.

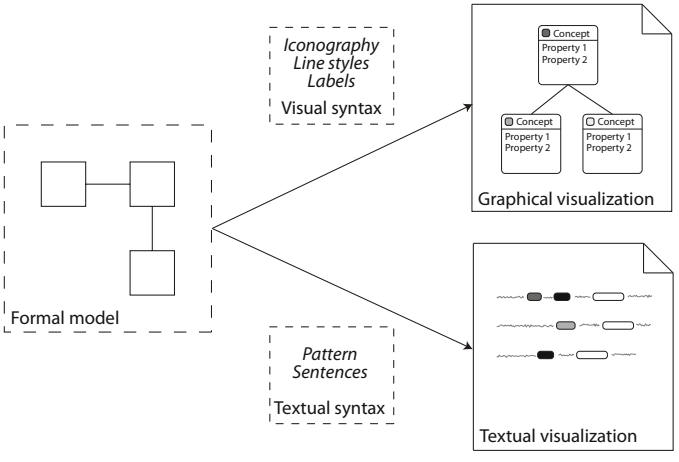


Fig. 1. Visualizing models using both visual and textual syntaxes

This paper proposes an alternative visualization of the graph based on natural language pattern sentences. The visualization is defined by a grammar of pattern sentences, which consists of natural language text with placeholders that map to the graph's concepts and properties by their types and relations. These sentences are handcrafted to communicate the semantics of the graph constructs they represent and are based on the best practices in the Business Rule community like for example RuleSpeak. This approach brings us several additional advantages:

1. Multiple visualisations: All kinds of textual constructs or graphical visualizations that are valid representations of the formal model can be constructed in order to be 'fit for purpose'.

2. Multiple languages: the pattern sentences are mapped to the formal model, and therefore it is possible to implement pattern sentences in other languages for the same formal model.
3. Multiple target groups: like multilinguality, the language levels are a major factor in communication. One could create multiple sentence templates for different CEF levels (Common European Framework Levels). An example would be the vocabulary used by helpdesk employees versus the vocabulary of domain experts.

3.2 Mapping between Graphs and Pattern Sentences

When visualizing a graph, the structural mapping constraints that are part of the pattern sentences are applied to the graph's concepts and relations. These pattern sentences are expressed in terms of the types in the meta model. Then, instances of the pattern sentences are shown for instances of the meta model types, which are the concepts and relations in the graph. These relations are visualized in Figure 2.

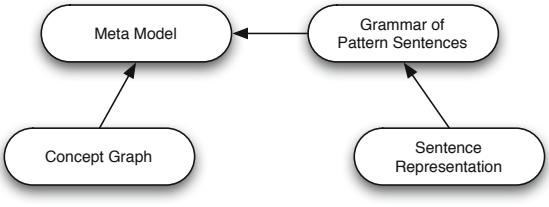


Fig. 2. Pattern based generation approach

Pattern sentences consist of different kinds of parts. They have their own properties that are used as constraints when mapping the pattern sentences to a graph, as seen in Figure 3.

1. Static text fragments contain the wording of the sentences;
2. Subject placeholders map to concepts that act as the subject of a sentence and include its label in the text;
   (a) The subject placeholder maps to subjects of specified type;
3. Object placeholders map to the objects of the sentences' subject relations and include their label in the text;
   (a) The object placeholder maps to objects of specified type;
   (b) The object placeholder maps to objects of relations with the subject of specified type;
   (c) The object placeholder concatenates the objects labels according to the number of objects and with configurable infixes;

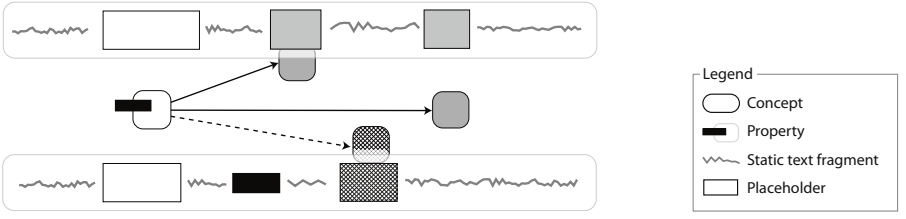


Fig. 3. Mapping pattern sentences to a concept graph

4. Property placeholders map to the subject's properties and include their value in the text;
   (a) The property placeholder maps to properties of specified type.

