Reasoning in Attempto Controlled English

Norbert E. Fuchs, Uta Schwertel

Institut für Informatik, Universität Zürich
{fuchs, uschwert}@ifi.unizh.ch
http://www.ifi.unizh.ch/attempto

Abstract. Attempto Controlled English (ACE) – a subset of English that can be unambiguously translated into first-order logic – is a knowledge representation language. To support automatic reasoning in ACE we have developed the Attempto Reasoner RACE (Reasoning in ACE). RACE proves that one ACE text is the logical consequence of another one, and gives a justification for the proof in ACE. Variations of the basic proof procedure permit query answering and consistency checking. Reasoning in RACE is supported by auxiliary first-order axioms and by evaluable functions. The current implementation of RACE is based on the model generator Satchmo.

1 Reasoning in Natural Language

Knowledge representation requires a language suited to the problem domain investigated. Traditional candidates for knowledge representation languages are natural language and formal languages. Discussing the pros and cons of natural language versus formal languages one could easily overlook that natural language is not on a par with these other languages, but plays an important and privileged role. First, it is the prototypical means of human communication, and also offers itself as a user-friendly means to interact with computers. Second, it serves as the meta-language for all other languages, informal or formal ones. Third, natural language effectively serves as its own meta-language, thus supporting representation, explanation, argumentation, and analysis all in one and the same notation. Fourth, natural language needs no extra learning effort, and – provided we exercise some care to avoid vagueness and ambiguity – is easy to use and to understand.

Some researchers go as far as to consider natural language "the ultimate knowledge representation language" [18]. Arguably, natural language also has a great potential for semantic web applications. This potential will be explored by the EU Network of Excellence "Reasoning on the Web with Rules and Semantics (REWERSE)".

Likewise, we use natural language to perform common sense reasoning that involves logical inference operations like deduction, abduction, and induction. Knowing that

```
Every company that buys a machine gets a discount.
Hardware Corporation is a company and buys a machine.
```

we deduce that

```
Hardware Corporation gets a discount.
```

Natural language and reasoning in natural language have concerned people since ancient times. In more recent years researchers have increasingly employed computers to investigate the potential of natural language for knowledge representation and its suitability for automated reasoning [9, 20]. The systems described in these publications usually process unrestricted natural language, typically do not use first-order logic but richer representations, and try to perform "one-step" inferences by introducing a large number of specialised inference rules that should closely mimic informal human reasoning in natural language.

These approaches have two major drawbacks. First, the large number of inference rules – requiring a combination of forward and backward reasoning – makes it hard to find an effective and efficient inference strategy, and can lead to a combinatorial explosion of inferences. Second, each new language construct can introduce additional inference rules threatening to further aggravate the inference process. Arguably, these objections apply to all approaches that perform one-step inferences – inferences that in first-order predicate logic would require several elementary steps.

Here we suggest an alternative approach to using natural language for knowledge representation and reasoning. Our approach differs in three essential aspects from the approaches mentioned above.

First, realising that we cannot hope to process full natural language on a computer, we use a tractable subset of English called Attempto Controlled English (ACE) – where tractable refers to both parsing and reasoning.

Second, ACE texts are translated into first-order logic. Our conviction that first-order logic is the adequate tool for our purposes is precisely expressed by "...besides expressive power, first-order logic has the best-defined, least problematic model theory and proof theory ..." [18] . Further support for the use of first-order logic in the context of natural language processing can be found in [8, 2].

Third, to show that a set T of theorems expressed in ACE is the logical consequence of a set A of ACE axioms we automatically translate A and T into their equivalent first-order representations LA and LT, try to deduce LT from LA with the help of a standard first-order theorem prover, and then report the success or failure of the proof – together with a justification – again on the level of ACE.

The advantages of our approach are twofold. First, we can rely on the correctness, completeness and efficiency of first-order theorem provers and model generators available off-the-shelf. Second, adding language constructs to ACE will not affect in any way the inference rules and the inference strategy used on the logical level. However, as will be seen in the sequel some language constructs of ACE require auxiliary first-order axioms that necessarily enlarge the search space for inferences.

