A Natural Language Front-End to Automatic Verification and Validation of Specifications

Norbert E. Fuchs, Uta Schwertel
Department of Computer Science, University of Zurich, Switzerland
{fuchs, uschwert}@ifi.unizh.ch

Sunna Torge
Department of Computer Science, University of Munich, Germany
torge@informatik.uni-muenchen.de

Abstract

Abstract. Often problem solving can be reduced to the search for finite models of first-order logic specifications. We describe the model generation method EP Tableaux that is complete for refutation and for finite satisfiability. To make EP Tableaux available to domain specialists unfamiliar with formal notations we added a natural language front-end that accepts Attempto Controlled English — a subset of standard English with a domain-specific vocabulary and a restricted grammar. ACE allows users to express specifications precisely and in the terms of the application domain. ACE specifications are unambiguously translated into logic languages. We specified in ACE a database example that was previously specified in the EP Tableaux language PRQ, translated the ACE specification into PRQ, and reproduced the previously found results. As a further test, we formulated Schubert's Steamroller in ACE, translated the ACE version into PRQ and successfully proved the Steamroller's conclusions with EP Tableaux.

1 Introduction

Though formal methods promise improved quality of software and partial automation of the software development process they are not readily accepted by domain specialists. This applies particularly to formal specifications that are at the very basis of any formal software development. The reasons are twofold — formal specifications are hard to understand and difficult to relate to the concepts of the application domain. Hall [10] expressed this quite succinctly when he wrote that formal specifications need to be accompanied by a paraphrase in natural language that "explains what the specification means in real-world terms and why the specification says what it does".

We are directly addressing both problems — incomprehensibility of formal specification languages and semantical distance between application domain and specification language — by replacing obviously formal specifications by specifications in Attempto Controlled English (ACE) — a subset of English that seems informal but is in fact formal and can be deterministically translated into logic languages [8]. ACE allows domain specialists to express specifications precisely and in the terms of the application domain. Thus ACE reduces the semantic distance between the domain specialists' mental model of the application domain and the formal specification, and almost completely eliminates the incomprehensibility problem. In brief, an ACE specification is formal and at the same time "explains what the specification means in real-world terms and why the specification says what it does".

To prove our point we show that (a subset of) Attempto Controlled English can replace logic as front-end to EP Tableaux [3] — a method to verify the finite satisfiability of finite sets of range restricted formulae by generating models. EP Tableaux has been implemented as a Prolog program and can be used to solve problems that are reducible to the search for finite models of first-order specifications.

The combination of EP Tableaux with an ACE front-end allows domain specialists to express, verify and validate first-order specifications in the familiar natural language terms of their application domain.

The rest of the paper is organized as follows. In section 2 we discuss classes of problems that can be solved by model generation. Section 3 introduces the EP Tableaux method that in section 4 is used to solve a data base problem expressed in first-order logic. In section 5 we motivate the need for natural language front-ends. Section 6 contains a brief introduction into Attempto Controlled English (ACE), and also describes the subset of ACE to be used as front-end for EP Tableaux. In section 7 we reformulate in ACE the data base example from section 4 and show its automatic translation into first-order logic and the input language PRQ of EP Tableaux. While our procedure of section 7 could be considered as putting the cart before the horse, in section 8 we proceed in a more natural way reformulating the ambiguous full natural language version of Schubert's Steamroller in ACE, translating ACE into PRQ and proving the conclusions of the Steamroller with EP Tableaux. In section 9 we summarize the main results and point to some open issues.

2 Model Generation for Applications

In several application areas — e.g. diagnosis, planning, database schema design and data base view updates — problem solving can be reduced to the systematic search for models of first-order logic specifications (see [3]). These applications have in common that the models being searched for have to be finite.

In *Diagnosis* a description of a system is given and symptoms of the system are observed. The generated models correspond to explanations of the observed symptoms and therefore must be finite (see e.g. [11, 1]).

Solving *Planning and Design* problems can as well be seen as model generation. The specifications can describe an environment, e.g. the possible movements of a robot, a starting position, and a goal to reach. Each finite model describes a solution while infinite models are meaningless.

Several *Database* issues are conveniently formalized seeing databases as models of finite sets of first-order formulae that express either static integrity constraints, views, updates, or dynamic integrity constraints [5].

With this formalization, the question whether static integrity constraints are correctly designed in the sense that there exist databases enforcing them can be formalized as a *satisfiability* problem. Here the question is whether the integrity constraints are satisfiable, i.e. whether they have a model at all. Since databases correspond to finite models, the models sought for have to be finite and the satisfiability notion of concern is *finite satisfiability*.

