PROC. WLP 95, 11TH LOGIC PROGRAMMING WORKSHOP, VIENNA, SEPTEMBER 1995

Attempto

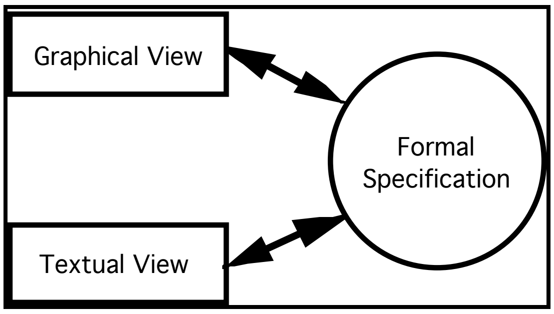
Specifications in Controlled Natural Language

Norbert E. Fuchs, Bernhard Hamberger, Rolf Schwitter
Department of Computer Science, University of Zurich
{fuchs, hambe, schwitter}@ifi.unizh.ch

Writing specifications for computer programs is not easy since one has to take into account the disparate conceptual worlds of the application domain and of software development. To bridge this conceptual gap we propose controlled natural language as a declarative and application-specific specification language. Controlled natural language is a subset of natural language that can be accurately and efficiently processed by a computer, but is expressive enough to allow natural usage by non-specialists. Specifications in controlled natural language are automatically translated into Prolog clauses, hence become formal and executable. The translation uses a definite clause grammar (DCG) enhanced by feature structures. Inter-text references of the specification, e.g. anaphora, are resolved with the help of discourse representation theory (DRT). The generated Prolog clauses are added to a knowledge base. We have implemented the prototypical specification system Attempto that successfully processes the specification of a simple automated teller machine.

1 Introduction: Views as Declarative Specifications

We develop formal specifications in logic languages, specifically first-order predicate logic and Prolog. To bridge the conceptual gap between application domains and formal specifications we introduce graphical and textual views of formal specifications as application-oriented, i.e. in the true sense declarative, specifications [Fuchs & Fromherz 94].



An automatic mapping between a view and its associated formal specification assigns a formal semantics to the view. Though views give the impression of being informal and having no intrinsic meaning, they are formal and have the semantics of their associated formal specification. This dual-faced appearance of views reduces the conceptual gap.

If the formal specification is executable the execution can be observed on the level of the view. Thus validation and prototyping in concepts close to the application domain become possible.

The rest of this paper is structured as follows: in section 2 we introduce controlled natural language as a view of a formal specification in a logic language; in section 3 we give an overview of the Attempto specification system; sections 4 - 7 describe the translation process from controlled English to Prolog; section 8 is dedicated to the lexical editor and the spelling checker; finally, in section 9 we conclude and outline further research.

2 Controlled Natural Language

Controlled natural language – a subset of natural language with restricted grammar and an application-specific vocabulary – can serve as a view for a formal specification in a logic language.

A specification in controlled natural language is a multi-sentential text consisting of

- simple declarative sentences of the form subject – verb – object
- *if ... then* sentences
- *yes/no* queries, *wh*-queries

The specification text can contain

- anaphoric references, e.g. pronouns
- relative clauses, both subject and object modifying
- comparative clauses like *bigger than*, *smaller than* and *equal to*
- elliptical compound phrases like *and*-lists, *or*-lists
- negation like *does not, is not* and *has not*

Constructs like anaphora, ellipsis, and abbreviations have been introduced to make the controlled language compact, concise and close to unrestricted natural language.

Controlled or simplified English is not a new idea. It has been used for quite some time for technical documentation [AECMA 89, Wojcik et al. 90, Adriaens & Schreurs 92], and as data base query language [Androutsopoulos 95]. Pulman and Rayner are suggesting a computer processable controlled language that could be used for various purposes ranging from structured documentation over access to information to the control of devices [Pulman & Rayner 94]. However, very few researchers have tried to employ controlled natural language for software specifications since this leads to additional syntactic and semantic constraints for the language especially if one requires the specifications to be executable [Ishihara et al. 92, Macias & Pulman 92, Pulman 94, Fuchs & Schwitter 95].

Users seem to be able to construct sentences in controlled natural language, and to avoid constructions that fall outside the bounds of the language, particularly when the system gives feedback of the analysed sentences in a paraphrased form using the same controlled language [Capindale & Crawford 89].