In section 2 we briefly describe Attempto Controlled English (ACE) and its translation into first-order logic. In section 3 we list and motivate our requirements for the Attempto Reasoner RACE (Reasoning in ACE), in section 4 we present our candidate Satchmo as the basis of RACE, and in section 5 we sketch how we satisfy the requirements for RACE. Section 6 presents RACE's basic functionality, while section 7 demonstrates RACE's application to the notoriously difficult processing of plurals using auxiliary first-order axioms and evaluable functions. In section 8 we briefly discuss the performance of RACE. Finally, in section 9 we conclude and point to open issues and further research.

2 Attempto Controlled English

Information on the language Attempto Controlled English (ACE) in particular, and on the project Attempto in general, can be found at the Attempto web-site [www.ifi.unizh.ch/attempto]. Here, we briefly recall ACE's main features.

ACE is a controlled subset of standard English that allows users to express technical texts precisely, and in the terms of the respective application domain. ACE texts are computer-processable and can be unambiguously translated into first-order logic. Concretely, ACE is equivalent to the subset of closed sentences of FOL. ACE appears perfectly natural, but – being a controlled subset of English – is in fact a formal language with the semantics of the underlying first-order logic representation. It is exactly this logic underpinning that allows us to reason in ACE.

The Attempto system and Attempto Controlled English are intended for domain specialists – e.g. engineers, economists, physicians – who want to use formal notations and formal methods, but may not be familiar with them. Thus the Attempto system has been designed in a way that allows users to work solely on the level of ACE without having to take recourse to its internal logic representation.

Here is an ACE text from the example domain used in the sequel.

```
Every company that buys a standard machine gets a discount.
A British company buys a standard machine.
```

The Attempto Parsing Engine APE translates this ACE text unambiguously into the following discourse representation structure [10]:

```
drs([A,B,C,D,E],[drs([F,G,H,I,J],[structure(G,atomic),
quantity(G,cardinality,count_unit,F,eq,1),
object(G,company),structure(I,atomic),
quantity(I,cardinality,count_unit,H,eq,1),
property(I,standard),object(I,machine),
predicate(J,event,buy,G,I)])=>drs([K,L,M],[structure(L,atomic),
quantity(L,cardinality,count_unit,K,eq,1),
object(L,discount),predicate(M,event,get,G,L)]),
structure(B,atomic),quantity(B,cardinality,count_unit,A,eq,1),
property(B,'British'),object(B,company),
structure(D,atomic),quantity(D,cardinality,count_unit,C,eq,1),
property(D,standard),object(D,machine),
predicate(E,event,buy,B,D)])
```

The discourse representation structure `drs/2` uses a syntactic variant of the language of first-order predicate logic. The first argument of `drs/2` is a list of discourse referents, i.e. quantified variables naming objects of the domain of discourse. In our example the discourse referents `A`, `B`, `C`, `D`, `E`, `K`, `L`, `M` are existentially quantified, and `F`, `G`, `H`, `I`, `J` being introduced in the precondition of an implication are universally quantified. The second argument of `drs/2` is a list of simple and complex conditions for the discourse referents. The list separator '`,`' stands for logical conjunction. Simple conditions are logical atoms, while complex conditions are built from other discourse representation structures with the help of the logical connectors negation '`-`', disjunction '`v`', and implication '`=>`'.

Logical atoms are formed from a small set of predefined predicates like `object/2`, `property/2`, or `predicate/5`. For example, instead of the usual `company(D)`, we reify the relation `company`, and write `object(D,company)`. This 'flat notation' allows us to quantify over the arguments of the predefined predicates and thus to express general aspects of relations in first-order axioms that otherwise would require higher-order logic [8].

The discourse representation structure gets a model-theoretic semantics [10] that assigns an unambiguous meaning to the ACE text from which it was derived. Thus the Attempto system treats every ACE sentence as unambiguous, even if people not familiar with ACE would perceive the same sentence as ambiguous with respect to full English.

3 Requirements for the Attempto Theorem Prover

For the Attempto Reasoner (RACE) we determined a number of requirements most of which take our decision for granted to base RACE on first-order logic. Some requirements reflect the particular needs of the typical users of the Attempto system, that is to say domain specialists who may be unfamiliar with logic and theorem proving. Other requirements concern complementing ACE as RACE's main input language by alternative notations. Still other requirements refer to the efficient and flexible implementation of RACE.