Fragments and placeholders are grouped into sentence parts, in order to make certain parts of the sentence optional. If the graph construct they map to does not exist, the other parts of the sentence might still apply and form a valid textual representation.

Pattern sentences have to deal with cardinality in the (meta)model. In a trivial case, where sentences encode for one relation instance only, having more than one relation is represented in language by introducing a sentence for each relation. Pattern sentences can also represent multiplicity within a sentence, so that one sentence encodes a number of relations and concepts. A grammar can contain one sentence to encode for more multiple relations of the same type. For any subject, all these relations are then represented in a single sentence with the objects of the relations enumerated inside. A sentence can also encode for relations of more than one type. In that case, sentence parts encode for the different relation types, and they are concatenated into a single sentence.

3.3 Example: A Telecom Product Model

In this section we present an example based on the product model of a telecom provider.

The meta model in Figure 4 is based on typical taxonomical structures for modeling products and associated discounts. The requirement relations connect the discounts with the products, or combinations of products they apply to. This basic meta model enables the modeling of both the provider's product portfolio structure and the requirements its client must meet to apply for specific target group discounts.

An example of a product and discount model in the traditional graphical notation is given in Figure 5. It describes the telephone, television and internet products the provider has and how customers apply for specific discounts. For example, a consumer ordering all three products (Internet, Telephone and TV) applies for a triple play discount.

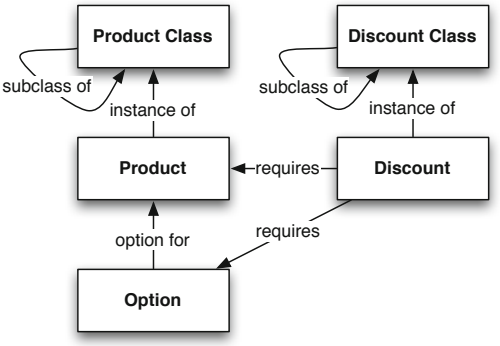


Fig. 4. Meta model for the Telecom example

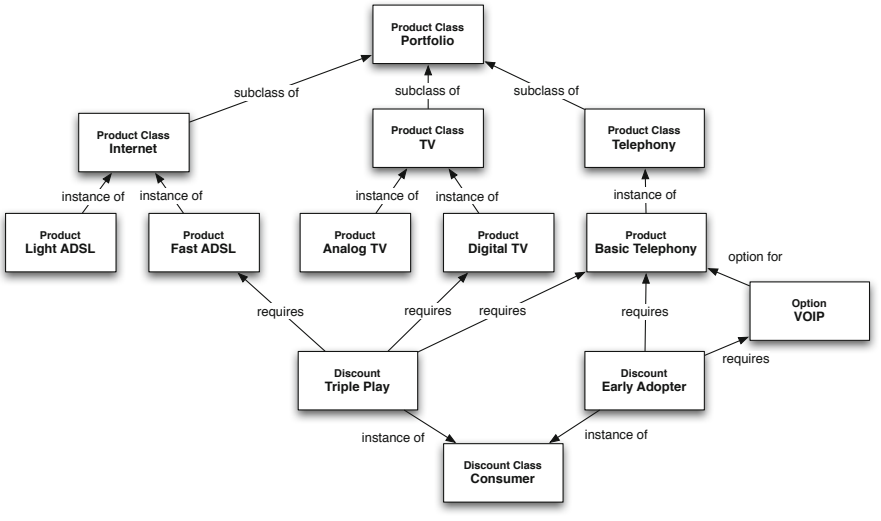


Fig. 5. Product model for the Telecom example

To visualize the product model in natural language, we need to specify a grammar of pattern sentences first; such a grammar should mirror the information in the meta model shown in Figure 4. The example grammar introduces product classes and individual products in short, structural sentences. The following two patterns are possible parts of this grammar:

1. "THERE IS A CLASS OF PRODUCTS NAMED $<ProductClass>$." $\leftrightarrow$ $\{ProductClass\}$.
2. "THE PRODUCT $<Product>$ IS A $<ProductClass>$ PRODUCT." $\leftrightarrow$ $\{Product, instanceof, ProductClass\}$.

