Dual tableau-based decision procedures for some relational logics*

Domenico Cantone[1], Marianna Nicolosi Asmundo[1], and Ewa Orłowska[2]

[1] Università di Catania, Dipartimento di Matematica e Informatica
email: cantone@dmi.unict.it, nicolosi@dmi.unict.it
[2] National Institute of Telecommunications, Warsaw, Poland
email: orlowska@itl.waw.pl

Abstract. We consider fragments of the relational logic $\mathsf{RL}(\mathbf{1})$ obtained by imposing some constraints on the relational terms involving relations composition. Such fragments allow to express several non classical logics such as the multi-modal logic K and the description logic $\mathcal{ALC}$ with union and intersection of roles. We show how relational dual tableaux can be employed to define decision procedures for each of them.

1 Introduction

In this paper we consider logics of binary relations which may serve as formalisms for representation of various theories, in particular some non-classical logics. These relational logics are based on languages whose formulae have the form xRy, where x and y are object variables and R is a term built from relation variables with the relational operations typical for binary relations as formalized in [15] (see also [10]). The semantics of these languages reflects the usual meaning of xRy as saying that two objects denoted by x and y stand in the relation denoted by R. The relational logics studied in the paper are fragments of the relational logic $\mathsf{RL}(\mathbf{1})$ presented in [12]. These fragments are obtained from $\mathsf{RL}(\mathbf{1})$ by posing some constraints on the relations involving relational composition. In particular, the first argument of the composition can only be a relational variable in the first fragment and a positive Boolean term in the second one. From the algebraic perspective, these fragments may be seen as the fragments of Peirce algebras [13]. A modern presentation of Peirce algebras can be found in [4]. A dual tableau for Peirce algebras is presented in [14].

The representation of non-classical logics with relational logics is based on the fact that logical formulae may be treated as relations once their Kripke-style semantics is known. In Kripke-style semantics, formulae are interpreted as sets of objects which may be identified with what is called right ideal relations, which in the binary case amounts to saying that the relations satisfy $R\,;\mathbf{1} = R$, where "$;$" is the composition of binary relations and $\mathbf{1}$ is the universal relation. For example, since the set of right ideal relations is closed on Boolean operations, the

* Work partially supported by GNCS, Programma Giovani Ricercatori 2009.

propositional connectives of disjunction, conjunction, and negation may be interpreted as union, intersection, and complement of relations, respectively. In modal (resp., description) logics, a possibility operator $\langle R \rangle$ (resp., a concept operator $\exists R$) determined by a relation (resp., role) R acting on a formula α, interpreted as a right ideal relation, may be understood as $R\,;\alpha$, as observed in [11]. It is known that the composition of a relation with a right ideal relation returns a right ideal relation. The relational interpretation of languages preserves validity of formulae. In [7] an implementation of the translation of modal languages into relational languages is presented.

Relational logics appear to be an adequate representation means for a great variety of theories as shown in [12]. Therefore any decision procedure for a relational logic is not just a single decision method for some theory but it may be applied to several theories which can be interpreted in this relational logic.

The paper is organized as follows. In Section 2 we recall the logic $\mathsf{RL}(\mathbf{1})$ and its dual tableau. In Sections 3 and 4 we present two fragments of $\mathsf{RL}(\mathbf{1})$ and we develop decision procedures for them based on dual tableaux.

2 The relational logic $\mathsf{RL}(1)$ and its dual tableau

2.1 Syntax

Let $\mathbb{OV}$ be a countably infinite set of object (individual) variables $x, y, z, w \ldots$, let $\mathbb{RV}$ be a countably infinite set of *relational variables* $\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}, \ldots$, and let $\mathbf{1}$ be the *relational constant*. The *relational operators* are $-$ (complementation), $\cap$ (intersection), $\cup$ (union), $;$ (composition), and $^{-1}$ (converse). The set of *relational terms* $\mathbb{RT}$ is the smallest set (with respect to inclusion) such that

(a) $\mathbb{RV} \subseteq \mathbb{RT}$,
(b) $\mathbf{1} \in \mathbb{RT}$, and
(c) $\mathbb{RT}$ is closed with respect to the relational operators.