Input and output of RACE should be in Attempto Controlled English. To accommodate the needs of the typical user of the Attempto system the input and output of RACE should be in ACE. Alternative forms of input should be available for users familiar with first-order logic.

RACE should generate all proofs. If an ACE text is unsatisfiable, RACE should generate all minimal unsatisfiable subsets of sentences of the text, i.e. sets of sentences that are unsatisfiable and all of whose strict subsets are satisfiable.

There can be several proofs of ACE theorems from ACE axioms, specifically several answers to ACE queries. Furthermore, an ACE text can have several inconsistent subsets. Users should be given the option to get all results.

RACE should give a justification of a proof. RACE should provide a justification of a successful proof, either as a trace of the proof or as a report which minimal subset of the axioms was used to prove the theorems. Especially the second alternative is of utmost practical relevance if the ACE text is a software specification and users want to trace requirements.

RACE should combine theorem proving with model generation. If an ACE text is satisfiable, RACE should generate a minimal finite model if there is one.

Theorem provers and model generators complement each other. If a problem is unsatisfiable a theorem prover will find a proof. If, however, the problem is satisfiable and admits finite models then a model generator will find a finite model. Finally, if the problem is satisfiable, but does only have infinite models, we can encounter non-termination for both theorem provers and model generators.

Besides complementing theorem provers, model generators generating (minimal) finite models offer additional advantages [3], foremost the possibility to construct comprehensive answers to queries.

RACE should allow for the definition of auxiliary axioms in first-order logic. ACE is primarily a language to express domain knowledge. However, deductions in ACE presuppose domain-independent knowledge, for instance general linguistic knowledge like the relations between plural and singular nouns, or mathematical knowledge like comparisons between natural numbers. This domain-independent knowledge is best expressed in auxiliary axioms using the language of first-order logic. Users may even prefer to state some domain knowledge, for instance ontologies, in first-order axioms instead of in ACE. In still other cases users may want to state something in first-order logic that cannot yet be conveniently expressed in ACE.

RACE should have an interface to evaluable functions and predicates. Auxiliary first-order axioms, but also ACE texts can refer to functions or predicates, for instance to arithmetic functions or Boolean predicates. Instead of letting users define these functions and predicates, it is much more convenient and certainly more efficient to use the evaluable functions and predicates that are provided by the execution environment.

Using RACE should not presuppose detailed knowledge of its workings. Many theorem provers allow users to control proofs through options and parameters. Often these options and parameters presuppose detailed knowledge of the structure of the problem, of the internal working of a theorem prover, or of theorem proving in general, that a typical user of the Attempto system may not have. Thus, RACE should preferably run automatically, and at most expect users to set familiar parameters like a runtime limit, or the number of solutions found.

4 A Basis for the Attempto Reasoner

Many first-order theorem provers and model generators are freely available off-the-shelf. Since these tools have already reached a high level of maturity, we decided to base RACE on one of them.

Since RACE's requirements imply extensions and possibly modifications of the selected tool we wanted to have the tool locally available. This decision precludes solutions like MathWeb [www.mathweb.org] that farms out an inference task simultaneously to theorem provers and model generators available on the internet and then uses the first result returned. However, this competitive parallelism that leads to super-linear speed-ups can also be implemented as parallel processes on one single machine [21].

After an extensive and necessarily subjective evaluation of candidates – involving deduction rules for discourse representation structures [15], leanT^AP [1], EP Tableaux [4, 22], Otter [13] and Mace [14] – we decided to base the implementation of RACE on the model generator Satchmo [12, 5].

Satchmo accepts first-order clauses in implication form, or Horn clauses in Prolog notation. Negation is expressed as implication to false.

If the set of clauses admits a finite model, Satchmo will find it. Satchmo is correct for unsatisfiability if the clauses are range-restricted – which can always be achieved – and complete for unsatisfiability if used in level-saturation manner – technically achieved with the help of Prolog's set predicates [5].

Satchmo is highly efficient since it delegates as much as possible to the underlying Prolog inference engine.

5 Fulfilling the Requirements for the Attempto Reasoner

RACE consists of a set of Prolog programs with an extended version of Satchmo at its core. Some of RACE's requirements are already fulfilled by Satchmo, while others are satisfied by RACE's Prolog code making use of Satchmo's basic functionality and of special features of the logical representation of ACE texts.