The first sentence pattern can be used to declare a class of products, while the second is meant to declare a specific product and to which class it belongs. The following two sentences are examples of such a specific declaration:

I There is a class of products named **Internet**.

II The product **Fast ADSL** is an **Internet** product.

Introducing discounts requires a more complex structure of several sentence parts that can be linked together.

3. a) "The discount $<Discount>$ is a $<DiscountClass>$ discount," $\leftrightarrow \{Discount, instanceof, DiscountClass\}$

   b) "and customers apply for it by ordering [the product/all of the products] $<Product>$," $\leftrightarrow \{Discount, requires, Product\}$

   c) "with the option[s] $<Option>$." $\leftrightarrow \{Discount, requires, Option\}$

The first part introduces a discount and encodes for its type relation, as it is mandatory. The second part encodes for the products required to apply for the discount. The third part encodes for the options that are required to apply for the discount, if any. Based on the product model from Figure 5, the following textual representation of the two types of discounts can be constructed:

III The discount **Triple Play** is a **Consumer** discount, and customers apply for it by ordering all of the products **Fast ADSL**, **Digital TV** and **Basic Telephony**.

IV The discount **Early Adopter** is a **Consumer** discount, and customers apply for it by ordering the product **Basic Telephony**, with option **VOIP**.

3.4 Example: Verbalizing a Process Ontology

In this second example, we show how a similar approach can be applied to an ontology of process concepts. In this example, the process of applying for a subsidy is described in terms of a case oriented meta model as shown in Figure 6. It introduces concepts for the activities that are performed, the prerequisites that have to be met for each activity to be performed and the consequences that each activity has, in terms of decisions taken and artifacts like documents and notes created.

The graphical visualization of the process of applying for a subsidy is shown in Figure 7. The example introduces four activities within the system. It is is not a classic process conceptualization in terms of activities and their order of execution, however. Note that the precondition is directed to the artifact of the previous activity and not the previous activity itself. This implicitly expresses an order relation too, but it also allows for exception flows and interventions, where pre conditions are met some other way, without the corresponding activities being performed.

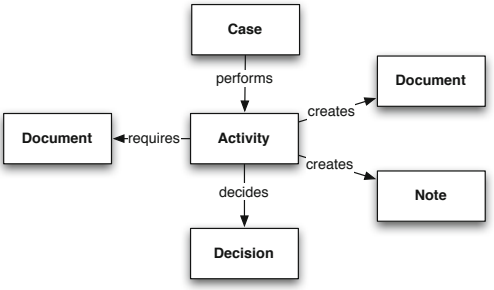


Fig. 6. Graphical representation of the process meta model

To visualize this process model in natural language, we again need to specify a grammar. This grammar verbalizes all constructs introduced by the used meta model shown in Figure 6.

1. "After the Case $<Case>$ is completed:" $\leftrightarrow \{Case\}$.
   (a) "the activity $<Activity>$ is completed."
   $\leftrightarrow \{Case, performs, Activity\}$.
2. "The activity $<Activity>$ may be performed if:" $\leftrightarrow \{Activity\}$.
   (a) "a document $<Document>$ is already present"
   $\leftrightarrow \{Activity, requires, Document\}$.
3. "After the activity $<Activity>$ is completed:" $\leftrightarrow \{Activity\}$.
   (a) "a document of type $<Document>$ is created"
   $\leftrightarrow \{Activity, creates, Document\}$
   (b) "the decision $<Decision>$ is taken"
   $\leftrightarrow \{Activity, decides, Decision\}$.

