First-Order Reasoning for Attempto Controlled English

Norbert E. Fuchs

Department of Informatics & Institute of Computational Linguistics
University of Zurich
fuchs@ifi.uzh.ch
http://attempto.ifi.uzh.ch/

Abstract. RACE is a first-order reasoner for Attempto Controlled English (ACE) that can show the (in-) consistency of a set of ACE axioms, prove ACE theorems from ACE axioms and answer ACE queries from ACE axioms. In each case RACE gives a proof justification in ACE and full English. This paper is a system description of RACE sketching its structure, its implementation, its operation and its user interface. The power and the limitations of RACE are demonstrated and discussed by concrete examples.

Keywords: controlled natural language, Attempto Controlled English, ACE, first-order reasoning, RACE

1 Introduction

Attempto Controlled English (ACE)[1] is a logic-based knowledge representation language that uses the syntax of a subset of English. ACE allows domain specialist to represent formal knowledge in familiar English without having to resort to visibly formal languages. Working with ACE is supported by a number of tools[2], foremost by the ACE Parsing Engine (APE)[3] that translates an ACE text into a discourse representation structure (DRS) that is expressed in a variant of the standard language of first-order logic.

The DRS can be translated into other logic languages, for instance into the standard and the clausal forms of first-order logic, into the TPTP[4] notation, and – with some syntactic restrictions – into the semantic web languages OWL and SWRL[5].

Once you have knowledge expressed as an ACE text you may want to reason with this knowledge, for instance to show its consistency. The large variety of logical representations of an ACE text allows you to use existing reasoning tools for this purpose. The translation of an ACE text into the TPTP notation gives you access to all reasoning tools provided by the TPTP system. In fact, the TPTP web-interface already

[1] http://attempto.ifi.uzh.ch/
[2] http://attempto.ifi.uzh.ch/site/tools/
[3] web-interface of APE: http://attempto.ifi.uzh.ch/ape/
[4] http://www.cs.miami.edu/~tptp/
[5] http://www.w3.org/TR/owl2-overview/, http://www.w3.org/Submission/SWRL/

accepts input in ACE. The translation of an ACE text into OWL and SWRL makes all theorem provers for these languages available. This approach is taken by Kaljurand's ontology editor ACE View[6] and by Kuhn's semantic wiki AceWiki[7] both of which also provide a back translation of the reasoning results into ACE.

This paper describes the reasoner RACE that allows users to show the (in-) consistency of an ACE text, to deduce one ACE text from another one, and to answer ACE queries from an ACE text.

RACE stands in the long tradition that investigates the relation between natural language, reasoning and logic – a tradition that started in ancient Greece, continued during the European middle ages, and thrived especially vigorously after the formalisation of logic and the invention of the digital computer. Note, however, that this paper is a system description of RACE. Thus the relation between natural language and logic will be alluded to occasionally, but will not be discussed in depth. Also you will not learn much about the state of the art of natural language processing and of theorem proving. Nor will this paper introduce you to the language Attempto Controlled English beyond what is needed to demonstrate features of RACE. To learn more about ACE refer to the relevant documentation[8].

The rest of this paper is organised as follows. In section 2, I will summarise general features of RACE. Section 3 very briefly shows how one works with RACE via its web-client. Section 4 gives a rather detailed look under the hood of RACE. In section 5 I investigate how RACE handles some typical proofs and how RACE processes specific ACE constructs. Section 6 discusses questions like decidability, termination, looping and efficiency. Section 7 concludes with a discussion of RACE's strengths and limitations, and with a preview of further research.

2 General Features of RACE

RACE has the following general features:

- RACE offers consistency checking, textual entailment and query answering of ACE texts.
- Conforming to the goal of the Attempto project – provide formal methods clad in (controlled) English – RACE does not presuppose any knowledge of formal logic or theorem proving, does not require users to understand RACE's workings, nor does it require users to control the reasoning process.
- All input is in ACE, all output is in ACE and full English.
- Consistency checking: For inconsistent ACE axioms RACE will list all minimal subsets of the axioms that lead to inconsistency.
- Textual entailment and query answering: If the ACE axioms entail the ACE theorems, respectively ACE queries, RACE will list all minimal subsets of the axioms that entail the theorems, respectively queries.
- Textual entailment and query answering: If the ACE axioms do not entail the

[6] http://attempto.ifi.uzh.ch/aceview/
[7] http://attempto.ifi.uzh.ch/acewiki/, https://launchpad.net/acewiki/
[8] http://attempto.ifi.uzh.ch/site/docs/

ACE theorems, respectively ACE queries, RACE will list all ACE words and ACE language constructs of the theorems, respectively queries, that could not be entailed.

- Textual entailment and query answering: If the ACE axioms are inconsistent, RACE outputs a warning message, and reports the results found so far.
- Non-termination: ACE is a superset of a fragment of English that [5] proved to be undecidable. For undecidable problems RACE terminates with a time-out.
- RACE covers the first-order subset of ACE, that is all ACE constructs with the exception of imperative sentences, negation-as-failure and the modal operators *may* and *should*. Currently RACE does not yet cover arithmetic, formulas and operations on lists, sets and strings.

3 Working With RACE

RACE is implemented as a Prolog program and can be accessed remotely via its web-client[9] or via its web-service[10]. Both offer interfaces for consistency checking, textual entailment and query answering. For convenience we will use screen-shots of the web-client to give a first impression how to work with RACE.

The consistency checking of figure 1 shows that the minimal subset {*Every man is a human.*, *John is a man.*, *John is not a human.*} of the axioms leads to inconsistency.

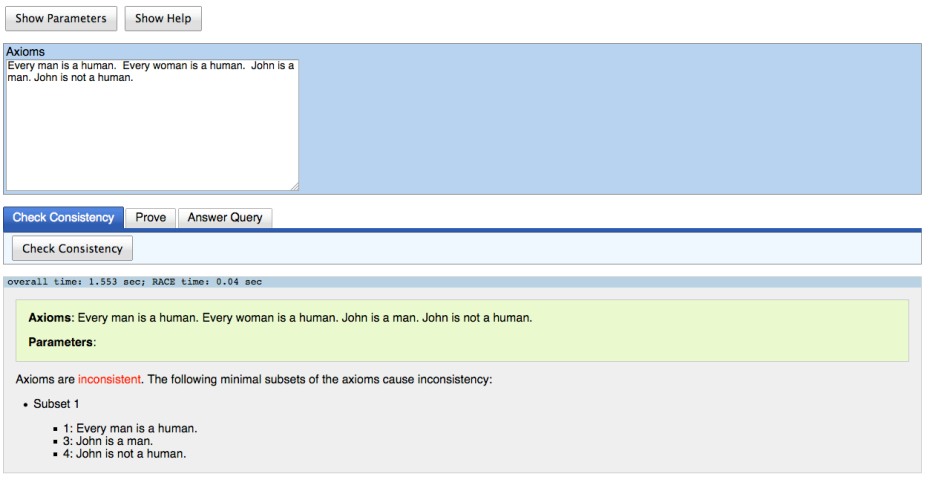

Figure 1. Consistency checking

In the following I will restrict the screen-shots to the result window since it also contains the input. Figure 2 shows that the theorem *There is a human.* is entailed by two minimal subsets of the axioms, namely {*Every man is a human.*, *John is a man.*} and {*Every woman is a human.*, *Mary is a woman.*}.