Input and output of RACE should be in Attempto Controlled English. RACE proves that ACE theorems T are the logical consequence of ACE axioms A, translating the ACE texts T and A into Satchmo clauses CT and CA, showing that $CA \cup \neg CT$ have no model, and then reporting the result of the proof $A \vdash T$ using the original texts T and A.

RACE should generate all proofs. As a model generator Satchmo searches for a model of a set of clauses. However, if the set is unsatisfiable, Satchmo will stop immediately once it detected unsatisfiability. The requirement to generate all proofs amounts to finding not just the first, but all cases of unsatisfiability. We have extended Satchmo so that it will find all minimal unsatisfiable subsets of the clauses, and thus all minimal unsatisfiable subsets of the ACE sentences from which the clauses were derived.

RACE should give a justification of a proof. RACE generates for each proof a report which minimal subset of the axioms was used to prove the theorems. The implementation of this feature relies on an extended internal representation of an ACE text called a paragraph. The example

```
Every company that buys a standard machine gets a discount.
A British company buys a standard machine.
```

of section 2 is actually translated into

```
paragraph(drs([A,B,C,D,E],[drs([F,G,H,I,J],[structure(G,atomic)-1,
quantity(G,cardinality,count_unit,F,eq,1)-1,object(G,company)-1,
structure(I,atomic)-1,quantity(I,cardinality,H,count_unit,H,eq,1)-1,
property(I,standard)-1,object(I,machine)-1,
predicate(J,event,buy,G,I)-1])=>drs([K,L,M],[structure(L,atomic)-1,
quantity(L,cardinality,count_unit,K,eq,1)-1,object(L,discount)-1,
predicate(M,event,get,G,L)-1]),structure(B,atomic)-2,
quantity(B,cardinality,count_unit,A,eq,1)-2,property(B,'British')-2,
object(B,company)-2,structure(D,atomic)-2,
quantity(D,cardinality,count_unit,C,eq,1)-2,property(D,standard)-2,
object(D,machine)-2,predicate(E,event,buy,B,D)-2]),text(['Every
company that buys a standard machine gets a discount.','A
British company buys a standard machine.']))
```

where `drs/2` is a slightly extended version of the discourse representation structure discussed in section 2. The structure `text/1` contains a list whose elements are the input sentences represented as character strings. A logical atom `Atom` occurring in `drs/2` is actually written as `Atom-I` where the index `I` refers to the `I`'th element of the list in `text/1`, i.e. to the sentence from which `Atom` was derived.

The discourse representation structure is translated into Satchmo clauses of the general form

```
satchmo_clause(Body,Head,Index)
```

where `Body`, respectively `Head`, are the body and head of the Satchmo clause, and Index is either `axiom(I)` or `theorem(I)` indicating that the clause was derived from the `I`'th ACE axiom or from the `I`'th ACE theorem.

During a proof RACE collects the indices of atoms participating in a proof in a sorted list. There is one list for each proof. These lists are then used to generate reports showing which ACE axioms were used to derive which ACE theorems.

RACE should combine theorem proving with model generation. Satchmo, and consequently RACE, can be used both as a theorem prover and a model generator. If a set of Satchmo clauses is satisfiable and admits a finite model then RACE will generate a minimal finite model that is returned as a list of ground instances of atoms.

RACE should allow for the definition of auxiliary axioms in first-order logic. RACE accepts auxiliary first-order axioms of the form

```
fol_axiom(Number,Formula,Text)
```

where `Number` labels the axiom, `Formula` is a closed first-order formula, and `Text` is a string describing the axiom. All auxiliary axioms are conjoined with the first-order formula derived from the ACE axioms, and the conjunction is translated into Satchmo clauses. While an auxiliary axiom is processed the atoms `A` of its `Formula` are changed into `A - fol_axiom(Number)` so that the `Text` of the auxiliary axiom becomes accessible to the justification of the proof via the index `fol_axiom(Number)`.

RACE should have an interface to evaluable functions and predicates. Satchmo accepts and executes any Prolog predicate – be it user defined or built-in.

Using RACE should not presuppose detailed knowledge of its workings. Satchmo does not offer any options or parameters, nor can users interact with Satchmo.