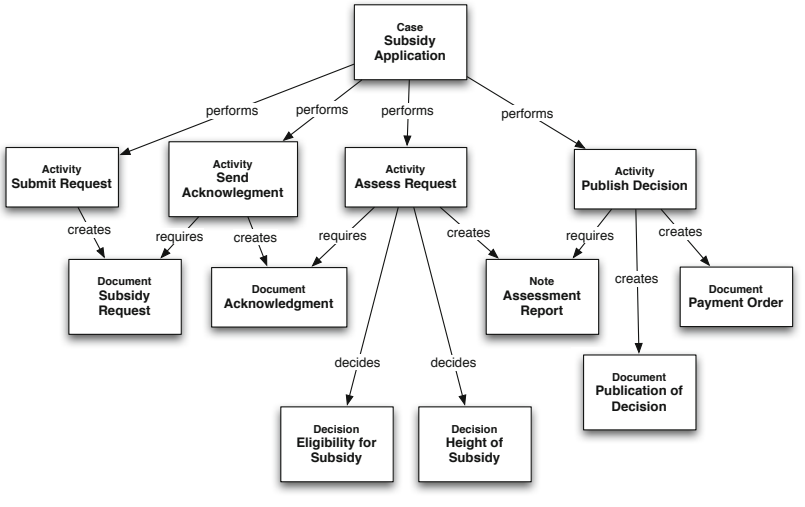


Fig. 7. Graphical representation of subsidy application process

The model from Figure 7 is then serialized into natural language as follows. "Subsidy Application" cases are completed by performing all four activities.

I After the Case **Subsidy Application** is completed:
(a) the activity **Submit Subsidy Request** is completed
(b) the activity **Send Acknowledgement** is completed
(c) the activity **Assess Subsidy Request** is completed
(d) the activity **Publish Decision** is completed.

The activity called "Assess Subsidy Request" is defined as an activity that can be performed if an "Acknowledgment" document is present. After completion, both eligibility and height of the subsidy is decided on, and a Assessment Report is written. The first sentence verbalizes the preconditions:

II The activity **Assess Subsidy Request** may be performed if :
(a) an **Acknowledgement** document is already present.

The consequences or postconditions are verbalized by the second pattern:

III After the activity **Assess Subsidy Request** is completed:
(a) a note **Assessment Report** is created
(b) the decision **Eligibility for Subsidy** is available
(c) the decision **Height of Subsidy** is available

Being able to review these kind of process rules is important as Be Informed's Case Management environment executes actual processes based on ontologies, like the one from this example.

4 Choosing Pattern Sentences

For this representation to enable business users to be involved in the relevant tasks, the pattern sentences have to be chosen carefully. Based on the presented examples, we will show how applying best practices from different fields can help to choose the most effective pattern sentences.

4.1 Structuring the Use of Natural Language

Methods such as RuleSpeak [18] and the OMG standard SBVR [15] have rationalized the use of natural language for the business by introducing syntactic guidelines and best practices. The two standards SBVR and RuleSpeak are related. Both are initiated by practitioners in the business rules community. Where SBVR provides a meta model that defines the semantics of vocabulary and rules, RuleSpeak is a set of sentence patterns and guidelines following that meta model. RuleSpeak differs from SBVR structured English (a non-normative appendix of the SBVR standard) in that it is pattern based and chooses readability over unambiguity.

To involve business users in the roles discussed earlier, it is important that the representation has the following features:

– does not require extensive training;
– is intuitive;
– emphasizes the meaning of the sentence;
– is a starting point to ask further questions;
– is precise and formal.

Early experiments with the patterns that are presented in Section 3.3 and 3.4 already demonstrate some of the above features. But it was also noted that a sentence that is not grammatically correct, for instance when patterns do not deal well with gender or plural forms as discussed in Section 3.2, or are just "not the way I would say it" disturbs the focus on the task at hand.

4.2 RuleSpeak Sentence Patterns

The most important guideline of RuleSpeak is that a sentence that expresses a rule must have a modal verb to indicate that something is required, obligatory or permitted. Among the different modal verbs available, RuleSpeak has chosen the verbs "must" and "may only" for obligations[1].

The practical value of having only 2 choices to express an obligatory modality is that all sentences follow a very similar pattern and readers quickly adopt to the specific semantics. The choice for modal verbs with a strict interpretation ("must", while the less stricter "should" is used by many requirements management practitioners) has the important benefit that it is a starting point for further questions and analysis.

The following sentence would be the RuleSpeak alternative for the pattern sentence from the example in Section 3.3:

Original The discount **Early Adopter** is a **Consumer** discount, and customers apply for it by ordering the product **Basic Telephony**, with option **VOIP**.