Further database issues that can be seen as finite satisfiability problems are the *view update problem*, i.e. if and how an update of views can be realized by updating the actually stored data, the *integrity constraint repair*, i.e. how an update could be modified so as not to violate some (static or dynamic) integrity constraints, and whether some (static or dynamic) integrity constraints might become redundant after some update (see [4]).

Since for all mentioned applications the models sought for have to be finite, a model generation method should not only be complete for refutation but also for finite satisfiability, i.e. the method should either determine that the given set of formulae is inconsistent, or generate its finite model.

3 The EP Tableaux Method

In [3, 13] a deduction method called Extended Positive Tableaux method (short EP Tableaux method) is proposed for verifying the finite satisfiability of finite sets of range restricted formulae by generating models. The method performs a systematic search for models of the considered formulae in the fashion introduced by the *tableaux methods* [15].

The input syntax for the EP Tableaux method is a fragment of the language of first-order logic called PRQ that has been shown to have the same expressive power as full first-order logic [3]. The language under consideration requires range-restricted quantification. Among other things it is required that the scope of a universal quantifier is an implication and the scope of an existential quantifier is a conjunction. In both cases nesting of quantifiers is allowed. The interpretations considered are *term interpretations* which — except for their domains — are defined like Herbrand interpretations [7]. In contrast to the domain of a Herbrand interpretation, the domain of a term interpretation is not necessarily infinite if the underlying language consists of infinitely many constants.

The EP tableaux method proceeds by decomposing the considered formulae till only literals remain.

In the following $\mathcal{S}$ denotes a finite set of formulae in the required syntax.

EP Tableaux for a set $\mathcal{S}$ are trees whose nodes are sets of closed formulae. They are inductively defined as follows:

1. The tree consisting of the single node $\mathcal{S}$ is an EP tableau for $\mathcal{S}$.
2. Let T be an EP tableau for $\mathcal{S}$, L a leaf of T, and ϕ a formula in L that is not satisfied in the term interpretation specified by the branch (i.e., by the set of ground atoms occurring in the branch). Then the tree obtained from T by appending one or more children to L according to an expansion rule applicable to ϕ is an EP tableau for $\mathcal{S}$. Each child of L consists of L and one more formula or two more formulae in the case of ϕ being a conjunction.

The expansion rules are:[1]

[1]If $\bar{c}$ is a tuple of constants and $\bar{x}$ a tuple of variables, then $F[\bar{c}/\bar{x}]$ denotes the formula F where every free occurrence of a member of x is replaced by the corresponding member of c.

$\wedge$ rule:	$\vee$ rule:	Extended PUHR rule:	
$\dfrac{E_1 \wedge E_2}{\begin{array}{c}E_1\\E_2\end{array}}$	$\dfrac{E_1 \vee E_2}{E_1 \mid E_2}$	$\dfrac{\forall \bar{x}(R(\bar{x}) \rightarrow F)}{F[\bar{c}/\bar{x}]}$	where $R[\bar{c}/\bar{x}]$ is satisfied by the interpretation specified by the branch.

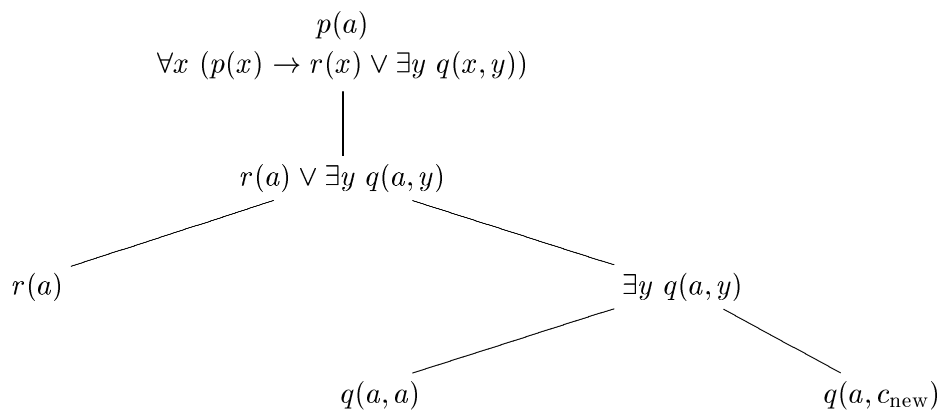
$\exists$ rule:

$$\frac{\exists x E(x)}{E[c_1/x] \mid \ldots \mid E[c_k/x] \mid E[c_{new}/x]}$$

where $\{c_1 \ldots c_k\}$ is the set of all constants occurring in the node and c_{new} is a constant distinct from $c_1, \ldots, c_k$.

The part above the horizontal line of each expansion rule describes the formula ϕ to which the rule is applied. Below the horizontal line are the additional formulae to be added to the child nodes. Alternatives corresponding to different children are separated by a vertical bar.

