Codeco: A Grammar Notation for Controlled Natural Language in Predictive Editors

Tobias Kuhn

Department of Informatics & Institute of Computational Linguistics,
University of Zurich, Switzerland
kuhntobias@gmail.com
http://www.tkuhn.ch

Abstract. Existing grammar frameworks do not work out particularly well for controlled natural languages (CNL), especially if they are to be used in predictive editors. I introduce in this paper a new grammar notation, called *Codeco*, which is designed specifically for CNLs and predictive editors. Two different parsers have been implemented 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) like Attempto Controlled English (ACE) [3] and others have been proposed to make knowledge representation systems more user-friendly [10, 5]. It can be very difficult, however, for users to write statements that comply with the restrictions of CNLs. Three approaches have been proposed so far to solve this problem: error messages [2], language generation [9], and predictive editors [11, 5]. Each of the approaches has its own advantages and drawbacks. In my view, the predictive editor approach — where the editor shows all possible continuations of a partial sentence — is the most elegant one. However, implementing lookahead features (i.e. retrieving the possible continuations for a given partial sentence on the basis of a given grammar) is not a trivial task, especially if the grammar describes complex nonlocal structures like anaphoric references. Furthermore, good tool integration is crucial for CNLs, 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.

These requirements are not satisfied by existing grammar frameworks. Natural language grammar frameworks like Head-driven Phrase Structure Grammars (HPSG) [8] are unnecessarily complex for CNLs and do not allow for the implementation of efficient and reliable lookahead features. Grammar frameworks for formal languages — called *parser generators* — like Yacc [4] do not allow for the declarative definition of complex (i.e. context-sensitive) structures. Definite clause grammars (DCG) [7] are a good candidate but they have the problem that they are hard to implement in programming languages that are not logic-based. Furthermore, all these grammar frameworks have no special support for the deterministic resolution of anaphoric references, as required by languages like ACE. The Grammatical Framework (GF) [1] is a recent grammar framework in competition with the approach to be presented here. It allows for declarative definition of grammar rules and can be used in predictive editors, but it lacks support for anaphoric references.

The following example illustrates the difficulty of lookahead in the case of complex nonlocal structures like anaphoric references. The partial ACE sentence "every man protects a house from every enemy and does not destroy ..." can be continued by several anaphoric structures referring to earlier objects in the sentence: "the man" or "himself" can be used to refer to "man"; "the house" or "it" can be used to refer to "house". It is not possible, however, to refer to "enemy", because it is under the scope of the quantifier "every" and such references from outside the scope of a quantified entity do not make sense logically.[1] Thus, a predictive editor should in this case propose "the man" and "the house" as possible continuations of the given partial sentence, but not "the enemy".

2 Codeco

In order to solve the problems sketched above, I created a new grammar notation called *Codeco*, which stands for "*co*ncrete and *de*clarative grammar notation for *co*ntrolled natural languages". This grammar notation is designed specifically for CNLs to be used in predictive editors. Below is a brief description of the most important features of Codeco.

In a nutshell, grammar rules in Codeco consist of grammatical categories with flat feature structures[2]. One of the important features of Codeco is that anaphoric references and their potential antecedents can be represented by special categories for backward references "<" and forward references ">", respectively. These special categories establish nonlocal dependencies across the syntax tree in the sense that each backward reference needs a matching (i.e. unifiable) forward reference somewhere in the preceding text.

Furthermore, scopes can be defined that make the contained forward references inaccessible from the outside. In Codeco, scopes are opened by the special category "//" and are closed by the use of scope-closing rules "$\overset{\sim}{\rightarrow}$". In contrast to normal rules "$\overset{:}{\rightarrow}$", scope-closing rules close at their end all scopes that have

[1] The difference between "every man" and "every enemy" in the given example is not obvious. The scope that is opened by the quantifier "every" of "every enemy" closes after "enemy", because the verb phrase ends there. In ACE, verb phrases — but not only verb phrases — close at their end all scopes that are opened within (which is, in general, also true for full natural English). Since "every man" is not part of that verb phrase, the scope for "every man" is still open at the end of the partial sentence. This is why "man" is accessible for anaphoric references, but "enemy" is not. See also Fig. 1.