RuleSpeak "The **Consumer** discount **Early Adopter** must be applied when a customer orders the product **Basic Telephony**, with option **VOIP.**"

Let's suppose that there are additional criteria to apply for the discount and the author of the rule was not aware of them. Since the second version of the rule is stronger it is more likely that the business expert will question the RuleSpeak sentence. Our practical experience confirms this intuition.

4.3 RuleSpeak Guidelines

RuleSpeak also provides guidelines that can be incorporated in the patterns. One guideline states that a rule may only state one modality. So if two things are obligatory, make two rules.

[1] In the SBVR appendix on RuleSpeak other verbs for permissions and necessities are included but they are not relevant for the scope of this article and therefore not further discussed.

Continuing on the example in the previous paragraph applying this guideline would result in the following two rules:

1. "The **Early Adopter** discount must be applied when a customer orders the product **Basic Telephony**, with option **VOIP.**"
2. "The **Early Adopter** discount must be considered a **consumer discount**."

Another RuleSpeak guideline that is applicable to the example rules from Section 3.3 requires avoiding long strings of "and-ed" and "or-ed" conditions. They can be extremely hard to follow. Instead, eliminate all significant conjunctions in a Business Rule Statement following the RuleSpeak guidelines above and use indentation for separate bullet lists as necessary. Applying this guideline on the following example from section 3.3

Original The discount **Triple Play** is a **Consumer** discount, and customers apply for it by ordering all of the products **Fast ADSL**, **Digital TV** and **Basic Telephony**.

RuleSpeak The discount **Triple Play** must be applied when a customer orders all of the following products:

- **Fast ADSL**,
- **Digital**,
- **TV**,
- **Basic Telephony**.

The RuleSpeak keyword "the following" is used to set off the list. This approach avoids ambiguity and allows for easier modification. In practice the business expert may use the list as a checklist during validation sessions or to assess individual cases.

5 Human Interfaces Based on Pattern Sentences

Based on the experiences with the representation described, user interfaces were developed to enable Be Informed users to use the natural language representations in their work.

5.1 Using Language Representation in Knowledge Base Access

Often, the results of knowledge representation efforts are published online to assist reviewing and querying of the models. Typically, such an interface is based on directory-style navigation and search. A concept, or topic, is represented by a page, containing all its properties and resources and navigation links to other related topics, as can be seen in Figure 8. The types of both concepts and relations are visualized through for instance icons or by grouping items in categories. The natural language representation presented in Section 3 can be integrated into such an interface. On each page, a textual representation of

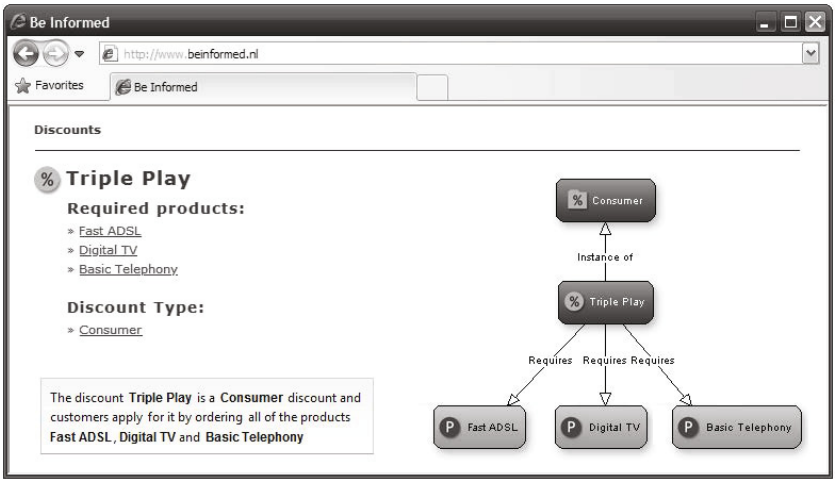


Fig. 8. Multi syntax knowledge base: Navigation, natural language and visual

its concept and all its relations is shown. The textual representation can offer hyperlinks for the concepts in the sentences, linking to their respective knowledge base pages.

