A Natural Language Front-End to Model Generation

NORBERT E. FUCHS, UTA SCHWERTEL, *Department of Computer Science, University of Zurich, Winterthurer Str. 190, 8057 Zurich, Switzerland. E-mail: {fuchs,uschwert}@ifi.unizh.ch*

SUNNA TORGE[1], *Department of Computer Science, University of Munich, Oettingenstr. 67, 80538 Munich, Germany. E-mail: torge@informatik.uni-muenchen.de*

Abstract

Currently, formal methods are not widely employed since many domain specialists are not familiar or comfortable with formal notations and formal tools. To address this problem we developed Attempto Controlled English (ACE), a subset of English that allows domain specialists to express problems in the language of their application domain, that can be unambiguously translated into first-order logic, and thus can replace first-order logic as a formal notation. In this paper we describe how ACE can be used as a front-end to EP Tableaux, a model generation method complete for unsatisfiability and for finite satisfiability. We specified in ACE a database example that was previously expressed in the EP Tableaux language PRQ, automatically translated the ACE specification into PRQ, and with the help of EP Tableaux 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.

Keywords: formal methods, automated reasoning, controlled natural language, Attempto Controlled English, model generation, EP Tableaux, database schema design, Schubert's Steamroller

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 [9] 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 semantic distance between application domain and specification language — by replacing visibly formal specifications by specifications in Attempto Controlled English (ACE), a subset of English that seems informal but is in fact formal and that can be deterministically translated into logic languages [7]. ACE allows domain specialists to express specifications precisely and in the terms of their

[1] Now with Sony International (Europe) GmbH, European Research & Development, Fellbach, Germany.

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".

There is a further incentive in employing ACE: formal tools like model generators become more and more important to database applications [2, 4]. With ACE they become available even to domain specialists who are not familiar with the formal notations that these tools normally require. With a similar motivation Holt et al. [10] developed a natural language interface to a model checker.

To prove our point we show that (a subset of) Attempto Controlled English can replace logic as a front-end to EP Tableaux [3, 16], a model generation method to verify the finite satisfiability of finite sets of range restricted formulae. 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.

Apart from other application areas (see Section 2), model generation and theorem proving are playing an increasingly important role in computational linguistics, particularly in semantic analysis [13]. First-order inference turns out to be a powerful and practical tool for problems like disambiguation, treatment of presupposition, pronoun resolution etc. [1, 12].

Though in the sequel we will address some of these linguistic problems, we hasten to emphasize that the work reported here has different goals: we are not using model generation to analyze ACE, but use ACE as a natural-language front-end to a model generator.

The rest of the paper is organized as follows. Section 2 introduces the model generation method EP Tableaux that in Section 3 is used to solve a data base problem expressed in first-order logic. Section 4 contains a brief introduction to Attempto Controlled English, and also describes the subset of ACE to be used as a front-end for EP Tableaux. In Section 5 we reformulate in ACE the data base example from Section 3 and show its automatic translation into first-order logic and the input language PRQ of EP Tableaux. While our procedure of Section 5 could be considered as putting the cart before the horse, in Section 6 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 7 we summarize the main results and point to some open issues.[2]

2 The Model Generation Method EP Tableaux

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 sought for have to be finite.

As an example we consider *database schema design*. Several database issues can be formalized by viewing databases as models of finite sets of first-order formulae

[2]A preliminary version of this paper appeared as ASE'99 short paper.

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.*

To apply a model generation method to the mentioned applications, the 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, if one exists.

In [3, 16] 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. This means the EP Tableaux method exactly provides what is needed by the applications mentioned above, namely completeness for both unsatisfiability and finite satisfiability. The method performs a systematic search for models of the considered formulae in a *tableau-like* fashion [17]. It proceeds by decomposing the formulae till only literals remain. Branches in the expanded EP tableau that do not contain the formula $\perp$ (which evaluates to false in every interpretation) represent a model of the given set of formulae.

