International Journal of Computer Mathematics
Vol. 00, No. 00, August 2009, 1–13

RESEARCH ARTICLE

Implementing a Relational Theorem Prover for Modal Logic K

Mora, A.[1], Muñoz-Velasco, E.[1] and Golińska-Pilarek, J.[2]
[1] Dept. Applied Mathematics. University of Málaga. Spain.
{amora, emilio}@ctima.uma.es
[2] Institute of Philosophy, Warsaw University.
National Institute of Telecommunications. Poland.
j.golinska-pilarek@itl.waw.pl
(*Received 00 Month 200x; in final form 00 Month 200x*)

An automatic theorem prover for a proof system in the style of dual tableaux for the relational logic associated with modal logic K has been introduced. Although there are many well known implementations of provers for modal logic, as far as we know, it is the first implementation of a specific relational prover for a standard modal logic. There are two main contributions in this paper. First, the implementation of new rules, called (k_1) and (k_2), which substitute the classical relational rules for composition and negation of composition in order to guarantee not only that every proof tree is finite but also to decrease the number of applied rules in dual tableaux. Second, the implementation of an order of application of the rules which ensures that the proof tree obtained is unique. As a consequence, we have implemented a decision procedure for modal logic K. Moreover, this work would be the basis for successive extensions of this logic, such as T, B and S4.

Keywords: Relational Logic; Modal Logic; Dual Tableau Methods; Implementation theorem provers.

1. Introduction

Implementation of theorem provers are required in many areas of computer science. They are required for performing the four major reasoning tasks: verification of validity, verification of entailment, model checking, and verification of satisfaction. Relational proof systems in the style of Rasiowa-Sikorski, called *dual tableaux*, are powerful tools for dealing with all these tasks. The system of the basic relational logic provides the common relational core of all dual tableaux. Therefore, for each particular theory we need only to expand the basic relational logic with specific relational constants and/or operators satisfying the appropriate axioms, and then we design specific rules corresponding to given properties of a logic and we adjoin them to the core set of the rules. Dual tableau systems have been constructed for many non-classical logics [6, 9, 11, 12, 16, 17, 25–27].

The election relational systems has many advantages [22]. Namely, it provides a clear-cut method of generating proof rules from the semantics and the resulting deduction system is well suited for automated deduction purposes. Moreover, it provides a standard and intuitively simple way of proving completeness and it enables an almost automatic way of transforming a complete dual tableau proof tree into a complete Gentzen calculus proof tree. Furthermore, for each particular theory we need only to expand the basic relational logic with specific relational constants and/or operators satisfying the appropriate axioms, then we design specific rules corresponding to given properties of a logic and we adjoin them to the core set of the rules.

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

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

In this paper, we introduce an automatic theorem prover, called $RePML_K$, for a relational proof system in the style of dual tableaux for the relational logic associated with standard modal logic K, given in [15]. As far as we know, it is the first implementation of a relational theorem prover for a standard modal logic. However, there are some related works in this subject. For example, an implementation of the proof system for the classical relational logic is described in [8] which could be used for modal logic, but as it considers the classical relational rules, it is not well suited for modal logic K. On the other hand, in [20] there have been proved many theorems of relational algebras. In [10], an implementation of translation procedures from non-classical logics to relational logic is presented. Moreover, in [7, 14] there are implementations of relational logics for order of magnitude reasoning.

Many provers have been designed which can deal with modal logic. Very optimised ones like MSPASS [9] and FaCT [8]; generic logical frameworks like Isabelle [29]; and some offer users the possibility to create a new prover, like LWB [19], LoTReC [13] LeanTAP [5] and TWB [2]. As said in [3], although efficiency is an important aspect, depending on the intended application, other qualities can be as important, such as portability, construction of counter-models, user-friendliness, or small size. In this line, $RePML_K$ has been developed in Prolog and tries to take advantage of the powerful capabilities of this language: fast prototyped, modular, and extensible to other modal logics. In fact, our prover could be useful for educational applications, i.e. where their purpose is as tools to teach theory proof. For example, by using its trace mode, it explains step by step the full process of the proof, indeed, it is very intuitive to see in every step which rule has been applied and how it works. Furthermore, there is an option which asks the user what rule should be applied in each step.

Our aim is to design a prover which applies the relational rules in a predefined order, without using any *external* strategy such as backtracking, loop-checking, etc. For example, in the case of general tableaux, if the current node is composed by $\Diamond a \wedge \Diamond b \wedge \Box\neg b$. Then choosing $\Diamond a$ first will erroneously lead to an open tableau. But choosing $\Diamond b$ first will give a closed tableau. The tableaux prover will backtrack over the choices of $\Diamond$-formulas to avoid this problem. In our case, in a similar situation (see example 3.2 below) we will obtain directly a closed tree because our rules consider all the choices of $\Diamond$-formulas in the same step, that is, we could say that our prover makes a breadth-first search. As we only apply the rules of the dual tableaux, the soundness and completeness of our dual tableaux, proved in [15], is also the soundness and completeness of our implementation.

There are two main contributions in this paper. First, the implementation of new rules, called (k_1) and (k_2), which substitute the classical relational rules for composition and negation of composition in order to guarantee not only that every proof tree is finite but also to decrease the number of applied rules in dual tableaux. This improvement comes from the fact that the classical relational rule for composition may be applied infinitely many times if a formula with composition appears in a branch, and the negation of composition rule introduces a new variable in each step [8]; our rules not introduce branching and their application for sets of formulas with many compositions and negation of composition at least may contribute to decrease the length of the proof. Second, the definition of an order of application of the rules which ensures that the proof tree obtained is unique. As a consequence of both contributions, we can say that $RePML_K$ is an implementation of a decision procedure for modal logic K. Moreover, this work would be the basis for successive extensions of modal logic K, such as T, B and S4, where we will have to modifiy/substitute the rules (k_1) and (k_2) for suitable ones for each type of logic. Finally, the dual-tableau presented is somewhat close to labelled tableau

[23], to sequent calculi [24] and to PDL [1]. However, as far a we know, our rules (k_1) and (k_2) have not any similar in these approaches.

The paper is organized as follows. In section 2, we present the relational proof system for modal logic K. In section 3, we present the prover $RePML_K$ on the basis of some examples. Finally, some conclusions and prospects of future work are presented in section 4.

2. Relational proof system for modal logic K

In this section, we sketch the construction of a relational proof system in the style of dual tableaux for the relational logic associated with standard modal logic K, presented in [15].

