Proceedings of the International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2008 13–17 June 2008.

Implementing a relational system for order of magnitude reasoning[*]

A. Burrieza[1], **A. Mora**[2], **M. Ojeda-Aciego**[2] **and E. Orłowska**[3]

[1] *Dept. Filosofía. , Univ. Málaga, (Spain)*

[2] *Dept. Matemática Aplicada., Univ. Málaga, (Spain)*

[3] *National Institute of Telecommunications, Warsaw, (Poland)*

emails: burrieza@uma.es, amora@ctima.uma.es, aciego@ctima.uma.es, orlowska@itl.waw.pl

Abstract

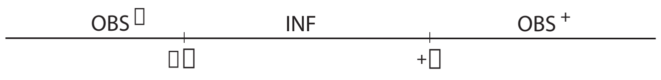
This work concentrates on the automated deduction of logics of order-of-magnitude reasoning. Specifically, a Prolog implementation is presented for the Rasiowa-Sikorski proof system associated to the relational translation $Re(OM)$ of the multimodal logic of qualitative order-of-magnitude reasoning OM.

Key words: Relational theorem proving, Rasiowa-Sikorski procedure.

1 Introduction

This paper concentrates on the logic approach to order-of-magnitude qualitative reasoning firstly introduced in [1], and further developed in [2]. 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.

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 OBS^+, negative observable OBS^- and non-observable numbers INF:



[*]Partially supported by TIC06-15455-C03-01 and P06-FQM-02049

In [3], the paradigm 'formulas are relations' formulated in [9] was applied to the modal logic for order-of-magnitude (OM) reasoning introduced in [2], 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 [10] 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 [3]; it is worth to note that our proof system is modular, in that adding new semantic constrainsts to the logic implies adding new deduction rules or axiomatic sets, and not implementing a new system from scratch.

The structure of the paper is the following: in Section 2, the language $Re(OM)$ is 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, some executions of the Prolog engine are shown, the input formulas are taken from the axiom system for the logic as presented in [2]; Section 5, finally, concludes and presents prospects for future work.

2 The language $Re(OM)$: the RS proof system

As stated in the introduction, the language OM was translated in [3] into a relational one in order to take 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. For limitation of the length of the paper, we reveal only the language $Re(OM)$.

We recall here the definition of The syntax of $Re(OM)$ [1].

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 for $-\alpha$ and $+\alpha$, and the three ordering relations related to the three 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:

[1]We show the syntax and for more details the reader is suggested to consult [3].

- 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.

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* (which denotes a tautological formula) 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 considers 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]). 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 denote *any* variable occurring in the branch.

The *axiomatic sets* of $Re(OM)$ shown below state valid formulas of the system which allows for stopping the procedure on a given 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 of the relational system

In this section, we introduce the Prolog implementation[2] of the relational system given above.

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

To begin with, the relations have to be encoded as predicates. This is done as follows: A relational formula xRy, where x, y are object variables and R is a relational term represented as the Prolog fact:

$$rel(address, R, x, y)$$

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

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

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

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

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

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

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

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

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

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

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

The first argument contains a list of integers which define the position of the node in the proof tree, as it has been generated during the proof process.

Example 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:*[3]

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

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 the validity of the set of relations 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.

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,

[3]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 operator $\sqsubset$ is translated into `sqsub`, etc.

if either $x1'x$ (`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)]),!.
```

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 ys consider a specific (non-standard) 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 three first 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 constant object 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 3). 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 `copyToLeaves` appends $x \prec y$ to the first leaf and $y < z$ to the second leaf.

The proof procedure

The implementation of a full and automated proof procedure is roughly sketched here. The inference engine examines the first leaf of the tree that the proof system needs to check and tries to apply the rules to the relations containing this leaf. As stated previously, the predicate `open_leaves` stores the leaves which has not been closed so far. The inference engine tries to apply some rule to the given leaf, while the tree has open leaves.

The order in which the engine tries to apply the rules is crucial. 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.

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 without relations in the numerator).

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 $\alpha^- \to \overrightarrow{\blacksquare} A$, see [2]):

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

The system traces, in reverse ordering, the rules applied in order to close the tree for the input term:

```
 OK. No more open leaves. VALID.
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)]).
```

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 the new variables introduced. 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 values can be chosen for y. If we do not take this into account, the proof tree might grow in an uncontrolled manner.

We introduce a non-instantiated variable (so-called "phantom variable") and delay its actual instantiation until we have some guarantees, by a unification process, that it will generate axiomatic sets. Thus, a *phantom variable* is a special case of variable whose possible instantiations are constrained to belong to the set of variables or constants occurring in the leaf.

The use of phantom variables is crucial for an adequate performance of the implementation, although it initially implied the need to rewrite the code for the axiomatic sets in order to make them parameterized. 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 by α^-.

4 Experimental results and examples

As the relational proof procedure was proved to be complete in [3], the first choice of formulas to prove with the implementation has been the set of axioms of the system given in [2]. The implementation has been tested against all the axioms in the system[4] with the result that every axiom has been automatically proved. This is an important matter, since so far no result about the decidability of $Re(OM)$ has been obtained.

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

Example 2 *Let us consider the formula $\alpha^- \to \overrightarrow{\blacksquare} A$, corresponding to Axiom c4 from [2]. Its relational translation is*

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

which, in turn, is translated into Prolog as:

[4]The full trace of execution of the procedure applied on all the axioms of [2] can be obtained from the address `http://homepage.mac.com/alicauchy/`.

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

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)]

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 is more complete than the previous one, as it branches the proof tree and, in addition, uses phantom variables.

Example 3 *Let us consider the formula $\overleftarrow{\Diamond}\alpha^- \vee \alpha^- \vee \overrightarrow{\Diamond}\alpha^-$, corresponding to Axiom c1 from [2]. Its relational translation is*

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

which in Prolog has the following form:

```
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 one phantom variable, the following information is displayed on the screen:

```
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. □

5 Conclusions and future work

We have presented a first 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 [2], and all the axioms of the system have been automatically proved. This is an important matter, since so far no result about the decidability of $Re(OM)$ has been obtained.

As future work, the implementation will be improved in several directions. On the one hand, we want to add more interaction with the user during the proof process. When the system does not close the proof tree, some cut-like rule might be needed and the user should be asked to provide some clue on this although, in some situations, it is possible for the system to suggest the use of some of these rules. On the other hand, 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 $Re(OM)$.

References

[1] A. Burrieza and M. Ojeda-Aciego. A multimodal logic approach to order of magnitude qualitative reasoning. *Lect. Notes in Artificial Intelligence*, 3040:431–440, 2004.

[2] A. Burrieza and M. Ojeda-Aciego. A multimodal logic approach to order of magnitude qualitative reasoning with comparability and negligibility relations. *Fundamenta Informaticae*, 68:21–46, 2005.

[3] A. Burrieza, M. Ojeda-Aciego, and E. Orłowska. Relational approach to order-of-magnitude reasoning. *Lect. Notes in Computer Science*, 4342:105-124, 2006.

[4] J. Dallien and W. MacCaull. RelDT—a dual tableaux system for relational logics, 2005. Available from `http://logic.stfx.ca/reldt/`

[5] 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 No 12, 2005, 1-10. System available from `http://www.di.univaq.it/TARSKI/transIt/`

[6] J. Golińska-Pilarek and E. Orłowska. Tableaux and dual tableaux: Transformation of proofs. *Studia Logica* 85(3):283–302, 2007.

[7] B. Konikowska. Rasiowa-Sikorski deduction systems in computer science applications. *Theoretical Computer Science* 286:323–366, 2002

[8] E. Orłowska. Relational interpretation of modal logics. In H. Andreka, D. Monk, and I. Nemeti, editors, *Algebraic Logic*, volume 54 of *Colloquia Mathematica Societatis Janos Bolyai*, pages 443–471. North Holland, 1988.

[9] E. Orłowska. Relational semantics for nonclassical logics: Formulas are relations. In J. Wolenski, editor, *Philosophical Logic in Poland*, page 167–186. Kluwer, 1994.

[10] H. Rasiowa and R. Sikorski. *Mathematics of Metamathematics.* Polish Scientific Publishers, 1963.