Reasoning in Attempto Controlled English

Norbert E. Fuchs, Uta Schwertel
Institut für Informatik, Universität Zürich
(fuchs, uschwert)@ifi.unizh.ch
www.ifi.unizh.ch

Abstract

Attempto Controlled English (ACE) – a subset of English that can be unambiguously translated into first-order logic – is a software specification and 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. Variations of the basic proof procedure permit query answering and consistency checking. Extending RACE by auxiliary first-order axioms and by evaluable functions we can perform complex deductions on ACE texts containing plurals and numbers. The implementation of RACE is based on the theorem prover Otter and on the model generator Satchmo, both available off-the-shelf.

1 Reasoning in Natural Language

Discussing the pros and cons of natural language versus other languages for specification or knowledge representation one could easily overlook that natural language is not on a par with these other languages, but plays two important and privileged roles. First, it is the prototypical means of human communication. Second, it serves as metalanguage for all other languages, informal or formal ones; even the discussion of the pros and cons of specification and knowledge representation languages takes place in natural language. Some researchers consider natural language "the ultimate knowledge representation language" since "after thousands of years of evolution in intimate contact with every aspect of human experience [it] has attained a greater flexibility and expressive power than any artificial language" [Sowa 2000]. Natural language effectively serves as its own meta-language, thus supporting representation, explanation, argumentation, and analysis all in one and the same notation. 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.

We also 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 can deduce that

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

Natural language and reasoning in natural language have fascinated 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. A monograph on the state of the art in this field [Iwanska & Shapiro 2000] describes among others the UNO Model by Iwanska, the Montagovian Syntax by McAllester & Givan, KRISP by McDonald, Episodic Logic by Schubert & Huang, and SnePS by Shapiro.

These systems tackle unrestricted natural language, typically do not use first-order logic but richer alternatives, and try to perform "one-step" inferences by introducing specialised inference rules that should closely mimic informal human reasoning in natural language.

The Montagovian Syntax of McAllester & Givan [McAllester & Givan 1992] is the system that Iwanska & Shapiro [Iwanska & Shapiro 2000, p. 3] consider closest to

natural language as knowledge representation and reasoning language. In the Montagovian Syntax the sentence

```
Every dog ate a bone.
```

is represented as the quantifier-free formula

```
(every dog (ate (some bone)))
```

whose denotation is true if the set of dogs is a subset of the set of individuals each of which has the relation denoted by `ate` to some member of the set of bones. McAllester & Givan introduced a tractable decision procedure based on a set of 16 inference rules for quantifier-free formulas. Here is an example of an inference rule

```
(at-most-one t), (every r t)
_____________________________

        (at-most-one r)
```

where `r` and `t` range over class expressions denoting sets.

Sukkarieh [Sukkarieh 2001a, Sukkarieh 2001b] extended the work of McAllester & Givan. She developed the sublange CLIP of English that is translated into a quasi-natural language knowledge representation McLogic based on Montagovian syntax. Sukkarieh performs reasoning in McLogic using a set of 34 inference rules that resemble those of McAllester & Givan.

Suggestive and attractive as the approaches of McAllester & Givan and of Sukkarieh appear, they seem to 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 unambiguously 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 …" [Sowa 2000].

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 formulas `LA` and `LT`, try to deduce `LT` from `LA` with the help of a standard first-order theorem prover or model generator, 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.

Theorem proving and model generation are increasingly being used in computational semantics (cf. www.sigsem.org) to solve problems like disambiguation, presupposition and resolution of anaphora. A recent publication in this field [Blackburn et al. 2001] discusses presupposition and contains many helpful pointers to relevant information. Our work reported here has a lot in common with computational semantics as far as the technical aspects of theorem proving and model generation are concerned

[Blackburn & Bos 2000a, Blackburn & Bos 2000b], but differs in its goals, namely to perform logical inferences on ACE texts.