The wiki-like interface presented here might be extended to allow editing of the models, in a similar way as editing is done in Ace Wiki [12]. As shown in Section 3.1, multiple representations can be presented in parallel. Even the visual graph representation can be integrated into a knowledge base, which can be useful if the knowledge base is used for analysts as well as end users. Typically, the appropriate representation is selected based on user profile or roles. However, adding representations in parallel allows users to focus on the representation of their personal choice, while possibly getting familiar with the other representations.

5.2 Editing Models by Manipulating Sentences

We have also developed an editor for knowledge models based on the pattern sentences representation. It uses the interaction metaphor of a text document with sentences, however without the cursor as the main means of manipulating the document. Instead, the user can edit the model by adding and removing pattern sentences and filling in the placeholders of these pattern sentences.

The biggest interaction challenge in such an interface is supporting users in selecting the pattern sentences that match the knowledge they want to express. At modeling time, users have little freedom in creating their own sentences, so only a carefully designed grammar that is offered to the user in a very contextual way leads to a good user experience.

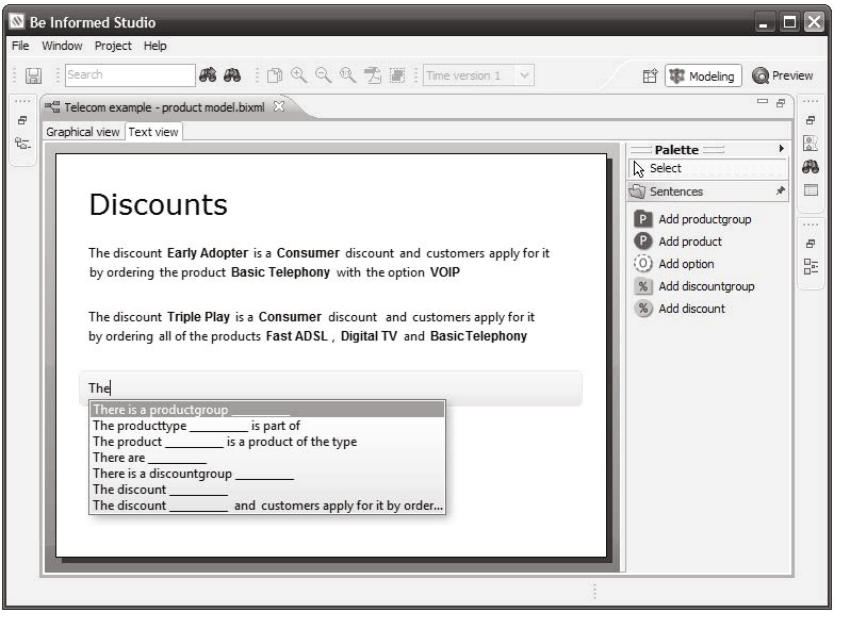


Fig. 9. Task centric and word processor style editing

In the current editor, available pattern sentences are offered to the user in two ways. The pattern sentences are offered in a task centric way in a Tool Palette, as displayed in Figure 9. Users can drag available pattern sentences from the palette onto the document, where they appear with variable parts that need to be completed. The Tool Palette presents the pattern sentences by a task-oriented name, that summarizes the effect or goal of the particular sentence.

They are also offered in a more word processor style: By typing at the end of the document, pattern sentences matching the typed text are presented in a popup menu. Selecting a pattern sentence from the menu inserts it into the document. As a result, the editor offers deep predictive behavior. Compared to more constraint free environments, prediction is trivial compared to grammar based look ahead, but the predictions are typically much deeper. Creating references to other concepts is also performed by combo boxes that show a relevant subset of the available concepts based on typed text and placeholder constraints.

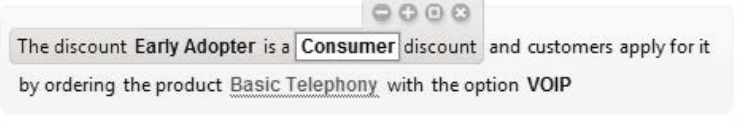


Fig. 10. Contextual controls and embedded error messages

To offer the user the experience of a text document, all controls are embedded in the document and are only visible when a specific sentence part or placeholder is selected, as seen in Figure 10. Error feedback is provided by showing annotations inside the text document. Sentence parts containing warnings or errors are underlined, orange and red respectively.