First of all, we define the relational logic, RL_K, appropriate for expressing formulas of the modal logic K. The language of the relational logic RL_K consists of the symbols from the following pairwise disjoint sets: $\mathbb{OV} = \{z_0, z_1, \ldots\}$, a countable infinite set of object variables; $\mathbb{RV} = \{S_1, S_2, \ldots\}$, a countable infinite set of relational variables; $\{R\}$, the set consisting of the relational constant R representing the accessibility relation from K-models; $\{-, \cup, \cap, ;\}$, the set of relational operations.

The set of relational terms, $\mathbb{RT}$, is the smallest set which includes $\mathbb{RV}$ and satisfies: If $P, Q \in \mathbb{RT}$, then $-P, P \cup Q, P \cap Q, (R; P) \in \mathbb{RT}$. RL_K-formulas are of the form $z_i T z_j$, where z_i, z_j are object variables and T is any relational term.

An RL_K-*model* is a structure $\mathcal{M} = (U, R, m)$, where U is a non-empty set, R is a binary relation on U, and m is a meaning function satisfying: $m(S) = X \times U$, where $X \subseteq U$, for every relational variable S; $m(R) = R$, i.e., R is the interpretation of the relational constant R; m extends to all the compound relational terms as follows: $m(-P) = (U \times U) - m(P)$; $m(P \cup Q) = m(P) \cup m(Q)$; $m(P \cap Q) = m(P) \cap m(Q)$; $m(R; P) = \{(x, y) \in U \times U : \exists z \in U\,((x, z) \in R \wedge (z, y) \in m(P))\}$. Let $\mathcal{M} = (U, R, m)$ be an RL_K-model. A *valuation* in $\mathcal{M}$ is any function $v \colon \mathbb{OV} \to U$.

An RL_K-formula $z_i T z_j$ is *satisfied* in an RL_K-model $\mathcal{M}$ by a valuation v, $\mathcal{M}, v \models z_i T z_j$ whenever $(v(z_i), v(z_j)) \in m(T)$. A formula is *true* in $\mathcal{M}$ whenever it is satisfied by all the valuations in $\mathcal{M}$, and it is RL_K-*valid* whenever it is true in all RL_K-models.

The translation of K-formulas into relational terms starts with a one-to-one assignment of relational variables to the propositional variables, denoted by τ'. Then the translation τ of formulas is defined inductively as follows: $\tau(p) = \tau'(p)$, for any propositional variable $p \in \mathbb{V}$; $\tau(\neg\varphi) = -\tau(\varphi)$; $\tau(\varphi \vee \psi) = \tau(\varphi) \cup \tau(\psi)$; $\tau(\varphi \wedge \psi) = \tau(\varphi) \cap \tau(\psi)$; $\tau(\langle R \rangle \varphi) = (R; \tau(\varphi))$. The desired result of preservation of validity via translation of formulas of modal logic into relational terms is as follows.

Theorem 2.1: *For every* K*-formula* φ*,* φ *is* K*-valid iff* $z_1 \tau(\varphi) z_0$ *is* RL_K*-valid.*

We present now the relational proof system for our logic given in [15]. First of all, we introduce an *order* of relational terms and relational formulas which will be useful in the rest of the paper. We define the *length* of a relational term T, $l(T)$, given by: $l(S) = 0$, for every relational variable S; $l(-T) = l(T) + 1$; $l(T \# T') = l(T) + l(T') + 1$, for $\# \in \{\cup, \cap\}$ and $l(R; T) = l(T) + 1$. For $i = 0, 1$ and a relational term T, we define: $-^i T \stackrel{\text{def}}{=} \begin{cases} T, \text{ if } i = 0 \\ -T, \text{ if } i = 1 \end{cases}$.

The *type* of a relational term is defined as follows. If a relational term is of the form $-^i S_j$, for $i = 0, 1$ and for some relational variable S_j, then it is said of type $(-^i S)$. If a relational term is of the form $--T$, for some relational term T, then it is said to be of type $(-)$. A relational term is said to be of type $(-^i \#)$,

$\# \in \{\cup, \cap\}$, (resp. $(-^i;)$) whenever it is of the form $-^i(P\#Q)$ (resp. $-^i(R;P)$), for some relational terms P and Q. We define a strict linear order $<$ on a finite set of types as follows: $(S) < (-S) < (-) < (\cup) < (-\cap) < (\cap) < (-\cup) < (;) < (-;)$. The type of a relational term T is denoted by $t(T)$.

Now, we define inductively an ordering $<$ on the set of all relational terms, $\mathbb{RT}$. We say that $T < T'$ if and only if either of the following possibilities holds:

(1) $t(T) < t(T')$, or

(2) T and T' are of the same type and $l(T) < l(T')$, or

(3) T and T' are of the same type and of the same length and satisfy either of the following: $j < k$, if $T = -^i S_j$ and $T' -^i S_k$, for some relational variables S_j, S_k and $i = 0, 1$; or $P < P'$, if $T = --P$ and $T' = --P'$ or $T = -^i(R;P)$ and $T' = -^i(R;P')$, for some relational terms P and P' and $i = 0, 1$; or $P < P'$ or both $P = P'$ and $Q < Q'$, if $T = -^i(P\#Q)$ and $T' = -^i(P'\#Q')$, for some relational terms P, P', Q, Q', $i = 0, 1$, and $\# \in \{\cup, \cap\}$.

We extend the ordering $<$ to all $\mathsf{RL_K}$-formulas as follows: $z_{k_1} T z_{k_2} < z_{l_1} T' z_{l_2}$ whenever either of the following conditions is satisfied: $k_1 < l_1$; or $k_1 = l_1$ and $T < T'$; or $k_1 = l_1$ and $T = T'$ and $k_2 < l_2$.

Let X be a finite set of $\mathsf{RL_K}$-formulas, let $\# \in \{\cup, \cap\}$, and let $i = 0, 1$. A formula φ of type $(-^i\#)$ (resp. $(-^i;)$) is said to be *minimal with respect to* X *and* $(-^i\#)$ (resp. $(-^i;)$) whenever for every formula $\psi \in X$ of the same type as φ, $\varphi < \psi$. Similarly, a formula φ of type $(-)$ is said to be *minimal with respect to* X *and* $(-)$ whenever for every formula $\psi \in X$ of type $(-)$, $\varphi < \psi$.