In the following we briefly describe Attempto Controlled English (ACE) and its translation into first-order logic (section 2). In section 3 we list and motivate our requirements for the Attempto Reasoner RACE (Reasoning in ACE), in section 4 we present our candidates Otter and Satchmo for the basis of RACE, and in section 5 we show how we satisfy the requirements for RACE. Section 6 presents RACE's basic functionality, while section 7 demonstrates RACE's application to the processing of plurals and numbers using first-order auxiliary axioms and evaluable functions and predicates. In section 8 we describe limitations of the current implementation of RACE. Finally, we conclude and point to open issues and further research (section 9). The appendix summarises our experiences with some other theorem provers and model generators that we investigated.

2 Attempto Controlled English

Attempto Controlled English (ACE) has been described in detail elsewhere [Fuchs et al. 1998; ACE Manual 1999; Schwertel 2000]. For the convenience of the reader we here briefly recall its main features.

ACE is a controlled subset of standard English that allows users to express technical texts, e.g. specifications, precisely, and in the terms of the respective application domain. ACE texts are computer-processable and can be unambiguously translated into first-order logic. ACE appears perfectly natural, but – being a controlled subset of English – it 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, e.g. to answer queries expressed in ACE and to check the consistency of an ACE text.

The Attempto system and Attempto Controlled English are intended for domain specialists – e.g. engineers, economists, physicians – who want to use 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 system translates this ACE text unambiguously into a discourse representation structure [Kamp & Reyle 1993]

```
drs([A,B,C],[drs([D,E,F],[object(D,company),structure(D,atomic),
object(E,machine),structure(E,atomic),property(E,standard),
predicate(F,event,buy,D,E)])=>drs([G,H],[object(G,discount),
structure(G,atomic),predicate(H,event,get,D,G)]),
object(A,company),structure(A,atomic),property(A,'British'),
object(B,machine),structure(B,atomic),property(B,standard),
predicate(C,event,buy,A,B)])
```

that constitutes the internal logical representation of the text.

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`, `G`, `H` are existentially quantified, and `D`, `E`, `F` 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 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 logic that otherwise would require higher-order logic.

The discourse representation structure gets a model-theoretic semantics [Kamp & Reyle 1993] 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 would perceive the same sentence as ambiguous in full English.

3 Requirements for the Attempto Reasoner RACE

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 implementation of RACE.

Input and output of RACE should be in Attempto Controlled English.

To accommodate the needs of the typical users 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.

There can be several proofs of ACE theorems from ACE axioms, specifically several answers to ACE queries. Or, an ACE text can have several inconsistent subsets. In all these cases users will profit from getting all results. Users should be given options to delimit or select the output of RACE.

RACE should give a justification of a proof.

RACE should provide a trace of a successful proof, or report which 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.

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 internal working of a theorem prover or of theorem proving in general that not all users of the Attempto system may have. Thus, RACE should preferably run automatically, and at most expect users to set familiar parameters like a runtime limit.

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. 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.

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 efficient to make use of evaluable functions and predicates made available by the execution environment.

RACE should combine theorem proving with model generation.

If a theorem `T` is the logical consequence of axioms `A` then `A` ∧ {¬`T`} is unsatisfiable. Unsatisfiability is semi-decidable and can be detected by a correct and complete theorem prover in finite time. Finding a single contradiction – for instance the always false formula ⊥, or contradictory formulas `F` and ¬`F` – suffices to show unsatisfiability. If, however, a theorem `T` is not the logical consequence of axioms `A` then `A` ∧ {¬`T`} is satisfiable, i.e. does have a model. Satisfiability is undecidable, while finite satisfiability is semi-decidable. A correct model generator that is complete for finite satisfiability will detect satisfiability in finite time. Finding a single (finite) model suffices to show satisfiability.

Thus theorem provers and model generators complement each other. If the problem is unsatisfiable a theorem prover will find a proof while a model generator has to do an exhaustive – and possibly non-terminating – search to find out that there are no models. If the problem is satisfiable and admits finite models then a model generator will find a finite model while a theorem prover must again do an exhaustive – and potentially non-terminating – search to find that there are no contradictions. Finally, if the problem is satisfiable but does only have infinite models, we can encounter non-termination for both theorem provers and model generators – after all satisfiability is undecidable.