The following is a small excerpt of the controlled natural language specification of a simple automated teller machine called SimpleMat.

```
% Example Specification
The customer enters a card and a personal code that is a number.
If the personal code is not valid then SM rejects the card.
```

The example specification text employs

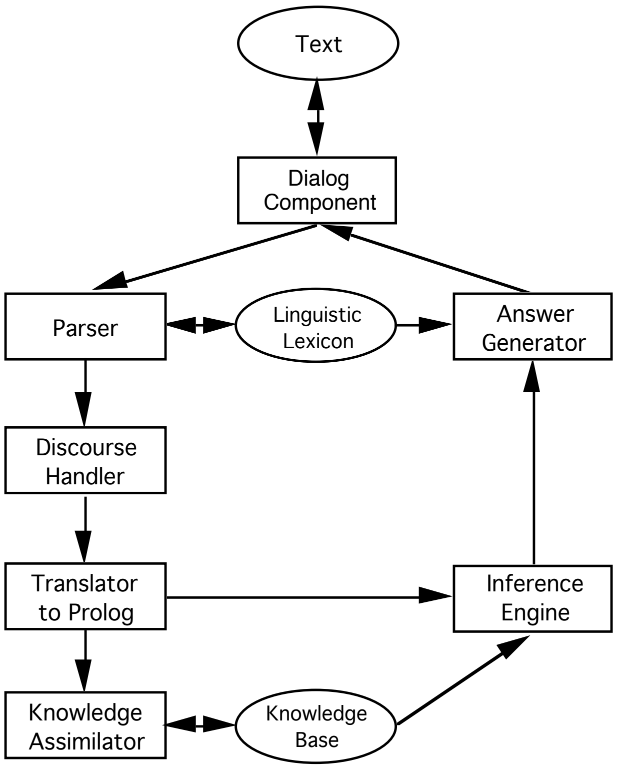
- declarative and *if-then* sentences
- ellipsis
- compound nouns, e.g. `personal code`
- relative clauses
- anaphoric reference by indefinite and definite determiners (`a card` - `the card`)
- negation
- abbreviations (`SM` standing for SimpleMat)

3 Overview of Attempto

We have implemented the Attempto system that accepts specifications in controlled natural language and translates them into Prolog.

The user enters specification text in controlled natural language that the *Dialog Component* forwards to the parser in tokenised form. Parsing errors and ambiguities to be resolved by the user are reported back by the dialog component. The user can also query the knowledge base in controlled natural language.

The *Parser* uses a predefined definite clause grammar enhanced by feature structures and a predefined linguistic lexicon to check sentences for syntactical correctness, and to generate syntax trees and sets of nested discourse representation structures.



The *Linguistic Lexicon* contains an application-specific vocabulary. The lexicon can be modified by a lexical editor invokable from the dialog component.

The *Discourse Handler* analyses and resolves inter-text references and updates the discourse representation structures generated by the parser.

The *Translator* translates discourse representation structures into Prolog clauses. These Prolog clauses are either passed to the knowledge assimilator, or – in case of queries – to the inference engine.

The *Knowledge Assimilator* adds new knowledge to the knowledge base.

The *Inference Engine* answers user queries with the help of the knowledge base. In a preliminary version the inference engine is just the Prolog interpreter.

The *Answer Generator* takes the answers of the inference engine, reformulates them in controlled natural language, and forwards them to the dialog component.

4 Parsing

The specification text is parsed by a top-down parser using a Definite Clause Grammar enhanced by feature structures [Covington 94].

The parser generates a syntax tree as syntactic representation, and concurrently a discourse representation structure as semantic representation.

The syntax tree for a multisentential specification text has the root `discourse` to which `s`-nodes for sentences are attached.

The following graph shows the `s`-node of the second sentence of the above example specification.

```
                                s
           |--------------------|--------------------|
           s1                                        s1
  |--------|--------|                         |------|------|
  cj                s                         cj            s
  |     |-----------|-----------|             |     |-------|-------|
  |     np                      vp            |     np              vp
  |   |-|--|              |-----|-|----|      |     |          |----|-----|
  |   det  n1             copula  neg  ap     |     pn         tv         np
  |   |    |              |       |    |      |     |          |        |-|--|
  |   |    nn             |       |    a      |     |          |        det  n1
  |   |    |              |       |    |      |     |          |        |    |
  |   |    |              |       |    |      |     |          |        |    cn
  |   |    |              |       |    |      |     |          |        |    |
  if  the  personal_code  is      not  valid  then  simplemat  rejects  the
card.
```

The parser generates the following paraphrase – displaying all substitutions and interpretations made – that explains how Attempto interpreted the user's input.

```
% Example Specification
the customer enters a card and the customer [same object] enters [same predicator] a personal_code that is a number.
if the personal_code [same object] is not valid then sm [simplemat] rejects the card [same object].
```

The user can now decide to accept Attempto 's interpretation, or to rephrase the input to achieve another interpretation. For ambiguous input Attempto always suggest one standard interpretation as default. It is up to the user to reformulate the input to achieve non-standard interpretations.