Relational proof systems are determined by the axiomatic sets of formulas and rules which apply to finite sets of relational formulas. The rules have the following general form: $(*)\ \dfrac{X \cup \Psi}{X \cup \Phi}$ or $(**)\ \dfrac{X \cup \Psi}{X \cup \Phi_1 \mid X \cup \Phi_2}$, where $X, \Psi, \Phi, \Phi_1, \Phi_2$ are finite non-empty sets of formulas such that $X \cap \Psi = \emptyset$. A rule of the form (**) is a branching rule. In a rule, the set above the line is referred to as its *premise* and the set(s) below the line is (are) its *conclusion(s)*. A rule of the form $(*)$ (resp. $(**)$) is *applicable* to a finite set Y if and only if $Y = X \cup \Psi$ and $\Phi \not\subseteq Y$ (resp. $\Phi_1 \not\subseteq Y$ or $\Phi_2 \not\subseteq Y$), that is an application of a rule must introduce a new formula. A *new variable* is a variable which appears in the conclusion of one rule but does not appear in its premise.

Decomposition rules of $\mathsf{RL_K}$-dual tableau are $(\cup)$, $(\cap)$, $(-\cup)$, $(-\cap)$, $(-)$, (k_1), and (k_2) of the following forms:

For every $k \geq 1$ and for all relational terms P and Q,

$$(\cup)\ \frac{X \cup \{z_k(P \cup Q)z_0\}}{X \cup \{z_k P z_0, z_k Q z_0\}} \qquad (\cap)\ \frac{X \cup \{z_k(P \cap Q)z_0\}}{X \cup \{z_k P z_0\} \mid X \cup \{z_k Q z_0\}} \qquad (-)\ \frac{X \cup \{z_k{-}{-}Pz_0\}}{X \cup \{z_k P z_0\}}$$

$$(-\cup)\ \frac{X \cup \{z_k{-}(P \cup Q)z_0\}}{X \cup \{z_k{-}Pz_0\} \mid X \cup \{z_k{-}Qz_0\}} \qquad (-\cap)\ \frac{X \cup \{z_k{-}(P \cap Q)z_0\}}{X \cup \{z_k{-}Pz_0, z_k{-}Qz_0\}}$$

For all $k, l, m \geq 1$ and for all relational terms P_i, Q_j, $1 \leq i \leq j$, $1 \leq j \leq m$,

$$(\mathsf{k}_1)\ \frac{X \cup \{z_k{-}(R;Q_1)z_0, \ldots, z_k{-}(R;Q_m)z_0\}}{X \cup \{z_{k_1}{-}Q_1 z_0, \ldots, z_{k_1+(m-1)}{-}Q_m z_0\}}$$

$$(\mathsf{k}_2)\ \frac{X \cup \{z_k(R;P_i)z_0\}_{i\in\{1,\ldots,l\}} \cup \{z_k{-}(R;Q_1)z_0, \ldots, z_k{-}(R;Q_m)z_0\}}{X \cup \{z_{k_1}P_i z_0, \ldots, z_{k_1+(m-1)}P_i z_0\}_{i\in\{1,\ldots,l\}} \cup \{z_{k_1}{-}Q_1 z_0, \ldots, z_{k_1+(m-1)}{-}Q_m z_0\}}$$

provided that $z_k(R;T)z_0 \notin X$ and $z_k{-}(R;T')z_0 \notin X$ for all terms T and T' and $k < k_1$ and k_1 is the minimum natural number such that z_{k_1} is a new variable.

The specific rule of $\mathsf{RL_K}$-dual tableau is of the form: For all $k \geq 1$, $j \geq 0$ and for every relational variable S, (right) $\dfrac{X \cup \{z_k S z_j\}}{X \cup \{z_k S z_l, z_k S z_j\}}$ provided that l is the minimum natural number such that z_l occurs in $X \cup \{z_k S z_j\}$ and $l \neq j$ and $z_k S z_l \notin X$.

Notice that rules (k_1) and (k_2) have been presented separately only for clarity. If we admit that some of the sets of formulas can be empty, rule (k_1) could be eliminated. In this case, we should say what sets in the premise and in the conclusion of (k_2) could be empty. On the other hand, rule (right) has been introduced in order to ensure the completeness of our prover. For details, see proof of Proposition 2.5 in [15].We could reduce the number of rules if we use only formulas in negation normal form, as in [5]. However, we give this more general approach in order to be able to extend our prover for logics which might not have involutive negation in a future work.

A finite set of $\mathsf{RL_K}$-formulas is said to be an $\mathsf{RL_K}$-*axiomatic set* whenever it is a superset of $\{z_kPz_j, z_k{-}Pz_j\}$, for some object variables z_k, z_j and for some relational term P. Let z_1Tz_0 be an $\mathsf{RL_K}$-formula.

An $\mathsf{RL_K}$-*proof tree of* z_1Tz_0 is a tree with the following properties: the formula z_1Tz_0 is at the root of this tree; each node except the root is obtained by an application of a rule to its predecessor node; the rules are applied with the following ordering: ($-$), ($\cup$), ($-\cap$), ($\cap$), ($-\cup$), (right), (k_1), and (k_2); a node does not have successors whenever its set of formulas is an $\mathsf{RL_K}$-axiomatic set or none of the rules is applicable to its set of formulas. A branch of an $\mathsf{RL_K}$-proof tree is *closed* whenever it contains a node with an $\mathsf{RL_K}$-axiomatic set of formulas. An $\mathsf{RL_K}$-proof tree is closed if and only if all of its branches are closed. An $\mathsf{RL_K}$-formula z_1Tz_0 is $\mathsf{RL_K}$-*provable* whenever there is a closed $\mathsf{RL_K}$-proof tree of it, which is then referred to as its $\mathsf{RL_K}$-*proof.*

The following result ensures the equivalence between validity of a K-formula and provability in our relational system.

Theorem 2.2: (Relational Soundness and Completeness of K) *For every* K*-formula* φ*, we have that* φ *is* K*-valid iff* $z_1\tau(\varphi)z_0$ *is* $\mathsf{RL_K}$*-provable.*

It is easy to prove that our system terminates, because the maximal modal degree always decreases after application of rules (k_1) and (k_2). Moreover, it is a decision procedure because the order of application of the rules ensures that the proof tree for every formula is unique. For details, see [15].