As an example, we give an EP tableau for $\mathcal{S} = \{p(a),\ \forall x(p(x) \rightarrow r(x) \vee \exists y q(x,y))\}$. For the sake of brevity, the non-root nodes are (in contrast to the definition above) not labelled with sets of formulae but only with the single formula added w.r.t. the node's parent.



A branch of an EP tableau is *closed* if it contains $\perp$. Otherwise, it is *open.* An EP tableau is *open* if at least one of its branches is open; otherwise, it is *closed.* Furthermore the notion of a *fair* EP tableau is needed which means that in every open branch every possible application of an expansion rule to a formula in the branch takes place after finitely many expansion steps.

The EP Tableaux method has the following properties [3]: *soundness for unsatisfiability* (if there exists a closed EP tableau for $\mathcal{S}$, then $\mathcal{S}$ is unsatisfiable) and *soundness for satisfiability* (if a fair EP tableau for $\mathcal{S}$ has an open branch, the open branch specifies a model of $\mathcal{S}$); an immediate consequence of the latter is *completeness for unsatisfiability* (if $\mathcal{S}$ is unsatisfiable, then every fair EP tableau for $\mathcal{S}$ is closed). Finally, the method also enjoys *completeness for finite satisfiability*: if $\mathcal{I}$ is a finite minimal[2] model of $\mathcal{S}$, then every fair EP tableau for $\mathcal{S}$ has a finite open branch $\mathcal{B}$ such that, up to a renaming of constants, the branch $\mathcal{B}$ represents $\mathcal{I}$. This property of EP Tableaux is essential with regard to the applications described in section 2. Thus the EP Tableaux method is not only complete for either unsatisfiability or finite satisfiability, but for both of them.

The EP Tableaux method is implemented in Prolog with a depth-first strategy as well as with a controlled breadth-first strategy (see [2]). Note, that while the latter implementation ensures completeness for finite satisfiability, the first one does not.

[2]A term model of $\mathcal{S}$ is *minimal* if no proper subset of the ground atoms it satisfies specifies a term model of $\mathcal{S}$.

4 A Database Example in Logic

In section 2 we described how to formalize the question whether a set of static integrity constraints is well-designed or not as a finite satisfiability problem. To detect ill-designed constraints we argued to use a model generation method that is complete for unsatisfiability and finite satisfiability. Since the EP Tableaux method has these properties it is appropriate to assist in the design of finitely satisfiable integrity constraints. In[2] an interactive prototype (SIC) is presented whose reasoning component is an implementation of the EP Tableaux method with a controlled breadth-first strategy in Prolog.

In this section a database example from [2] — slightly modified — is stated to demonstrate the use of the EP Tableaux method during the design of integrity constraints.

The following formulae are database integrity constraints, formalized in first order logic. The formula $\perp$ evaluates to false in every interpretation. Negation will be handled as implication, i.e. instead of $\neg\phi$ the formula $\phi \rightarrow \perp$ is written.

$$
\begin{array}{ll}
(1) & \forall A\ (department(A) \rightarrow (employee(A) \rightarrow \perp)) \\
 & \forall A\ (employee(A) \rightarrow (department(A) \rightarrow \perp)) \\
(2) & \forall A\ \forall B\ ((manager(A) \wedge department(B) \wedge of(A,B)) \rightarrow \\
 & \qquad\qquad (employee(A) \wedge member(A) \wedge of(A,B)))) \\
(3) & \forall A\ \forall B\ ((member(A) \wedge department(B) \wedge of(A,B)) \rightarrow \\
 & \qquad\qquad \forall C\ ((manager(C) \wedge of(C,B)) \rightarrow work\_for(A,C))) \\
(4) & \forall A\ (employee(A) \rightarrow \exists B\ (department(B) \wedge member(A) \wedge of(A,B))) \\
(5) & \forall A\ (department(A) \rightarrow \exists B\ (employee(B) \wedge manager(B) \wedge have(A,B) \wedge of(B,A)))) \\
(6) & \forall A\ (employee(A) \rightarrow (work\_for(A,A) \rightarrow \perp))
\end{array}
$$

These integrity constraints are ill-defined in the sense that they are satisfiable, but only by meaningless models like the empty model. In fact, application of the EP Tableaux method yields the empty model. But if e.g. the formula

$$(7) \quad employee(anne)$$

is added, the set of formulae is unsatisfiable which will be detected by applying the EP Tableaux method. This means that as soon as any employee is inserted into the database the integrity constraints are violated, or — in other words — there is no database of employees that will enforce the given integrity constraints.