In addition, the parser informs the user about spelling and parsing errors, e.g. if the user had entered

```
The customer enters a card. It is checked for validity.
```

After parsing the first sentence successfully the system finds an unknown word in the second sentence that makes the sentence unparsable, and replies

```
First Unparsable Sentence: it is checked for validity.
Unknown word:                     checked
```

With the help of a lexical editor the user can immediately add the unknown word to the lexicon and resubmit the input to the parser.

5 Contextual Semantic Translation

The specification text is translated into a discourse representation structure (DRS) which contains discourse referents representing the objects of the discourse, and conditions for these discourse referents [Covington et al. 88, Kamp & Reyle 93].

The first sentence of our example contributes the discourse referents `A`, `B`, `C` and `D` and the conditions

```
[A, B, C, D]
gender(A, masc)
customer(A)
gender(B, neut)
card(B)
enter(A, B)
gender(C, neut)
personal_code(C)
gender(D, neut)
number(D)
be(C, D)
enter(A, C)
```

The second sentence is analysed in the context of the first sentence thus making the resolution of references, e.g. anaphora, possible. This sentence contributes further discourse referents `E`, `F` and `G`.

```
[E]
gender(E, neut)
named(E, simplemat)
IF:
  [F]
  gender(F, neut)
  the(personal_code(F))
  F=C
  NOT:
    []
    valid(F)
THEN:
  [G]
  gender(G, neut)
  the(card(G))
  G=B
  reject(E, G)
```

Conditions can be simple – e.g. `customer(A)` – or complex, i.e. DRSs. This can lead to nested DRSs. In our case, the topmost DRS contains an `IF-THEN` sub-DRS which itself contains a `NOT` sub-DRS.

Note that discourse referents and conditions for proper names, e.g. `named(E, simplemat)` appear always in the topmost DRS.

Anaphoric references, e.g. the phrase `the card` of the second sentence referring to the phrase `a card` of the first sentence, are represented as the conjunctive condition `the(card(G))` and `G=B`. References are only possible to discourse referents in super-ordinate DRSs. The resolution algorithm always picks the closest referent that agrees in gender and number.

6 Semantic Representation

Now the DRSs can be combined and simplified yielding the final semantic representation of the complete specification text as one (nested) DRS

```
[A, B, C, E]
customer(A)
card(B)
enter(A, B)
personal_code(C)
number(C)
enter(A, C)
named(E, simplemat)
IF:
  []
  NOT:
    []
    valid(C)
THEN:
  []
  reject(E, B)
```

The gender information that was only necessary for anaphoric resolution is eliminated, and all unifications including `be(C, D)` are performed.

7 Translation into Prolog

Finally, the discourse representation structure is translated into Prolog clauses which are asserted as `fact/1` to the knowledge base.

```
fact(customer(0)).
fact(card(1)).
fact(enter(0, 1)).
fact(personal_code(2)).
fact(number(2)).
fact(enter(0, 2)).
fact(named(3, simplemat)).
fact((reject(3, 1):-neg(valid(2)))).
```

Discourse referents – being existentially quantified variables – are replaced by Skolem constants `0`, `1`, ..., or – if they are in the scope of a universal quantor – by Skolem functions.

`IF-THEN` DRSs with disjunctive consequences cannot directly be translated into Prolog since they would lead to disjunctive clauses. Instead they are represented by sets of Prolog clauses, one clause for each disjunct.

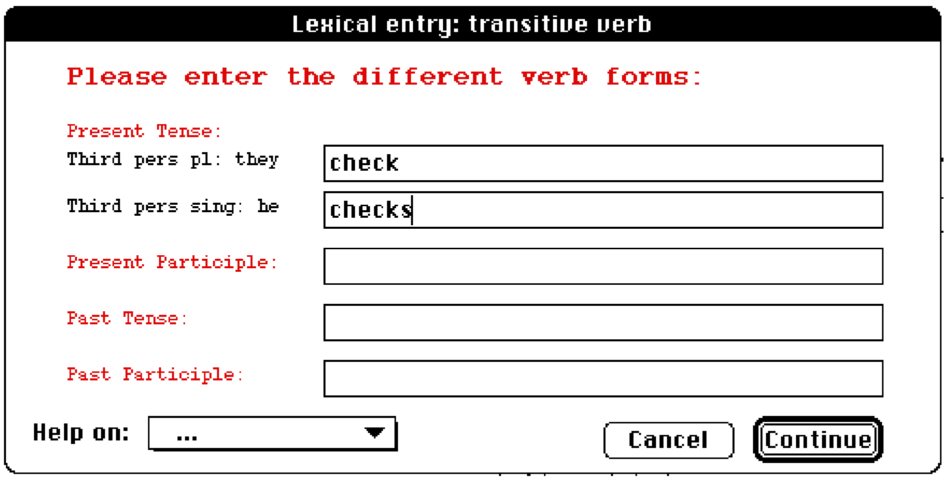
Questions (*yes/no* and *wh*-queries) can be used to interrogate the contents of the knowledge base. Questions are translated first into `QUERY` DRSs and then into Prolog queries, and are answered by logical inference.

