International Journal of Computer Mathematics
Vol. 00, No. 00, December 2008, 1–15

RESEARCH ARTICLE

An implementation of a dual tableaux system for order-of-magnitude qualitative reasoning

A. Burrieza[a] and A. Mora[b†] and M. Ojeda-Aciego[b‡] and E. Orłowska[c]

[a]*Dept. Filosofía, Univ. Málaga, Spain*; [b]*Dept. Matemática Aplicada, Univ. Málaga, Spain*; [c]*National Institute of Telecommunications, Warsaw, Poland*

(*December 2008*)

Logic programming has been used as a natural framework to automate deduction in the logic of order-of-magnitude reasoning. Specifically, we introduce a Prolog implementation of the Rasiowa-Sikorski proof system associated to the relational translation $Re(OM)$ of the multimodal logic of order-of-magnitude qualitative reasoning OM.

Keywords: Relational theorem proving; Rasiowa-Sikorski procedure; Modal Logic; Tableaux procedure; Automated theorem proving.

AMS Subject Classification: 03B45; 05E10;68T15

1. Introduction

Qualitative reasoning is a broad area of AI whose main aim is to deal with physical systems where either an information of numerical data is not sufficiently precise, or the physical situation cannot be precisely quantified. In any case, qualitative reasoning is a tool which enables us to reason and make inferences from abstractions of quantitative values instead of the numerical values themselves. A variety of applications of qualitative reasoning have been developed, such as robotic navigation [17], spatial reasoning [2], diagnosis of systems [24], reasoning about economic models [14] and many others.

A variant of qualitative reasoning, is the family of order of magnitude representations. In this variant, coarse values are considered as abstractions of precise quantities taken from a totally ordered set of numbers [7, 16, 18, 22]. In this context, two approaches have been identified: Absolute Order of Magnitude (AOM) and Relative Order of Magnitude (ROM). The former stratifies quantities by means of some kind of scale, whereas the latter introduces a family of relations which enable to compare numbers in a different way. In general, both models, AOM and ROM, need to be combined in order to capture all relevant information (see, for example [25]).

Several logics have been defined to deal with qualitative reasoning, such as the Region Connection Calculus [3] for managing qualitative spatial reasoning; or the multimodal logics used in [26] to deal with qualitative spatio-temporal representations.

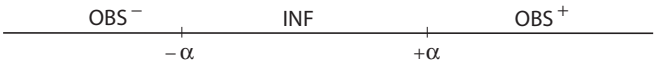
†Partially supported by P06-FQM-02049.
‡Partially supported by TIC06-15455-C03-01. Corresponding author. Email: aciego@uma.es

ISSN: 0020-7160 print/ISSN 1029-0265 online

DOI: 0020716YYxxxxxxxx
http://www.informaworld.com

This paper concentrates on the logic approach to order-of-magnitude qualitative reasoning firstly introduced in [4], and further developed in [5]. These approaches are based on a system with two landmarks and with relations of comparability and negligibility. Roughly speaking, the approach is based on a system with two landmarks, $-\alpha$ and $+\alpha$, which is both simple enough to keep under control the complexity of the system and rich enough so as to permit the representation of a subset of the usual language of qualitative order-of-magnitude reasoning.

A formal representation of systems whose behavior is described up to an order of magnitude usually is based on a linearly ordered set of real numbers with two numbers distinguished as landmarks. The intuitive representation of the underlying frames is given in the picture below, where $-\alpha$ and $+\alpha$ represent respectively the greatest negative observable and the least positive observable, partitioning the real line in classes of positive observable $\textsc{Obs}^+$, negative observable $\textsc{Obs}^-$ and non-observable numbers Inf (note that this choice makes sense, in particular, when considering physical metric spaces in which we always have a smallest unit which can be measured; however, it is not possible to identify a least or a greatest non-observable number).



Consider the following example which illustrates a concept of comparability. Assume one aims at specifying the behavior of a device for automatic control of the speed of a car; assume the system has to maintain the speed close to some speed limit v. For practical purposes, any value in an interval $[v-\varepsilon, v+\varepsilon]$ for small ε is admissible. The extreme points of this interval can then be considered as the landmarks $-\alpha$ and $+\alpha$; on the other hand, the sets $\textsc{Obs}^-$, Inf, and $\textsc{Obs}^+$ can be interpreted as Slow, Ok and High speed.

Regarding negligibility, the representation capabilities of a pocket calculator provide an illustrative example of a relation of that kind. In such a device, it is not possible to represent any number whose absolute value is less than 10^{-99}. Therefore, it makes sense to consider $-\alpha = -10^{-99}$ and $+\alpha = +10^{-99}$ since any number between -10^{-99} and $+10^{-99}$ cannot be either observed or represented. On the other hand, a number x can be said to be negligible with respect to y provided that the difference $y-x$ cannot be distinguished from y. Numerically, and assuming an $8+2$ (digits and mantissa) display, this amounts to state that x is negligible wrt y iff $y - x > 10^8$. Furthermore, this example suggests a real-life model in which, for instance, -1000 is negligible with respect to -1. This is even more suggestive if we interpret the numbers as exponents, since 10^{-1000} certainly can be considered negligible with respect to 10^{-1}.

In [6], the paradigm 'formulas are relations' formulated in [20] was applied to the modal logic for order-of-magnitude (OM) reasoning introduced in [5], obtaining a relational logic $Re(OM)$ based on algebras of relations generated by some relations specific to the frames of OM-logics; after a translation from the language of OM-logics to the language of $Re(OM)$, a deduction system for $Re(OM)$ in the Rasiowa-Sikorski style [23] was presented, paving the way for applicative research on the implementation of the proof procedure. The main contribution of this work consists in the development of a Prolog implementation of the Rasiowa-Sikorski proof procedure introduced in [6]; it is worth noting that our proof system is modular, in that adding new constraints to semantics of the logic implies adding new deduction rules or axiomatic sets, and not implementing a new system from scratch.

In a nutshell, the Rasiowa-Sikorski method (named after a simple and universal deduction mechanism introduced in [23], and which we will denote RS, for short) is a powerful, simple and flexible methodology of developing deduction systems for various non-classical logics based on the analysis of their semantics. Its key idea is to obtain an adequate and complete proof mechanism for the given logic in a systematic way, which is achieved by representing the semantics of all the logical constructs through invertible rules operating on sequences of formulae of the logic.