Besides complementing theorem provers model generators generating (minimal) finite models offer additional advantages [Bos 2001], foremost the possibility to construct comprehensive answers to queries.

4 A Basis for the Attempto Reasoner

The computational aspects of first-order logic have been intensively investigated by many researchers. To profit from the accumulated experience of these researchers we decided to base RACE on first-order theorem provers and model generators freely available off-the-shelf. While for theorem provers there is a large variety of sophisticated candidates, for model generators the choice is still limited.

Since RACE's requirements imply extensions and possibly modifications of the selected tools we wanted to have these tools locally available. This decision precludes solutions like MathWeb (cf. 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 [Sutcliffe & Suttner 1997].

After an extensive – necessarily subjective – evaluation of selected candidates (cf. Appendix) we chose to base RACE on the theorem prover Otter [McCune 1994] and on the model generator Satchmo [Manthey & Bry 1988].

Otter. McCune describes Otter the following way: "Otter is a resolution-style theorem-proving program for first-order logic with equality. Otter includes the inference rules binary resolution, hyperresolution, UR-resolution, and binary paramodulation. Some of its other abilities and features are conversion from first-order formulas to clauses, forward and back subsumption, factoring, weighting, answer literals, term ordering, forward and back demodulation, evaluable functions and predicates, and Knuth-Bendix completion."

After nearly 20 years of development Otter is very stable, and – due to its implementation in C – very efficient. For the fine-tuning of proofs Otter offers a rich set of options and parameters that can even be modified interactively by users during a proof. This fine-tuning, however, and the possibility of user-interactions during proofs prevent general statements concerning the correctness and completeness of Otter.

Otter accepts first-order clauses, or first-order formulas that are automatically translated into clauses.

Satchmo. Manthey & Bry write: "Satchmo is a theorem prover consisting of just a few and simple Prolog programs. Prolog may be used for representing problem clauses as well. Satchmo is based on a model-generation paradigm. It is refutation complete if used in a level saturation manner."

Satchmo owes its high efficiency to the power of the underlying Prolog inference engine. Satchmo accepts first-order clauses in implication form, or Horn clauses in Prolog notation. Negation is expressed as implication to `false`.

If the clauses admit a finite model, Satchmo will find it. Satchmo is correct for unsatisfiability if the input 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 [Bry & Yahya 1996].

Otter & Satchmo. Currently we are using Otter and Satchmo independently but we will later combine them into one integrated inference system.

5 Fulfilling the Requirements for the Attempto Reasoner

In this section we will only discuss technical aspects of fulfilling the requirements of RACE while examples of using RACE will be presented in the following sections.

RACE consists of a set of Prolog programs with Otter, respectively Satchmo, at its core. Some of RACE's requirements are already fulfilled by Otter, respectively Satchmo, while others are satisfied by RACE's Prolog code making use of Otter's and 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.

Otter & Satchmo. RACE proves that one ACE text `T` is the logical consequence of another ACE text `A` by translating `T` and `A` into their first-order representations `LT` and `LA`, performing `LA |– LT`, and then reporting the result of the proof using again `T` and `A`.

The Attempto parser generates discourse representation structures from ACE texts (cf. section 2). These discourse representation structures are converted into the standard form of the language of first-order logic and then into Otter or Satchmo clauses that are passed to Otter, respectively to Satchmo. Otter, respectively Satchmo, is executed, and its output is scanned for the results of the proof that are then reported to the user reusing the original ACE input.

RACE should generate all proofs.

Otter. Otter can find all proofs of a problem. It provides a parameter `max_proofs` that delimits the number of proofs Otter should try to find. If the parameter is set to -1 then Otter will find as many proofs as it can within other constraints, for instance the setting of the parameter `max_seconds` that delimits Otter's runtime.

Satchmo. The original version of Satchmo is designed to detect unsatisfiability as quickly as possible, i.e. it will only find the very first proof. We modified Satchmo's Prolog code so that Satchmo will find all proofs.

RACE should give a justification of a proof.