8 Lexical Editor and Spelling Checker

Specification texts are incrementally developed by domain specialists. Though Attempto's lexicon contains entries of the closed word classes, e.g. determiners and prepositions, the entries for domain specific subsets of the open word classes, e.g. nouns and verbs, have to be added incrementally as needed for the specification text. A lexical editor – exhibiting interfaces for linguistic experts and non-experts – allows users to interactively modify and extend the lexicon while the system parses the specification text.

The expert interface represents lexical entries as complete feature structures and allows experts to freely modify any lexical entry. The interface for non-experts employs templates that help users to enter a minimum of information. The rest of the information is automatically derived. Help texts and balloon help support both groups of users.

The following screen shots show how a non-expert would add the transitive verb `check` to the lexicon. Not all fields need to be filled out. The translation into the predicate `check/2` is automatically derived.

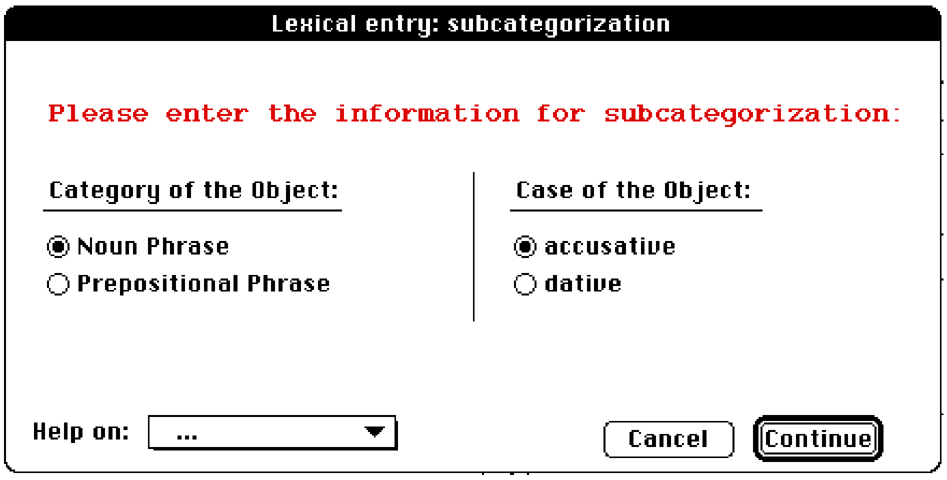


Lexical entry: transitive verb

Please enter the different verb forms:

Present Tense:

Third pers pl: they	check
Third pers sing: he	checks
Present Participle:	
Past Tense:	
Past Participle:	



Lexical entry: subcategorization

Please enter the information for subcategorization:

Category of the Object: [✓ Noun Phrase] Prepositional Phrase

Case of the Object: [✓ accusative] dative

A spelling checker allows users to determine whether all words of a specification text are in the lexicon. This spelling checker is invoked automatically if (part of) a specification text cannot be parsed.

9 Conclusions and Further Research

The present prototypical implementation of Attempto proves that controlled natural language can be used for the non-trivial specification of an automated teller machine, and that the specification can be translated as coherent text into Prolog clauses. Much more work needs to be done, however.

Controlled Natural Language

Our current version of controlled English was derived in an attempt to represent typical constructs in natural language specifications in a structured and concise way. It seems that other researchers have chosen similar ad hoc approaches to define their versions of controlled or restricted natural languages. However, a more systematic definition of controlled English has to be found that not only results in a highly expressive language, but also makes it easier to learn and remember it.

Retranslation

To hide the internal representation of a formal specification it must be retranslated into controlled natural language when the user wants to examine or query the knowledge base. Formal specifications in the form of a DRS can – at least partially – be retranslated into their equivalent controlled natural language text since the grammar of the Attempto system is reversible. Another approach would use predefined translation schemata that access the lexicon to fill in variable parts of the schemata.