6 Using the Attempto Reasoner

In its basic form RACE proves that one ACE text – the theorems – follows logically from another ACE text – the axioms – by showing that the conjunction of the axioms and the negated theorems leads to a contradiction. Variations of this basic proof procedure allow users to check the consistency of an ACE text, or to answer queries expressed in ACE. These two forms of deduction are especially interesting for the analysis of specifications written in ACE, for instance for their validation with respect to requirements.

The following examples are deliberately simple to clearly demonstrate the basic usage of RACE. We will only show the ACE input and output of RACE and omit the internal logical representation into which the ACE input is transformed by the Attempto parser before being fed to RACE.

Given the three ACE axioms

```
Every company that buys a standard machine gets a discount.
A British company buys a standard machine.
A French company buys a special machine.
```

and the ACE theorem

```
A company gets a discount.
```

RACE will prove the theorem and generate the following output

```
RACE proved that the sentence(s)
  A company gets a discount.
can be deduced from the sentence(s)
  Every company that buys a standard machine gets a discount.
  A British company buys a standard machine.
```

Note that since RACE generates minimal unsatisfiable subsets, we only see the two axioms actually used in the proof.

Given the same three axioms and the ACE query

```
Who buys a machine?
```

RACE generates the two answers

```
RACE proved that the query (-ies)
  Who buys a machine?
can be answered on the basis of the sentence(s)
  A British company buys a standard machine.

RACE proved that the query (-ies)
  Who buys a machine?
can be answered on the basis of the sentence(s)
  A French company buys a special machine.
```

All possible answers are generated, and for each answer we see only the ACE axiom(s) used to derive that answer.

Similarly we can check the consistency of an ACE text. If the text is inconsistent, RACE will identify all minimal unsatisfiable subsets. Given the ACE text

```
Every company that buys a standard machine gets a discount.
A British company buys a standard machine.
A French company buys a standard machine.
There is no company that gets a discount.
```

we get the two results

```
RACE proved that the sentence(s)
  Every company that buys a standard machine gets a discount.
  A French company buys a standard machine.
  There is no company that gets a discount.
are inconsistent.

RACE proved that the sentence(s)
  Every company that buys a standard machine gets a discount.
  A British company buys a standard machine.
  There is no company that gets a discount.
are inconsistent.
```

showing that the text contains two inconsistent subsets.

The preceding examples demonstrated the basic usage of RACE. More advanced applications making use of auxiliary first-order axioms and evaluable functions will be shown in the next section.

7 Plural Inferences via Auxiliary First-Order Axioms

Plural constructions in natural language raise hard linguistic and semantic problems [17], and trigger complex inferences. There are for example plural disambiguation processes activated by world-knowledge, inferences induced by linguistic knowledge about the structure and interpretation of plurals, or inferences that are driven by mathematical knowledge. Linguistic and mathematical inferences can be modelled by extending RACE with auxiliary domain-independent first-order axioms for lattice-theory, equality and integer arithmetic. The examples used in this section may give the impression of being simple, and are in fact easily solved by human beings. However, as will become apparent, they are not at all trivial when solved by a computer.

Lattice Theoretic Axioms. The representation and processing of natural language plurals in first-order logic requires additional axioms describing the properties of plural entities. For the practical implementation we had to settle with an axiom system that provides a good trade-off between empirical adequacy and computational tractability. For conciseness we can only show a selection of the axioms that were implemented in RACE.

From the two ACE sentences

```
Every company that buys a machine gets a discount.
Six Swiss companies each buy a machine.
```

where the second sentence contains a plural construction we want to infer the singular sentence

```
A company gets a discount.
```

To perform this inference we need to deduce from the second sentence the existence of a company that buys a machine. The logical representation of the second sentence is

```
paragraph(
drs([A,B],[structure(B,group)-2,
drs([C],[structure(C,atomic)-2,part_of(C,B)-2])=>
drs([],[object(C,company)-2,property(C,'Swiss')-2]),
quantity(B,cardinality,count_unit,A,eq,6)-2,
drs([D],[structure(D,atomic)-2,part_of(D,B)-2])=>
drs([E,F,G],[structure(F,atomic)-2,object(F,machine)-2,
quantity(F,cardinality,count_unit,E,eq,1)-2,
predicate(G,event,buy,D,F)-2])]),
text(['...','Six Swiss companies each buy a machine.']))
```