[9] http://attempto.ifi.uzh.ch/race/
[10] http://attempto.ifi.uzh.ch/ws/race/racews.perl

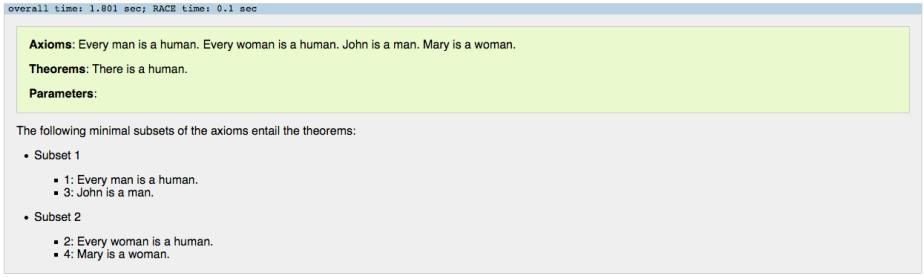

Figure 2. Textual entailment

Figure 3 contains a case of query answering. The results show that the query *Who tires how?* can be answered from the axioms {*If John waits then he tires easily*., *John waits*.} Also, the query word *who* is substituted by the proper name *John*, and the query word *how* by the positive form of the adverb *easily*. Note that the substitution refers to *how/when/where* since these three query words are treated as equivalent.

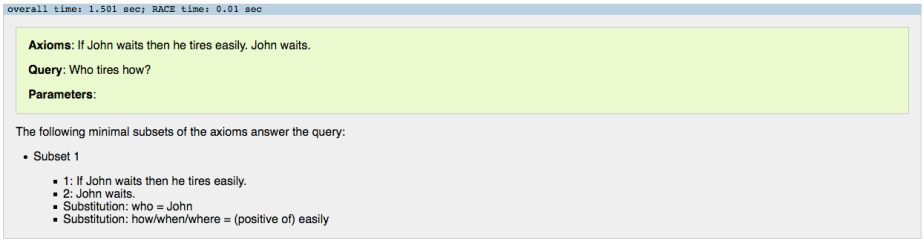

Figure 3. Query answering

As we have seen, RACE answers for succeeding proofs the question "Why?" by listing the axioms that are inconsistent or that are needed to prove a theorem or answer a query. For failing proofs RACE answers the question "Why Not?" by listing the ACE words and ACE constructs of the theorem or query that could not be proved. Figure 4 shows an example.

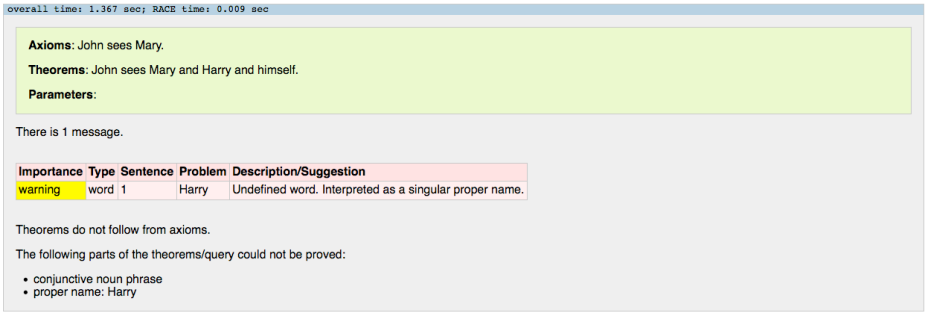

Figure 4. "Why Not?" answer for a failing proof

The proof failed, and RACE list the proper name *Harry* and the ACE construct *noun phrase conjunction* as not provable. Since the proper name *Harry* was not found in the ACE lexicon, a warning is generated that *Harry* was automatically interpreted as proper name.

4 A Look Under the Hood of RACE

Satchmo. RACE is implemented as a Prolog program on the base of the model generator Satchmo [1]. Satchmo was chosen since it is available as a small Prolog program (see figure 5) that lent itself to be locally modified and extended to provide RACE's functionality. As will be seen in the next sections, RACE is not simply a wrapper of Satchmo since the modifications applied concern also core functionality of Satchmo.

```
satisfiable :-
  setof(Clause, violated_instance(Clause), Clauses),
  !,
  satisfy_all(Clauses),
  satisfiable.
satisfiable.

violated_instance((B ---> H)) :-
  (B ---> H),
  B,
  \+ H.

satisfy_all([]).
satisfy_all([(_B ---> H) | RestClauses]) :-
  H,
  !,
  satisfy_all(RestClauses).

satisfy_all([(_B ---> H) | RestClauses]) :-
        satisfy(H),
  satisfy_all(RestClauses).

satisfy((A;B)) :-
  !,
  (satisfy(A) ; satisfy(B)).

satisfy(Atom) :-
  \+ Atom = fail,
  assume(Atom).

assume(Atom) :-
  asserta(Atom).

assume(Atom) :-
  retract(Atom),
  !,
  fail.
```

Figure 5. Satchmo model generator [1]

Satchmo works with first-order clauses of the form *Body* ---> *Head* where *Body* is *true* or a conjunction of logical atoms, and *Head* is *fail* or a disjunction of logical atoms. There is not explicit negation, instead one uses an implication to *fail*.

Satchmo executes the clauses by forward-reasoning. Once the *Body* of a clause can be proved from the Prolog database, then its *Head* is asserted to the Prolog data base if not already there. If the *Head* of a clause is *fail* then Satchmo backtracks, retracting previously asserted atoms from the Prolog database.

Satchmo generates a minimal finite Herbrand model of the clauses – if finite models exist [2]. If the model of the clauses is infinite then Satchmo loops. If the clauses are unsatisfiable, Satchmo just fails. Satchmo is correct for unsatisfiablity if the clauses are range-restricted, i.e. all variables of *Head* occur already in *Body* [1]. Satchmo is complete for unsatisfiability if it is used level-saturated, i.e. clauses are tried bottom-up, level by level [1].

Satchmo is a very efficient program. Its efficiency can be enhanced in various ways [1], most effectively when first-order clauses are replaced by Prolog clauses that are directly executed without incurring the overhead of forward-reasoning.

From Satchmo to RACE. The modifications and extension applied to Satchmo in order to achieve RACE's functionality resulted in a program that is two orders of magnitude larger than Satchmo. All modifications and extensions of RACE take into account the preservation of Satchmo's theoretical attributes – correctness, completeness, minimal model generation – though this has not yet been proven.

Here are the similarities and the main differences between Satchmo and RACE.