To use one of the Prolog implementations of the EP Tableaux method the input formulae must conform to the following format:

Every formula is represented as `axiom()`. Implications are handled as universally quantified formulae without quantified variables. If there are no or more than one universally quantified variables, the variables will be represented in a list. E.g. Formula (1) is represented as

```
axiom(all(A,department(A) => all([],employee(A) => false))).
```

Conjunctions are represented by commata and disjunctions by semicolons. E.g. Formula (4) becomes

```
axiom(all(A,department(A) => exists(B,(department(B),member(A),of(B,A))))).
```

Clearly, this syntax is not very accessible to domain specialist unfamiliar with formal notations. In section 7, the same example will be formulated in Attempto Controlled English, a language that allows domain specialists to express formal specifications in the familiar terms of their application domain.

5 Formal or Informal Specification Languages?

Verifying a specification, i.e. checking its internal consistency, and validating it, i.e. making certain that it coincides with or closely corresponds to the domain specialists' mental model of the problem domain, necessarily involves human domain specialists who read and review the specification.

To enhance readability and reviewability of a specification, and thus to ease verification and validation, it is of utmost importance to minimize the semantic distance between the problem domain and the specification — where semantic distance stands for the mental effort to relate two different conceptualizations or two different notations, or to translate one conceptualization or notation into the other.

Since the language of the problem domain is usually given, one must carefully select — or perhaps even construct — a specification language that reduces the semantic distance.

Formal specification languages are based on mathematics or logic. These languages have the advantage of an unambiguous syntax and a clean semantics, and support the automatic analysis of specifications. Formal specification languages seem appropriate if the domain specified is well-defined — perhaps already formalized — since the semantic distance can clearly be minimized. Popular examples of problems from well-defined domains are the greatest common divisor, the mutilated checker board, send-more-money, and the zebra puzzle. If the problem domain is ill-defined, however, formal specification languages do not normally fulfil the demand of minimizing the semantic distance since mapping the concepts of the domain to the constructs of the formal specification language can be extremely difficult, and the resulting formal specification can be incomprehensible to anybody besides the authors. Note, that most specifications of real-world problems — including our data base example — arise in ill-defined domains.

On the other hand, in many domains natural language is used as informal specification language since it is the fundamental means of human communication. Since any problem can be expressed in natural language to any required level of detail, the semantic distance between a problem domain and a specification expressed in natural language can be made arbitrarily small. However, not being formal, natural language can be used in uncontrolled ways leading to ambiguous, imprecise and unclear statements mixing relevant and irrelevant information, different levels of abstraction, and different degrees of granularity. This has been clearly exhibited when problem descriptions in natural language — e.g. the Library Data Base Problem [14], or Schubert's Steamroller [6] — were manually translated into their formal equivalents. Different researchers achieved widely differing results depending on their understanding, ad hoc interpretation and shallow parsing of the natural language texts.

To minimize the semantic distance between problem domains and formal specifications, to combine the advantages of formal and natural languages, and to avoid many of their disadvantages, we have constructed the specification language Attempto Controlled English [8, 12].

6 Overview of Attempto Controlled English

Attempto Controlled English (ACE) is a controlled natural language specifically constructed to write specifications [8, 12]. ACE allows users to express specifications precisely, and in the terms of the application domain. ACE specifications are computer-processable and can be unambiguously translated into a logic language. Though ACE may seem informal, it is a formal language with the semantics of the underlying logic language. This also means that ACE has to be learned like other formal languages. Though initially developed as a specification language, ACE has since been used for other purposes, e.g. as input language of a program synthesizer. Here we introduce ACE as a front-end to EP Tableaux.

What exactly does it mean that ACE is a controlled natural language?

ACE is a subset of standard English, i.e. every ACE sentence is correct English though not every English sentence is allowed in ACE. ACE uses a domain-specific vocabulary, i.e. predefined function words like determiners, prepositions and conjunctions, and user-defined content words like nouns, verbs, and adjectives. Users can extend and modify the lexicon via a simple interface requiring little more than basic grammar knowledge. ACE employs a restricted grammar in the form of a small set of construction and interpretation rules. Construction rules define the form of ACE sentences and state restrictions intended to remove imprecision and to restrain ambiguities. Interpretation rules control the semantic analysis of grammatically correct ACE sentences and resolve remaining ambiguities.

The most important construction rules are:

- ACE specifications are sequences of anaphorically interrelated simple and composite sentences.

- Simple sentences have the form *subject + verb + complements + adjuncts*, where complements (noun phrases, prepositional phrases) are required for transitive and ditransitive verbs, and adjuncts (adverbs, prepositional phrases) are optional.

  Here is a simple sentence paraphrasing a part of a formula of the data base example from section 4:

  > A manager of a department is an employee.