This representation assumes a lattice-theoretic structure of the domain of discourse partially ordered by the relation `part_of/2`. Additionally it is assumed that for any subset S of the domain there exists a unique least upper bound (supremum) of the elements of S with respect to `part_of/2`. Thus, apart from atomic individuals (atoms) there are complex individuals (groups) formed by the supremum operation which serve as the denotation of plural nouns. This lattice-theoretic approach allows for a first-order treatment of plural objects [11]. In the above representation each object variable is typed according to its position in the lattice. Lines 2–4 of the structure express that there is a group `B` the atomic parts of which are Swiss companies, the fifth line that the cardinality `A` of `B` equals 6, and lines 6–9 express the distributive reading triggered by the cue word `each`.

Since from this representation the existence of a company that buys a machine cannot be directly deduced we add to RACE the following auxiliary first-order lattice-theoretic axiom stating that each group consists of atomic parts:

```
fol_axiom(1,forall([X],(structure(X,group) =>
exists([Y],(part_of(Y,X) & structure(Y,atomic))))),
‘Every group consists of atomic parts.’).
```

Note that the axiom is not domain specific since it models the meaning of plurals in natural language. Hence the axiom has to be available for each proof in each domain. Calling RACE with the conjunction of the clauses derived from the ACE text and from the auxiliary axiom we get the desired deduction and RACE outputs

```
RACE proved that the sentence(s)
  A company gets a discount.
can be deduced from the sentence(s)
  Every company that buys a machine gets a discount.
  Six Swiss companies each buy a machine.
using the auxiliary axiom(s)
  Every group consists of atomic parts.
```

RACE includes other lattice-theoretic axioms, e.g. the reflexivity, transitivity and antisymmetry of the part-of relation, the proper-part-of relation, or an axiom that states that atoms do not have proper parts. Commutativity, associativity and idempotence of the lattice-theoretic join operation – needed for the representation of noun phrase coordination – are not directly implemented via first-order axioms but more efficiently simulated by list processing operations like permutation.

Equality. Many inferences require the interaction of several auxiliary axioms whereby equality plays an important role. Asking the query

```
Who buys machines?
```

we expect to retrieve the second sentence

```
Six Swiss companies each buy a machine.
```

of the above example since the bare plural `machines` in the query is indeterminate as to whether one or more machines are bought. To model this we represent both the query word `who` and the bare plural `machines` as underspecified with respect to the position in the lattice (`structure(V,dom)`). The query representation is

```
paragraph(
drs([A,B,C,D],[structure(A,dom)-1,query(A,who)-1,
structure(C,dom)-1,quantity(C,cardinality,count_unit,B,geq,1)-1,
drs([E],[structure(E,atomic)-1,part_of(E,C)-1])=>
drs([],[object(E,machine)-1]),predicate(D,event,buy,A,C)-1]),
text([‘Who buys machines?’])).
```

Using the auxiliary axiom 1 introduced above and additionally the following three auxiliary axioms

```
fol_axiom(2,forall([X],(structure(X,atomic) => structure(X,dom))),
‘Atom is a subtype of the general type dom.’)

fol_axiom(3,forall([X,Y],(structure(X,atomic) & part_of(Y,X) =>
is_equal(X,Y))), ‘Atoms do not have proper parts.’)

fol_axiom(4,forall([X,Y,P],(is_equal(X,Y) & object(X,P) =>
object(Y,P))), ‘Identical objects have the same properties.’)
```

will licence the deduction of the query from the second sentence. The relation `is_equal/2` models equality and is defined as reflexive, symmetric and transitive via other auxiliary axioms. Equality substitution axioms like the fourth axiom can be formalised directly in first-order logic due to our flat notation. Defining equality in this way may seem naïve, but since Satchmo does not provide methods like paramodulation or demodulation there is no alternative.

Mathematical Axioms. Assume the slightly modified ACE text

```
Every company that buys at least three machines gets a discount.
Six Swiss companies each buy one machine.
A German company buys four machines.
```

Answering the query

```
Who gets a discount?
```

needs mathematical knowledge about natural numbers.