- Like Satchmo RACE works with first–order clauses.
- For satisfiable clauses both Satchmo and RACE generate minimal finite Herbrand models – if they exists.
- For infinite models Satchmo loops, while RACE stops with a time-out.
- For unsatisfiable clauses Satchmo stops immediately when it detects unsatisfiability, while RACE finds all minimal unsatisfiable subsets of the clauses.
- While Satchmo works on clauses found in the Prolog data base and stores the model of the clauses as Prolog facts, RACE translates ACE axioms, theorems, and questions into clauses and outputs its results in ACE and full English.
- While Satchmo just succeeds or fails indicating that the clauses are satisfiable or not, RACE generates for each sucessful proof a report showing which minimal subsets of the ACE axioms are inconsistent, respectively entail the ACE theorems or queries. For a failing proof RACE will list those ACE text fragments of the theorems or queries that could not be proved.

In light of RACE's additional functionality and increased size it is obvious that it cannot be as efficient as Satchmo. Concerning RACE's efficiency see section 6.

Structure and operation of RACE. To best understand the structure and the operation of RACE we will consider a concrete example.

Given the ACE text T
There is a cat. Every cat is an animal.
answer the ACE query Q
Is a cat an animal?

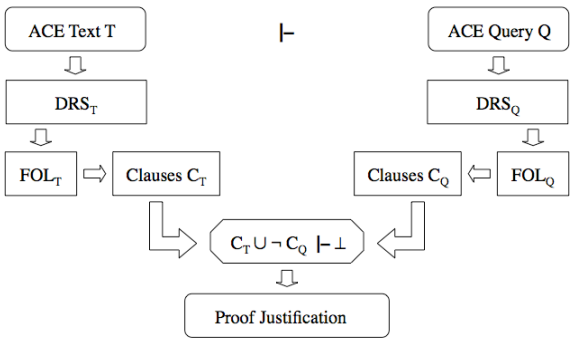

Figure 6. RACE answers an ACE query Q from an ACE text T

The query answering is summarised in Figure 6. In a first step both the ACE text T and the ACE query Q are submitted to the Attempto Parsing Engine APE that translates them into the Discourse Representation Structures DRS_T, respectively DRS_Q.

DRS_T:
```
[A]
 object(A,cat,countable,na,eq,1)-1/4
    [B]
    object(B,cat,countable,na,eq,1)-2/2
    =>
    [C,D]
    object(C,animal,countable,na,eq,1)-2/5
    predicate(D,be,B,C)-2/3
```

DRS_Q:
```
[]
    QUESTION
    [A,B,C]
    object(A,cat,countable,na,eq,1)-1/3
    object(B,animal,countable,na,eq,1)-1/5
    predicate(C,be,A,B)-1/1
```

These are pretty-printed versions of the DRSs internally represented as Prolog terms. The language of the DRSs – described in Fuchs et al. [7] – uses a compact, reified variant of the standard language of first-order logic. Countable noun phrases like *a cat* are represented as `object(Ref,Noun,countable,na,Operator,Count)` where `object/6` is a predefined predicate, `Ref` is a quantified variable called discourse referent, `Noun` stands for the represented noun, `countable` classifies the noun as a countable common noun, `na` is a place-holder otherwise used for measurement nouns, and `Operator` and `Count` express the cardinality. Transitive predicates like the transitive form of *to be* are represented as `predicate(Ref,Verb,Ref1,Ref2)` where `predicate/3` is a predefined predicate, `Ref` is a discourse referent, `Verb` stands for the represented transitive verb, and `Ref1` and `Ref2` are the discourse referents of two `object/6` definitions. The structure `S/T` attached to each logical atom indicates that the atom was derived from token `T` in ACE sentence `S`. The sentence indices `S` are later used by RACE to label individual axioms, theorems and queries.

In a next step the two DRSs are individually translated into the standard language of first-order logic, and we get FOL_T and FOL_Q.

FOL_T:
```
exists(A,&(object(World,A,cat,countable,na,geq,1)-axiom(1),forall(B,
=>(object(World,B,cat,countable,na,geq,1)-axiom(2),exists(C,exists(D,
&(object(World,C,animal,countable,na,geq,1)-axiom(2),
predicate(World,D,be,B,C)-axiom(2)))))))
```

FOL_Q:
```
exists(A,exists(B,exists(C,&(object(World,A,cat,countable,na,geq,1)-
theorem(1),&(object(World,B,animal,countable,na,geq,1)-theorem(1),
predicate(World,C,be,A,B)-theorem(1))))))
```

During the translation each logical atom received an additional argument `World` that stands for a possible world needed to process modality within first-order logic.

Operators `eq` were replaced by `geq` to express the meaning of, for instance, *a cat* as *at least one cat*. Also the sentence indices of the logical atoms have been wrapped by `axiom/1`, respectively `theorem/1` to distinguish axioms from theorems.

In a final step the formula `exists(World, FOL`$_T$ `& ¬ FOL`$_Q$`)` is translated into clauses using a variant of the standard clausification procedure. The general form of the clauses used by RACE is

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

where – as in Satchmo – *Body* is *true* or a conjunction of logical atoms and *Head* is *fail* or a disjunction of logical atoms. Negation is expressed as implication to *fail*. *Body* implies *Head*. *Index* is a list containing the term *axiom(N)* or *theorem(N)* where *N* is the sentence index carried over from the DRS.

The three clauses derived from `exists(World, FOL`$_T$ `& ¬ FOL`$_Q$`)` are

```
satchmo_clause(true, object(sk1, sk2, cat, countable, na, geq, 1),
[axiom(1)])
satchmo_clause(object(sk1, D, cat, countable, na, EQ, Count),
(object(sk1, sk3(D), animal, countable, na, EQ, Count), predicate(sk1,
sk4(D), be_NP, D, sk3(D))), [axiom(2)])
satchmo_clause((object(sk1, A, cat, countable, na, geq, 1), object(sk1,
B, animal, countable, na, geq, 1), predicate(sk1, C, be_NP, A, B)),
fail, [theorem(1)])
```

During clausification further transformations were performed. The argument `World` – being existentially quantified – was skolemised. The arguments `be` were replaced by `be_NP` – the reason for which is explained in section 5. The operator and count arguments of the second clause – derived from *Every cat is an animal*. – were replaced by the variables `EQ`, respectively `Count`. This allows RACE to flexibly match the clause body to all values of operators and counts. Furthermore, clauses are checked for range-restriction, i.e. all variables in the head of a clause must already occur in its body.