Relational terms are indicated with the letters P, Q, R,... Examples of relational terms are $(\mathsf{p} \cap \mathsf{q})\,;\mathsf{s}$ and $-(P \cup Q)$, where p, q, and s are relational variables and P, Q are relational terms. $\mathsf{RL}(\mathbf{1})$-formulae have the form xRy, where $x, y \in \mathbb{OV}$ and $R \in \mathbb{RT}$. The $\mathsf{RL}(\mathbf{1})$-formulae $x\mathbf{1}y$ and $x\mathsf{r}y$, with $\mathsf{r} \in \mathbb{RV}$, are called *atomic* $\mathsf{RL}(\mathbf{1})$-formulae. A *literal* is an atomic formula ($x\mathbf{1}y$ or $x\mathsf{r}y$) or its complementation ($x(-\mathbf{1})y$ or $x(-\mathsf{r})y$). $\mathsf{RL}(\mathbf{1})$-formulae are also denoted using as metavariables the greek letters φ and ψ. For a relational operation $\sharp$, by a $(\sharp)$-*formula* we mean a formula built with a relational term whose principal operation is $\sharp$ whereas by a $(-\sharp)$-*formula* we denote a formula obtained from a relational term with principal operation $-$ followed by $\sharp$. A *Boolean term* is a relational term such that all the relational operations in it are among the Boolean operations $-$, $\cup$, and $\cap$.

2.2 Semantics

$\mathsf{RL}(\mathbf{1})$-formulae are interpreted in $\mathsf{RL}(\mathbf{1})$-*models*. An $\mathsf{RL}(\mathbf{1})$-model is a structure $\mathcal{M} = (U, m)$, where U is a nonempty universe and $m : \mathbb{RV} \to U \times U$ is a given map which is naturally extended to the whole collection $\mathbb{RT}$ of relational terms as follows:

- $m(\mathbf{1}) = U \times U$;
- $m(-R) = (U \times U) \setminus m(R)$;
- $m(R \cup S) = m(R) \cup m(S)$;
- $m(R \cap S) = m(R) \cap m(S)$;
- $m(R\,;S) = m(R)\,;m(S)$
  $= \{(x,y) \in U \times U : (x,z) \in m(R) \text{ and } (z,y) \in m(S), \text{ for some } z \in U\}$;
- $m(R^{-1}) = \{(y,x) \in U \times U : (x,y) \in m(R)\}$.

Let $\mathcal{M} = (U,m)$ be an $\mathsf{RL}(\mathbf{1})$-model. An *evaluation* in $\mathcal{M}$ is any function $v : \mathbb{OV} \to U$. Given an object variable z in $\mathbb{OV}$, an evaluation v_1 is a *z-variant* of an evaluation v if $v_1(x) = v(x)$, for every $x \in \mathbb{OV}$ such that $x \neq z$. *Satisfaction* of an $\mathsf{RL}(\mathbf{1})$-formula xRy by an $\mathsf{RL}(\mathbf{1})$-model $\mathcal{M} = (U,m)$ and by an evaluation v in $\mathcal{M}$ is defined as:

$$\mathcal{M}, v \models xRy \text{ iff } (v(x), v(y)) \in m(R).$$

An $\mathsf{RL}(\mathbf{1})$-formula xRy is *true* in a model $\mathcal{M} = (U,m)$ if $\mathcal{M}, v \models xRy$, for every evaluation v in $\mathcal{M}$. An $\mathsf{RL}(\mathbf{1})$-formula xRy is said *valid* if it is *true* in all $\mathsf{RL}(\mathbf{1})$-models. An $\mathsf{RL}(\mathbf{1})$-formula xRy is *falsified* by a model $\mathcal{M} = (U,m)$ and by an evaluation v in $\mathcal{M}$ if $\mathcal{M}, v \not\models xRy$. It is *falsifiable* if there are a model $\mathcal{M}$ and an evaluation v in $\mathcal{M}$ such that $\mathcal{M}, v \not\models xRy$.