Executing the Specification

The internal representation of a specification can be used for simulation or prototyping by executing it. In our example specification, this means executing/running the specification of the automated teller machine. As it stands the specification does not provide all the necessary information and needs to be enhanced in three ways.

- First, an order of events has to be established, e.g. we have to make sure that during the simulation the event of entering a card has to precede the event of checking it. [Ishihara et al. 92] who translate natural language specifications into algebraic ones use contextual dependency and properties of data types to establish the correct order of events. In our approach based on discourse representation theory the order of events is to a great extent established when we introduce eventualities (events and states) into the processing of our controlled natural language.
- Second, many relations representing events are not only truth-functional, but also cause side-effects, e.g. I/O operations. The required side-effects can be defined by interface predicates that depend on the simulation environment. One could, for example, envisage that the interface predicates do not simply simulate the automated teller machine but cause the execution of a real automated teller machine.
- Third, the execution needs some situation specific information, or scaffolding. We can either provide the relevant facts in the knowledge base, or more conveniently, get the information by querying the user.

10 Acknowledgements

We would like to thank the anonymous referees for their helpful comments on an earlier version of this paper.

11 References

[Adriaens & Schreurs 92] G. Adriaens, D. Schreurs, From Cogram to Alcogram: Towards a Controlled English Grammar Checker, Proceedings COLING 92, pp. 595-601, 1992

[AECMA 89] Association Européenne des Constructeur de Matériel Aéronautique, AECMA – A Guide for the Preparation of Aircraft Maintenance Documentation in the International Aerospace Maintenance Language, 1989.

[Androutsopoulos 95] I. Androutsopoulos, G. D. Ritchie, P. Thanisch, Natural Language Interfaces to Databases – An Introduction, Journal of Natural Language Engineering, Cambridge University Press (to appear)

[Capindale & Crawford 89] R. A. Capindale, R. G. Crawford, Using a natural language interface with casual users, International Journal Man-Machine Studies, 32, pp. 341-362, 1989

[Covington 94] M. A. Covington, GULP 3.1: An Extension of Prolog for Unification-Based Grammar, Report AI-1994-06, Artificial Intelligence Center, University of Georgia, 1994

[Covington et al. 88] M. A. Covington, D. Nute, N. Schmitz, D. Goodman, From English to Prolog via Discourse Representation Theory, Research Report 01-0024, Artificial Intelligence Programs, University of Georgia, 1988

[Fuchs & Fromherz 94] N. E. Fuchs, M. P. J. Fromherz, Transformational Development of Logic Programs from Executable Specifications, in C. Beckstein, U. Geske (eds.), Entwicklung, Test und Wartung deklarativer KI-Programme, GMD Studien Nr. 238, 1994

[Fuchs & Schwitter 95] N. E. Fuchs, R. Schwitter, Specifying Logic Programs in Controlled Natural Language, CLNLP 95, Workshop on Computational Logic for Natural Language Processing, Edinburgh, 1995

[Ishihara et al. 92] Y. Ishihara, H. Seki, T. Kasami, A Translation Method from Natural Language Specifications into Formal Specifications Using Contextual Dependencies, in: Proceedings of IEEE International Symposium on Requirements Engineering, 4-6 Jan. 1993, San Diego, IEEE Computer Society Press, pp. 232 - 239, 1992

[Kamp & Reyle 93] H. Kamp, U. Reyle, From Discourse to Logic, Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory, Kluwer Academic Publishers, Dordrecht, 1993

[Macias & Pulman 92] B. Macias, S. Pulman, Natural Language Processing for Requirements Specifications, in: F. Redmill, T. Anderson (eds.), Safety-Critical Systems, Current Issues, Techniques and Standards, Chapman & Hall, pp. 67-89, 1993

[Pulman 94] S. G. Pulman, Natural Language Processing and Requirements Specification, Presentation at the Prolog Forum, Department of Computer Science, University of Zurich, February 1994

[Pulman & Rayner 94] S. Pulman, M. Rayner, Computer Processable Controlled Language, SRI International Cambridge Computer Science Research Centre, 1994

[Wojcik et al. 90] R. H. Wojcik, J. E. Hoard, K. C. Holzhauser, The Boeing Simplified English Checker, Proc. Internatl. Conf. Human Machine Interaction and Artificial Intelligence in

Aeronautics and Space, Centre d'Etude et de Recherche de Toulouse, pp. 43-57, 1990