The input language 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 PRQ 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. Negation will be handled as implication, i.e. instead of $\neg\phi$ the formula $\phi \rightarrow \perp$ is written. The formal definition of the PRQ-Syntax is the following:

Definition

Positive conditions are inductively defined as follows:

1. Atoms except $\perp$ are positive conditions.
2. Conjunctions and disjunctions of positive conditions are positive conditions.
3. $\exists y\ F$ is a positive condition if F is a positive condition.

Ranges for variables $x_1, \ldots, x_n$ are inductively defined as follows:

1. An atom in which all of $x_1, \ldots, x_n$ occur is a range for $x_1, \ldots, x_n$.
2. $A_1 \vee A_2$ is a range for $x_1, \ldots, x_n$ if both A_1 and A_2 are ranges for $x_1, \ldots, x_n$.
3. $A_1 \wedge A_2$ is a range for $x_1, \ldots, x_n$ if A_1 is a range for $x_1, \ldots, x_n$ and A_2 is a positive condition.
4. $\exists y\ R$ is a range for $x_1, \ldots, x_n$ if R is a range for $y, x_1, \ldots, x_n$ and if $x_i \neq y$ for all $i = 1, \ldots, n$.

Positive Formulae with Restricted Quantifications (short PRQ formulae) are inductively defined as follows:

1. Atoms (in particular $\perp$ and $\top$) are PRQ formulae.
2. Conjunctions and disjunctions of PRQ formulae are PRQ formulae.
3. A formula of the form $P \rightarrow F$ is a PRQ formula if P is a positive condition and F a PRQ formula.
4. A formula of the form $\forall x_1 \ldots x_n\ (R \rightarrow F)$ $(n \geq 1)$ is a PRQ formula if R is a range for $x_1, \ldots, x_n$ and if F is a PRQ formula.
5. A formula of the form $\exists x\ (R \wedge F)$ is a PRQ formula if R is a range for x and if F is a PRQ formula.

Example
The formula $F = \forall xy\ (p(x) \wedge q(y) \vee r(x,y) \rightarrow \exists z\ (s(z) \wedge t(z)))$ is a PRQ formula, because $p(x) \wedge q(y) \vee r(x,y)$ is a range for x and y and $s(z)$ is a range for z. The formula $G = \forall xy\ (p(x) \vee q(y) \rightarrow s(y))$, in contrast, is not a PRQ formula, since the premise of the implication is not a range for x and y.

The interpretations considered in the following are *term interpretations* which — except for their domains (they may be finite, even if the underlying language consists of infinitely many constants) — are defined like Herbrand interpretations [6].

The EP tableaux expansion rules are:[3]

$\exists$ 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 $c_1, \ldots, c_k$.

Extended PUHR rule:

$$\frac{\forall \bar{x}(R(\bar{x}) \rightarrow F)}{F[\bar{c}/\bar{x}]}$$

where $R[\bar{c}/\bar{x}]$ is satisfied by the interpretation specified by the branch.

$\wedge$ rule:

$$\frac{E_1 \wedge E_2}{\begin{array}{c} E_1 \\ E_2 \end{array}}$$

$\vee$ rule:

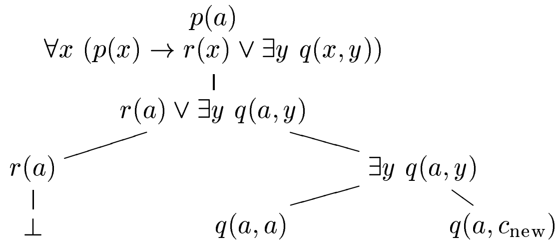
$$\frac{E_1 \vee E_2}{E_1 \mid E_2}$$

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.

Instead of the exact definition of EP tableaux we give an example. For details concerning the definition, see [3].

[3] 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.

The following figure presents the EP tableau for $\mathcal{S} = \{p(a),\ \forall x(p(x) \to r(x) \vee \exists y q(x,y)),\ r(a) \to \bot\}$, where the rightmost and the middle branch represent models of $\mathcal{S}$.



A branch of an EP tableau is *closed* if it contains $\bot$. 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[4] 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 mentioned at the beginning of this section. 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 search strategy as well as with a controlled breadth-first search strategy (see [2]). Note, that while the latter implementation ensures completeness for finite satisfiability, the first one does not.