2.3 RL(1)-dual tableau

Proof development in dual tableaux proceeds by systematically decomposing the (disjunction of) formula(e) to be proved till a validity condition is detected by means of axiomatic sets. Such an analytic approach is similar to the one adopted by the tableau method with the difference that the two systems work in a dual way. Duality of tableaux and of dual tableaux has been deeply analyzed in [9].

$\mathsf{RL}(\mathbf{1})$-dual tableau consists of decomposition rules to analyze the structure of the formula to be proved valid, and of axiomatic sets which specify the closure conditions. The decomposition rules for $\mathsf{RL}(\mathbf{1})$ are illustrated in Table 1. In these rules, "," is interpreted as disjunction and "|" as conjunction.

$\mathsf{RL}(\mathbf{1})$-axiomatic sets are sets of $\mathsf{RL}(\mathbf{1})$-formulae including a subset of one of the following forms:

(Ax 1) $\{xRy, x(-R)y\}$,
(Ax 2) $\{x\mathbf{1}y\}$.

Let xPy be an $\mathsf{RL}(\mathbf{1})$-formula. An $\mathsf{RL}(\mathbf{1})$-*proof tree for xPy* is an ordered tree whose nodes are labelled by disjunctive sets of formulae. By a *branch* of a proof tree we mean any maximal path in it. We require a proof tree for xPy to satisfy the following properties:

- the formula xPy is at the root of this tree,

Table 1. RL(**1**) decomposition rules.

($\cup$)	$\dfrac{x(R\cup S)y}{xRy, xSy}$	($-\cup$)	$\dfrac{x(-(R\cup S))y}{x(-R)y \mid x(-S)y}$
($\cap$)	$\dfrac{x(R\cap S)y}{xRy \mid xSy}$	($-\cap$)	$\dfrac{x(-(R\cap S))y}{x(-R)y, x(-S)y}$
($--$)	$\dfrac{x(--R)y}{xRy}$		
($^{-1}$)	$\dfrac{x(R^{-1})y}{yRx}$	($-\,^{-1}$)	$\dfrac{x(-(R^{-1}))y}{y(-R)x}$
($;$)	$\dfrac{x(R;S)y}{xRz, x(R;S)y \mid zSy, x(R;S)y}$	z, any object variable	
($-;$)	$\dfrac{x(-(R;S))y}{x(-R)z, z(-S)y}$	z, a new object variable	

– each node, with the exception of the root, is obtained from its predecessor node by an application of a decomposition rule of Table 1,
– a node does not have successors (i.e. it is a leaf node) whenever its set of formulae is an axiomatic set or none of the rules of Table 1 can be applied to its set of formulae.

A node of an RL(**1**)-proof tree is *closed* if its associated set of formulae contains an axiomatic set. A branch is closed if one of its nodes is closed. A proof tree is closed if all of its branches are closed. An RL(**1**)-formula is provable if there is a closed RL(**1**)-proof tree for it, referred to as an RL(**1**)-proof.

A node of an RL(**1**)-proof tree is *falsified* by a model $\mathcal{M} = (U, m)$ and by an evaluation v if every formula xRy in its set of formulae is falsified by $\mathcal{M}$ and v. A node is *falsifiable* if there are a model $\mathcal{M}$ and an evaluation v such that it is falsified by $\mathcal{M}$ and v. A branch of an RL(**1**)-proof tree is *falsified* by a model $\mathcal{M}$ and by an evaluation v if each node in it is *falsified* by $\mathcal{M}$ and v. A branch of an RL(**1**)-proof tree is *falsifiable* if there is a model and an evaluation which falsify every node in the branch. An RL(**1**)-proof tree is *falsified* by a model $\mathcal{M} = (U, m)$ and by an evaluation v if one of its branches is falsified by $\mathcal{M}$ and v. Finally, an RL(**1**)-proof tree is *falsifiable* if one of its branches is *falsifiable*.

Correctness and completeness of RL(**1**)-dual tableau are proved in [12]. The logic RL(**1**) is undecidable. Such result follows from the undecidability of the equational theory of representable relation algebras discussed in [16]. In the following sections we present some of its decidable fragments. Other decidable fragments of RL(**1**) can be found in [12].