Otter & Satchmo. RACE generates for each proof a report which 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],[drs([D,E,F],[object(D,company)-1,structure(D,atomic)-1,
object(E,machine)-1,structure(E,atomic)-1,property(E,standard)-1,
predicate(F,event,buy,D,E)-1])=>drs([G,H],[object(G,discount)-1,
structure(G,atomic)-1,predicate(H,event,get,D,G)-1]),
object(A,company)-2,structure(A,atomic)-2,property(A,'British')-2,
object(B,machine)-2,structure(B,atomic)-2,property(B,standard)-2,
predicate(C,event,buy,A,B)-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. Logical atoms occurring in `drs/2` are 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.

During the translation of the discourse representation structure into a first-order formula the indices of the logical atoms are wrapped by `axiom/1`, respectively `theorem/1`, to indicate whether an atom was derived from the ACE axioms or from the ACE theorems.

The following steps are different in Otter and Satchmo.

Otter. During the translation of the first-order formula into a set of Otter clauses the just wrapped indices will be wrapped once more by `$ans/1` – Otter's notation for answer literals [Green 1969]. The thus generated answer literals will be added as extra disjuncts to their respective clauses. Thus we arrive at a set of clauses, that for example could look like

```
object(sk1,company)|$ans(axiom(1)).
predicate(sk2,state,be,sk1)|$ans(axiom(2)).
-object(A,company)|-predicate(B,state,be,A)|$ans(theorem(1)).
```

The added answer literals are carried through all lines of the refutation proof. Then the last line of a successful proof is not empty as usual, but consists of the answer literals of the clauses participating in the proof, namely

```
$ans(axiom(2))|$ans(theorem(1))|$ans(axiom(1)).
```

Since the answer literals contain the wrapped original indices we can easily determine which ACE axioms were used to derive which ACE theorems, and generate the pertinent reports.

Satchmo. During the translation of the first-order formula into a set of Satchmo clauses the wrapped indices remain attached to their atoms. Our modified version of Satchmo collects the wrapped 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.

Using RACE should not presuppose detailed knowledge of its workings.

Otter. Otter offers a large set of options and parameters and furthermore allows users to intervene during proofs. For RACE we use Otter's autonomous mode that – after syntactically scanning the input – automatically selects appropriate options, inference rules, and search strategies. The selected search strategy is usually refutation complete, but may not be especially effective and efficient. Thus the convenience of the autonomous mode is bought at a price. This is particularly noticeable in the case of Otter's built-in equality. Equality triggers paramodulation that, unfortunately, cannot be efficiently controlled in the autonomous mode. Thus we decided to represent equality by the reflexive, symmetric and transitive defined relation `is_equal/2`.

Satchmo. Satchmo does not offer any options or parameters, nor can users interact with Satchmo.

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

Otter & Satchmo. Both Otter and Satchmo accept auxiliary first-order axioms of the form

```
aux_axiom(Number,Formula,Text)
```

where `Number` labels the auxiliary axiom, `Formula` is a 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 in Otter clauses, respectively Satchmo clauses. While an auxiliary axiom is processed the atoms `A` of its `Formula` are changed into `A - aux_axiom(Number)` so that the `Text` of the auxiliary axiom becomes accessible to the justification of the proof via the index `aux_axiom(Number)`.

RACE should have an interface to evaluable functions and predicates.

This is the only requirement that Otter fulfils less well than Satchmo.

Otter. Otter provides a very limited set of evaluable functions and predicates for integer and floating point arithmetic, Boolean operations and lexical comparisons that allow users to program aspects of the deduction process – for instance list operations – in a style similar to Prolog. To go beyond the limits of Otter's evaluable functions and predicates, and to draw level with Satchmo, we will use Otter's gateway to foreign evaluable functions to access predicates defined externally in Prolog.

Satchmo. Satchmo accepts any Prolog predicate – be it user defined or built-in.

RACE should combine theorem proving with model generation.

Otter. Otter functions only as a theorem prover.