3. The implementation of $RePML_K$

In the previous section, we have presented a new proof system based on relational dual tableaux for modal logic K. The decision procedure developed in the theoretical framework has improved the rules and the engine of the prover, and the result is a new ATP, called $RePML_K$ [1]. In this section, we summarize how $RePML_K$ works in three levels: representation of the formulas, rules of the new proof system, and significant enhancements in the engine of the prover. From now on, we will work with the relational translation modal formulas as explained in the previous section. A formula is represented as the Prolog fact: $rel([1], T, z_1, z_0)$. In node [1] it stores the formula z_1Tz_0.
Prolog knows the leaf in which it must apply any rule, because the predicate `leaves([[1,...,1],..., [1,...,k]])` stores the leaves that the tool must close. Prolog will try to satisfy the relations in the leaf nodes. If the tool can close all the leaves in the tree, then *formula* is valid. As said above, rules of $\mathsf{RL_K}$ have the following general form:

$$(*)\,\frac{X \cup \Psi}{X \cup \Phi} \quad \text{or} \quad (**)\,\frac{X \cup \Psi}{X \cup \Phi_1 \mid X \cup \Phi_2}$$

[1]The full implementation (developed in SWI-Prolog Version 5.6.33 for Windows and Mac platforms) is available from the address http://files.getdropbox.com/u/1639661/Klogicv2.zip where can be seen outputs for different formulas introduced to the prover.

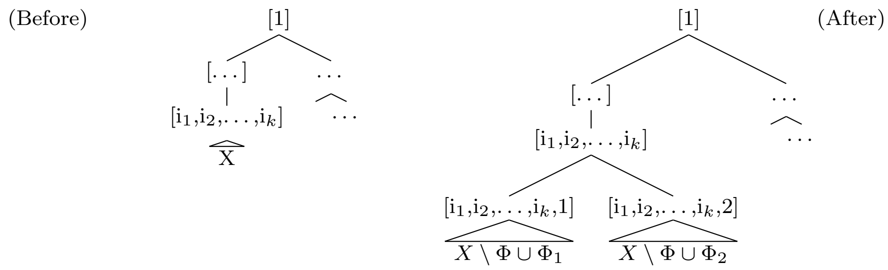

Figure 1. Division of a leaf of the tree.

Now, we explain how our prover works when $(**)$ is applicable to a set of formulas $Y = X \cup \Psi$, the case of $(*)$ is similar. If Y appears in the leaf $[i_1, i_2, \ldots, i_k]$, the system divides this leaf in 2 new leaves, labeled as $[i_1, i_2, \ldots, i_k, 1]$ and $[i_1, i_2, \ldots, i_k, 2]$ by copying $X \cup \Phi_1$ to the node $[i_1, i_2, \ldots, i_k, 1]$, and $X \cup \Phi_2$ to the node $[i_1, i_2, \ldots, i_k, 2]$ (see Figure 1).

We have translated the rules for $\mathsf{RL_K}$ to clauses in Prolog[1] and now, we outline the implementation of the powerful new rules (k_1) and (k_2). These rules are related to the composition and opposite composition relations. The predicate `k1_or_k2` will call the (k1) rule when in a set of formulas, there is an opposite composition formula with any variables but none composition formula with a similar pattern of variables. And if both patterns exist then the (k2) rule will be called.

```
k1_or_k2(IdLeaf,rel(Leaf,opp(comp(r,Q)),Zk, Z0)):-
          \+rel(IdLeaf,comp(r,_),Zk, Z0),
           k1(IdLeaf,rel(IdLeaf,opp(comp(r,Q)),Zk, Z0)),!.
k1_or_k2(IdLeaf,rel(IdLeaf,opp(comp(r,Q)),Zk, Z0)):-
          rel(IdLeaf,comp(r,_),Zk, Z0),
          k2(IdLeaf,rel(IdLeaf,opp(comp(r,Q)),Zk, Z0)),!.
```

When Prolog tries to apply the (k_1) rule in the leaf selected by the engine (depth first search), it checks if in `IdLeaf` leaf any formula matches with `rel(IdLeaf,opp(comp(r,Q)),Zk,Z0)` and in the same `IdLeaf` leaf that none formula matches with `rel(IdLeaf,comp(r,_), Zk, Z0)`. Then Prolog searches a list of formulas `ListRels` using `allOppCompRels` predicate, with the pattern `rel(IdLeaf,opp(comp(r,Qi)),Zk,Z0)` $(i = 1 \ldots n)$ for the instantiated variables `Zk,Z0`. If this rule has not been applied previously in `IdLeaf` for these variables (`rule_used`), then the list of formulas `rel(IdLeaf,opp(Qi)),Zki,Z0)` (being `Zki` new variables) are deduced and stored in `ListRelsDeduced` using `add_relsK1` predicate. Also, the order of the formulas of the `IdLeaf` leaf is held using `actualize_list_rels_ordered` that insert the deduced formulas in the adequate place, and `ListRelsDeduced` formulas are added to node `IdLeaf` with the predicate called `add_list_of_relations`.

```
k1(IdLeaf,rel(IdLeaf,opp(comp(r,Q)),Zk, Z0)):-
         \+rel(IdLeaf,comp(r,_),Zk, Z0),
          allOppCompRels(rel(IdLeaf,opp(comp(r,Q)),Zk, Z0), ListRels),
         \+rule_used(Leaf,k1, ListRels),!,
          add_relsK1(ListRels,ListRelsDeduced),
          write_rule('k1 ', ListRels, ListRelsDeduced),
          actualize_list_rels_ordered(IdLeaf,ListRelsDeduced),
          add_list_of_relations(ListRelsDeduced),!.
```

Now, we focus our attention in the rule (k_2). When Prolog matches any formula in `IdLeaf` leaf with `rel(IdLeaf,opp(comp(r,Q)), Zk, Z0)` and any formula with `rel(IdLeaf, comp(r,P) ,Zk,Z0)`, (k_2) decomposition rule can be applied. Then Prolog searches a list of formulas `ListOppcomRels` with the pattern `rel(IdLeaf, opp(comp(r,Qi)), Zk,Z0)` $(i = 1 \ldots m)$ using `allOppCompRels` predicate, and a list of formulas `ListComRels` with the pattern `rel(Leaf, comp(r, Pj), Zk, Z0)`

[1] In [14], an explanation of the common rules for modal logics (union, intersection, etc.) is available.