From the semantic viewpoint, an RS system is dual to the well-known tableau system, as shown in [12]: In order to prove the validity of a complex formula, it is decomposed step by step into a sequence or sequences of simpler formulas, the validity of which is equivalent to that of the original formula. The resulting decomposition tree has the structure of a finitely branching tree with vertices labelled by sequences of formulae; the tree is a proof if it is finite and all its leaves are labelled with so-called "axiomatic-sets" of the logic. To get more insight on the method and on its scope in computer science applications, the interested reader is referred to [13, 21].

The structure of the paper is the following: in Section 2, the OM language and its relational translation $Re(OM)$ are introduced, together with the relational proof system; then, the main contribution of the paper is presented in Section 3, which contains the implementation in Prolog of the relational procedure; in Section 4 we concentrate on a particular aspect of the implementation, the use of phantom variables, which allows for important improvements both in execution time and space requirements; in Section 5, some executions of the Prolog engine are shown, the input formulas are taken from the axiom system for the logic as presented in [5]; Section 6, finally, concludes and presents prospects for future work.

2. The languages OM and $Re(OM)$: the RS proof system

For the sake of making this paper self-contained, let us introduce briefly the syntax and semantics of the underlying language OM. More details and intuitions on the language OM can be found in [5].

In our syntax, we consider three types of modal connectives, each one associated to certain order relation: $\overrightarrow{\Box}$ and $\overleftarrow{\Box}$ to deal with an ordering $<$, the connectives $\overrightarrow{\blacksquare}$ and $\overleftarrow{\blacksquare}$ to deal with a second ordering $\sqsubset$ and the connectives $\overrightarrow{\boxed{n}}$ and $\overleftarrow{\boxed{n}}$ to deal with a third order relation $\prec$ (the specific conditions required on comparability and negligibility relations, $\sqsubset$ and $\prec$, will be stated later).

Syntax of OM:

The alphabet of the language OM is defined by using:

- A stock of atoms or *propositional variables*, $\mathcal{V}$.
- The *classical connectives* $\neg, \wedge, \vee$ and $\rightarrow$ and the constants $\top$ and $\bot$.
- The *unary modal connectives* $\overrightarrow{\Box}$, $\overleftarrow{\Box}$, $\overrightarrow{\blacksquare}$, $\overleftarrow{\blacksquare}$, $\overrightarrow{\boxed{n}}$ and $\overleftarrow{\boxed{n}}$.
- The *constants* α^+ and α^-.
- The *auxiliary symbols*: (,).

Formulas are generated from $\mathcal{V} \cup \{\alpha^+, \alpha^-, \top, \bot\}$ by the construction rules of classical propositional logic adding the following rule: If A is a formula, then so are $\overrightarrow{\Box}A$, $\overleftarrow{\Box}A$, $\overrightarrow{\blacksquare}A$, $\overleftarrow{\blacksquare}A$, $\overrightarrow{\boxed{n}}A$ and $\overleftarrow{\boxed{n}}A$.

Semantics of OM:

As our language is based on a multi-modal approach, its semantics is given by using the concept of frame.

A *frame* for OM is a tuple $\Sigma = (\mathbb{S}, +\alpha, -\alpha, <, \prec)$, where

(1) $(\mathbb{S}, <)$ is a linearly ordered set.
(2) $+\alpha$ and $-\alpha$ are designated points in $\mathbb{S}$ (called *frame constants*) which allow to form the sets $\textsc{Obs}^+$, $\textsc{Inf}$, and $\textsc{Obs}^-$ that are defined as follows:

$$\textsc{Obs}^- = \{x \in \mathbb{S} \mid x \leq -\alpha\}; \qquad \textsc{Inf} = \{x \in \mathbb{S} \mid -\alpha < x < +\alpha\};$$
$$\textsc{Obs}^+ = \{x \in \mathbb{S} \mid +\alpha \leq x\}$$

(3) The negligibility relation $\prec$ is a restriction of $<$, i.e. $\prec \subseteq <$, and satisfies:
  (i) If $x \prec y < z$, then $x \prec z$
  (ii) If $x < y \prec z$, then $x \prec z$
  (iii) If $x \prec y$, then either $x \notin \textsc{Inf}$ or $y \notin \textsc{Inf}$

The comparability relation $x \sqsubset y$ is defined by: $x < y$ and $x, y \in \textsc{Eq}$, where $\textsc{Eq} \in \{\textsc{Inf}, \textsc{Obs}^+, \textsc{Obs}^-\}$.

Given a frame Σ, a *model based on* Σ is an ordered pair $\mathcal{M} = (\Sigma, h)$, where h is a *meaning function* (or, *interpretation*) $h\colon \mathcal{V} \longrightarrow 2^{\mathbb{S}}$.

Any interpretation can be uniquely extended to the set of all formulas in OM (also denoted by h) by means of the usual conditions for the classical boolean connectives and the constants $\top$ and $\bot$, and the following conditions for the modal operators and frame constants:

$$h(\overrightarrow{\Box} A) = \{x \in \mathbb{S} \mid y \in h(A) \text{ for all } y \text{ such that } x < y\}$$
$$h(\overrightarrow{\blacksquare} A) = \{x \in \mathbb{S} \mid y \in h(A) \text{ for all } y \text{ such that } x \sqsubset y\}$$
$$h(\overrightarrow{\boxed{n}} A) = \{x \in \mathbb{S} \mid y \in h(A) \text{ for all } y \text{ such that } x \prec y\}$$
$$h(\overleftarrow{\Box} A) = \{x \in \mathbb{S} \mid y \in h(A) \text{ for all } y \text{ such that } y < x\}$$
$$h(\overleftarrow{\blacksquare} A) = \{x \in \mathbb{S} \mid y \in h(A) \text{ for all } y \text{ such that } y \sqsubset x\}$$
$$h(\overleftarrow{\boxed{n}} A) = \{x \in \mathbb{S} \mid y \in h(A) \text{ for all } y \text{ such that } y \prec x\}$$
$$h(\alpha^+) = \{+\alpha\}$$
$$h(\alpha^-) = \{-\alpha\}$$

The concepts of truth and validity are defined in a usual manner, i.e., a formula A is true in a model $M = (\Sigma, h)$ whenever $h(A) = \mathbb{S}$, and A is OM-valid if it is true in all OM-models.