Satchmo. Satchmo 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 Satchmo will generate a model that is returned as a list of ground instances of atoms.

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 as a consequence of RACE's justification facility 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.
```

and

```
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 axiom(s) used to derive that answer.

Similarly we can check the consistency of an ACE text. If the text is inconsistent, RACE's justification facility will identify all its inconsistent 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.
```

and

```
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 often raise hard linguistic problems and trigger complex inferences. There are for example disambiguation presumptions activated by world-knowledge, inferences induced by linguistic knowledge about the structure and interpretation of plurals, or inferences that are driven by mathematical knowledge. The treatment of disambiguation inferences has been addressed in [Schwertel 2000]. Here we will show how 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 introduction of auxiliary predicates necessarily increases the search space. However – as we noticed during our investigations – the larger search space does not inevitably result in longer runtimes of RACE. This is most probably due to the efficient implementations of Otter and Satchmo.

Lattice Theoretic Axioms

Various axiom systems governing the behaviour of plurals in natural language have been proposed [Link 1998, chap. 2 and 10; Kamp & Reyle 1993]. For a 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 small selection of the many 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.
```

we want to infer

```
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(A,group)-2,drs([C],[structure(C,atomic)-2,
part_of(C,A)-2])=>drs([],[object(C,company)-2,property(C,'Swiss')-2]),
quantity(A,cardinality,B,count_unit)-2,value(B,eq,6)-2,
drs([D],[structure(D,atomic)-2,part_of(D,A)-2])=>drs([E,F],
[object(E,machine)-2,structure(E,atomic)-2,
predicate(F,event,buy,D,E)-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. In the above representation each object variable is typed according to its position in the lattice. Lines 2 and 3 of the structure express that there is a group `A` the atomic parts of which are Swiss companies, the fourth line that the cardinality of `A` equals 6, and lines 5 to 7 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 axiom stating that each group consists of atomic parts:

```
aux_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.

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. The reverse direction of the last axiom, viz. if an object does not have proper parts it is an atom, is not implemented since it explodes

the size of the search space. Also, the existence of a supremum is not axiomatised. These restrictions are not harmful since we have not (yet) found empirical needs for the introduction of the axioms. Commutativity, associativity and idempotence of the lattice-theoretic join operation are not directly implemented via first-order axioms but more efficiently simulated by list processing operations like permutation. In Otter, the relevant list processing operations were simulated by further auxiliary axioms, for Satchmo Prolog's built-in procedures could be used.

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],[query(A,who)-1,structure(A,dom)-1,
structure(B,dom)-1,drs([D],[structure(D,atomic)-1,part_of(D,B)-1])
=>
drs([],[object(D,machine)-1]),predicate(C,event,buy,A,B)-1]),
text(['Who buys machines?'])).
```

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

```
aux_axiom(2,forall([X],(structure(X,atomic) => structure(X,dom))),
'Atom is a subtype of the general type dom.')
aux_axiom(3,forall([X,Y],(structure(X,atomic) & part_of(Y,X) =>
is_equal(X,Y))), 'Atoms do not have proper parts.')
aux_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.

Otter. To process equality Otter provides paramodulation and demodulation. However, paramodulation cannot be effectively controlled in Otter's autonomous mode. Using the defined relation `is_equal/2` instead of Otter's built-in equality avoids triggering paramodulation, and turned out to be a viable – though perhaps debatable – approach.

Satchmo. Since Satchmo has no built-in theory of equality we have no alternative than to use the defined relation `is_equal/2`.

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.

Otter. In Otter the modelling of mathematical knowledge turned out to be rather tricky. We combined Otter's evaluable relations like `>/2` with auxiliary first-order axioms, e.g.