- Composite sentences are built from other sentences through coordination (and, or), subordination by if ... then ... , subordination by relative sentences (who, which, that), verb phrase negation (does not, is not), noun phrase negation (no), or quantification (a, there is a, every, for every).

  Here are two composite sentences paraphrasing two formulae of the data base example:

  > Every department has an employee who is the manager of a department.
  > Every manager of a department is an employee and a member of the department.

  The second sentence shows that verb phrase coordination can be simplified by leaving out the repeated verb.

- ACE sentences can be interrelated by anaphora, i.e. personal pronouns or definite noun phrases.

  Here is an example showing both possibilities:

  > A manager of a department is an employee. He leads the department.

  The personal pronoun he of the second sentence is an anaphoric reference to the noun phrase a manager of the first sentence. Similarly, the department is an anaphoric reference to a department.

- Verbs are only used in the simple present tense, the active voice, the indicative mood, and the third person.
- Modal verbs (can, must etc.), intensional verbs (hope, know etc.), and modal adverbs (possibly, probably etc.) are not allowed.

Following are essential interpretation rules:

- Verbs denote events or states, and the textual order of verbs determines the default temporal order of the associated events and states.
- Prepositional phrases in adjunct position always modify the verb, while relative sentences modify the immediately preceding noun phrase.
- The textual occurrence of a quantifier opens its scope that extends to the end of the sentence; thus any following quantifier is within the scope of the preceding ones.
- The antecedent of anaphoric reference is always the most recent suitable noun phrase that agrees in number and gender.

The construction and interpretation rules are realized as a unification-based phrase structure grammar that is used by the chart-parser of the Attempto system. The parser deterministically translates ACE texts into discourse representation structures — a syntactic variant of first-order predicate logic (FOL) — into the standard form of FOL, and optionally into clausal form. Furthermore, a paraphrase is generated that shows how the Attempto system interprets the ACE input. For further details of the Attempto system cf. [8].

Translating the sentences

> The manager of a department is an employee. He leads the department.

yields the discourse representation structure

```
[A,B,C,D]
manager(A)
department(B)
of(A,B)
state(C,be(employee(A)))
state(D,lead(A,B))
```

where the discourse referents `[A,B,C,D]` are existentially quantified variables representing objects of the discourse domain and the other lines represent atomic conditions for these discourse referents. Note that the second sentence is translated in the context of the first one so that the anaphora (he, the department) are automatically resolved.

The discourse representation structure is further translated into the equivalent FOL formula

```
exists(A,manager(A) & exists(B,department(B) & of(A,B) &
exists(C,state(C,be(employee(A))) & exists(D,state(D,lead(A,B))))))
```

The translation also generates the paraphrase

> The manager of a department is an employee. [The manager] leads [the department].

where the bracketed noun phrases replace the anaphoric references of the second sentence.

Since verbs denote events and states, ACE sentences are translated into discourse representation structures that contain discourse referents and conditions for reified verbs. As demonstrated in the example above some conditions are constructed from the subject, the verb and the complements of the sentence. Other conditions — that did not occur in the example — are derived from optional adjuncts, i.e. adverbs and prepositional phrases.

It is important to note that the interpretation of verbs as events and states results in a representation that uses function symbols. For example, in the FOL formula `exists(D,state(D,lead(A,B)))` the term `lead(A,B)` is built-up by the function symbol `lead`.

Since the language PRQ of EP Tableaux does not allow function symbols, we use as a front-end to EP Tableaux a subset of ACE without events and states, and consequently without adjuncts. This subset can be translated into a function-free subset of FOL and subsequently into PRQ. Thus the translation of the sentences

> The manager of a department is an employee. He leads the department.

in the ACE subset yields now the discourse representation structure

```
[A,B]
manager(A)
department(B)
of(A,B)
employee(A)
lead(A,B)
```

and the equivalent FOL formula

```
exists(A,manager(A) & exists(B,department(B) & of(A,B) & employee(A)
& lead(A,B)))
```

In the sequel, ACE stands for the subset of ACE in which verbs are not interpreted as events or states, and in which adjuncts are not allowed.

7 Database Example in ACE and Translation into Logic

As a first test for the successful interplay between the EP Tableaux method and the Attempto system we specify the database example from section 4 in ACE. The translation of the FOL formulae derived from ACE sentences into the Prolog format of PRQ requires just a few systematic transformations:

FOL	PRQ
`-S`	`S => false`
`S => R`	`all([],S => R)`
& (conjunction)	,
$ (disjunction)	;
`all(A,all(B,S => Q))`	`all([A,B],S => Q)`
`exists(A,S & Q)`	`exists(A,(S,Q))`
`S`	`axiom(S).`

We give the ACE formulation of the database integrity constraints, their automatic translation into FOL formulae, and the corresponding PRQ formulae.