As stated in the introduction, the language OM was translated in [6] into a relational one in order to benefit from the RS proof procedure. As usual, the main idea of the relational formalisation is to interpret formulas of nonclassical logics as relations which are the elements of algebras of relations from a suitable class.

The syntax of $Re(OM)$:

Again with the goal of maintaining this work self-contained, we recall here the definition of the relational language $Re(OM)$, for more details the reader is suggested to consult [6].

The alphabet of $Re(OM)$ consists of the disjoint sets listed below:

- A (nonempty) set $\mathbb{OV} = \{x, y, z, \dots\}$ of object variables.
- A set $\mathbb{OC} = \{\alpha^-, \alpha^+\}$ of object constants.

- A (nonempty) set $\mathbb{RV} = \{P, Q, R, \dots\}$ of binary relation variables.
- A set $\mathbb{RC} = \{1, 1', \aleph^-, \aleph^+, <, \sqsubset, \prec\}$ of relation constants denoting, respectively, the universal relation, the identity relation, the constant relations being the relational counterparts to the frame constants $-\alpha$ and $+\alpha$, and the three ordering relations related to the three pairs of modalities of the OM language.
- A set $\mathbb{OP} = \{-, \cup, \cap, ;, ^{-1}\}$ of relational operation symbols which are interpreted as the opposite, the union, the intersection, the composition and the inverse of a relation.

Now, the set of relation terms and formulas of $Re(OM)$ is given as follows:

- The set of *relation terms* $\mathbb{RT}$ is the smallest set of expressions that includes all the relational variables and relational constants, and is closed with respect to the operation symbols from $\mathbb{OP}$.
- The set $\mathbb{FR}$ of *formulas*, consists of expressions of the form xRy where x, y denote individual (or object) variables or constants and R is a relational term built from the relational variables and the relational operators.

Semantics of $Re(OM)$:

A model for $Re(OM)$ is a pair $M = (W, m)$ where $W = W' \cup \{-\alpha, +\alpha\}$ for $W' \neq \varnothing$, and m is a meaning function satisfying:

(1) Assigns elements of W to object constants as follows:

a) $m(\alpha^-) = -\alpha$ b) $m(\alpha^+) = +\alpha$

(2) Assigns binary relations on W to relation constants as follows:
For relation constants we should have:

- $m(1) = W \times W$
- $m(1') = \{(w, w) \mid w \in W\}$
- $m(\aleph^-) = \{-\alpha\} \times W$
- $m(\aleph^+) = \{+\alpha\} \times W$

- $m(<)$ is a strict linear ordering in W satisfying $(-\alpha, +\alpha) \in m(<)$
- $m(\sqsubset) = m(<) \cap \big((\textsc{Obs}^- \times \textsc{Obs}^-) \cup (\textsc{Inf} \times \textsc{Inf}) \cup (\textsc{Obs}^+ \times \textsc{Obs}^+)\big)$
- $m(\prec)$ is a restriction of $m(<)$ satisfying the following frame conditions, which mimic items (3.i)–(3.iii) in the definition of a frame for OM:

$$\forall x, \forall y \text{ if } (x, y) \in m(\prec) \text{ and } (y, z) \in m(<), \text{ then } (x, z) \in m(\prec)$$

$$\forall x, \forall y \text{ if } (x, y) \in m(<) \text{ and } (y, z) \in m(\prec), \text{ then } (x, z) \in m(\prec)$$

$$\forall x, \forall y \text{ if } x \in \textsc{Inf} \text{ and } (x, y) \in m(\prec), \text{ then } (+\alpha, y) \in m(< \cup 1')$$

$$\forall x, \forall y \text{ if } x \in \textsc{Inf} \text{ and } (y, x) \in m(\prec), \text{ then } (y, -\alpha) \in m(< \cup 1')$$

(3) Assigns binary relations on W to relation variables.
(4) Assigns operations on binary relations to the relational operation symbols in $\mathbb{OP}$.
(5) Extends homomorphically to the set of terms in the usual manner.

The RS proof system for $Re(OM)$:

We will now concentrate on the presentation of the RS proof system for $Re(OM)$. Let us recall that, given a relational formula xAy, where A may be a compound relational expression, we successively apply *decomposition or specific rules*. In this way a tree is formed whose root consists of xAy and each node (except the root) is obtained by an application of a rule to its predecessor node. The application of rules

is stopped on a node when an *axiomatic set* (a set of formulas whose first order disjunction is valid) has been obtained, or when none of the rules is applicable to the formulas in this node. Such a tree is referred to as a *proof tree* for the formula xAy. A branch of a proof tree is said to be *closed* whenever it contains a node with an axiomatic set of formulas. A tree is closed iff all of its branches are closed.

Our system includes the usual rules for the calculus of binary relations with equality (these rules are not shown explicitly here due to length restrictions, see for instance [6, 19]). Moreover, some new specific rules are included in order to handle the specific object and relation constants of the language $Re(OM)$. These rules are shown in Fig. 1, in which the new variables occurring in the denominator of some rules may be instantiated with any variable, usually one of those occurring in the branch. In the rules, comma is interpreted as first order disjunction of formulas and | is interpreted as branching.