Once we have the clauses we can call RACE's main predicate

```
satchmo(Clauses, Models, Inconsistencies)
```

where `Clauses` is a list of clauses, `Models` is a list of models of the clauses, and `Inconsistencies` is a list of list of indices of clauses that led to inconsistency.
The predicate `satchmo/3` executes clauses bottom-up by forward-reasoning. If the *Body* of a clause `satchmo_clause(Body, Head, Index)` is *true* or can be proved from the Prolog data base then *Head* is asserted to the Prolog database together with a list that contains the element of *Index* plus the sentence indices of all clauses that were used to prove *Body*. This amounts to building a proof-tree labelled with lists of indices. Range restriction ensures that all atoms added to the Prolog data base are ground. If *Head* is a disjunction – meaning that the proof-tree branches – then the asserted index list also contains an identification of the respective disjunctive branch. If *Head* is *false* then `satchmo/3` backtracks and removes atoms added previously to the Prolog data base. This also indicates that a branch of the proof-tree is closed, and the indices of the leaf of the closed branch are stored. Forward-reasoning ends when no clause can be executed. The result is either a set of ground atoms in the Prolog data

base that constitute minimal models of `Clauses` or an empty Prolog data base plus a set of closed branches. In the second case `satchmo/3` checks whether all branches of the proof-tree are closed and then collects in `Inconsistencies` the indices of the leaves of all closed branches. Thus for terminating proofs we have the following outcomes:

- `Clauses` are consistent: `Inconsistencies = []`
- `Clauses` are inconsistent: `Models = []`, `Inconsistencies ≠ []`

Tracing a proof. Here is an annotated proof of our example displaying the assertions and retractions mentioned above. The argument `Clauses` of `satchmo/3` is the list of clauses derived before.

```
call: satchmo(Clauses, Models, Inconsistencies)

% satchmo_clause(true, object(sk1, sk2, cat, countable, na, geq, 1),
[axiom(1)]) can fire, yielding ...
Asserting Head and Indices:
object(sk1,sk2,cat,countable,na,geq,1),[axiom(1)]

% satchmo_clause(object(sk1, D, cat, countable, na, EQ, Count),
(object(sk1, sk3(D), animal, countable, na, EQ, Count), predicate(sk1,
sk4(D), be_NP, D, sk3(D))), [axiom(2)]) can fire, yielding ...
Asserting Head and Indices:
object(sk1,sk3(sk2),animal,countable,na,geq,1),[axiom(1),axiom(2)]
Asserting Head and Indices:
predicate(sk1,sk4(sk2),be_NP,sk2,sk3(sk2)),[axiom(1),axiom(2)]

% satchmo_clause((object(sk1, A, cat, countable, na, geq, 1),
object(sk1, B, animal, countable, na, geq, 1), predicate(sk1, C, be_NP,
A, B)), fail, [theorem(1)]) can fire, yielding ...
Asserting: closed_branch([axiom(1),axiom(2),theorem(1)])
Retracting Head and Indices:
predicate(sk1,sk4(sk2),be_NP,sk2,sk3(sk2)),[axiom(1),axiom(2)]
Retracting Head and Indices:
object(sk1,sk3(sk2),animal,countable,na,geq,1),[axiom(1),axiom(2)]
Retracting Head and Indices:
object(sk1,sk2,cat,countable,na,geq,1),[axiom(1)]

exit: satchmo(Clauses, [], [[axiom(1), axiom(2), theorem(1)]])
```

Since `Models=[]` and `Inconsistencies=[[axiom(1), axiom(2), theorem(1)]]` `satchmo/3` proved that the clauses derived from the two axioms and the theorem are inconsistent, meaning that the ACE axioms *There is a cat. Every cat is an animal.* answer the ACE query *Is a cat an animal?*.

Note that if the query could not have been answered, then the asserted logical atoms `Head` would not have been retracted, and would have constituted a minimal model of `Clauses`, respectively the ACE axioms.

Interface predicates. RACE – for example its web-client – does not call `satchmo/3` directly. The translation of ACE texts into clauses, the call of `satchmo/3` and the back translation of `Inconsistencies` into the respective ACE texts are performed by three

interface predicates that take care of the peculiarities of consistency checking, theorem proving and query answering. These predicates are

- *check_consistency(Axioms, Parameters, Messages, RunTime, AxiomsUsed)* that checks the consistency of a set of ACE *Axioms* and returns a list of lists *AxiomsUsed* of all minimal inconsistent subsets of the *Axioms*.
- *prove(Axioms, Theorems, Parameters, Messages, RunTime, AxiomsUsed, WhyNot)* proves a set of ACE *Theorems* from a set of ACE *Axioms* and returns a list of lists *AxiomsUsed* that contains all minimal subsets of the *Axioms* needed to prove the *Theorems*.
- *answer_query(Axioms, Queries, Parameters, Messages, RunTime, AxiomsUsed, WhyNot)* answers a set of ACE *Queries* from a set of ACE *Axioms* and return a list of lists *AxiomsUsed* that contains all minimal subsets of the *Axioms* needed to answer the *Queries*. For *wh*-questions *AxiomsUsed* contains also the substitutions of the query words.

RACE's interface predicates have some common arguments. The argument *Parameters* is a list containing RACE's parameters. Currently, there is but one parameter *raw* used for testing. The argument *Messages* contains warning and error messages produced during a run of RACE. If there are error messages then RACE reports that the proof failed. The argument *RunTime* returns the overall run time in milliseconds. If *Axioms* cannot be proved or *Queries* cannot be answered then the argument *WhyNot* contains a list of ACE text fragments of the *Theorems* or *Queries* that RACE was unable to prove on the basis of the *Axioms*. For demonstration purposes here is the call of *answer_query/7* for our example.

```
answer_query('There is a cat. Every cat is an animal.','Is a cat an
animal?', [], Messages, RunTime, AxiomsUsed, WhyNot).
Messages = []
RunTime = 0
AxiomsUsed = [proof(['1: There is a cat.', '2: Every cat is an
animal.'], [])]
WhyNot = []
```

Since the identification of the minimal subsets of inconsistent axioms, or the identification of minimal subsets of axioms needed to entail a theorem or a query depends essentially on the threading and processing of sentence indices one could say that RACE uses a form of labelled deduction [3].

Auxiliary axioms. RACE uses about 50 auxiliary axioms as meaning postulates for plurals and singulars, to handle generalised quantifiers, to interpret the copula *be* that in ACE is underspecified, for summation, for the substitution of query words, and for other purposes. Auxiliary axioms are implemented as Prolog predicates that access elements of the DRS representation of ACE texts. Being implemented in Prolog, the auxiliary axioms do not participate in the eager forward-reasoning of the clauses, but – called only on demand – are executed by lazy backward-reasoning. Auxiliary predicates are also labelled, and the RACE parameter *raw* – introduced for testing purposes – enables the output of the auxiliary axioms used together with the ACE axioms. Individual auxiliary axioms will be presented in the next section.

5 All Things Considered

In this section I will investigate in greater detail how RACE handles some typical proofs and how RACE processes specific language constructs of ACE. I will use RACE's interface predicates instead of the more space-consuming screen-shots of the web-client. Irrelevant argument substitutions will be suppressed.

Plurals. As Schwertel discusses in great detail in her thesis [8], English plural nouns are extremely complex. To eliminate some of this complexity, ACE offers only collective and distributive plurals. The sentence

```
Three men lift two pianos.
```

introducing the collective plurals *three men* and *two pianos* describes a complex lifting event involving a group of three men and a group of two pianos. The lifting event is underspecified concerning the number of individual lifting actions and the number of men and pianos involved in each lifting action. Compare this to the sentence

```
Each of three men lifts each of two pianos.
```

involving the two distributive plurals *each of three men* and *each of two pianos*. The lifting relation consists of six lifting actions each involving one of the three men and one of the two pianos. One could say that the lifting relation is fully specified as far as the number of lifting actions and the number of participants are concerned.