($j = 1 \ldots n$) using `allCompRels` predicate. Finally, if this rule has not been applied in the node `IdLeaf`, then the list of formulas with `rel(IdLeaf,Pj, Zki,Z0)` and `rel(IdLeaf, opp(Qi), Zki,Z0)` (`ListRelsDeduced`) are deduced where `Zki` are new variables using `add_relsK2` predicate. In the same way that (k_1) rule, `ListRelsDeduced` formulas are added to node `IdLeaf` and the order of the relations of the IdLeaf leaf is held using `actualize_list_rels_ordered`.

```
k2(Leaf,rel(IdLeaf,opp(comp(r,Q)), Zk, Z0)):-
          rel(IdLeaf,comp(r,P), Zk, Z0),
          allOppCompRels(rel(IdLeaf,opp(comp(r,Q)), Zk, Z0), ListOppcomRels),
          allCompRels(rel(IdLeaf,comp(r,P), Zk, Z0), ListComRels),
          append(ListOppcomRels, ListComRels,ListRelsK),
          \+rule_used(IdLeaf,k2, ListRelsK),
          add_relsK2(ListOppcomRels, ListComRels,ListRelsDeduced),
          write_rule('k2 ', ListRelsK, ListRelsDeduced),
          actualize_list_rels_ordered(IdLeaf,ListRelsDeduced),
          add_list_of_relations(ListRelsDeduced),!.
```

The next problem is how $RePML_K$ selects the adequate rule for the list of relational formulas in a leaf of the tree. We implement the ordering defined in section 2 as follows. First, we define the length of a relational term using `lengthGreater, lengthEqual, lengthRel` predicates:

```
lengthGreater(T1,T2):-
       lengthRel(T1,Length1),
       lengthRel(T2,Length2),
       Length1 > Length2,!.
lengthEqual(T1,T2):-
       lengthRel(T1,Length1),
       lengthRel(T2,Length2),
       Length1 = Length2,!.
lengthRel(rel(_, Term,_,_),0):-
    atom(Term),!.
```

```
...
lengthRel(rel(_,inter(P,Q),_,_),Length2):-
   lengthRel(rel(_,P,_,_),Lengthp),
   lengthRel(rel(_,Q,_,_),Lengthq),
   Length1 is Lengthp+Lengthq,
   Length2 is Length1+1,!.
lengthRel(rel(_,comp(r, T),_,_),Length1):-
   lengthRel(rel(_, T,_,_),Length),
   Length1 is Length+1,!.
```

Then, as we have described in section 2, the strict linear order $<$ on a finite set of types is defined by the `typeGreater, typeEqual, linearorderTypes` predicates as follows:

```
typeGreater(Term1, Term2):-
       linearorderTypes(Term1,NumTerm1),
       linearorderTypes(Term2,NumTerm2),
         NumTerm1 > NumTerm2,!.
typeEqual(X,H):-
       linearorderTypes(Term1,NumTerm1),
       linearorderTypes(Term2,NumTerm2),
         NumTerm1=NumTerm2,!.
```

```
linearorderTypes(rel(_,Term,_,_),1):-
        atomic(Term).
...
linearorderTypes(rel(_,inter(_,_),_,_),8).
linearorderTypes(rel(_,comp(_,_),_,_),9).
linearorderTypes(
           rel(_,opp(comp(_,_)),_,_),10).
```

And now, we implement the ordering $<$ on the set of all relational terms given above, by using the `relationalTermGreather` predicate. A relational term is greater than another one, if the type of the first one is greater than the type of the second one; or they have the same type, and the length of the first one is greater than the length of the second one; or they have the same type and the same length, and the relational variables of the first one are greater than relational variables of the second one (in terms of the lexicographic order).

We extend this order to all relational formulas by using the `relationalFormGreather` predicate as follows. A formula is greater than another one if the first variable of this formula is greater than the first variable of the second formula; or both formulas have the same first variable and the relational term of the first one is greater than the relational term of the second one; or the first variable and the relational term are equal and the second variable of the first formula is greater than the second variable of the second formula.

```
relationalTermGreather(rel(Leaf,R1,Zi1,Zj1), rel(Leaf,R2,Zi2,Zj2)):-
```

```
        typeGreater(rel(Leaf,R1,Zi1,Zj1), rel(Leaf,R2,Zi2,Zj2)),!.
relationalTermGreather(rel(Leaf,R1,Zi1,Zj1), rel(Leaf,R2,Zi2,Zj2)):-
        typeEqual(rel(Leaf,R1,Zi1,Zj1), rel(Leaf,R2,Zi2,Zj2)),
        lengthGreater(rel(Leaf,R1,Zi1,Zj1), rel(Leaf,R2,Zi2,Zj2)),!.
relationalTermGreather(rel(Leaf,R1,Zi1,Zj1), rel(Leaf,R2,Zi2,Zj2)):-
        typeEqual(rel(Leaf,R1,Zi1,Zj1), rel(Leaf,R2,Zi2,Zj2)),
        lengthEqual(rel(Leaf,R1,Zi1,Zj1), rel(Leaf,R2,Zi2,Zj2)),
        relationalvariableGreather(rel(Leaf,R1,Zi1,Zj1), rel(Leaf,R2,Zi2,Zj2)),!.
relationalvariableGreather(rel(Leaf,R1,Zi1,Zj1), rel(Leaf,R2,Zi2,Zj2)):-
         rel(Leaf,R1,Zi1,Zj1) @> rel(Leaf,R2,Zi2,Zj2).
relationalFormGreather(rel(Leaf,_,Zi1,_), rel(Leaf,_,Zi2,_)):- compare(>,Zi1,Zi2),!.
relationalFormGreather(rel(Leaf,R1,Zi,Zj1), rel(Leaf,R2,Zi,Zj2)):-
        relationalTermGreather(rel(Leaf,R1,Zi,Zj1), rel(Leaf,R2,Zi,Zj2)),!.
relationalFormGreather(rel(Leaf,_,Zi,Zj1), rel(Leaf,_,Zi,Zj2)):- compare(>,Zj1,Zj2),!.
```

To obtain a sort of the list of the relations of a leaf, we use the classical *quick sort algorithm* which uses the ordering defined previously. The list of relations ordered (`LRO`) is stored in `rels2List` fact.

```
order_List_Rels(Leaf):-
   initializeList(rels2List(Leaf,_)),
   rels2list(Leaf),
   retract(rels2List(Leaf, LR)),
   relationalQuick_sort(LR,LRO),
   asserta(rels2List(Leaf, LRO)).
relationalQuick_sort(List,S):-
  q_sort(List,[],S),!.
q_sort([],A,A).
q_sort([H|T],A,S):-
       pivoting(H,T,L1,L2),
       q_sort(L1,A,S1),q_sort(L2,[H|S1],S).
pivoting(_,[],[],[]).
pivoting(H,[X|T],[X|L],G):-
         relationalFormGreather(X,H),
         pivoting(H,T,L,G).
pivoting(H,[X|T],L,[X|G]):-
         \+relationalFormGreather(X,H),
         pivoting(H,T,L,G).
```