$$\frac{x\aleph^- y}{x1'\alpha^-, x\aleph^- y}\ \textbf{(c1a)} \qquad \frac{x{-}\aleph^- y}{x{-}1'\alpha^-, x{-}\aleph^- y}\ \textbf{(c1b)} \qquad \frac{x\aleph^+ y}{x1'\alpha^+, x\aleph^+ y}\ \textbf{(c2a)} \qquad \frac{x{-}\aleph^+ y}{x{-}1'\alpha^+, x{-}\aleph^+ y}\ \textbf{(c2b)}$$

$$\frac{x < \alpha^+}{x1'\alpha^-, x < \alpha^+}\ \textbf{(c3)} \qquad \frac{x{-}\sqsubset y}{x1'\alpha^-, x{-}\sqsubset y}\ \textbf{(c4)} \qquad \frac{x{-}\sqsubset y}{y1'\alpha^+, x{-}\sqsubset y}\ \textbf{(c5)}$$

$$\frac{x \leq \alpha^-, \alpha^+ \leq x, x- \sqsubset y}{x \leq \alpha^-, \alpha^+ \leq x, x- \sqsubset y, y \leq \alpha^-}\ \textbf{(c6)} \qquad \frac{x \leq \alpha^-, \alpha^+ \leq x, x- \sqsubset y}{x \leq \alpha^-, \alpha^+ \leq x, x- \sqsubset y, \alpha^+ \leq y}\ \textbf{(c7)}$$

$$\frac{\alpha^- \leq x, x- \sqsubset y}{\alpha^- \leq x, x- \sqsubset y, \alpha^- < y}\ \textbf{(c8)} \qquad \frac{x- < y, \alpha^- < y}{x- < y, \alpha^- < y, x- \sqsubset y}\ \textbf{(c9)} \qquad \frac{x- < y, x < \alpha^+}{x- < y, x < \alpha^+, x- \sqsubset y}\ \textbf{(c10)}$$

$$\frac{x \leq \alpha^-, \alpha^+ \leq x, y \leq \alpha^-, \alpha^+ \leq y, x \sqsubset y}{x \leq \alpha^-, \alpha^+ \leq x, y \leq \alpha^-, \alpha^+ \leq y, x \sqsubset y, x < y}\ \textbf{(c11)} \qquad \frac{x{-}\sqsubset y}{x{-}\sqsubset y, x{-}{<}y}\ \textbf{(c12)}$$

$$\frac{}{x < x}\ \textbf{(Iref)} \qquad \frac{}{y{-}{<}x \mid x{-}{<}y \mid x{-}1'y}\ \textbf{(Lin)} \qquad \frac{}{x \sqsubset y \mid x - \sqsubset y}\ \textbf{(cut-}\sqsubset\textbf{)} \qquad \frac{xRy}{xRy, xRz, \mid xRy, zRy}\ \textbf{(Tran)}$$

$$\frac{x < y}{x \prec y, x < y}\ \textbf{(n-0)} \qquad \frac{x \prec z}{x \prec y, x \prec z \mid y < z, x \prec z}\ \textbf{(n-i)} \qquad \frac{x \prec z}{x < y, x \prec z \mid y \prec z, x \prec z}\ \textbf{(n-ii)}$$

$$\frac{\alpha^+ \leq y}{\alpha^- < x, \alpha^+ \leq y \mid x < \alpha^+, \alpha^+ \leq y \mid x \prec y, \alpha^+ \leq y}\ \textbf{(n-iii)}$$

$$\frac{y \leq \alpha^-}{\alpha^- < x, y \leq \alpha^- \mid x < \alpha^+, y \leq \alpha^- \mid y \prec x, y \leq \alpha^-}\ \textbf{(n-iv)}$$

Figure 1. Specific rules for $Re(OM)$

The *axiomatic sets* of $Re(OM)$ are presented below; their occurrence in a node of a branch allows for stopping the further application of the rules to formulas of that branch.

$$\{x1y\} \qquad \{x1'x\} \qquad \{x{-}Ry, xRy\} \qquad \{\alpha^- < \alpha^+\}$$

where $x, y \in \mathbb{OS}$ and $R \in \mathbb{RT}$.

3. Prolog implementation

In this section, we introduce the Prolog implementation of the relational system given above.[1]

3.1 *Translating to a tree of formulas in Re(OM)*

To begin with, the relations have to be encoded as predicates. This is done as follows:

A formula in OM is represented using the Prolog fact $formulaOM(formula)$ where the argument is a well formed formula in OM. A relational formula xRy in $Re(OM)$, where x, y are object variables and R is a relational term, is represented as the Prolog fact $rel(address, R, x, y)$. In this case, the first argument contains a list of integers which defines the position of the node in the proof tree, as it has been generated during the proof process.

Example 3.1 The formulas contained in a leaf of a proof tree are read disjunctively, hence an expression as $xRy \cup xSy \cup x\aleph^{-}y \cup x(\sqsubset;(a;1)^{-})^{-}y$ is translated into the following four facts in Prolog:

```
rel([1],r,x,y).
rel([1],s,x,y).
rel([1],opp(alephm),x,y).
rel([1],opp(comp(sqsub,opp(comp(a,univ)))),x,y).
```

Note that, as Prolog only manipulates text, some symbols are renamed accordingly to its reading. For instance, $\aleph^{-}$ is translated into `alephm`; the composition operator `;` is translated into *comp*, the relation $\sqsubset$ is translated into `sqsub`, etc.

The `omToreom` predicate reads the formulas in OM and renders the set of formulas in $Re(OM)$.

```
omToreom:-
          formulaOM(FormulaOM),
          translate(FormulaOM,FormulaReOM),
          retract(formulaOM(FormulaOM)),
          asserta(rel([1],FormulaReOM,x,y)),
          fail.
omToreom.

translate(alpham,alephm).
translate(or(P,Q),uni(Pnew,Qnew)):-
         translate(P,Pnew),
         translate(Q,Qnew).
translate(implication(P,Q),uni(opp(Pnew),Qnew)):-
         translate(P,Pnew),
         translate(Q,Qnew).
translate(wF(P),comp(<,Pnew)):-
         translate(P,Pnew).
```

The following example outlines a formula in $Re(OM)$ rewritten from OM.

Example 3.2 Consider now the formula $\overleftarrow{\Diamond}(\alpha^{-}) \wedge \overrightarrow{\Box}(\alpha^{-}) \rightarrow \overleftarrow{\boxed{n}}(\alpha^{+} \vee \overleftarrow{\Diamond}(\alpha^{+}))$ in OM (axiom N6 in [4]). The representation of this axiom in Prolog, together with the output of the `omToreom` predicate is given below:

[1]The full implementation (developed in SWI-Prolog Version 5.6.33 for Windows platform) is available from the address `http://homepage.mac.com/alicauchy/`.

```
% Input formula
formulaOM(implication(and(wP(alpham), wF(alphap)),
                      nG(or(alphap, wP(alphap))))).
% Relational version
rel([1], uni(opp(inter(comp(>, alephm), comp(<, alephp))),
         opp(comp(prec, opp(uni(alephp, comp(>, alephp)))))), x, y).
```

Once the system receives as an input the relational formula to be checked, it generates a proof tree, whose leaves contain sets of relational formulas whose disjunctions are to be proved. The input formula gets proved when Prolog closes all the leaves in the proof tree.