3 The (r ; _)-fragment of RL(1) and its decision procedure

The (r ; _)-fragment is the collection of the RL(**1**)-formulae xPy in which the composition operator "$;$" can occur only in the following restricted way. For each subterm of P of the form $R;S$, R must belong to a designated nonempty proper subset of $\mathbb{RV}$, $\mathbb{RV}_1$, whereas S can involve all the relational operators

Table 2. $(\mathsf{r}\,;\,\_)$-fragment decomposition rules.

$(\,;\,)$	$\dfrac{x(\mathsf{r}\,;S)y}{zSy,x(\mathsf{r}\,;S)y}$	$x(-\mathsf{r})z$ a literal in the current node
$(-\,;\,)$	$\dfrac{x(-(\mathsf{r}\,;S))y}{x(-\mathsf{r})z,z(-S)y}$	z, a new object variable

used to construct $\mathsf{RL}(\mathbf{1})$-formulae, with the exception of the converse operator $^{-1}$. In the relation interpretation of logics, the elements of $\mathbb{RV}_1$ are meant to denote accessibility relations (resp., roles) in modal (resp., description) logics.

A formal description of the set of relational terms $\mathbb{RT}_{(\mathsf{r}\,;\,\_)}$ is given in what follows.

Let $\mathbb{RV}_1$ be as above, then we define the set of terms $\mathbb{RT}_{(\mathsf{r}\,;\,\_)_1}$ as the smallest set of terms containing $\mathbb{RV}_1$ which is closed with respect to the complementation operator "$-$".

Likewise, we define $\mathbb{RT}_{(\mathsf{r}\,;\,\_)_2}$ as the smallest set of terms containing the constant $\mathbf{1}$ and the relational variables in $\mathbb{RV}\setminus\mathbb{RV}_1$, and such that if $R, S \in \mathbb{RT}_{(\mathsf{r}\,;\,\_)_2}$ and $\mathsf{r} \in \mathbb{RV}_1$, then $-R, R\cup S, R\cap S$, $\mathsf{r}\,;S \in \mathbb{RT}_{(\mathsf{r}\,;\,\_)_2}$. Finally we put

$$\mathbb{RT}_{(\mathsf{r}\,;\,\_)} =_{\mathrm{Def}} \mathbb{RT}_{(\mathsf{r}\,;\,\_)_1} \cup \mathbb{RT}_{(\mathsf{r}\,;\,\_)_2}.$$

This logic allows to express the multi-modal logic K and, therefore, also the description logic $\mathcal{ALC}$ [2, 1]. The translation of such logics in relational terms is carried out along the lines of [12], Chapter 7. In particular, the relational variables in $\mathbb{RV}_1$ represent the accessibility relations of the multi-modal logic K and the roles of the logic $\mathcal{ALC}$. A relational dual tableau style decision procedure for the logic K can be found in [8]. The procedure defined there is inspired by [3].

3.1 A dual tableau decision procedure for the $(\mathsf{r}\,;\,\_)$-fragment

Dual tableaux for the $(\mathsf{r}\,;\,\_)$-fragment can be obtained by adapting the system introduced in Section 2.3 as we describe below.

Axiomatic sets are defined as in Section 2.3. The set of decomposition rules for the Boolean operators, namely the $(\cup)$, $(\cap)$, $(-\cap)$, $(-\cup)$, $(--)$-rules, are identical to the ones presented in Table 1. The other decomposition rules, that is the $(\,;\,)$-rule and the $(-\,;\,)$-rule, are displayed in Table 2.[3] The notion of proof tree is identical to the one given in Section 2.3 with the exception that each node can be obtained from its predecessor (if any) by the application of a Boolean decomposition rule of Table 1 or a decomposition rule of Table 2. In particular, the $(\,;\,)$-rule of Table 2 can be applied to a formula $x(\mathsf{r}\,;S)y$ of a node of a proof tree only in case the literal $x(-\mathsf{r})z$ occurs in the same node. Such side condition makes this variant of the $(\,;\,)$-rule less liberal than the corresponding

[3] Table 2 does not contain any decomposition rule for the converse operation $^{-1}$ because it is not a constructor of the terms belonging to the $(\mathsf{r}\,;\,\_)$-fragment.

rule presented in Table 1, since it restricts the choice of the variable which can be used in the decomposition step. Moreover, such a rule variant does not perform any branch splitting and therefore the overall number of branches in the proof tree is generally smaller.