Deductions from Collective Plurals. Collective plurals like *three men* are ambiguous between the reading *at least 3 men* and *exactly three men* (= *at least three men and at most three men*). This ambiguity also affects the deductions that can be made from collective plurals. Given the attempted deductions

```
(1) Three men lift a piano. |?- A man lifts a piano.

(2) Three men see a piano. |?- A man sees a piano.
```

humans – on the basis of their world knowledge – would probably consider deduction (1) as incorrect and deduction (2) as correct.

The reasoner RACE does not have any world knowledge and thus needs to take recourse to syntactic means to enable/disable deductions from collective plurals. By default RACE interprets collective plurals like *three men* as *at least 3 men*. (Note that ACE also provides the explicit construct *at least 3 men*.) The alternative "exactly" reading is explicitly expressed as *exactly three men*. To enforce the expected deductive behaviour we can reformulate the deductions (1) and (2) as

```
(1') Exactly three men lift a piano. |/- Exactly one man lift a piano.

(2') Three men see a piano. |- A man sees a piano.
     (= At least three men see a piano.|- At least one man see a piano.)
```

Note that the determiners *a* and *one* are treated as synonyms, that is *a man* and *one man* get the same logical representation.

To perform deduction (2') RACE needs a meaning postulate relating the noun phrases *at least three men* and *at least one man*. To formulate this meaning postulate we rely on RACE's logical representations of noun phrases. As introduced in section 4, nouns are represented by the predefined predicate *object/7*, concretely

- *at least three men* as *object(World, A, man, countable, na, geq, 3)*
- *at least one man* as *object(World, B, man, countable, na, geq, 1)*

Thus *at least three men* and *at least one man* get the same representation with appropriate cardinalities. This allows us to state the relation between singulars and collective plurals of all countable nouns as the first-order formula

```
∀ A, Noun, N, M (object(World, A, Noun, countable, na, geq, N) ∧ N≥M
→
object(World, A, Noun, countable, na, geq, M))
```

where *Noun* stands for any noun, and N and M are the respective cardinalities. RACE expresses this formula as the auxiliary Prolog axiom `cd5`

```
prolog_axiom(object(World,A,Noun,countable,na,geq,M), Block,
[prolog_axiom(cd5)|Indices]) :-
 nonvar(M),
 exists_asserted_atom(object(World,A,Noun,countable,na,geq,N), Indices),
 N>=1,
 M<N.
```

The logical atom `object(World, A, Noun, countable, na, geq, M)` can be proved and gets the index `[prolog_axiom(cd5)|Indices]` if the logical atom `object(World, A, Noun, countable, na, geq, N)` with the index `Indices` can be found in the Prolog data base, and the conditions `N>=1` and `M<N` are fulfilled. The argument `Block` is used to block unwanted combinations of auxiliary axioms.

Calling the RACE interface predicate *prove/7* with the parameter *raw* set we get the expected results for deductions (1') and (2'). While (1') fails

```
prove('Exactly three men lift a piano.','Exactly one man lifts a
piano.', [raw], Messages, RunTime, AxiomsUsed, WhyNot).

WhyNot = ['countable common noun: (at least 1) man']
```

(2') succeeds with the help of auxiliary Prolog axiom `cd5` as shown in the proof trace

```
prove('Three men see a piano.', 'A man sees a piano.', [raw], Messages,
RunTime, AxiomsUsed, WhyNot).

Clauses=[satchmo_clause(true, (object(sk1, sk2, man, countable, na, geq,
3), object(sk1, sk3, piano, countable, na, geq, 1), predicate(sk1, sk4,
see, sk2, sk3)), [axiom(1)]), satchmo_clause((object(sk1, A, man,
countable, na, geq, 1), object(sk1, B, piano, countable, na, geq, 1),
predicate(sk1, C, see, A, B)), fail, [theorem(1)])]

call: satchmo(Clauses, Models, Inconsistencies)
```

```
Asserting Head and Indices:
object(sk1,sk2,man,countable,na,geq,3),[axiom(1)]
Asserting Head and Indices:
object(sk1,sk3,piano,countable,na,geq,1),[axiom(1)]
Asserting Head and Indices: predicate(sk1,sk4,see,sk2,sk3),[axiom(1)]
Auxiliary Prolog axiom cd5: Body object(sk1,sk2,man,countable,na,geq,1)
is proved from asserted atom object(sk1,sk2,man,countable,na,geq,3)
Asserting: closed_branch([axiom(1),prolog_axiom(cd5),theorem(1)])
Retracting Head and Indices: predicate(sk1,sk4,see,sk2,sk3),[axiom(1)]
Retracting Head and Indices:
object(sk1,sk3,piano,countable,na,geq,1),[axiom(1)]
Retracting Head and Indices:
object(sk1,sk2,man,countable,na,geq,3),[axiom(1)]
exit: satchmo(Clauses, [], [[axiom(1), prolog_axiom(cd5), theorem(1)]])

AxiomsUsed = [proof(['1: Three men see a piano.'], ['Prolog Axiom cd5:
at least M objects |- at least 1, 2, ..., M-1 objects'])]
```

Deductions from Distributive Plurals. As we have seen distributive plurals are fully specified concerning the structure of the relation between the participants. Thus deductions to distributive plurals with smaller cardinalities and to the singular are always allowed. Again meaning postulates, i.e. auxiliary Prolog axioms, based on the representation of distributive plurals are needed to enable the deductions. As a demonstration an example using distributive plurals in four different syntactic roles.

```
prove('Each of 6 men gives each of 5 students each of 4 books on each of
3 mornings.', 'Each of 5 men gives each of 4 students each of 3 books on
1 morning.', [raw], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: Each of 6 men gives each of 5 students each of
4 books on each of 3 mornings.'], ['Prolog Axiom cd0: enable deductions
from distributive plurals', 'Prolog Axiom cd1: each of M objects |- each
of M-1, ..., each of 2 objects, 1 object']), proof(['1: Each of 6 men
gives each of 5 students each of 4 books on each of 3 mornings.'],
['Prolog Axiom cd1: each of M objects |- each of M-1, ..., each of 2
objects, 1 object'])]
```

There are two results involving the auxiliary axioms `cd0` and `cd1`.

Deductions Involving Collective and Distributive Plurals. Deductions from collective to distributive plurals are not possible, while deductions from distributive to collective plurals are allowed. Here are two examples demonstrating this behaviour. The first example

```
prove('3 women wait.', 'Each of 3 women waits.', [raw], Messages,
RunTime, AxiomsUsed, WhyNot).
```

correctly fails. The second example shows the other direction

```
prove('Each of 3 women waits.', '3 women wait.', [raw], Messages,
RunTime, AxiomsUsed, WhyNot).
```

```
AxiomsUsed = [proof(['1: Each of 3 women waits.'], ['Prolog Axiom cd0:
enable deductions from distributive plurals'])],
```

This proof succeeds with the help of the auxiliary Prolog axiom `cd0`.

Deductions from Conjunctive Plurals. Collective and distributive plurals can also be formed by conjunctions of noun phrases leading to so called conjunctive plurals. Since conjuncts can again be plurals we may end up with rather complex plural structures.