The addresses of the open leaves are stored in a list, which is handled by the predicate `open_leaves`. For instance, the predicate `open_leaves([n])` states that it is necessary to prove validity of the set of formulas stored in node `[n]`. As expected, the initial relational terms are valid if and only if all the leaves in the tree can be closed.

In other alternative approaches, such as [9, 10], a Prolog program has been introduced to translate from various logics to the relational formalism. In the same way, a translator from OM to $Re(OM)$ has been developed. The translator described in [9] is generic, so in principle it can be applied to OM. In our paper, we have designed a specific translation for the fixed particular language of OM, thus obtaining a simpler translation. On the other part, in [10], a deductive tool is presented where decomposition rules for the classical operations of the calculus of relations are implemented. In our work, apart for the implementation of these classical rules, there is an implementation of many other rules that are specific for logic OM. These rules characterize properties of the constants corresponding to the landmarks of the assumed order-of-magnitude reasoning model and the properties of the relations of negligibility and comparability of the model. Summarizing, we provide a specific, dedicated system, while those in [9, 10] provide a general framework which in each particular case must be appropriately adapted.

Expressing axiomatic sets and rules

When Prolog detects a relation representing an *axiomatic set*, the corresponding leaf is deleted and the user informed by means of the `remove_leaf` predicate. For instance, if $x1'x$ (`rel(Leaf,equal,X,X)`) occurs in the set of relations of the leaf `Leaf`, it is removed because of the occurrence of an axiomatic set.

```
axiomatic_set:- rel(Leaf,equal,X,X),
                  remove_leaf(Leaf,[rel(Leaf,equal,X,X)]),!.

axiomatic_set:- rel(Leaf,univ,X,Y),
                remove_leaf(Leaf,[rel(Leaf,univ,X,Y)]),!.

axiomatic_set:- rel(Leaf,<,alpham,alphap),
                remove_leaf(Leaf,[rel(Leaf,univ,X,Y)]),!.
```

As noticed in Figure 1, a *rule* in $Re(OM)$ has the following general form: $\frac{\Phi}{\Phi_1|\ldots|\Phi_n}$ where $\Phi_1, \ldots, \Phi_n$ are non-empty sets of formulas and Φ is a finite (possibly empty) set of formulas.

The application of a rule like the previous one to a leaf assumes it is labelled by a set X of formulas satisfying $\Phi \subseteq X$; then, the leaf branches into n new branches, each one with the set of formulas $(X \setminus \Phi) \cup \Phi_i, i = 1, \ldots, n$.

In general, due to the particular nature of the rules of $Re(OM)$, whenever a rule is applicable, it can be applied again on the resulting leaves, but this kind of

behaviour is obviously undesirable. In order to avoid repeated applications of rules against the same formulas each application of a rule is stored in a list.

The implementation of a rule can be roughly stated as follows: firstly, the preconditions (contained in the numerator of the rule) are checked, in order to know whether the rule is applicable; if affirmative, and provided that the rule has not been previously applied against the same arguments, the rule is displayed on the screen and stored as used; finally, the leaf is branched and new labels are attached to each new leaf as stated above.

In order to obtain a rough idea of how a rule is encoded, let us consider the standard rule for the union of relations $\dfrac{x(R \cup S)y}{xRy, xSy}$ (**uni**), its encoding is:

```
uni(Leaf):- rel(Leaf,uni(R,S),X,Y),
            new_deduced_rels([rel(Leaf,R,X,Y),rel(Leaf,S,X,Y)]),
            \+rule_used(Leaf,uni,[rel(uni(R,S),X,Y)]),
            write_rule('Union', [rel(Leaf,uni(R,S),X,Y)],
                                [rel(Leaf,R,X,Y), rel(Leaf,S,X,Y)]),
            update_leaf([rel(Leaf,R,X,Y),rel(Leaf,S,X,Y)]).
```

In order to start explaining the most interesting features of the implementation, let us consider the specific rule (n-i) below:

$$\frac{x \prec z}{x \prec y, x \prec z \mid y < z, x \prec z} \text{ (n-i)} \quad y \text{ any variable}$$

This rule is implemented by using the following code:

```
ni(Leaf):- rel(Leaf,prec,X,Z),
         new_deduced_rels([rel(Leaf,prec,X,Y), rel(Leaf,<,Y,Z)]),
         \+rule_used(Leaf,ni,[rel(prec,X,Z)]),
         any_variable('ni  (prec) ',Leaf,[rel(Leaf,prec,X,Z)],Y),!,
         write_rule('ni (prec) ', [ rel(Leaf,prec,X,Z)],
                                  [ rel(Leaf,prec,X,Y),rel(Leaf,<,Y,Z)]),
         branch(Leaf,2),
         update_leaf(Leaf,2,[[rel(Leaf,prec,X,Y)]
                                 ,[rel(Leaf,<,Y,Z)]]),!.
```

In the first three lines, the rule checks that $x \prec z$ is in the set of relations, that the relations introduced by the rule are new (`new_deduced_rels`) and that the rule has not been previously applied (`rule_used`). Then, note that, as stated in the rule, the variable y in the denominator has to be any of the variables or object constants occurring in the branch (this situation is similar to that of the free tableaux systems, in which the γ rule instantiates a variable by any of the constants occurring in the branch, whereas the δ rule always introduces a new constant). The predicate `any_variable` chooses some constant or variable occurring in the branch (an optimized version of this task is given in Section 4). The predicate `branch(Leaf,2)` branches the current leaf into two new leaves, and copies all the formulas of the current leaf to the two new leaves. The predicate `update_leaf` appends $x \prec y$ to the first leaf and $y < z$ to the second leaf.