3 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 for 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 search strategy in Prolog.

In this section a slightly modified version of the database example from [2] is stated

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

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.

(1) $\forall A\ (department(A) \rightarrow (employee(A) \rightarrow \bot))$
$\forall A\ (employee(A) \rightarrow (department(A) \rightarrow \bot))$

(2) $\forall A\ \forall B\ ((manager(A) \wedge department(B) \wedge of(A,B)) \rightarrow$
$(employee(A) \wedge member(A) \wedge of(A,B)))$

(3) $\forall A\ \forall B\ ((member(A) \wedge department(B) \wedge of(A,B)) \rightarrow$
$\forall C\ ((manager(C) \wedge of(C,B)) \rightarrow work\_for(A,C)))$

(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) $\forall A\ (employee(A) \rightarrow (work\_for(A,A) \rightarrow \bot))$

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 for instance the formula

(7) $employee(anne)$

is added, the set of formulae is unsatisfiable, because for every department a manager is required who in turn is an employee. Since every employee works for the manager and nobody works for her/himself, there is a contradiction, 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. For instance, formula (1) is represented as

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

Conjunctions are represented by commas and disjunctions by semicolons. For instance, 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 specialists unfamiliar with formal notations. In Section 5, the same example will be formulated more directly and more naturally in Attempto Controlled English.

4 Overview of Attempto Controlled English

Attempto Controlled English (ACE) is a controlled natural language specifically constructed to write specifications [7, 14]. 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.

Using ACE, we have specified a simple automated teller machine, Kemmerer's library database problem, and several other problems. Though initially developed as a specification language, ACE has since been used for other purposes, e.g. as input language of a program synthesizer and to describe diseases. 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.

The 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. Following closely the standard algorithms of Kamp & Reyle [11] the parser deterministically translates ACE texts into discourse representation structures, and then into the standard form of first-order logic, 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 see [7, 14].

It is important to note that in ACE the interpretation of verbs as events and states results in logic representations that use function symbols. 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. This subset can be translated into a function-free subset of first-order logic and subsequently into PRQ.

The translation of the sentences

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

in the ACE subset yields the discourse representation structure

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

and the equivalent first-order 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.

5 The Database Example in ACE

To demonstrate ACE as a natural language front-end to the EP Tableaux method we express the database example from Section 3 in ACE. We give the ACE formulation of each constraint together with its automatic translation into first-order logic (FOL) and PRQ formulae. The conversion from the FOL syntax generated by Attempto into the PRQ syntax (see end of Section 3) requires just a few systematic steps.

Constraint (1) states that departments and employees are different entities. In ACE this is expressed using the quantifier no which is logically treated as every ... not.

(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 which relates to the previously occurring noun phrase a department. This means that the two noun phrases refer to the same object.