The first example with a collective conjunctive plural correctly fails – though in full English the deduction is valid.

```
prove('John and Mary wait.', 'Mary waits.', [raw], Messages, RunTime,
AxiomsUsed, WhyNot).
```

Replacing the collective conjunctive plural by a distributive one the proof succeeds.

```
prove('Each of John and Mary waits.', 'Mary waits.', [raw], Messages,
RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: Each of John and Mary waits.'], ['Prolog Axiom
cd0: enable deductions from distributive plurals'])],
```

Finally an example with a complex conjunctive plural.

```
prove('Each of 2 men and 3 women waits.', 'A man waits.', [raw],
Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: Each of 2 men and 3 women waits.'], ['Prolog
Axiom cd0: enable deductions from distributive plurals', 'Prolog Axiom
cd1: each of M objects |- each of M-1, ..., each of 2 objects, 1
object'])],
```

Deductions from Generalised Quantifiers. RACE allows deductions involving generalised quantifiers for both collective and distributive plurals if and only if the mathematical relations between the generalised quantifiers are correct. But before we come to this there is an additional aspect of generalised quantifiers to be taken into account. ACE provides five generalised quantifiers that – according to their deductive behaviour – can be split into three groups.

- monotone increasing: *at least N*, *more than N*
- monotone decreasing: *at most N*, *less than N*
- non-monotone: *exactly N*

A generalised quantifier *Q* is called monotone increasing if for all sets *Smaller* and *Larger* with *Smaller* $\subseteq$ *Larger* we have that *Q(Smaller)* entails *Q(Larger)*. Here is an example with the monotone increasing generalised quantifier *at least* where *Smaller = {3 tall men that John sees}* and *Larger= {3 men that John sees}*.

```
prove('John sees at least 3 tall men.', 'John sees at least 3 men.', [],
Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: John sees at least 3 tall men.'], [])]
```

A generalised quantifier *Q* is called monotone decreasing if for all sets *Smaller* and *Larger* with *Smaller* $\subseteq$ *Larger* we have that *Q(Larger)* entails *Q(Smaller)*. Here is an example with the monotone decreasing generalised quantifier *at most* where *Smaller = {3 tall men that John sees}* and *Larger= {3 men that John sees}*.

```
prove('John sees at most 3 men.', 'John sees at most 3 tall men.', [],
Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: John sees at most 3 men.'], [])]
```

While RACE achieves the monotone increasing effect directly, the monotone decreasing effect is created by interpreting *at most N* as *not more than N* and *less than N* as *not at least N* – producing a deductive behaviour similar to the determiner *no*.

The generalised quantifier *exactly N* is neither monotone increasing nor monotone decreasing. RACE interprets *exactly N* as *at least N & at most N*, i.e. as *at least N & not more than N*. This explains why the generalised quantifier *exactly N* allows us only to deduce a set from itself. While the following deduction succeeds

```
prove('John sees exactly 3 men.', 'John sees exactly 3 men.', [],
Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: John sees exactly 3 men.'], [])]
```

this one fails

```
prove('John sees exactly 3 men.', 'John sees exactly 2 men.', [],

WhyNot = ['countable common noun: (at least 2) man'].
```

Finally, one example demonstrating the mathematical properties of generalised quantifiers.

```
prove('At least 3 boys run fast.', 'More than 2 boys run. A boy runs
fast.', [raw], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: At least 3 boys run fast.'], ['Prolog Axiom
cd5: at least M objects |- at least 1, 2, ..., M-1 objects', 'Prolog
Axiom cd6: at least M objects |- more than 1, 2, ..., M-1 objects'])]
```

Summation. RACE performs summations on individuals found in an inheritance tree. Here is a simple example showing that *2 male cats* and *3 tricolor cats* are *5 animals*.

```
prove('There are at least 2 male cats. There are at least 3 tricolor
cats. Every tricolor cat is a female cat. Every male cat is a cat. Every
female cat is a cat. Every cat is an animal. No male cat is a female
cat.', 'There are at least 5 animals.', [raw], Messages, Time,
AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: There are at least 2 male cats.', '2: There are
at least 3 tricolor cats.', '3: Every tricolor cat is a female cat.',
'4: Every male cat is a cat.', '5: Every female cat is a cat.', '6:
Every cat is an animal.', '7: No male cat is a female cat.'], ['Prolog
Axiom agg1: aggregation', 'Prolog Axiom c2: Identity of attributive
adjectives.'])]
```

The proof uses the auxiliary Prolog axiom `agg1` that implements the following algorithm. First, `agg1` extracts the inheritance tree (figure 7) from the axioms

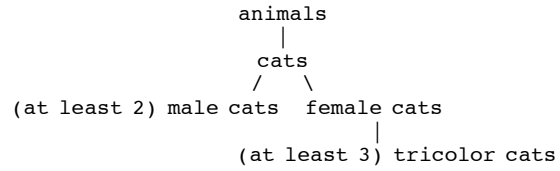

Figure 7. Inheritance tree of cats and animal

checking that the classes involved – *male cats* and *female cats* – are disjoint. Then `agg1` adds the cardinalities of the leaf nodes. The runtime of this algorithm depends on the size of the inheritance tree – for a tree with the height H and the width W the worst-case runtime is $O(H^2 \times W^2)$ – and not on the cardinalities of the leaf nodes. If the relevant classes are not disjoint then `agg1` fails.

Modality. ACE provides modal constructs for *possibility*, *necessity* and *sentence subordination* represented in the DRS language [7] by the three modal operators *diamond* <>, *box []* and *colon :*.

Modal logic can be mapped to first-order predicate logic via the so-called *standard translation*[11] that adds to each logical atom an argument that stands for a *possible world*. Bos [4] extended the standard translation to cover also sentence subordination.

As mentioned in section 4, the possible world argument is added to each logical atom during the translation of DRSs into FOL, meaning that RACE uses possible world semantics even in the absence of modal constructs. This does not seem to have any adverse effect on performance. Possible worlds are connected by a binary accessibility relation that RACE assumes to be *reflexive*, *symmetric* and *transitive*, i.e. an *equivalence relation*. The accessibility relation is expressed as three auxiliary Prolog axioms.

Here are three examples two of which simulate standard axioms of modal logic.

modality axiom: If A then it is possible that A.

```
prove('John waits patiently in the hall.', 'John can wait.', [raw],
Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: John waits patiently in the hall.'], ['Prolog
Axiom pw1: Accessibility relation is reflexive.'])]
```

modality axiom: If A is necessary then it is not possible that not A.

```
prove('John must wait.', 'It is not possible that John cannot wait.',
[raw], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: John must wait.'], ['Prolog Axiom pw1:
Accessibility relation is reflexive.']), proof(['1: John must wait.'],
['Prolog Axiom pw1: Accessibility relation is reflexive.', 'Prolog Axiom
pw2: Accessibility relation is symmetric.'])]
```

[11] https://secure.wikimedia.org/wikipedia/en/wiki/Standard_translation

sentence subordination

```
prove('John sincerely promises to wait for Mary.', 'John promises to
wait.', [raw], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: John sincerely promises to wait for Mary.'],
[])]
```

Copula. In ACE the copula *to be* is used to express

- identity, for instance *John is Harry*.
- class membership, for instance *Every cat is an animal*.
- predication, for instance *The cat is black*. or *John is in the garden*.