The proof procedure

A highly automated implementation of the proof procedure is sketched here. The main predicate in the inference engine is `run_engine`, its implementation is given in Fig. 2; it examines the first leaf of the tree that the proof system needs to check and tries to apply the rules to the formulas containing this leaf.

```
run_engine:-
    leaves(L),
   \+is_list_null(L),
   apply_rules,!.

run_engine:-
    leaves(L),
   \+is_list_null(L),
   try_axioms,!.

run_engine:-
  write(' OK. No leaves in the proof tree.'),
  write(' VALID. '), nl,    !.

apply_rules:-
 first_leaf([FirstLeaf]),
 maplist(select_rule,[FirstLeaf]),!,
  run_engine.

select_rule(FirstLeaf):-
     non_branching_rules(FirstLeaf).

select_rule(FirstLeaf):-
     non_branching_derived_rules(FirstLeaf).

select_rule(FirstLeaf):-
    branching_rules(FirstLeaf).

select_rule(FirstLeaf):-
    explosive_rules(FirstLeaf).

non_branching_rules(FirstLeaf):-
 uni(FirstLeaf)->           axiomatic_set;
 % rest of rules

non_branching_derived_rules(FirstLeaf):-
  greater(FirstLeaf)->      axiomatic_set;
 % rest of rules

branching_rules(FirstLeaf):-
 notuni(FirstLeaf)->        axiomatic_set;
 % rest of rules

explosive_rules(FirstLeaf):-
 ni(FirstLeaf)->            axiomatic_set;
 % rest of rules
```

Figure 2. The engine of the system

As stated previously, the predicate `leaves` stores the leaves which have not been closed so far. If it is a non-null list, then the corresponding nodes must be visited. Then, the predicate `apply_rules` is called recursively as long as the tree has open leaves. The predicate `apply_rules` tries to apply some rule to the given leaf by using the `select_rule` predicate.

The rules are organized into several categories which are ordered as follows: firstly, primitive rules that do not branch a leaf, then derived rules that do not branch a leaf, rules that branch a leaf into several leaves and, finally, explosive rules. The latter contains rules such as transitivity which can generate an exponential number of new branches.

The order in which the engine tries to apply the rules is crucial. The predicate `select_rule` is implemented as a list of rules which are to be tried consecutively. Clearly, the rules which do not generate new branches are at the beginning; among these rules we have some primitive rules (either standard or specific), then some selected derived rules have been implemented directly as primitive, in order to avoid excessively long proofs. Finally, the system tries to apply the rules that generate new branches. Note that the order of rules within a given category is not essential.

Whenever a non-closed leaf does not admit any of the rules in the list, then the system asks the user about considering some cut-like rule (a rule with the empty numerator) by using the predicate `try_cuts`; obviously, the unrestricted use of cut-like rules might generate excessively big trees, thus its use has to be strictly controlled.

```
try_cuts:-
    ...
    write('Try any of the goals --> lin(Leaf,Var1,Var2).'),nl,
    write('                     --> cut(Leaf,Var1,Var2).'),nl,    !.
```

After an application of the procedure, and provided that a closed tree has been obtained, the system provides a list of the rules used in the proof; this is done by the predicate `table_of_used_rules`.

As an example, consider the output obtained from the following relational formula (which corresponds to the Axiom c4 of the system for the logic OM, the

formula $\alpha^- \rightarrow \overrightarrow{\blacksquare} A$, see [5]):

```
rel([1], uni(opp(alephm),opp(comp(sqsub,opp(comp(p,univ))))),x,y).
```

The system traces, in reverse ordering, the rules applied in the process of building a proof tree for the input term:

```
 OK. No more open leaves.
 VALID.
----------------------------        SUMMARY          ------------------------
table_of_used_rules([1], c5, [rel(opp(sqsub), z, x)]).
table_of_used_rules([1], c4, [rel(opp(sqsub), z, x)]).
table_of_used_rules([1], c2b, [rel(opp(alephp), x, y)]).
table_of_used_rules([1], notinverse,[rel(opp(inv(sqsub)), x, z)]).
table_of_used_rules([1], not2, [rel(opp(opp(comp(p, univ))), z, y)]).
table_of_used_rules([1], notcomp,[rel(opp(comp(inv(sqsub), opp(comp(p, univ)))), x, y)]).
table_of_used_rules([1], uni, [rel(uni(opp(alephp), opp(comp(inv(sqsub),
                                                  opp(comp(p, univ))))), x, y)]).

leaves([]).
```

4. Phantom variables: postponing the choice

There are several rules in the relational system for $Re(OM)$ which exhibit the same behavior that Rule (n-i) regarding an introduction of new variables. We saw that the rule branches the leaf into two new leaves, and appends $x \prec y$ to the first leaf and $y < z$ to the second leaf, where y is "*any variable*" occurring in the branch. In principle, we have as many different instantiations of the rule as the values that can be chosen for y. If we do not take this into account, the proof tree might grow in an uncontrolled manner.

The naive approach to the implementation of this behaviour is to allow the user to introduce a particular variable when a rule is being applied. If, eventually, the formula could not be proved, the system would have to return to the previous leaf and, then, choose another variable. Obviously, when the number of variables existing in the leaf is big, the system performance will be poor, to say the least.

Anyway, a much smarter solution is possible if we introduce a non-instantiated variable (so-called "phantom variable") and delay its actual instantiation until we get a guarantee that, by a unification process, an axiomatic set will be generated. Thus, a *phantom variable* is a special variable whose possible instantiations are constrained to belong to the set of variables or constants occurring in the leaf.

An optimized implementation of predicate `any_variable`, as in rule (n-i) above, encodes the introduction of the phantom variable `Y`, whose instantiation is delayed until a suitable value would be chosen in order to generate an axiomatic set.

```
any_variable('ni  (prec) ',Leaf,[rel(Leaf,prec,X,Z)],Y)
```

If this choice fails and the branch will not close, then the system backtracks and undoes the instantiation. This way, the growth of the tree is controlled.

For instance, recall that if the axiomatic set $\alpha^- < \alpha^+$ is present in a leaf, then the leaf will be closed; as a result, $X < \alpha^+$ will be an axiomatic set provided that X is a phantom variable which can be instantiated as α^-. This particular case is illustrated in the piece of code below:

```
axiomatic_set:-
         rel(Leaf,<,X,alphap),
         is_variable_phantom(X),
         substitute([X],[alpham]),
         remove_leaf(Leaf,[rel(Leaf,<,X,alphap)]),!,
         axiomatic_set.
```

In addition, the implementation introduces a predicate (`interactivemode`) which allows to switch between a fully automated handling of phantom variables and an interactive mode. In the latter, the procedure can stop in order to get feedback from the user as to an appropriate choice of a value to be used in the instantiation.