In RACE mathematical knowledge about natural numbers can be straightforwardly implemented by triggering the execution of Prolog predicates during the proof. For the current example we need the user-defined predicate

```
quantity(_A,_Dimension,_Unit,Cardinality,geq,NewNumber):-
  number(NewNumber),
  quantity(_A,_Dimension,_Unit,Cardinality,eq,GivenNumber),
  number(GivenNumber),
  NewNumber =< GivenNumber.
```

With this predicate it can be proved that an object has a `Cardinality` greater or equal to `NewNumber` (here 3) if that object has a `Cardinality` that equals `GivenNumber` (here 4) and if `NewNumber` is less or equal than `GivenNumber`. Instantiation problems – that we encountered when working with the theorem prover Otter – can be easily avoided by the Prolog predicate `number/1`.

Domain-Specific Axioms. Even domain-specific knowledge – for instance ontologies – could analogously be formalised as auxiliary first-order axioms, although the formulation in ACE is preferable.

8 Performance of RACE

Since RACE is an extended and modified version of Satchmo, we will compare its performance with that of the original version of Satchmo. To put the performance figures into the correct perspective, some comments are in order, though.

If a set of clauses is unsatisfiable, Satchmo will stop immediately once it detects the first inconsistency. Since RACE generates all minimal unsatisfiable subsets it has to cope with a much larger search space that must be traversed exhaustively. This affects RACE's performance negatively.

Satchmo owes much of its efficiency to the underlying Prolog inference engine. While implementing RACE we have tried to preserve as much as possible of Satchmo's original code. However, to implement RACE's additional functionality we had to operate on a meta-level that necessarily introduces a performance loss.

Satchmo has one source of inefficiency, though. While trying to find a model, Satchmo again and again checks the same clauses for (un-) satisfiability. RACE eliminates some of this inefficiency by pruning all clauses `satchmo_clause(true,Head,Index)` from the Prolog data base once they have been used. Furthermore, RACE uses a simple, but effective algorithm [16] to select relevant clauses for the (un-) satisfiability check. Other authors have proposed alternative algorithms to identify relevant clauses. A recent proposal for such an algorithm, and references to older ones, can be found in [7].

Since the original version of Satchmo used Schubert's steamroller [19] as one of its examples, we will report here RACE's performance figures for the steamroller.

We compared the original version of Satchmo with two versions of RACE, one that generates all solutions and a modified version that stops after the first solution. We used these three versions with the standard representation of the steamroller and contrasted it to the flat Attempto representation introduced in section 2. The following times were measured on a Macintosh 500 MHz G4 running under Mac OS X 10.2.6 and using SICStus Prolog 3.10.

Representation	Standard	Attempto
Satchmo (original)	15 ms	2050 ms
RACE (all)	230 ms	2100 ms
RACE (first only)	70 ms	990 ms

The runtimes for the Attempto representation may seem excessive when compared to the runtimes of the standard representation of the steamroller. However, we have to realise that this is a consequence of the much richer first-order language necessary to adequately represent aspects of natural language, for instance verb phrase modification and plural.

Furthermore, the comparison between the original Satchmo and RACE (all) is not completely fair since Satchmo finds the single solution of the steamroller and then stops, while RACE – after finding this solution – searches for alternative solutions. Thus we slightly modified RACE to stop after the first solution, and got the results listed for RACE (first only).

A more thorough investigation of RACE's performance remains to be done.

9 Conclusions

The Attempto Reasoner (RACE) proves that one text in Attempto Controlled English (ACE) can be deduced from another one and gives a justification for the proof in ACE. Variations of the basic proof procedure permit query answering and consistency checking. Extending RACE by auxiliary first-order axioms and evaluable functions and predicates we can perform complex deductions on sentences with plurals and numbers.

Though small, the examples of this paper already exhibit the practicality and potential of our approach. Much more remains to be done, though, besides investigating the scaling up of RACE. To support the analysis of ACE specifications hypothetical reasoning ('What happens if . . . ?'), abductive reasoning ('Under which conditions does . . . occur?'), and the execution of ACE specifications [6] using ACE scenarios would be helpful. These and other problems will be investigated within the EU Network of Excellence REWERSE.