To reason correctly, RACE needs to distinguish the various uses of the copula and to support them by appropriate meaning postulates. During the translation into clauses step RACE replaces the verb *be* by variants that express the respective use. Concretely by

- *be_MOD* if the copula *be* is modified by adverbs or prepositional phrases
- *be_ID* if the copula *be* links two proper names or a proper name and *something*/*somebody*
- *be_NP* if the copula *be* links a proper name or a noun phrase to a noun phrase
- *be_ADJ* if the copula *be* links a proper name or a noun phrase to an adjective

For each variant of the copula RACE provides respective meaning postulates in the form of auxiliary Prolog axioms.

The variant *be_NP* can serve as a concrete example. Since *be_NP* is meant to express class membership, it needs to have the attributes of a partial order, i.e. the auxiliary axioms must ensure that *be_NP* behaves as a *reflexive*, *antisymmetric* and *transitive* relation. While reflexivity is trivial, here are examples showing the antisymmetry

```
prove('Every man is a male. Every male is a man.', 'Every man is a
man.', [raw], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: Every man is a male.', '2: Every male is a
man.'], ['Prolog Axiom c1c: Identity of countable objects.']), ...]
```

and the transitivity

```
prove('Every man is a person. Every person is a human.', 'Every man is a
human.', [raw], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: Every man is a person.', '2: Every person is a
human.'], ['Prolog Axiom c1c: Identity of countable objects.'])]
```

Obviously, if the need arose it would also be possible to interpret *be_NP* as an equivalence relation by defining appropriate auxiliary Prolog axioms.

Variations of Query Answering. RACE allows for three different types of queries: *yes/no* queries ask for the existence or non-existence of a specific situation, *wh*-queries (*who*, *whose*, *what*, *which*) ask for noun phrases, and *wh*-queries (*how*, *when*, *where*) ask for adverbs and prepositional phrases. Successful answers to *wh*-queries not only report the axioms used during the proof, but also the substitutions of the query words. For this purpose RACE uses yet another set of auxiliary Prolog predicates.

Here are examples for each of the query types.

```
answer_query('A man sleeps. A woman does not sleep.', 'Is there somebody
who does not sleep?', [], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['2: A woman does not sleep.'], [])]


answer_query('John sees a red book that lies on a table.', 'Who sees
which book?', [raw], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: John sees a red book that lies on a table.',
'Substitution: who = John', 'Substitution: which = (at least 1) book,
(positive of) red'], ['Prolog Axiom w11: If there are countable objects
then the question "which" can be answered.', 'Prolog Axiom w2: If there
are named objects then the question "who" can be answered.'])]


answer_query('Mary sleeps silently. John sleeps with a heavy dream.',
'How does somebody sleep?', [], Messages, RunTime, AxiomsUsed, WhyNot).

AxiomsUsed = [proof(['1: Mary sleeps silently.', 'Substitution:
how/when/where = (positive of) silently'], []), proof(['2: John sleeps
with a heavy dream.', 'Substitution: how/when/where = (with) (at least
1) dream, (positive of) heavy'], [])]


answer_query('Mary sleeps silently. John sleeps in a bedroom.', 'Where
does somebody sleep?', [], Messages, RunTime, AxiomsUsed, WhyNot).

Messages = [message(warning, 'query answering', ''-'', 'The query word
"where" was interpreted as the query word "how".', '')]

AxiomsUsed = [proof(['1: Mary sleeps silently.', 'Substitution:
how/when/where = (positive of) silently'], []), proof(['2: John sleeps
in a bedroom.', 'Substitution: how/when/where = (in) (at least 1)
bedroom'], [])]
```

Note that the last proof generated a warning message. Since ACE does not use thematic roles – that would allow RACE to distinguish location, time, manner, instrument etc. – the *where*- and *when*-queries are interpreted as the less specific *how*-query. As the example shows this can lead to possibly surprising answers.

6 Decidability, Termination, Efficiency, Looping and All That

Decidability of ACE. Pratt-Hartman and Third [5] investigated several fragments of natural language with respect to their complexity and decidability and found that the fragment containing singular quantified nouns, predicative adjectives, copula with/without negation, relative clauses, transitive verbs and reflexive/non-reflexive pronouns as anaphors resolved by co-indexing is undecidable.

This means that ACE is undecidable since already its first-order subset is larger than the above fragment. However, ACE contains some decidable – though less expressive – subsets, for instance the subset defined by its translation into OWL.

Termination of RACE. Since ACE is not decidable, RACE would not terminate for axioms with an infinite model. To prevent this, RACE is equipped with a time-out whose limit depends on the size of the problem. Figure 8 shows a set of axioms – originally devised by Pratt-Hartman and Third [5] and translated into ACE by Mandl [6] – that would generate an infinite model, if not stopped by RACE after 10 s.

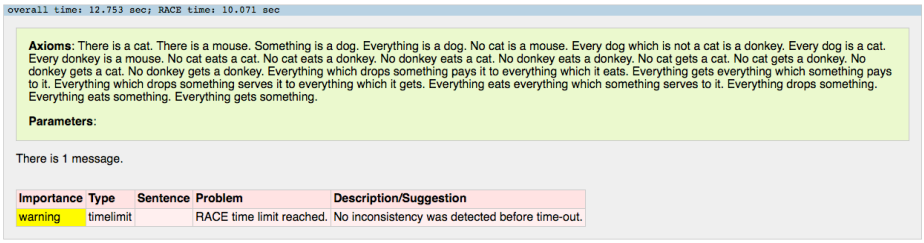

Figure 8. Termination in spite of an infinite model

Efficiency of RACE. If RACE terminates then its run-time can mainly be attributed to the forward-reasoning of clauses, i.e. to unifying logical atoms of clause bodies with logical atoms previously asserted to the Prolog data base. If N is the number of clauses, B is the average number of body atoms and A is the number of atoms asserted to the Prolog data base, then in the worst-case $O(N \times A^B)$ unifications are required. To reduce this number, RACE uses the following means:

- The DRS representation of ACE texts is very compact reducing the number of clauses and the number of body atoms.
- RACE represents auxiliary axioms as Prolog predicates instead of clauses thus reducing the number of clauses, and furthermore replacing eager forward-reasoning by lazy backward-reasoning.
- RACE reduces the number of clauses that at any moment participate in forward-reasoning.
    - RACE blocks after the first round of forward-reasoning the clauses with the body *true* since these clauses are only used once.
    - Before a clause is selected for execution RACE checks whether at least one logical atom of the body of the clause was asserted to the

Prolog database in the preceding round; if not then the clause can certainly not be executed.

- To minimise the number of atoms asserted to the Prolog data base RACE ensures that no logical atom is asserted that would subsume an atom already in the database.
- All remaining unifications profit from Prolog's indexing.