When an application of a rule requires the introduction of a variable already occurring on the current branch, and we are in the interactive mode, the system provides a list of formulas occurring in that branch and the user can choose an adequate variable for instantiation. Anyway, the user can refuse to provide feedback and let the system to introduce a phantom variable. In automatic mode, `any_variable` always returns a new phantom variable that is used in the application of the rule.

5. Experimental results and examples

As the relational proof procedure was proved to be complete in [6], the first choice of formulas to prove with the implementation has been the set of axioms of the system given in [5]. The implementation has been tested against all the axioms in the system[1] with the result that every axiom has been automatically proved.

In this section we comment in detail on the performance of the implementation on the relational translation of two specific axioms of OM.

Example 5.1 Let us consider the formula $\alpha^- \rightarrow \overrightarrow{\blacksquare} A$, corresponding to Axiom c4 from [5]. Its relational translation is

$$x(-\aleph^- \cup -(\sqsubset; -(A; 1)))y$$

which, in turn, is translated into Prolog as:

```
rel(1,uni( opp(alephm), opp(comp(sqsub,opp(comp(a,univ)))),x,y) .
```

which is stored in the file `reomAxiomc4.pl`. Now, the program is called to satisfy the predicate:

```
?engine('reomAxiomc4.pl','logc4.txt').}
```

The following report in logc4.txt file is returned:

```
------>Input file: reomAxiomc4.pl
THE ENGINE IS RUNNING
--->opp composition Rule
[rel(1, opp(comp(sqsub, opp(comp(a, univ)))), x,y)]
------------------------------------------------------------
[rel(1, opp(sqsub), x, z), rel(1,opp(opp(comp(a, univ))), z, y)]

---->c1b (opp aleph-)  Rule
[rel(1, alephm, x, y)]
------------------------------------------------
[rel(1, equal, x, alpham), rel(1, alephm, x, y)]

---->c4 (notsqsubset)  Rule
[rel(1, opp(sqsub), x, z)]
------------------------------------------------
[rel(1, equal, x, alpham), rel(1, opp(sqsub), x,z)]
```

[1]The full trace of execution of the procedure applied on all the axioms of [5] can be obtained from the address `http://homepage.mac.com/alicauchy/`.

```
Found axiomatic set. Branch: 1
    - Axiomatic set: [rel(1, opp(equal), x, alpham),
                      rel(1, equal, x, alpham)]
    - Deleted relations in branch 1
 OK. No more open leaves.
```

The following example illustrates branching of a proof tree and a use of phantom variables.

Example 5.2 Let us consider the formula $\overleftarrow{\Diamond}\alpha^- \vee \alpha^- \vee \overrightarrow{\Diamond}\alpha^-$, corresponding to Axiom (c1) from [5]. Its relational translation is

$$x((>;\aleph^+) \cup \aleph^+ \cup (<;\aleph^+))y$$

which has the following form in Prolog:

```
rel([1], comp(>, alephp), x, y).
rel([1], alephp, x, y).
rel([1], comp(<, alephp), x, y).
```

Now, the program is called to satisfy the predicate:

```
?engine('reomAxiomc1.pl','logc1.txt').}
```

After applying some rules, the system detects the possibility of using a phantom variable; in the interactive mode, the following information is displayed on the screen:

```
Information for user
 - Relations in [1]
rel([1], alephp, x, y).
rel([1], comp(<, alephp), x, y).
rel([1], equal, x, alphap).
rel([1], equal, alphap, x).
rel([1], comp(>, alephp), x, y).

 Now, we can apply the following rules:
---->comp Rule
[rel([1], comp(>, alephp), x, y)]
------------------------------------------------------------
rel(new_leaf1, >, x, var) | rel(new_leaf2, alephp, var, y)

where var can be either:
   - any variable from: [x, y]
   - or alpham or  alphap.
We can use a non-instantiated variable (phantom).
Introduce the desired var or 0 for phantom variable.
```

Now, the user can either introduce any of the possible values, or let the system introduce a phantom variable. In this example, the system is always said to introduce phantom variables (which are denoted as `t1`, `t2`, etc). Thus, the log file of this example continues as follows:

```
|: 0
---->comp Rule
[rel([1], comp(>, alephp), x, y)]
------------------------------------------------
rel([1, 1], >, x, t1) | rel([1, 2], alephp, t1, y)
```

The system continues applying rules automatically until a new composition

(`comp`) rule is applied. Note that, in the leaf (1,1,2) we would obtain an axiomatic set if `t2` is substituted by `alphap`.

```
---->comp Rule
[rel([1, 1], comp(<, alephp), x, y)]
------------------------------------------------------------
rel([1, 1, 1], <, x, t2) | rel([1, 1, 2], alephp, t2, y)

Substitute in all relations variable phantom:t2 by alphap
```

This instantiation provides an extra piece of information which allows eventually to close all the open branches of the proof tree. More details can be seen in the demos available in the web.

6. Conclusions and future work

We have presented an implementation in Prolog of the relational proof system for the logic of qualitative order-of-magnitude reasoning. The system has been tested against the axiom system provided in [5], and all the axioms of the system have been automatically proved.

Further work is planned on studying the relationship with other approaches [8], as well as in developing relational proof systems for variants of OM logic and on implementation of the specific rules for OM within the latter system, as already done in [11].

As future work, the implementation will be improved in several directions. The main point to be investigated arises when the system does not close the proof tree, since some cut-like rule might be needed to continue the expansion; the goal would be to detect situations in which the system can automatically suggest the user to choose some of these rules. Despite of the improvement in automatization provided by the use of phantom variables, our next aim is to enhance the interaction with the user during the proof process whenever the automatic mode does not allow to close the tree. Moreover, the graphical aspect of the interface should be enhanced, allowing the user to specify directly the requirements by using the standard OM logic, which is more intuitive than its relational translation into $Re(OM)$.