```
aux_axiom(5,forall([V,N,M],(value(V,eq,N)&-value(V,geq,M)=>M>N)),
'Number Axiom 5.')
```

RACE then outputs

```
RACE proved that the query (-ies)
  Who gets a discount?
can be answered on the basis of the sentence(s)
  If a company buys at least three machines it gets a discount.
  A German company buys four machines.
using the auxiliary axiom(s)
  Number Axiom 5.
```

One has to be careful though. The Otter manual recommends using evaluable functions and relations only if one knows how the inferences are going to occur when the formulas are conceived. This is difficult if you are using Otter's autonomous mode. For example, we have found no general solution to prevent evaluable conditions to be executed before all variables are instantiated. Due to these problems we consider our treatment of evaluable functions and relations in Otter only preliminary. Currently, the axioms work for all examples but scaling up to more complex examples cannot be guaranteed.

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

```
value(Var,geq,NewNumber)-Index:-
  number(NewNumber),
  value(Var,eq,GivenNumber)-Index,
  number(GivenNumber),
  NewNumber =< GivenNumber.
```

Instantiation problems can be easily avoided by the Prolog predicate `number/1`.

Domain-Specific Axioms

Even domain-specific knowledge – e.g. special ontologies – could analogously be formalised as auxiliary first-order axioms, although the representation in ACE is preferable.

8 Temporary Limitations of RACE

Since we are still experimenting with the Attempto Reasoner RACE its current implementation has the following temporary limitations some of which were already mentioned in the preceding sections.

Otter and Satchmo are still used independently but will later be combined into one integrated inference system.

Satchmo does not deliver all solutions, but only the first solution, if unsatisfiability is caused by a Prolog built-in predicate.

Otter and Satchmo do not yet accept the same set of evaluable functions and predicates. Otter uses its own set of built-in functions and predicates – that lead to instantiation problems – while Satchmo uses Prolog predicates – that are unproblematic. We plan to make Prolog predicates accessible to Otter via its interface to foreign evaluable functions.

Since RACE is still in an experimental phase we decided to postpone the introduction of settings to limit the runtime, or to restrict RACE's output in any way.

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.

Being based on the powerful theorem prover Otter and the efficient model generator Satchmo, RACE is very fast and solves all examples of this paper in a few milliseconds. Schubert's Steamroller [Stickel 1986] – a standard theorem proving example – takes 70 milliseconds on a 500 MHz Macintosh running Mac OS X. As a first check of RACE's scaling up capability we contrived a contradictory example consisting of 10 complex ACE sentences and altogether 56 auxiliary first-order axioms. RACE found all 5 contradictions in 2 seconds.

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 [Fuchs 1992] using ACE scenarios would be helpful. Further extensions will be necessary for the practical applications of ACE that we are investigating together with our partners in the fields of medical documentation, synthesis of constraint logic programs, web site synthesis and knowledge sharing, teaching logic and using ACE to control agents.

Acknowledgements

We would like to thank Johan Bos, François Bry, Michael Kohlhase, William McCune, John Sowa and Sunna Torge for fruitful discussions concerning automatic theorem proving. Our research was partially funded by grant 2000-061932.00 of the Swiss National Science Foundation.

Appendix

Besides Otter and Satchmo we evaluated a – necessarily subjective – selection of other theorem provers and model generators. Here we briefly describe our experiences with these tools, and explain why we decided not to use them to implement the Attempto Reasoner.

During our evaluation we learned two lessons. First, excellent theoretical credentials of a tool do not necessarily mean that it is also practical and efficient. Second, not every theorem prover or model generator is equally qualified for any type of problem, specifically for the problems occurring in the processing of natural language. Both insights are not actually new and have been stated previously by other researchers – we were even forewarned [personal communication by M. Kohlhase] – but we obviously needed to learn them from firsthand experience.