A proof procedure for the dual tableau system just defined, that we call $(\mathsf{r}\,;\_)$-dual tableau, can be designed by giving a description of the proof tree construction process together with the constraints which limit the application of the decomposition rules.

For this purpose, we introduce the notion of *deduction tree*. As proof trees, deduction trees are ordered trees whose nodes are labelled with disjunctive sets. However, deduction trees may have some leaf nodes that do not contain any axiomatic set and such that decomposition rules can still be applied to them. As it is clarified below, deduction trees can be seen as "approximations" of proof trees with the property that they can be completed to proof trees.

Definition 1. *Let xPy be a formula of the $(\mathsf{r}\,;\_)$-fragment of $\mathsf{RL}(\mathbf{1})$. A deduction tree $\mathcal{T}$ for xPy is recursively defined as follows:*

(a) the tree with only one node labelled with $\{xPy\}$ is a deduction tree for xPy (initial deduction tree);

(b) let $\mathcal{T}$ be a deduction tree for xPy and let θ be a branch of $\mathcal{T}$ whose leaf node N does not contain an axiomatic set.[4] The tree obtained from $\mathcal{T}$ by applying one of the Boolean decomposition rules of Table 1 or one of the decomposition rules of Table 2, as illustrated by items 1-5 below, is a deduction tree for xPy:

1. *if any formula of type $x(R \cup S)y$ (resp., $x(-(R \cap S))y$) occurs in N, we add $N' = (N \setminus \{x(R \cup S)y\}) \cup \{xRy, xSy\}$ (resp., $N' = (N \setminus \{x(-(R \cap S y))\}) \cup \{x(-R)y, x(-S)y\}$) as the successor of N in θ;*
2. *if any formula of type $x(R \cap S)y$ (resp., $x(-(R \cup S))y$) occurs in N, we simultaneously add $N' = (N \setminus \{x(R \cap S)y\}) \cup \{xRy\}$ (resp., $N' = (N \setminus \{x(-(R \cup S))y\}) \cup \{x(-R)y\}$) as left successor of N, and $N'' = (N \setminus \{x(R \cap S)y\}) \cup \{xSy\}$ (resp., $N'' = (N \setminus \{x(-(R \cup S))y\}) \cup \{x(-S)y\}$) as right successor of N in θ;*
3. *if any formula of type $x(--R)y$ occurs in N, we add $N' = (N \setminus \{x(--R)y\}) \cup \{xRy\}$ as the successor of N in θ;*
4. *if any formula of type $x(-(\mathsf{r}\,;S))y$ occurs in N, we add $N' = (N \setminus \{x(-(\mathsf{r}\,;S))y\}) \cup \{x(-\mathsf{r})z, z(-S)y\}$ as the successor of N in θ;*
5. *if any formula of type $x(\mathsf{r}\,;S)y$ occurs in N and a literal $x(-\mathsf{r})z$ occurs in N we add $N' = N \cup \{zSy\}$ as the successor of N in θ.*

We further require that the following *strictness hypotheses* are satisfied: on each branch of a deduction tree

- all the decomposition rules, with the exception of the $(\,;\,)$-rule, can be applied at most once to the same non-literal formula,
- the $(\,;\,)$-rule can be applied at most once with the same premises.

[4] From now on we identify nodes with the (disjunctive) sets labelling them.

It is easy to see that if all the branches of a deduction tree $\mathcal{T}$ are either closed or, according to the strictness hypotheses, not further expansible, then $\mathcal{T}$ is a proof tree. The proof construction in Definition 1 is sound and complete even under the above strictness hypotheses. We will limit ourselves in showing only its termination, thus obtaining a decision procedure for the $(\mathsf{r}\,;\,\_)$-fragment.