Another topic for future work is to study the computational complexity of the validity problem in OM. There are some results in the literature which give some initial clues: on the one hand, [15] proves that validity is coNP-complete for all finitely axiomatizable tense logics of linear time flows. As our logic is a conservative extension of the type of logics studied above, we get a lower bound for its complexity. Moreover, OM includes nominals and a negligibility relation; regarding nominals, in [1] we have that the inclusion of nominals in logics whose class of frames is that of strict total orderings does not increase the complexity. As a result, a possible increase over coNP-completeness should be related solely to the behaviour of the negligibility relation.

7. Acknowledgements

The authors acknowledge the anonymous referees for providing valuable suggestions on how to improve the final version of this paper.

This work has been partially funded by the Spanish Ministry of Science and Innovation through grant TIN06-15455-C03-01 and by the Andalusian Government through grant P06-FQM-02049.

References

[1] C. Areces, P. Blackburn, and M. Marx. *The computational complexity of hybrid temporal logics*. Logic Journal of the IGPL 8(2000):653–679.
[2] C. Bailey-Kellogg and F. Zhao. *Qualitative spatial reasoning: extracting and reasoning with spatial aggregates*, AI Magazine, Vol. 24, No. 4 (2003), pp. 47–60
[3] B. Bennett. *Modal logics for qualitative spatial reasoning*. Bull. of the IGPL, 3(1995):1–22.
[4] A. Burrieza and M. Ojeda-Aciego, *A multimodal logic approach to order of magnitude qualitative reasoning*, Lect. Notes in Artificial Intelligence, 3040 (2004), pp. 431–440.
[5] A. Burrieza and M. Ojeda-Aciego. *A multimodal logic approach to order of magnitude qualitative reasoning with comparability and negligibility relations*, Fundamenta Informaticae, 68 (2005), pp. 21–46.
[6] A. Burrieza, M. Ojeda-Aciego, and E. Orłowska. *Relational approach to order-of-magnitude reasoning*, Lect. Notes in Computer Science, 4342 (2006), pp. 105-124.
[7] P. Dague. *Numeric reasoning with relative orders of magnitude*. In Proc. 11th National Conference on Artificial Intelligence, pages 541–547. The AAAI Press/The MIT Press, 1993.
[8] J. Dallien and W. MacCaull. *RelDT—a dual tableaux system for relational logics*, Available from `http://logic.stfx.ca/reldt/`, 2005.
[9] A. Formisano, E. Omodeo, and E. Orłowska. *A PROLOG tool for relational translation of modal logics: A front-end for relational proof systems*, In: B. Beckert (ed) TABLEAUX 2005 Position Papers and Tutorial Descriptions, Universität Koblenz-Landau, Fachberichte Informatik 12 (2005), pp. 1-10. System available from `http://www.di.univaq.it/TARSKI/transIt/`
[10] A. Formisano, E. Omodeo, and E. Orłowska. *An environment for specifying properties of dyadic relations and reasoning about them II: Relational presentation of non-classical logics*, Lect. Notes in Artificial Intelligence, 4342 (2006), pp. 89-104.
[11] J. Golińska-Pilarek, A. Mora and E. Muñoz-Velasco. *An ATP of a Relational Proof System for Order of Magnitude Reasoning with Negligibility, Non-closeness and Distance*. Lecture Notes in Computer Science 5351(2008):128–139
[12] J. Golińska-Pilarek and E. Orłowska. *Tableaux and dual tableaux: Transformation of proofs*, Studia Logica, 85(3) 2007, pp. 283–302.
[13] B. Konikowska. *Rasiowa-Sikorski deduction systems in computer science applications*, Theoretical Computer Science 286 (2002), pp. 323–366.
[14] K.R. Lang, A.B. Whinston, and A. Hinkkanen. *A Set-Theoretic Foundation of Qualitative Reasoning and its Application to the Modeling of Economics and Business Management Problems*. Information Systems Frontiers, Vol. 5, No. 4 (2003), pp. 379–399.
[15] T. Litak and F. Wolter. *All finitely axiomatizable tense logics of linear time flows are coNP-complete*. Studia Logica. 75(2005): 1–13.
[16] M.L. Mavrovouniotis and G. Stephanopoulos. *Reasoning with orders of magnitude and approximate relations*. In Proc. 6th National Conference on Artificial Intelligence. The AAAI Press/The MIT Press, 1987.
[17] R. Müller, T. Röfer, A. Lankenau, R. Musto, K. Stein, and A. Eisenkolb. *Coarse qualitative descriptions in robot navigation*, Lecture Notes in Artificial Intelligence 1849 (2000), pp. 265–276
[18] P. Nayak. *Order of Magnitude Reasoning Using Logarithms*. Proc. of Principles of Knowledge Representation and Reasoning (1992), pp. 201–210.
[19] E. Orłowska. *Relational interpretation of modal logics*, In H. Andreka, D. Monk, and I. Nemeti, editors, Algebraic Logic, Colloquia Mathematica Societatis Janos Bolyai, 54 (1988), pp. 443–471.
[20] E. Orłowska. *Relational semantics for nonclassical logics: Formulas are relations*, In J. Wolenski, editor, Philosophical Logic in Poland, Kluwer, 1994, pp. 167–186.
[21] E. Orłowska and J. Golińska-Pilarek. *Dual tableaux and their Applications*. Draft of a book, 2008.
[22] O. Raiman. *Order of magnitude reasoning*. Artificial Intelligence. 51(1991): 11–38.
[23] H. Rasiowa and R. Sikorski. *Mathematics of Metamathematics*, Polish Scientific Publishers, (1963).
[24] S. Subramanian and R. Mooney. *Multiple-Fault Diagnosis Using General Qualitative Models with Fault Models*. Working Notes of the IJCAI-95 Workshop on Engineering Problems for Qualitative Reasoning, pp. 321-325, 1995.
[25] L. Travé-Massuyès, F. Prats, M. Sánchez, and N. Agell. *Relative and absolute order-of-magnitude models unified*. Annals of Mathematics and Artificial Intelligence 45(2005):323–341
[26] F. Wolter and M. Zakharyaschev. *Qualitative spatio-temporal representation and reasoning: a computational perspective*. In G. Lakemeyer and B. Nebel (eds), Exploring Artificial Intelligence in the New Millenium. Morgan Kaufmann, 2002.