DRS Deduction Rules

In our first attempt to develop the Attempto Reasoner we implemented the deduction rules for discourse representation structures [Reyle & Gabbay 1994] in Prolog. This attempt differs from all our others in two aspects: first we performed deduction directly in the language of discourse representation structures, and second we developed a theorem prover virtually from scratch.

The seven DRS deduction rules were proven correct and complete for first-order logic. They use run-time skolemisation to avoid losing structural information of DRSs.

As we found out the correct application of the DRS deduction rules requires a combination of forward and backward reasoning which adversely affected the efficiency of our implementation. In fact, eventually we had to realise that we were unable to find a proof strategy that gave our implementation of the DRS deduction rules an acceptable performance. Our negative experience correlates with the cautionary remarks in [Blackburn et al. 2001, pp. 17-18].

leanTAP

leanTAP [Beckert & Posegga 1995] is a correct, complete and efficient theorem prover based on free-variable semantic tableaux. Input to leanTAP are first-order formulas in negation normal form. Like Satchmo leanTAP consists of a few lines of Prolog, and like Satchmo leanTAP owes its efficiency to exploiting the Prolog inference engine as much as possible.

To achieve the completeness that the Prolog inference engine lacks, leanTAP uses limited depth-first search that can be combined with iterative deepening. Beckert & Posegga write "… it is important in practice that the limit is not chosen too high, as the search space grows exponentially with [the limit]. A good solution for this problem is […] iterative deepening".

Recall that RACE should hide from its users internal parameters like a search limit. Thus for our purposes leanTAP could only be used in the iterative deepening mode. However, if a proof fails then leanTAP could run forever unless we enforced termination by an upper limit for iterative deepening or – perhaps better – for the run-time. This upper limit would have to be sufficiently high not to miss any proofs, meaning that many failing proofs would have to run for a long time until they finally reached the upper limit. We felt that this was asking too much of the users of RACE.

EP Tableaux

Extended Positive Tableaux [Bry & Torge 1998; Torge 1998] is a model generation method in the tradition of Satchmo. EP Tableaux is complete for unsatisfiability and for finite satisfiability. The input language of EP Tableaux is a fragment of the language of first-order logic called Positive Formulas with Restricted Quantification

(short PRQ formulas) that has the expressive power of full first-order logic. Torge implemented EP Tableaux as the Prolog program Finfimo.

In an earlier publication [Fuchs et al. 2000] we used ACE as a front-end to EP Tableaux and successfully solved a data base example and Schubert's steamroller [Stickel 1986]. However, as we found out, EP Tableaux as implemented in Finfimo does not scale up.

Here is why. EP Tableaux processes existentially quantified variables at run-time by the $\exists$ expansion rule

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

where $c_1 \ldots c_k$ are constants that occurred so far while c_{new} is a new constant distinct from $c_1 \ldots c_k$. Since each existentially quantified variable can be substituted by a previously used or by a new constant, each application of the $\exists$ expansion rule introduces a choice-point in Finfimo resulting in an exponentially growing search space.

Non-trivial ACE texts are translated into DRSs – and eventually into PRQ formulas – with a large number of existentially quantified variables. Using Finfimo for deductions from these ACE texts resulted in exploding run-times that we considered unacceptable.

Mace

The theorem prover Otter is complemented by Mace [McCune 2001] – a generator for finite models that accepts almost the same language as Otter.

Closer inspection revealed, however, that Mace is much more restrictive than Otter concerning the language it accepts, and also interprets some symbols differently. For instance, Mace does not accept relation symbols with arity greater than four, interprets natural numbers not as constants, does not accept Otter's evaluable functions and relations, and ignores Otter's answer literals.