Now, we show the engine of $RePML_K$. The main predicate in the inference engine is `rlk_proof_tree` that is executed recursively until all the leaves are closed. This predicate calls to `execute_rules_guided_by_order` which sorts the formulas of the first leaf and applies the rules using this ordering. If a rule add new formulas to a leaf, the engine of the prover adds these relations in the adequate position in order to preserve the ordering defined previously.

```
rlk_proof_tree:-
    leaves(L),
   \+is_list_null(L),
   execute_rules_guided_by_order,!.
rlk_proof_tree:-
   write(' VALID. All leaves closed').
execute_rules_guided_by_order:-
 first_leaf([FirstLeaf]),
 order_List_Rels(FirstLeaf),
 apply_rules_in_order(FirstLeaf),!,
 press_a_key,!,
 rlk_proof_tree.
execute_rules_guided_by_order:-
 rlk_proof_tree.
apply_rules_in_order(FirstLeaf):-
 rels2List(FirstLeaf,[]),!.
apply_rules_in_order(FirstLeaf):-
 rels2List(FirstLeaf,[FirstRel|_]),
 linearorderTypes(FirstRel,Num),
 apply_one_rule(FirstLeaf,FirstRel,Num),!,
 apply_rules_in_order(FirstLeaf),!.

apply_one_rule(FirstLeaf,FirstRel,2):-
 not2(FirstLeaf,FirstRel)->axiomatic_set,!.
...
apply_one_rule(FirstLeaf,FirstRel,10):-
 k2(FirstLeaf,FirstRel)->axiomatic_set,!.
```

While the tree has opened leaves, `rlk_proof_tree` is recursively called. If all leaves are closed, then the system informs to the user that the proof is finished and it is possible to trace (`used_rules` predicate) what rules have been applied. The engine of $RePML_K$ uses the mechanism of pattern machine of Prolog to detect if exists an axiomatic set in any leaf of the tree. In this case, it deletes the corresponding leaf and informs to the user.

```
axiomatic_set:-
         rel(NumLeaf,R,Zi,Zj),rel(NumLeaf,opp(R),Zi,Zj),
         remove_leaf(NumLeaf,[rel(NumLeaf,R,Zi,Zj),rel(NumLeaf,opp(R),Zi,Zj)]),!.
```

Example 3.1 In this example, the modal formula $\Diamond p \rightarrow \Box(\Box\neg q \vee \Diamond q)$ stored in 'twb3.pl' is satisfied by $RePML_K$ with the Prolog predicates:

```
? toReom('twb3.pl','reomtwb3.pl'),
  run('reomtwb3.pl','logreomtwb3.txt').
```

The predicate (`toReom`) translate the modal formula to a relational formula, then the predicate (`prove`) call the inference engine of $RePML_K$. The following report in logreomtwb3.txt is returned:

```
Select:
  ->  0 for trace mode and ouput by screen,
  ->  1 for no trace mode and ouput by screen,
  ->  2 for trace mode and ouput by file,
  ->  3 for no trace mode and ouput by file,
  ->  4 for configuration,
  ->  another character for full mode and output by file,
|: 2
 :::::  Input file: c:\myprograms\logickv2\axiomsreom\reomtwb3.pl
 :::::  Ejemplo 3 de twb
 :::::  http://twb.rsise.anu.edu.au/modal_logic_k_0
 :::::  Input en TWB:  ~ ( <> p & [] ~ p )

formulaKLogic(not(and(diamond(p),square(not(p))))).

 :::::  Translated automatically to the following relational formula:

RELATIONS ORDERED IN LEAF [1]
rel([1], opp(inter(comp(r, p), opp(comp(r, opp(opp(p)))))), x, y)

[rel([1], opp(inter(comp(r, p), opp(comp(r, opp(opp(p)))))), x, y)]
____________________________________________________________      Opposite Intersection Rule
[rel([1], opp(comp(r, p)), x, y), rel([1], opp(opp(comp(r, opp(opp(p))))), x, y)]

RELATIONS ORDERED IN LEAF [1]
rel([1], opp(opp(comp(r, opp(opp(p))))), x, y)
rel([1], opp(comp(r, p)), x, y)

 :::::  NEXT   STEP. Press Enter key to continue.

[rel([1], opp(opp(comp(r, opp(opp(p))))), x, y)]
_____________________________________      2 Not Rule
[rel([1], comp(r, opp(opp(p))), x, y)]

RELATIONS ORDERED IN LEAF [1]
rel([1], comp(r, opp(opp(p))), x, y)
rel([1], opp(comp(r, p)), x, y)

 :::::  NEXT   STEP. Press Enter key to continue.

[rel([1], opp(comp(r, p)), x, y), rel([1], comp(r, opp(opp(p))), x, y)]
____________________________________________________     k2  Rule
[rel([1], opp(p), a1, y), rel([1], opp(opp(p)), a1, y)]

RELATIONS ORDERED IN LEAF [1]
rel([1], opp(opp(p)), a1, y)
rel([1], opp(p), a1, y)

 :::::  Axiomatic set (close leaf): [1]
[rel([1], opp(opp(p)), a1, y), rel([1], opp(p), a1, y)]

 :::::  NEXT   STEP. Press Enter key to continue.
 :::::  Variables used: [a1, x, y]
 :::::  VALID. Total Rules applications: 3
used_rules([1], notinter, [rel(opp(inter(comp(r, p), opp(comp(r, opp(opp(p)))))), x, y)])
used_rules([1], not2, [rel(opp(opp(comp(r, opp(opp(p))))))])
used_rules([1], k2, [rel(opp(comp(r, p)), x, y), rel(comp(r, opp(opp(p))), x, y)])
```

Notice that after the application of the k2 rule, the engine of $RePML_K$ detects an axiomatic set and all the leaves of the tree are finally closed. Then the formula is valid and a trace of the rules applied is returned.