RACE's efficiency and scaling-up behaviour have not yet been systematically investigated. On a MacBook Pro most examples presented in this paper use less than 10 ms, while Lewis Carroll's *Grocer Puzzle*

Given
Every honest and industrious person is healthy. No grocer is healthy. Every industrious grocer is honest. Every cyclist is industrious. Every unhealthy cyclist is dishonest. No healthy person is unhealthy. No honest person is dishonest. Every grocer is a person. Every cyclist is a person.
show that
No grocer is a cyclist.

needs 40 ms. As figure 9 shows the *Grocer Puzzle* executed via the web-client needs an order of magnitude more time because the RACE server runs on an older machine. Note that RACE reports one proof based on five of the original nine axioms.

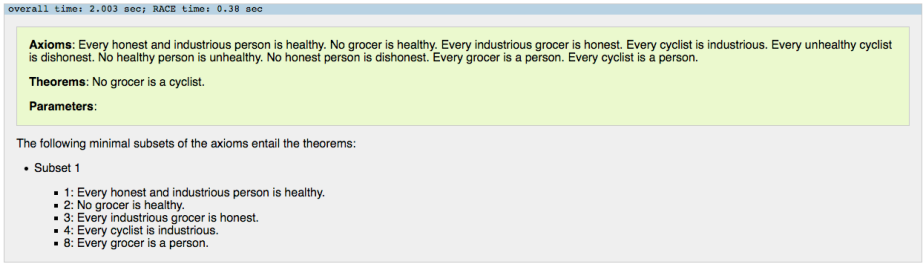

Figure 9. Lewis Carroll's Grocer Puzzle

Loop Prevention. Forward-reasoning systems are prone to loops. Given the clauses

```
satchmo_clause(true, A, ...)
satchmo_clause(A, B, ...)
satchmo_clause(B, A, ...)
```

with the circular definitions *A*, *A => B*, *B => A*, RACE would loop infinitely producing A, B, A, B, ... if it did not contain an effective way of loop prevention. This loop prevention is the effect of functionality introduced for other purposes. RACE uses a subsumption check to prevent asserting the same logical atom twice to the Prolog data base, and selects a clause only if the body of the clause contains a logical atom asserted to the Prolog database in the preceding round of forward-reasoning. Thus in our example RACE prevents the second occurrence of *A* to be asserted to the

database with the result that nothing is asserted and that no clause can be selected in the next round. Figure 10 is an example showing that RACE can reason correctly even in the presence of circular definitions.

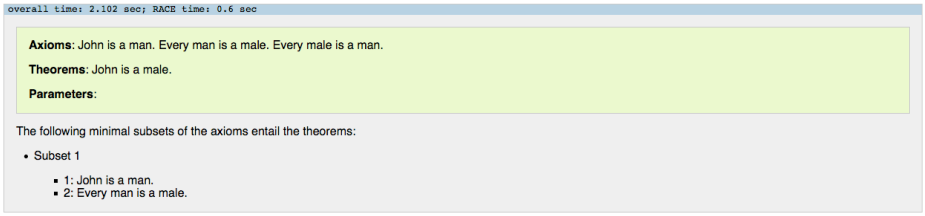

Figure 10. Deduction in the presence of loops

Inconsistent Axioms. From inconsistent axioms it is possible to deduce anything. If RACE detects that the axioms are inconsistent then it outputs the generated results together with a warning message.

... and All That. By default RACE uses the

- Unique name assumption, i.e. different proper names – *Robert*, *Bob*, ... – refer to different entities unless explicitly stated otherwise, for instance by *Robert is Bob*.
- Open world assumption, i.e. missing knowledge of the truth of an ACE sentence *A*, respectively a failed proof of *A*, is not interpreted as ¬ *A*.

7 Conclusions

I presented the reasoner RACE that can show the (in-) consistency of a set of ACE axioms, prove ACE theorems from ACE axioms and answer ACE queries from ACE axioms. In each case RACE gives a proof justification in ACE and full English. Most deductions are supported by auxiliary Prolog axioms that provide meaning postulates and other functionality.

The next version of RACE will cover the still missing language features of ACE, namely arithmetic, formulas, sets, lists and strings. Future extensions of RACE will focus on executable specifications and temporal reasoning. This will also involve larger examples possibly using external knowledge sources.

Furthermore, an open issue will have to be investigated that is due to RACE's flexibility. Obviously, much of the reasoning power of RACE stems from the more than 50 auxiliary Prolog axioms. The flexibility and power of Prolog allows us to craft auxiliary axioms to virtually deduce anything. Thus what we actually can deduce in a given situation raises not only the question of correctness and completeness, but also the question of need and requirement, and consequently the question of the availability of adequate auxiliary axioms.

Here is a case that illustrates this problem. Given that different proper names refer to different entities unless explicitly stated otherwise, RACE could easily be extended by an auxiliary axiom to perform the following two deductions.

Robert is a man. Bob is a man. |– There are two men.

Robert is a man. Bob is a man. Robert is Bob. |– There is one man.

This would introduce non-monotonic reasoning that the current version of RACE does not support. Future research will decide whether non-monotonicity is needed.

Here is another case. While English provides query words for most constituents of a sentence, it does not provide a query word for the verb. To get information on the verb you have to formulate non-specific questions, for instance *What does ... do?*. Given the sentence *John stands at the window and looks into the garden*. the question *What does John do?* can be answered by a number of complete sentences that cover at least part of the state of affairs. To equip RACE with this functionality we need additional auxiliary axioms, and furthermore a standardisation of the questions that can legitimately be asked. Again, future research will show whether this extension of RACE is useful.

Acknowledgements

I very much appreciate the constructive comments and helpful suggestions of the three reviewers of a previous version of this paper. Thanks also go to my colleagues Kaarel Kaljurand, Tobias Kuhn, Ian Pratt-Hartmann, Aarne Ranta and Rolf Schwitter for their valuable feedback.

References

1. Manthey, R., Bry, F.: SATCHMO: A Theorem Prover Implemented in Prolog. In: Lusk, E.L., Overbeek, R.A. (eds.). Proc. CADE 88, LNCS 310, pp. 415—434. Springer (1987)
2. Bry, F., Yahya, A.; Positive Unit Hyperresolution Tableaux and Their Application to Minimal Model Generation; Journal of Automated Reasoning 25: 35–82 (2000)
3. Basin, D., D'Agostino, M., Gabbay, D.M., Matthews, S., Viganò, L. (eds.). Labelled Deduction, Applied Logic Series, vol. 17, Springer (2000)
4. Bos, J.: Computational Semantics in Discourse: Underspecification, Resolution, and Inference. Journal of Logic, Language and Information, vol. 13, pp. 139—157 (2004)
5. Pratt-Hartmann, I., Third, A. More fragments of language. Notre Dame Journal of Formal Logic, vol. 47(2), pp. 151—177 (2006)
6. Mandl, M; Zur Komplexität des Erfüllbarkeitsproblems von Sprachfragmenten; Hauptseminar Kontrollierte Sprache CIS, SS 2010; Universität München (2010)
7. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Discourse Representation Structures for ACE 6.6, Technical Report ifi-2010.0010, Department of Informatics, University of Zurich (2010)
8. Schwertel, U.: Plural Semantics for Natural Language Understanding — A Computational Proof-Theoretic Approach, PhD thesis, University of Zurich (2004)