6 Conclusions and Future Work

In this paper we have presented a mechanism for representing formal concept graphs as natural language, using natural language pattern sentences. In its current form, it has already proven to be a valuable tool to our clients. First evaluations show it helps them to get in control in a model driven environment, by validating and communicating the models that drive the services they use in their daily operations.

The RuleSpeak sentence patterns and writing guidelines have helped to improve the quality of the generated text. The new patterns make sentences easier to read and place a natural and intuitive emphasis on the fact that the sentence introduces an obligation (and is not "just" a potential statement) for business experts with no background in formal logics. The sentence patterns also provide guidance to direct people into being more formal and precise.

We intend to integrate this technology into our product suite and to put it to broader use. In this paper, we applied it to the verbalization of knowledge models. In an in-house prototype, we have experimented with explaining the behavior of the services that are based on the models. As a consequence, we are then able to explain individual cases as they are handled by the services. For instance, the outcome of a decision that is taken based on models can be explained to the applicant.

The intended audience of these explanations range from business users responsible for modeling their business, employees such as call centre agents that have to explain the products and processes to their clients and ultimately the clients themselves, for instance in letters sent to them to communicate decisions. For generated texts to be acceptable to these different stakeholders, the generated texts must be of high quality.

The mechanism we have presented has some limitations in terms of the grammatical quality of generated text. Addressing in pattern sentences the variations that follow for instance from multiplicity and gender often leads to the creation of large numbers of patterns, and unacceptable amounts of work as a consequence.

Discussions at the workshop have convinced us that technologies presented there, such as the Grammatical Framework [1] currently being developed in the Molto Project [13], could help us face these challenges. We would like to investigate hybrid scenario's where the mechanism we presented is enriched with NLG technology to:

1. Better deal with morphology aspects, such as, verb forms, plurals and gender;
2. Remove sentence planning aspects from the patterns: As we generate sentences based on graphs, we typically have a lot of triple-related text

fragments to combine. Doing this inside the sentence pattern has proven to be impractical and probably language specific. Using small patterns to generate a number of small sentences for each triple and having GF consolidate those into larger sentences looks promising.

3. Deal with the lexical properties of the labels that are to be included in the pattern: When the concepts in the ontology have real names, these are often used as label and they are easily included into a pattern sentence because they are often nouns. However, often legal objects such as norms do not have names and are referred to by a definition, i.e. "Applicants must be over 25". Including them into an abstract syntax tree requires additional linguistic processing.
4. Provide the verbalization in multiple languages: Special attention is needed for the challenge of contextual translation. Many multilingual NLG technologies are based on aligned lexicons, which potentially is a problem when conceptualizations do not align well across different languages. This challenge is the scope of the Monnet Project [14], and its work could help to solve this challenge.
5. Generate sentence variants as they are used in end user dialog: An important part of ontology verbalization in service context is supporting dialog with users. This requires ontology based sentences to be available in different form, also referred to as mood. For instance "Applicants must be over 25" may be rewritten to "Is the applicant over 25?" when asked,"Applicant is over 25" when explained, or also "Applicant is not over 25", when explaining failure to comply (or even "Applicant is 25 or younger"). When explaining an individual case, it could read "Mr Johnson is over 25". Initial work in this area is already being performed within the mentioned Monnet Project.

We would like to study how to deal with the trade-off that is inherent to the restriction to a well defined domain defined by a pattern sentences grammar. Although the resulting consistency and productivity in authoring is an important asset to the approach, being able to express knowledge outside the scope of the pattern sentence grammar could sometimes be useful. The Ace Wiki [12] demonstrated at the workshop showed a relatively constraint free environment to express English sentences. In a hybrid approach, pattern sentences would allow for deep editing prediction, while still being able to express facts that are not covered by the current pattern sentences grammar. In addition, mining the sentences in such a knowledge base could be a way to maintain the pattern sentences catalog by suggesting frequently used patterns in the texts present in the database.