Example 3.2 In this example, the modal formula $(\neg\Diamond p \wedge \neg\Diamond q \wedge \neg\Diamond s \wedge \neg\Diamond t) \vee \neg\Diamond r \vee \Diamond r$ is proved by $RePML_K$ by applying only 3 rules, while other tableaux based provers as Lotrec and TWB need to apply 17 and 18 rules, respectively. As said previously, our rules take all the $\Diamond$-formulas in the same step.

```
formulaKLogic(or(or(not(diamond(u)), diamond(u)), and(not(diamond(p)), and(not(diamond(q)),
              and(not(diamond(t)), not(diamond(t))))))).
 :::::  Translated automatically to the following relational formula:
```

```
RELATIONS ORDERED IN LEAF [1]
rel([1], uni(uni(opp(comp(r, u)), comp(r, u)), inter(opp(comp(r, p)), inter(opp(comp(r, q)),
         inter(opp(comp(r, t)), opp(comp(r, t)))))), x, y)

% - Rule applied

[rel([1], uni(uni(opp(comp(r, u)), comp(r, u)), inter(opp(comp(r, p)), inter(opp(comp(r, q)),
          inter(opp(comp(r, t)), opp(comp(r, t)))))), x, y)]
______________________________________________________________   Union Rule
[rel([1], uni(opp(comp(r, u)), comp(r, u)), x, y), rel([1], inter(opp(comp(r, p)),
          inter(opp(comp(r, q)),  inter(opp(comp(r, t)), opp(comp(r, t))))), x, y)]


RELATIONS ORDERED IN LEAF [1]
rel([1], uni(opp(comp(r, u)), comp(r, u)), x, y)
rel([1], inter(opp(comp(r, p)), inter(opp(comp(r, q)), inter(opp(comp(r, t)),
         opp(comp(r, t))))), x, y)

:::::  NEXT   STEP. Press Enter key to continue.

[rel([1], uni(opp(comp(r, u)), comp(r, u)), x, y)]
______________________________________________________________    Union Rule
[rel([1], opp(comp(r, u)), x, y), rel([1], comp(r, u), x, y)]

RELATIONS ORDERED IN LEAF [1]
rel([1], inter(opp(comp(r, p)), inter(opp(comp(r, q)), inter(opp(comp(r, t)),
         opp(comp(r, t))))), x, y)
rel([1], opp(comp(r, u)), x, y)
rel([1], comp(r, u), x, y)

:::::  NEXT   STEP. Press Enter key to continue.

:::::  Axiomatic set (close leaf): [1]
[rel([1], opp(comp(r, u)), x, y), rel([1], comp(r, u), x, y)]

 :::::  NEXT   STEP. Press Enter key to continue.
 :::::  Variables used: [a1, x, y]
 :::::  VALID. Total Rules applications: 2
used_rules([1], union, [rel(uni(uni(opp(comp(r, u)), comp(r, u)), inter(opp(comp(r, p)),
  inter(opp(comp(r, q)), inter(opp(comp(r, t)), opp(comp(r, t)))))), x, y)])
used_rules([1], union, [rel(uni(opp(comp(r, u)), comp(r, u)), x, y)])
```

Let us consider now Figure 2 where we show the result of a small comparative of the number of rules applied by our prover, TWB [1] and Lotrec [2]. The first 9 formulas are valid while the following 3 are non-valid. These 12 formulas have been taken from the demo of TWB for logic K . Notice that $RePML_K$ applies the lowest number of rules in 6 of the 12 formulas.

4. Conclusions and Future Work

We presented an implementation in Prolog, called $RePML_K$, of a relational dual tableau for modal logic K. The key steps of this work are: first, the implementation of new rules (k_1) and (k_2) which guarantee that every proof tree is finite and improves the efficiency of our prover; second, the implementation of an order of application of rules that allows us to ensure that there is a unique proof tree for every formula. As a consequence of both contributions, we have implemented a decision procedure for our logic. $RePML_K$ makes use of backtracking of Prolog, matching mechanism for free variables, and of the logic programming techniques in order to obtain an easy and modular prover. We remark that the results are

[1] http://twb.rsise.anu.edu.au/
[2] http://www.irit.fr/ACTIVITES/LILaC/Lotrec/

Modal formula	Lotrec	TWB	$RePML_K$	Output
$\Diamond p \to \Diamond p$	1	3	1	valid
$\Diamond p \to \Box(\Box\neg q \vee \Diamond q)$	6	5	6	valid
$\neg(\Diamond p \wedge \Box\neg p)$	4	3	3	valid
$\neg(\Diamond p \wedge \Diamond(\Diamond q \wedge \neg\Diamond q))$	4	5	3	valid
$\Box(a \to b) \to (\Box a \to \Box b)$	7	7	6	valid
$(\Box(p0 \wedge p1) \leftrightarrow (\Box p0 \wedge \Box p1))$	19	18	15	valid
$(\Diamond(p0 \vee p1) \leftrightarrow (\Diamond p0 \vee \Diamond p1))$	16	18	11	valid
$(\neg\Box\neg p0 \leftrightarrow \Diamond p0)$	7	8	8	valid
$(\Box p0 \to \Diamond p0) \wedge (\Box p0 \to \Box\Box p0) \wedge$ $\wedge(\neg p0 \to \Box\Diamond\neg p0) \to (\Box p0 \to p0)$	12	31	14	valid
$\Box p1 \to p1$	1	1	2	not valid
$\Box p1 \to \Box\Box p1$	6	3	6	not valid
$\neg\Box p1 \to \Box\neg\Box p1$	4	3	6	not valid

Figure 2. Comparative of number of rules applied in provers for modal logic K.

promising. All the axioms and examples executed with $RePML_K$ are proved in a few steps and it works efficiently with the new rules (k_1) and (k_2).

It is easy to prove that the complexity of our prover is now suboptimal, because our breadth-first search implies a big use of memory. However, we have planned some strategies in order to improve it, such as the use of modal clauses [18] and the modification of our rules in order to eliminate redundant repetitions. For example, when we have formulas type $\Diamond a, \Diamond b, \Box\Diamond c$ in the world i, we should create two new worlds j and k such as a is true in j and b is true in k. However, it could be unnecessary the repetition of $\Diamond c$ in both worlds j, k. Other improvements could come from exploiting Prolog's built-in clause indexing scheme, as in [28].

We are working in a comparison of our implementation with other provers for modal logic K in order to show how the prover scales as the problem size increases (*LWB benchmark* - `http://www.lwb.unibe.ch/`). Moreover, we are studying the extension of this prover to other modal logics as T and S4. The case of T seems to be not very difficult. We are thinking to add a new rule such as

$$(\mathsf{k}_3) \quad \frac{X \cup \{z_k(R;P_i)z_0\}_{i\in\{1,\dots,l\}}}{X \cup \{z_k(R;P_i)z_0\}_{i\in\{1,\dots,l\}} \cup \{z_k P_i z_0\}_{i\in\{1,\dots,l\}}}$$

, provided that $z_k(R;T)z_0 \notin X$ for all terms T and always applied before (k_1) and (k_2). However, the case of S4 is not straightforward if we want to maintain the advantages of not using *external* strategies such as backtracking, loop checking, etc. Our idea is to modify/extend the rules (k_1) and (k_2) in order to obtain it.

