Controlled Natural Language Can Replace First-Order Logic

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

Many domain specialists are not familiar or comfortable with formal notations and formal tools like theorem provers or model generators. To address this problem we developed Attempto Controlled English (ACE), a subset of English that can be unambiguously translated into first-order logic and thus can conveniently replace first-order logic as a formal notation. In this paper we describe how ACE has been 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.

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. The reasons are twofold — formal notations, e.g. formal specifications or integrity constraints, are hard to understand and difficult to relate to the concepts of an application domain, and consequently formal tools like theorem provers or model generators are not easily accessible to people unfamiliar with formal notations.

We are directly addressing both problems by substituting obviously formal notations by Attempto Controlled English (ACE), a subset of full English that can be deterministically translated into first-order logic [5]. Here we show that ACE can conveniently replace first-order logic as input language of the model generation method EP Tableaux [3].

The combination of EP Tableaux with an ACE front-end allows domain specialists to express, verify and validate their problems in familiar natural language without having to translate the problems into an unfamiliar formal notation.

In section 2 we introduce 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 into ACE. In section 5 we reformulate the data base problem in ACE and show its automatic translation into the input language PRQ of EP Tableaux. In section 6 we conclude and point to some open issues.

2. The Model Generation Method EP Tableaux

In several application domains — 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 [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 seeing databases as models of finite sets of first-order formulae that express either static integrity constraints, views, updates, or dynamic integrity constraints [4].

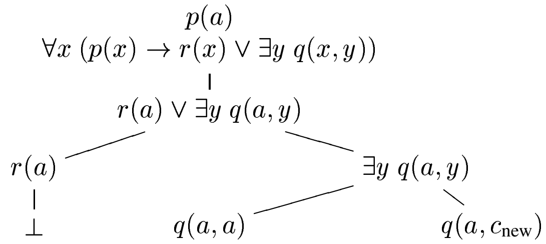
With this formalization, the question whether static integrity constraints are correctly designed in the sense that there exist databases enforcing them can be expressed as a *finite satisfiability* problem. Therefore a model generation method to be applied should not only be complete for unsatisfiability but also for finite satisfiability.

In [3] 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. The method performs a systematic search for models of the considered formulae in the fashion introduced by the *tableaux methods* [8]. The EP tableaux method proceeds by decomposing the considered formulae until only literals remain. Branches in the expanded EP tableau that do not contain the formula $\perp$ represent a model of the given set of formulae.

The input language PRQ of EP Tableaux is a fragment of first-order logic that has the same expressive power as full first-order logic.

As an example, we give an EP tableau for $\mathcal{S}$ $=$

$\{p(a),\ \forall x(p(x) \rightarrow r(x) \vee \exists y q(x,y)),\ r(a) \rightarrow \bot\}$, where the rightmost and the middle branch represent models of $\mathcal{S}$.



The EP Tableaux method has the following properties: *soundness for unsatisfiability*, *soundness for satisfiability*, and *completeness for unsatisfiability*. Additionally, the EP Tableaux method enjoys *completeness for finite satisfiability*, i.e. if the given set of formulae has a finite minimal model (a model of $\mathcal{S}$ is *minimal* if no proper subset of the ground atoms it satisfies represents a model of $\mathcal{S}$), then this model will be generated by an application of the EP Tableaux method. Thus the EP Tableaux method is not only complete for either unsatisfiability or finite satisfiability, but for both of them. This property of EP Tableaux is essential with regard to the applications mentioned above.

The EP Tableaux method is implemented in Prolog [2].

3. A Database Example in Logic

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. Negation will be handled as implication, i.e. instead of $\neg\phi$ the formula $\phi \rightarrow \bot$ is written.

(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 e.g. the formula

(7) $employee(anne)$

is added, the set of formulae is unsatisfiable which will be detected by applying the EP Tableaux method. The reason is that for every department a manager is required who is an employee at the same time (5). Since every member of a department is working for the manager (3) and since nobody is working for her/himself (6), we obtain a contradiction. 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 like the interactive prototype SIC [2] the input formulae must conform to the PRQ 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 specialists unfamiliar with formal notations. In section 5, 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.

4. Overview of Attempto Controlled English

Attempto Controlled English (ACE) is a controlled natural language specifically constructed to write specifications [5]. 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 first-order logic. This means that users can work solely with ACE without having to take recourse to the internal logic representation. Though ACE seems perfectly natural, it is a formal language with the semantics of the underlying logic language. ACE needs to be learned. We claim, however, that learning ACE needs less effort than learning an obviously formal language. 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 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 the user how the Attempto system interprets the ACE input. This reflection effectively reinforces the learning of ACE.

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 that can be translated into a function-free subset of FOL and thus into the language PRQ. In the sequel, ACE stands for this subset.

Translating the sentences

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

yields the discourse representation structure

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

where the discourse referents `[A,B]` 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) &
employee(A) & lead(A,B)))
```

The translation also generates the paraphrase

A 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.

There have been several projects with similar aims — SAFE [1] is an earlier one, while CLARE [6] is more recent — but in most cases the subsets of English were not systematically and clearly defined, and problems like ambiguity not effectively solved.

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 the PRQ syntax. Internally, ACE sentences are first translated into FOL formulae and then converted into PRQ formulae requiring just a few systematic transformations addressed in section 3.

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

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

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 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.

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.

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 (2) without conjunction.

(4) ACE: Every employee is a member of a department.
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.
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.
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 do not occur literally in the logical formula; they 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.

Taking the above PRQ formulae as input to the EP Tableaux method we can reproduce the results of the original formulation in section 3. This shows that the difficult and unfamiliar formal statement of the database example can indeed be replaced by a more natural formulation without losing precision.

6. Conclusion

We have shown that Attempto Controlled English (ACE) can replace first-order logic as input language to the model-generator EP Tableaux.

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 [7]. We rephrased the original natural language version unambiguously in ACE and then successfully proved the conclusions with EP Tableaux — thus challenging Stickel's [7] warning of "the danger of using natural language to try to convey problem statements unambiguously".

Both examples support our claim that domain specialists who may not be familiar with formal methods can formulate their problems in controlled natural language, and can then verify and validate them with the help of tools like EP Tableaux.

Though we demonstrated the viability and flexibility of our approach, there is room for improvement. Concerning EP Tableaux, the limitation to function-free subsets of first-order logic should be removed. Introducing types into ACE and PRQ would allow users to express problems more naturally and more concisely. Furthermore, it should be noted that ACE is currently being extended to achieve greater expressivity.

Acknowledgements

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

References

[1] R. M. Balzer. A 15 year perspective on automatic programming. *IEEE Transactions Software Engineering*, 11(11), 1985.

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

[5] 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, 8th International Workshop LOPSTR '98*, LNCS 1559. Springer-Verlag, 1999.

[6] B. Macias and S. G. Pulman. A method for controlling the production of specifications in natural language. *The Computer Journal*, 38(4), 1995.

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

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