(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)))).
```

Constraint (3) contains the full verb work for both argument positions of which are universally quantified.

(3) 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) contains a relative sentence that further modifies the immediately preceding noun employee. The conditions derived from the relative sentence 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 constraint is 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))).`

To express reflexivity, sentence (6) employs so-called *dynamic names* (here X). Dynamic names in ACE distinguish single instances of the set of objects denoted by the preceding noun (here employee). Dynamic names thus allow users to explicity refer to different objects falling under the same predicate. Used without a preceding noun, a dynamic name relates to the same discourse referent as the noun phrase in which it was introduced (here the second X relates to the discourse referent introduced by no employee X). Dynamic names do not occur literally in the logical formula but just guarantee correct variable bindings. Though sentences may sound less natural, dynamic names are a powerful and necessary means to express mathematical or logical problems in ACE. For another example using dynamic names, see sentence (3) of Section 6.

Taking the above PRQ formulae generated from the ACE sentences as input to the EP Tableaux method we can reproduce the results of the original formulation in Section 3.

Constraints (1)–(6) are satisfiable but only by the empty model. The addition of the ACE sentence

(7) Anne is an employee.

renders the set of automatically generated PRQ formulae unsatisfiable.

The above results demonstrate that the difficult and unfamiliar formal statement of the database example can indeed be replaced by a more natural formulation without losing precision.

6 Schubert's Steamroller in ACE

As a further test for the productive interplay between ACE and EP Tableaux we chose *Schubert's Steamroller* — a well-known problem for automated reasoning systems [15]. The problem is interesting for our approach as it raises the issues of how to express problems unambiguously and of which theorem-proving technique is best applied to prove the desired facts.

Each sentence of the original natural language version of the Steamroller problem is given below along with its unambiguous reformulation in ACE, its translation into FOL and into PRQ formulae. Note, that we used the same simplifications of the original version as other authors, e.g. [8]. For example, we simplified *A likes to eat B* to *A eats B*, or we replaced *much smaller than* by *smaller than*.

(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')).` ...

In full natural language bare plurals (e.g. *wolves*, *animals*) are ambiguous. They can have universal, existential or other readings. For example, constructions of the form *P's are Q's* (e.g. *Wolves are animals.*) are typically interpreted as *Every P is a Q*, where Q-instances are existentially quantified. The construction *P's like Q's*, however, prefers a universal quantification over Q-instances. In ACE the intended interpretation has to be made explicit; bare plurals are replaced by overtly quantified singular noun phrases (e.g. Every wolf is an animal.).

ACE does not yet allow users to express existential claims directly. Instead, as examples (1) and (2) show, one has to explicitly introduce at least one object. In example (1) this is done via the proper noun Wolf1 which is interpreted as a constant `'Wolf1'`.

(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')).`

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

(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 the dynamic names A and D in the ACE formulation (see Section 5). 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 is 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 in the Steamroller problem 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)))))))).`

With the help of the EP Tableaux method we can show that the automatically generated PRQ formulae (1)–(7) are satisfiable. Moreover, a model is generated in which the following two formulae are true:

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

In this model a fox eats a bird, and the bird eats a grain, i.e. there is an animal that eats a grain-eating animal which is required in (7) as a solution to the Steamroller problem. 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 formulae (1)–(6) and (8).

By rephrasing the original natural language version of the Steamroller problem unambiguously in ACE and then successfully proving its conclusions with EP Tableaux we challenge Stickel's [15] warning of "the danger of using natural language to try to convey problem statements unambiguously".

7 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 database example given originally in first-order logic, and the ACE version of Schubert's Steamroller problem originally formulated in full natural language demonstrate the viability and flexibility of our approach.

Nevertheless, there is room for improvement.

In 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. As an incentive, the full version of ACE could then be used as a 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, Rolf Schwitter for his contributions to the development of ACE, and the participants of ICoS-1 for useful discussions. Furthermore, we express our gratitude to the anonymous reviewers for their helpful comments. The work presented is partially funded by the Swiss National Science Foundation (SNF 20-53806.98).

References

[1] P. Blackburn, J. Bos, M. Kohlhase, and H. de Nivelle. Inference and Computational Semantics. In *Third International Workshop on Computational Semantics (IWCS-3)*, Tilburg, The Netherlands, 1999.

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

[7] N. E. Fuchs, U. Schwertel, and R. Schwitter. Attempto Controlled English — Not Just An-

other Logic Specification Language. In P. Flener, editor, *Logic-Based Program Synthesis and Transformation, 8th International Workshop LOPSTR '98*, LNCS 1559. Springer-Verlag, 1999.

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

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

[10] A. Holt, E. Klein, and C. Grover. Natrual language for hardware verification: Semantic interpretation and model checking. In Monz and de Rijke [13], pages 133–137.

[11] H. Kamp and U. Reyle. *From Discourse to Logic, Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory.* Studies in Linguistics and Philosophy 42. Kluwer, Dordrecht, 1993.

[12] K. Konrad. Model generation for natural-language semantic analysis. Technical report, Department of Computer Science, Saarland University, 1999. TABLEAUX'99 position paper.

[13] C. Monz and M. de Rijke, editors. *Proc. First Workshop on Inference in Computational Semantics (ICoS-1)*, Institute for Logic, Language and Computation (ILLC), Amsterdam, August 1999.

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

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

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

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

Received 1 March 2000