Furthermore, we are working in the improvement of our prover in other aspects, many of them related to its user-friendliness, the construction of counter-models and the possibility of create new logics, as in Lotrec. Last, but not least, we are considering the possibility of using OCaml language instead of Prolog as in TWB, because it allows much more advanced data-structures instead of naive lists and, as said in [21], a desirable feature in theorem provers, to allow reuse of previously proved theorems (abstraction).

Acknowledgements

This work is partially supported by the Spanish research projects TIN2006-15455-C03-01, TIN07-65819, and the second author is partially supported also by project P6-FQM-02049. The last author of the paper is partially supported by the Polish Ministry of Science and Higher Education grant N N206 399134. Finally, we would like to thank the anonymous reviewers for their careful reading and very helpful

comments, which have made possible this version of the paper.

References

99

[1] Abate, P. and Goré, R. and Widmann, F. An On-the-fly Tableau-based Decision Procedure for PDL-satisfiability. Electronic Notes in Theoretical Computer Science volume 231, pp 191–209, Elsevier 2009.

[2] Abate, P. and Goré, R. The Tableau Workbench. Electronic Notes in Theoretical Computer Science volume 231, pp 55–67, Elsevier 2009.

[3] P. Balsiger and A. Heuerding. Comparison of theorem provers for modal logics–Introduction and summary. In H. de Swart, editor, Automated Reasoning with Analytic Tableaux and Related Methods. In TABLEAUX 1998, LNAI 1397, pages 25–26, 1998.

[4] Peter Balsiger, Alain Heuerding, and Stefan Schwendimann. A benchmark method for the propositional modal logics k, kt, s4. Journal of Automated Reasoning, 24(3):297–317, 2000.

[5] B. Beckert , J. Posegga: leanTAP: Lean, Tableau-based Deduction. Journal of Automated Reasoning, Vol. 15, No. 3, pages 339-358, 1995.

[6] D. Bresolin, J. Golińska-Pilarek, and E. Orłowska, *Relational dual tableaux for interval temporal logics*, Journal of Applied Non-Classical Logics 16, No. 3-4 (2006), 251–277.

[7] A. Burrieza, A. Mora, M. Ojeda-Aciego, and E Orlowska. *An implementation of a dual tableaux system for order-of-magnitude qualitative reasoning.* I.J.of Computer Mathematics, 2009. To appear.

[8] Dallien, J. and MacCaull, W., *RelDT: A relational dual tableaux automated theorem prover*, `http://logic.stfx.ca/reldt.html`

[9] I. Düntsch and E. Orłowska, *A proof system for contact relation algebras*, Journal of Philosophical Logic 29 (2000), 241–262.

[10] 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. `http://www.di.univaq.it/TARSKI/transIt/`

[11] A. Formisano, M. Nicolosi Asmundo, *An efficient relational deductive system for propositional non-classical logics*, Journal of Applied Non-Classical Logics 16, No. 3–4 (2006), 367–408.

[12] 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*, Lecture Notes in Artificial Intelligence 4342 (2006), 89–104.

[13] Olivier Gasquet, Andreas Herzig, Dominique Longin, and Mohamad Sahade. Lotrec: Logical tableaux research engineering companion. In TABLEAUX 2005, LNAI 3702, pages 318–322, Springer, 2005.

[14] Golińska-Pilarek, J. Mora, A; Muñoz-Velasco, E.: *An ATP of a Relational Proof System for Order of Magnitude Reasoning with Negligibility, Non-Closeness and Distance*. Lecture Notes in Artificial Intelligence 5351, pp. 129-139, 2008.

[15] Golińska-Pilarek, J.; Muñoz-Velasco, E. and Mora, A: *A New Decision Procedure for Modal Logic* K. Technical Report, 2009. `http://www.matap.uma.es/~emilio/TechRepDPK09`.

[16] Golińska-Pilarek, J.; Muñoz-Velasco, E.: Dual tableau for a multimodal logic for order of magnitude qualitative reasoning with bidirectional negligibility. International Journal of Computer Mathematics Volume 86, Issue 10–11, pp. 1707 - 1718, 2009.

[17] Golińska-Pilarek, J.; Muñoz-Velasco, E.: Relational approach for a logic for order of magnitude qualitative reasoning with negligibility, non-closeness and distance. Logic Journal of IGPL 17(4):375-394, 2009.

[18] Goré, R. and Nguyen, L.A. Clausal Tableaux for Multimodal Logics of Belief, Fundamenta Informaticae, vol. 94 (1), pp. 21–40, 2009.

[19] A Heuerding. LWBtheory: information about some propositional logics via the WWW. Logic Journal of the IGPL, 4: 169–174, 1996.

[20] Hofner, P. and Struth, G. On automating the calculus of relations. LNAI 5195, pages 50–66, Springer, 2008.

[21] Kadoda, G., Stone, R., and Diaper, D. Desirable features of educational theorem provers: A cognitive dimensions viewpoint. Paper presented at the 11th Annual Workshop of the Psychology of Programming Interest Group, Leeds, UK, 1999.

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

[23] Massacci, F., Single Step Tableaux for Modal Logics. J. Autom. Reasoning 24(3): 319-364 (2000)

[24] Negri, S.: Proof analysis in modal logic. J. Philos. Log. 34(5?6), 507?544 (2005)

[25] E. Orłowska, *Relational proof systems for relevant logics*, Journal of Symbolic Logic 57 (1992), 1425–1440.

[26] E. Orłowska, *Temporal logics in a relational framework*, in: Bolc, L. and Szałas, A. (eds) Time and Logic-a Computational Approach, University College London Press (1995), 249–277.

[27] E. Orłowska and J. Golińska-Pilarek, *Dual Tableaux and their Applications*, A draft of the book, 2009.

[28] Otten, J.: ileanTAP: An intuitionistic theorem prover, in Proceedings, International Conference on Theorem Proving with Analytic Tableaux and Related Methods, Pont- a-Mousson, France, LNCS 1227, Springer, 1997.

[29] L. C. Paulson. Isabelle: The Next 700 Theorem Provers. ArXiv Computer Science e-prints, October 1993.