For this reason we decided not to use Mace.

References

[ACE Manual 1999]

N. E. Fuchs, U. Schwertel, and R. Schwitter, Attempto Controlled English (ACE) Language Manual, Version 3.0, Technical Report ifi-99.03, University of Zurich, August 1999

[Beckert & Posegga 1995]

B. Beckert & J. Posegga, leanTAP: Lean Tableau-Based Deduction, Journal of Automated Reasoning, 15(3), pp. 339-358, 1995

[Blackburn & Bos 2000a]

P. Blackburn & J. Bos, Representation and Inference in Natural Language: A First Course in Computational Semantics (draft at www.comsem.org)

[Blackburn & Bos 2000b]

P. Blackburn & J. Bos, Working with Discourse Representation Theory: An advanced Course in Computational Semantics (draft at www.comsem.org)

[Bos 2001]

J. Bos, Model Building for Natural Language Understanding (draft at www.iccs.informatics.ed.ac.uk/~jbos)

[Blackburn et al. 2001]

P. Blackburn, J. Bos, M. Kohlhase, H. de Nivelle, Inference and Computational Semantics, in H. Bunt, R. Muskens, E. Thijsse (eds.), Computing Meaning, vol. 2, Kluwer Academic Publishers, pp. 11-28, 2001

[Bry & Torge 1998]

F. Bry & S. Torge, A Deduction Method Complete for Refutation and Finite Satisfiability, Proc. 6th European Workshop on Logics in Artificial Intelligence, LNAI 1489, Springer-Verlag, 1998

[Bry & Yahya 1996]

F. Bry & A. Yahya, Minimal Model Generation with Positive Unit Hyper-Resolution Tableaux, Research Report PMS-FB-1996-1, Institut für Informatik, Ludwig-Maximilians Universität München, 1996

[Fuchs 1992]

N. E. Fuchs, Specifications Are (Preferably) Executable, Software Engineering Journal, September 1992, pp. 323-334; reprinted in: J. P. Bowen, M. G. Hinchey, High-Integrity System Specification and Design, Springer-Verlag London, 1999

[Fuchs et al. 1998]

N. E. Fuchs, U. Schwertel, R. Schwitter. Attempto Controlled English – Not Just Another Logic Specification Language. in P. Flener (ed.), Logic-Based Program Synthesis and Transformation, 8th International Workshop LOPSTR '98, LNCS 1559, Springer-Verlag, 1999

[Fuchs et al. 2000]

N. E. Fuchs, U. Schwertel, S. Torge, A Natural Language Front-End to Model Generation, Journal of Language and Computation, Volume 1, Number 2, pp. 199-214, December 2000

[Green 1969]

C. Green, Theorem proving by resolution as a basis for question-answering systems, Machine Intelligence, 4:183--205, 1969

[Iwanska & Shapiro 2000]

L. M. Iwanska & S. C. Shapiro (eds.), Natural Language Processing and Knowledge Representation, AAAI Press/MIT Press, 2000

[Kamp & Reyle 1993]

H. Kamp & U. Reyle, From Discourse to Logic, Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory, Kluwer, 1993

[Link 1998]

G. Link, Algebraic Semantics in Language and Philosophy, CSLI Lecture Notes 74, CSLI Publications, Stanford, 1998

[Manthey & Bry 1988]

R. Manthey & F. Bry, SATCHMO: A Theorem Prover Implemented in Prolog, in E. L. Lusk and R. A. Overbeek (eds.), Proceedings CADE 88, Ninth International Conference on Automated Deduction, LNCS, Springer Verlag, pp. 415-434, 1988

[McAllester & Givan 1992]

D. McAllester & R. Givan, Natural Language Syntax and First Order Inference, Artificial Intelligence vol. 56, pp. 1-20, 1992; reprinted in [Iwanska & Shapiro 2000]

[McCune 1994]

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

[McCune 2001]

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

[Reyle & Gabbay 1994]

U. Reyle & D. M. Gabbay, Direct Deductive Computation on Discourse Representation Structures, Linguistics & Philosophy, 17:343--390, 1994

[Schwertel 2000]

U. Schwertel, Controlling Plural Ambiguities in Attempto Controlled English, in 3rd International Workshop on Controlled Language Applications, Seattle, Washington, 2000

[Sowa 2000]

J. F. Sowa, Knowledge Representation – Logical, Philosophical and Computational Foundations, Brooks/Cole, 2000

[Stickel 1986]

M. E. Stickel, Schubert's Steamroller Problem: Formulations and Solutions, Journal of Automated Reasoning, vol. 2, pp. 89-101, 1986

[Sukkarieh 2001a]

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

[Sukkarieh 2001b]

J. Sukkarieh, Natural Language for Knowledge Representation, PhD Thesis, Computer Laboratory, University of Cambridge, 2001

[Sutcliffe & Suttner 1997]

G. Suttcliffe & C. Suttner, Results of the CADE-13 ATP System Competition, Journal Automated Reasoning, 18(2), pp. 259-264, 1997

[Torge 1998]

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