Constraint (1) expresses that departments and employees are different entities.

(1) ACE: No department is an employee. No employee is a department.

FOL:
```
all(A,department(A) => -employee(A))
all(A,employee(A) => -department(A))
```
PRQ:
```
axiom(all(A,department(A) => all([],employee(A) => false))).
axiom(all(A,employee(A) => all([], department(A) => false))).
```

In constraint (2) the definite noun phrase the department is used as an anaphor, i.e. it refers back to the previously occurring noun phrase a department. Logically, this means that the two noun phrases relate to the same variable.

(2) ACE: Every manager of a department is an employee and a member of the department.

FOL:
```
all(A,all(B,manager(A) & department(B) & of(A,B)
          => (employee(A) & member(A) & of(A,B))))
```
PRQ:
```
axiom(all([A,B],(manager(A),department(B),of(A,B))
          => (employee(A),member(A),of(A,B)))).
```

A "natural" formulation of the third constraint is:

(3a) ACE: Every member of a department works for the manager of the department.

FOL:
```
all(A,all(B, member(A) & department(B) & of(A,B)
          => exists(C,manager(C) & of(C,B) & work_for(A,C))))
```
PRQ:
```
axiom(all([A,B],(member(A),department(B),of(A,B))
          => exists([C],(manager(C),of(C,B)),work_for(A,C)))).
```

However, the formulation (3a) does not yield the desired results since now the manager is existentially quantified. In the original formulation in section 4 the manager is universally quantified which is necessary to prove the unsatisfiability of the set of formulae. To get all variables universally quantified we write the somewhat "unnatural" sentence (3b):

(3b) ACE: Every member of a department works for every manager of the department.

FOL:
```
all(A,all(B,member(A) & department(B) & of(A,B)
            => all(C,manager(C) & of(C,B) => work_for(A,C))))
```
PRQ:
```
axiom( all([A,B],(member(A),department(B),of(A,B))
            => all(C,(manager(C),of(C,B)) => work_for(A,C)))).
```

Sentence (4) is analogous to sentence (1) without conjunction.

(4) ACE: Every employee is a member of a department.

FOL:
```
all(A,employee(A) => exists(B,department(B) & member(A) & of(A,B)))
```
PRQ:
```
axiom(all(A,employee(A) => exists(B,(department(B),member(A),of(A,B))))).
```

Sentence (5) employs a relative sentence that further modifies the noun employee. The conditions are conjunctively added to the formula `employee(B)`.

(5) ACE: Every department has an employee who is a manager of the department.

FOL:
```
all(A,department(A) => exists(B,employee(B) &
            manager(B) & of(B,A) & have(A,B)))
```
PRQ:
```
axiom(all(A,department(A) => exists(B,(employee(B),
            manager(B),of(B,A),have(A,B))))).
```

The last constraint expresses that nobody works for herself/himself. As ACE does not yet handle reflexive pronouns the sentence has to be less naturally stated as:

(6) ACE: No employee X works for X.

FOL:
```
all(A,employee(A) => -work_for(A,A))
```
PRQ:
```
axiom(all(A,employee(A) => all([],work_for(A,A) => false))).
```

In ACE *dynamic names* (here X) can be added after a noun to distinguish single instances of the set of objects denoted by the preceding common noun (here employee). The dynamic name does not occur literally in the logical form, it just guarantees the correct variable bindings. Used alone, the dynamic name refers back to the whole noun phrase in which it was introduced. Though the sentences sound less natural, dynamic names are a powerful and necessary means to express mathematical or logical problems. For another example using dynamic names, see sentence (3) of section 8.

Taking the PRQ formulae generated from the ACE sentences as an input to EP Tableaux we can prove the same results as with the original formulation in section 4. Thus, we have shown that the difficult and unfamiliar formal statement of the database example can indeed be replaced by a more natural formulation without losing precision.

8 Steamroller Example

We chose *Schubert's Steamroller* — a well-known problem for automated reasoning systems [6, 9] — as a further test for the amalgamation of Attempto with EP Tableaux. The problem is interesting for our approach in that on the one hand it raises issues of how to express problems unambiguously and on the other hand of which theorem-proving technique is best applied. We formulated the problem in ACE and successfully proved its conclusions with the EP Tableaux method — thus challenging Stickel's [6] warning of "the danger of using natural language to try to convey problem statements unambiguously".

Each sentence of the original English formulation of the problem is given below along with its reformulation in ACE, its translation into FOL and into PRQ.

(1) Wolves, foxes, birds, caterpillars, and snails are animals, and there are some of each of them.