Acknowledgements

We would like to thank Johan Bos, François Bry, Stefan Höfler, Michael Kohlhase, William McCune, Rolf Schwitter, John Sowa and Sunna Torge for fruitful discussions. We also thank the anonymous reviewers of a first draft of this paper for their constructive comments. Our research was partially funded by grant 2000-061932.00 of the Swiss National Science Foundation.

References

[1] B. Beckert and J. Posegga. leanT^AP: Lean, Tableau-Based Deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995.

[2] P. Blackburn and J. Bos. Computational Semantics. *Theoria*, 18(1):27–45, 2003.

[3] J. Bos. Model Building for Natural Language Understanding, ms., 2001. Draft at www.iccs.informatics.ed.ac.uk/∼jbos.

[4] F. Bry and S. Torge. A Deduction Method Complete for Refutation and Finite Satisfiability. In *Proc. 6th European Workshop on Logics in Artificial Intelligence*, volume 1489 of *Lecture Notes in Artificial Intelligence*. Springer, 1998.

[5] F. Bry and A. H. Yahya. Positive Unit Hyperresolution Tableaux and Their Application to Minimal Model Generation. *Journal of Automated Reasoning*, 25(1):35–82, 2000.

[6] N. E. Fuchs. Specifications Are (Preferably) Executable. *Software Engineering Journal*, 7(5):323–334, 1992. Reprinted in: J. P. Bowen, M. G. Hinchey, *High-Integrity System Specification and Design*, Springer-Verlag London, 1999.

[7] L. F. He. UNSATCHMO: Unsatisfiability Checking by Model Rejection. In R. Goré, A. Leitsch, and T. Nipkov, editors, *International Joint Conference on Automated Reasoning IJCAR 2001, Short Papers*, pages 64–75, Siena, Italy, 2001.

[8] J. R. Hobbs. Ontological Promiscuity. In *23rd Annual Meeting of the ACL*, pages 61–69, University of Chicago, Illinois, 1985.

[9] L. M. Iwanska and S. C. Shapiro, editors. *Natural Language Processing and Knowledge Representation.* AAAI Press/MIT Press, Menlo Park, CA, 2000.

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

[11] G. Link. The Logical Analysis of Plurals and Mass Terms: A Lattice-Theoretical Approach. In R. Bäuerle, C. Schwarze, and A. von Stechow, editors, *Meaning, Use, and Interpretation of Language*, pages 302–323. de Gruyter, Berlin, 1983.

[12] R. Manthey and F. Bry. SATCHMO: A Theorem Prover Implemented in Prolog. In E. L. Lusk and R. A. Overbeek, editors, *Proc. CADE 88, Ninth International Conference on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 415–434. Springer, Argonne, Illinois, 1988.

[13] W. W. McCune. Otter 3.0 Reference Manual and Guide. Technical Report ANL-94/6, Argonne National Laboratory, 1994.

[14] W. W. McCune. Mace 2.0 Reference Manual and Guide. Technical Memorandum ANL/MCS-TM-249, Argonne National Laboratory, 2001.

[15] U. Reyle and D. M. Gabbay. Direct Deductive Computation on Discourse Representation Structures. *Linguistics and Philosophy*, 17:343–390, 1994.

[16] H. Schütz and T. Geisler. Efficient Model Generation through Compilation. *Information and Computation*, 162(1-2):138–157, 2000.

[17] U. Schwertel. Controlling Plural Ambiguities in Attempto Controlled English. In *3rd International Workshop on Controlled Language Applications (CLAW)*, Seattle, Washington, 2000.

[18] J. F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations.* Brooks Cole Publishing Co., Pacific Grove, CA, 2000.

[19] M. E. Stickel. Schubert's Steamroller Problem: Formulation and Solutions. *Journal of Automated Reasoning*, 2(1):89–101, 1986.

[20] J. Sukkarieh. Quasi-NL Knowledge Representation for Structurally-Based Inferences. In *3rd Workshop on Inference in Computational (ICoS-3)*, Siena, Italy, 2001a.

[21] G. Suttcliffe and C. Suttner. Results of the CADE-13 ATP System Competition. *Journal Automated Reasoning*, 18(2):259–264, 1997.

[22] S. Torge. *Überprüfung der Erfüllbarkeit im Endlichen: Ein Verfahren und seine Anwendung.* Ph.D. thesis, University of Munich, 1998.