[2] The restriction to flat feature structures (instead of general ones) restricts the expressive power of Codeco while making it easier to process and implement. The results suggest that Codeco is sufficiently expressive for common CNLs.

been opened by one of their child categories. In this way, Codeco allows us to define the positions where scopes open and close in a simple and fully declarative way. Other extensions that are not discussed here are needed to handle pronouns, variables, and references to proper names in the desired way. See [6] for the details.

The following grammar rules are examples that show how the special structures of Codeco are used:

$$np \quad \xrightarrow{:} \quad det\left(\text{exist: } +\right) \quad \underline{noun}\left(\text{text: } \boxed{\text{Noun}}\right) \quad >\left(\begin{array}{l}\text{type: noun}\\ \text{noun: } \boxed{\text{Noun}}\end{array}\right)$$

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

$$det\left(\text{exist: } -\right) \quad \xrightarrow{:} \quad /\!\!/ \quad [\,\text{every}\,]$$

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

The first rule represents existentially quantified noun phrases like "a house" and establishes a forward reference to define its status as a potential antecedent for subsequent anaphoric references. The second rule describes a definite noun phrase like "the house" and declares its anaphoric property by a backward reference. The third rule describes the determiner "every" opening a scope through the use of the scope opener category. The last rule describes a verb phrase and is scope-closing, which means that the end of such a verb phrase also marks the end of all scopes that have been opened within.

Codeco grammars allow for the correct and efficient implementation of lookahead features. Figure 1 shows a possible syntax tree of the example above and visualizes how the possible anaphoric references can be found.

Two independent parsers have been implemented for Codeco: one that executes Codeco grammars as Prolog DCGs and another one that is a chart parser implemented in Java. These two implementations have been shown to be equivalent and efficient [6]. The chart parser for Codeco is used in the predictive editors of AceWiki [5] and the ACE Editor[3]. The DCG parser is used for regression and ambiguity tests for the grammars of both tools.

The Codeco grammar that is used in the ACE Editor consists of 164 grammar rules and describes a large subset of ACE covering nouns, proper names, intransitive and transitive verbs, adjectives, adverbs, prepositions, variables, plurals, negation, prepositional phrases, relative clauses, modality, numerical quantifiers, various kinds of coordination, conditional sentences, questions, anaphoric pronouns, and more. Currently unsupported ACE features are 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. Nevertheless, this subset of ACE is — to

[3] `http://attempto.ifi.uzh.ch/webapps/aceeditor/`

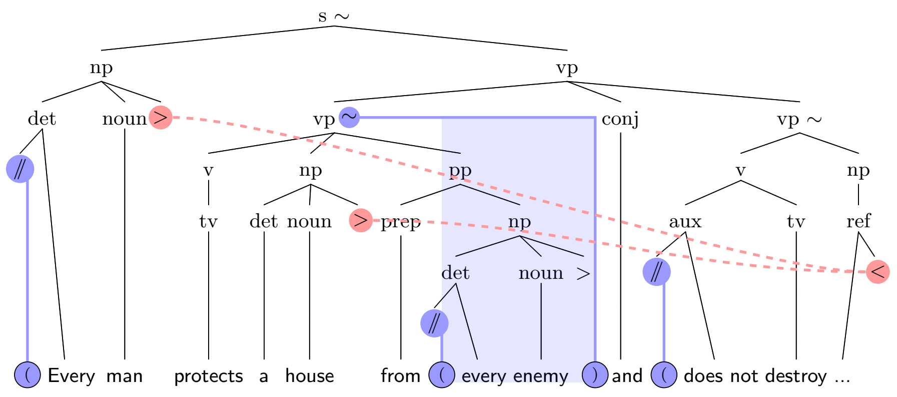

Fig. 1. This figure shows an exemplary syntax tree of a partial ACE sentence. Head nodes of scope-closing rules are marked with "∼". The positions where scopes open and close are marked by parentheses. The shaded area marks a closed scope, which is not accessible from the outside. The dashed lines indicate the possible references.

my knowledge — the broadest subset of English that has ever been defined in a formal and fully declarative way and that includes complex issues like anaphoric references.