ACE:	Every wolf is an animal.	Wolf1 is a wolf.
FOL:	`all(A,wolf(A) => animal(A))`	`wolf('Wolf1')`
	...	...
PRQ:	`axiom(all(A,wolf(A) => animal(A))).`	`axiom(wolf('Wolf1')).`
	...	...

We replaced the imprecise natural language construction *P's are Q's* (e.g. *Wolves are animals.*) with the non-ambiguous ACE formulation Every P is a Q (e.g. Every wolf is an animal.). Furthermore, since ACE does not allow to express existential claims directly, one has to explicitly introduce at least one wolf, in our case this is the proper noun Wolf1 which is interpreted as a constant `'Wolf1'`.

Sentence (2) is analogous to sentence (1).

(2) Also, there are some grains, and grains are plants.

ACE:	Every grain is a plant.	Grain1 is a grain.
FOL:	`all(A,grain(A) => plant(A))`	`grain('Grain1')`
PRQ:	`axiom(all(A,grain(A) => plant(A))).`	`axiom(grain('Grain1')).`

ACE expresses the existential quantification over plants in (3) by using the singular noun phrase a plant.

(3) Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants.

ACE: Every animal A eats every plant or eats every animal D that is smaller than A and that eats a plant.

FOL:
```
all(A,animal(A) => (all(B,plant(B) => eat(A,B)) $ all(D,all(F,animal(D) &
                smaller_than(D,A) & plant(F) & eat(D,F) => eat(A,D)))))
```

PRQ:
```
axiom(all(A,animal(A) => (all(B,plant(B) => eat(A,B));all([D,F],(animal(D),
                smaller_than(D,A),plant(F),eat(D,F)) => eat(A,D))))).
```

To get correct variable bindings in sentence (3) we employ dynamic names in the ACE formulation (see section 7). Moreover, repeating the relative pronoun that after the conjunction and ensures that eats a plant modifies animal D.

(4) Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which in turn are much smaller than wolves.

ACE: Every caterpillar is smaller than every bird. Every snail is smaller than every bird.
Every bird is smaller than every fox.
Every fox is smaller than every wolf.

FOL:
```
all(A,caterpillar(A) => (all(B,bird(B) => smaller_than(A,B))))
...
```
PRQ:
```
axiom(all(A,caterpillar(A) => all(B,bird(B) => smaller_than(A,B)))).
...
```

In sentences (4) and (5) full natural language *P's are smaller than Q's* or *P's eat Q's* are to be interpreted as universally quantifying over both P-instances and Q-instances. In ACE this universal quantification has to be expressed *explicitly* — and therefore unambiguously — by adding the universal quantifier every. Sentence (5) shows furthermore that in ACE No P eats a Q is interpreted equivalently to Every P does not eat a Q.

(5) Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails.

ACE: No wolf eats a fox. No wolf eats a grain. No bird eats a snail.
Every bird eats every caterpillar.

FOL:
```
all(A,wolf(A) => -(exists(B,fox(B) & eat(A,B))))
...
all(A,bird(A) => all(B,caterpillar(B) => eat(A,B)))
```
PRQ:
```
axiom(all(A,wolf(A) => all([],exists(B,(fox(B),eat(A,B))) => false))).
...
axiom(all(A,bird(A) => all(B,caterpillar(B) => eat(A,B)))).
```

As in previous examples, in sentence (6) the conjunction and — a possible source of ambiguity — is removed. Instead, ACE uses a separate sentence for each conjunct.

(6) Caterpillars and snails like to eat some plants.

ACE: Every caterpillar eats a plant. Every snail eats a plant.

FOL:
```
all(A,caterpillar(A) => exists(B,plant(B) & eat(A,B)))
...
```
PRQ:
```
axiom(all(A,caterpillar(A) => exists(B,(plant(B),eat(A,B))))).
...
```

The sentence to be proved is:

(7) Therefore, there is an animal that likes to eat a grain-eating animal.

ACE: An animal eats an animal that eats a grain.

FOL:
```
exists(A,animal(A) & exists(B,animal(B) & exists(C,grain(C) &
          eat(B,C) & eat(A,B))))
```
PRQ:
```
axiom(exists(A,(animal(A),exists(B,(animal(B),exists(C,(grain(C),
          eat(B,C),eat(A,B)))))))).
```

From the above statement of the problem the EP Tableaux method generates a model, in which a fox eats a bird, and a bird eats a grain, or more precisely, a model where the following two formulae are true:

```
eat('Fox1','Bird1')        eat('Bird1','Grain1')
```

Furthermore, replacing sentence (7) with its negation (8)

(8) ACE: No animal eats an animal that eats a grain.