Finally, when these challenges in verbalization are dealt with, it would be interesting to study the performance of textual vs graphical model visualization in a more systematical way. Especially, we are very interested in studying which representation is most suitable to which type of user, as evaluated in the context of their specific tasks. These might range from both domain experts and information professionals modeling knowledge to consumers and citizens interpreting the explanation of decisions taken for them based on this formal knowledge.

Acknowledgements. The authors would like to thank Norbert Fuchs and Mike Rosner for organizing the CNL workshop. It has proven to be a productive environment for discussion and creation of new ideas. Also, the fact that both industry and academics participate has been of great value. Although aims and methods may differ, the synergy is obvious. Moreover, cooperation is necessary for the adoption of a maturing domain like CNL.

The authors also wish to gratefully acknowledge the support for this work by the European Commission (EC). This work was partially funded by the EC within the EU FP7 Multilingual Ontologies for Networked Knowledge (MONNET) Project under Grant Agreement No. 248458.

References

1. Angelov, K., Ranta, A.: Implementing Controlled Languages in GF. In: Fuchs, N.E. (ed.) CNL 2009. LNCS (LNAI), vol. 5972, pp. 82–101. Springer, Heidelberg (2010)
2. Clark, P., Murray, W.R., Harrison, P., Thompson, J.: Naturalness vs. Predictability: A Key Debate in Controlled Languages. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 65–81. Springer, Heidelberg (2010)
3. Eclipse Modeling Project, http://www.eclipse.org/modeling/
4. Funk, A., Tablan, V., Bontcheva, K., Cunningham, H., Davis, B., Handschuh, S.: CLOnE: Controlled Language for Ontology Editing. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 142–155. Springer, Heidelberg (2007)
5. van Grondelle, J., Heller, R., van Haandel, E., Verburg, T.: Involving Business Users in Formal Modeling Using Natural Language Pattern Sentences. In: Cimiano, P., Pinto, H.S. (eds.) EKAW 2010. LNCS, vol. 6317, pp. 31–43. Springer, Heidelberg (2010)
6. Grönniger, H., Krahn, H., Rumpe, B., Schindler, M., Völkel, S.: Text-based Modeling. In: Proceedings of the 4th International Workshop on Software Language Engineering (2007)
7. Halpin, T.: Business Rule Verbalization. In: Proceedings of ISTA (2004)
8. Halpin, T., Curland, M.: Automated Verbalization for ORM 2. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops, Part II. LNCS, vol. 4278, pp. 1181–1190. Springer, Heidelberg (2006)
9. Heller, R., van Teeseling, F., Gülpers, M.: A Knowledge Infrastructure for the Dutch Immigration Office. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010, Part II. LNCS, vol. 6089, pp. 386–390. Springer, Heidelberg (2010)
10. Jarrar, M., Keet, M., Dongilli, P.: Multilingual verbalization of ORM conceptual models and axiomatized ontologies. Technical report. STARLab, Vrije Universiteit Brussel (2006)
11. Kaljurand, K., Fuchs, N.E.: Verbalizing OWL in Attempto Controlled English. In: Proceedings of Third International Workshop on OWL: Experiences and Directions, Innsbruck, Austria (2007)
12. Kuhn, T.: AceWiki: Collaborative Ontology Management in Controlled Natural Language. In: Proceedings of the 3rd Semantic Wiki Workshop. CEUR Workshop Proceedings (2008)

13. Molto Project: Multilingual On-Line Translation, http://www.molto-project.eu/
14. Monnet Project: Multilingual Ontologies for Networked Knowledge, http://www.monnet-project.eu/
15. Object Management Group: Semantics of Business Vocabulary and Rules (2008), http://www.omg.org/spec/SBVR/1.0
16. Power, R., Scott, D., Evans, R.: What you see is what you meant: direct knowledge editing with natural language feedback. In: Proceedings of the 13th Biennial European Conference on Artificial Intelligence, Brighton, UK, pp. 675-681 (1998)
17. Reiter, E.: NLG vs. Templates. In: Proceedings of the 5th European Workshop on Natural Language Generation, Leiden, The Netherlands, pp. 95–105 (1995)
18. Ross, R.G.: RuleSpeak (2009), http://www.rulespeak.com

Author Index