Codeco grammars can be checked for syntactic ambiguity by exhaustive language generation up to a certain sentence length. Due to combinatorial explosion, it is advisable not to check the complete language but to define a core subset thereof. Such a core subset should use a minimal lexicon and should have a reduced number of grammar rules, in a way that language structures that could possibly lead to complex cases of ambiguity are retained. For the ACE Codeco grammar introduced above, such a core subset has been defined, consisting of 97 of the altogether 164 grammar rules. Exhaustive language generation of this core subset up to ten tokens leads to 2'250'869 sentences, which are all different. Since syntactically ambiguous sentences would be generated more than once (because they have more than one syntax tree), this proves that at least the core of the ACE Codeco grammar is unambiguous, at least for sentences up to ten tokens. In the same way, it can be proven that the core of the ACE Codeco grammar is indeed a subset of ACE.

See my thesis [6] for the complete description of Codeco, the full ACE Codeco grammar, complexity considerations, the details and results of the performed tests and the involved algorithms.

3 Conclusions

In summary, the Codeco notation allows us to define controlled subsets of natural languages in an adequate way. The resulting grammars are fully declarative, can be efficiently interpreted in different kinds of programming languages, and allow for lookahead features, which is needed for predictive editors. Furthermore, it allows for different kinds of automatic tests, which is very important for the development of reliable practical applications.

Codeco is a proposal for a general CNL grammar notation. It cannot be excluded 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. Krasimir Angelov and Aarne Ranta. Implementing controlled languages in GF. In *Proceedings of the Workshop on Controlled Natural Language (CNL 2009)*, volume 5972 of *Lecture Notes in Computer Science*, pages 82–101. Springer, 2010.
2. Peter Clark, Shaw-Yi Chaw, Ken Barker, Vinay Chaudhri, Phil Harrison, James Fan, Bonnie John, Bruce Porter, Aaron Spaulding, John Thompson, and Peter Yeh. Capturing and answering questions posed to a knowledge-based system. In *K-CAP '07: Proceedings of the 4th International Conference on Knowledge Capture*, pages 63–70. ACM, 2007.
3. Juri Luca De Coi, Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Controlled English for reasoning on the Semantic Web. In *Semantic Techniques for the Web — The REWERSE Perspective*, volume 5500 of *Lecture Notes in Computer Science*, pages 276–308. Springer, 2009.
4. Stephen C. Johnson. Yacc: Yet another compiler-compiler. Computer Science Technical Report 32, Bell Laboratories, Murray Hill, NJ, USA, July 1975.
5. Tobias Kuhn. How controlled English can improve semantic wikis. In *Proceedings of the Forth Semantic Wiki Workshop (SemWiki 2009)*, volume 464 of *CEUR Workshop Proceedings*. CEUR-WS, 2009.
6. Tobias Kuhn. *Controlled English for Knowledge Representation*. Doctoral thesis, Faculty of Economics, Business Administration and Information Technology of the University of Zurich, Switzerland, to appear.
7. Fernando Pereira and David H. D. Warren. Definite clause grammars for language analysis. In *Readings in Natural Language Processing*, pages 101–124. Morgan Kaufmann Publishers, 1986.
8. Carl Pollard and Ivan Sag. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. Chicago University Press, 1994.
9. Richard Power, Robert Stevens, Donia Scott, and Alan Rector. Editing OWL through generated CNL. In *Pre-Proceedings of the Workshop on Controlled Natural Language (CNL 2009)*, volume 448 of *CEUR Workshop Proceedings*. CEUR-WS, April 2009.
10. Rolf Schwitter, Kaarel Kaljurand, Anne Cregan, Catherine Dolbear, and Glen Hart. A comparison of three controlled natural languages for OWL 1.1. In *Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Directions*, volume 496 of *CEUR Workshop Proceedings*. CEUR-WS, 2008.
11. Rolf Schwitter, Anna Ljungberg, and David Hood. ECOLE — a look-ahead editor for a controlled language. In *Controlled Translation — Proceedings of the Joint Conference combining the 8th International Workshop of the European Association for Machine Translation and the 4th Controlled Language Application Workshop (EAMT-CLAW03)*, pages 141–150, Ireland, 2003. Dublin City University.