FOL:
```
all(A,animal(A) => -(exists(B,animal(B) &
           exists(C,grain(C) & eat(B,C) & eat(A,B)))))
```

PRQ:
```
axiom(all(A,animal(A) => all([],exists(B,(animal(B),
           exists(C,(grain(C),eat(B,C),eat(A,B))))) => false))).
```

the EP Tableaux method proves — as desired — the unsatisfiability of the set of formulae.

9 Conclusion and Outlook

We have shown that Attempto Controlled English (ACE) can serve as a natural language front-end to the model-generator EP Tableaux. Domain specialists who may not be familiar with formal specification methods can formulate specifications in natural language, i.e. in the concepts and the terms of their application domain, and can then verify and validate the specifications with the help of EP Tableaux.

A "backward" reformulation in ACE of a data base example given originally in first-order logic, and the ACE version of Schubert's Steamroller problem originally formulated in full natural language demonstrated the viability and flexibility of our approach.

Nevertheless, there is room for improvement.

Concerning EP Tableaux, the limitation to function-free subsets of first-order logic should be removed. One way to achieve this, is the representation of functions as relations. This conservative extension has the advantage that the theoretical foundations of EP Tableaux would not have to be changed. A further advantage would be that the full version of ACE could be used as front-end to EP Tableaux.

Both ACE and PRQ are untyped languages. Introducing types in both languages would allow users to express specifications in a more natural and more concise way, thus leading to a further reduction of the semantic distance. In addition, types would render EP Tableaux more efficient.

While the ACE front-end of EP Tableaux allows users to conveniently formulate their input, the output of EP Tableaux — listings of models — is rather terse. It would certainly help if the output, too, would consist of ACE sentences.

We are working on other extensions of Attempto Controlled English — e.g. plurality, alternative notations, structuring of large specifications — that are, however, not directly relevant to the topic of this paper.

Acknowledgements

We would like to thank François Bry for his support and for useful discussions, and Rolf Schwitter for his essential contributions to the development of Attempto Controlled English.

References

[1] P. Baumgartner, P. Fröhlich, U. Furbach, and W. Nejdl. Tableaux for diagnosis applications. In *6th Workshop on Theorem Proving with Tableaux and Related Methods*, LNAI, pages 76–90. Springer-Verlag, 1997.

[2] F. Bry, N. Eisinger, H. Schütz, and S. Torge. SIC: Satisfiability checking for integrity constraints. In *Proc. Deductive Databases and Logic Programming, Workshop at the Joint International Conference and Symposium on Logic Programming*, 1998.

[3] F. Bry and S. Torge. A deduction method complete for refutation and finite satisfiability. In *Proc. 6th European Workshop on Logics in Artificial Intelligence*, LNAI 1489. Springer-Verlag, 1998.

[4] F. Bry and S. Torge. Solving database satisfiability problems (extended abstract). In *Proc. 11. Workshop "Grundlagen von Datenbanken"*, 1999.

[5] C. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases.* Springer-Verlag, 1990.

[6] Stickel M. E. Schubert's steamroller problem: Formulations and solutions. *Jour. of Automated Reasoning*, 2, 1986.

[7] M. Fitting. *First Order Logic and Automated Theorem Proving.* Springer, 1990.

[8] N. E. Fuchs, U. Schwertel, and R. Schwitter. Attempto Controlled English — Not Just Another Logic Specification Language. In P. Flener, editor, *Logic-Based Program Synthesis and Transformation, Eighth International Workshop LOPSTR'98*, Lecture Notes in Computer Science 1559, pages 1–20, Manchester, UK, 1999. Springer-Verlag.

[9] D. Givan R., McAllester and Shalaby S. Natural language based inference procedures applied to Schubert's Steamroller. In *Proc. AAAI*, 1991.

[10] A. Hall. Seven Myths of Formal Methods. *IEEE Software*, 7(5):11–19, 1990.

[11] D. Poole. Normality and faults in logic-based diagnosis. In *11th Int. Joint Conf. on Artificial Intelligence*, pages 1304–1310, 1985.

[12] R. Schwitter. *Kontrolliertes Englisch für Anforderungsspezifikationen.* PhD thesis, Department of Computer Science, University of Zurich, 1998.

[13] S. Torge. *Überprüfung der Erfüllbarkeit im Endlichen: Ein Verfahren und seine Anwendung.* PhD thesis, Computer Science, University of Munich, 1998.

[14] J. M. Wing. A Study of 12 Specifications of the Library Problem. *IEEE Software*, 5(4):66–76, 1988.

[15] G. Wrightson, ed. Special issue on automated reasoning with analytic tableaux – part I, part II. *Jour. of Automated Reasoning*, 13(2,3), 1994.