3.2 Termination

The proof procedure presented in Section 3.1 adds to the current deduction tree one or two new nodes at each decomposition step. Thus, in order to show that it always terminates, it is enough to prove that, given a formula xPy of the $(\mathsf{r}\,;\,\_)$-fragment, every proof tree for xPy that can be constructed according to the procedure described in Section 3.1 is finite. Before going into details it is useful to introduce the notion of *open saturated branch*. We characterize an open saturated branch θ_S of a deduction tree $\mathcal{T}$ for a formula xPy of the $(\mathsf{r}\,;\,\_)$-fragment as a set of nodes such that:

- $x'\mathbf{1}y' \notin N$, for every node $N \in \theta_S$;
- if $x'Ry'$, (resp., $x'(-R)y'$) occurs in a node $N \in \theta_S$, then $x'(-R)y'$ (resp., $x'Ry'$) does not occur in any other node $N' \in \theta_S$;
- if $x'(--R)y'$ occurs in a node $N \in \theta_S$, then there is a node $N' \in \theta_S$ such that $x'Ry' \in N'$;
- if $x'(R \cap S)y'$ occurs in a node $N \in \theta_S$, then there is a node $N' \in \theta_S$ such that either $x'Ry' \in N'$ or $x'Sy' \in N'$;
- if $x'(R \cup S)y'$ occurs in a node $N \in \theta_S$, then there is a node $N' \in \theta_S$ such that $x'Ry' \in N'$ and $x'Sy' \in N'$;
- if $x'(-(R\cap S))y'$ occurs in a node $N \in \theta_S$, then there is a node $N' \in \theta_S$ such that $x'(-R)y', x'(-S)y' \in N'$;
- if $x'(-(R\cup S))y'$ occurs in a node $N \in \theta_S$, then there is a node $N' \in \theta_S$ such that either $x'(-R)y'$, or $x'(-S)y' \in N'$;
- if $x'(\mathsf{r}\,;S)y'$ occurs in a node $N \in \theta_S$, then for every z such that $x'(-\mathsf{r})z \in N'$, for some $N' \in \theta_S$, there is an $N'' \in \theta_S$ such that $zSy' \in N''$;
- if $x'(-(\mathsf{r}\,;S))y'$ occurs in a node $N \in \theta_S$, then there is a node $N' \in \theta_S$ such that $x'(-\mathsf{r})z, z(-S)y' \in N'$, for some object variable z.

The proof can be carried out by contradiction, assuming that one can construct an infinite proof tree for xPy under the strictness hypotheses. By König's Lemma, such a proof tree must have an infinite branch. This branch cannot be closed because once a branch is closed, it cannot be further expanded. Thus it can be embedded in an open saturated branch.

We devote the rest of this section to proving that under the strictness hypotheses every open saturated branch of a proof tree for xPy has to be finite. This result is sufficient to assert, in contradiction with our hypothesis, that each branch of a proof tree for a formula xPy has to be finite. Thus, each proof tree for xPy has to be finite and therefore the proof procedure of Section 3.1 always terminates.

To carry out our proof, it is useful to consider that since nodes of a proof tree are finite sets of formulae, a branch containing a finite number of nodes is finite.

Let θ_S be a saturated branch of a proof tree $\mathcal{T}$ for a formula xPy. We define a total order $<_{\theta_S}$ on $W_{\theta_S} \setminus \{y\}$ as follows: for $z, w \in W_{\theta_S} \setminus \{y\}$ we let $z <_{\theta_S} w$ if and only if z has been introduced before w in the construction of the branch θ_S.

Lemma 1. *The number of formulae in $\bigcup \theta_S$ with left variable w is finite, for every $w \in W_{\theta_S}$.*

Proof: The lemma is trivially true for the variable y, since $\bigcup \theta_S$ contains no formula with left variable y. Concerning the variables in $\bigcup \theta_S$, we proceed by induction over the ordered set $(W_{\theta_S} \setminus \{y\}, <_{\theta_S})$.

- **Base case.** The initial formula xPy can generate, by Boolean decomposition, a finite number of subformulae with left variable x. Moreover, each application of the $(-\,;)$-decomposition rule introduces a literal of type $x(-\mathsf{r})z$ that, however, cannot be further decomposed, and every application of the $(\,;)$-rule does not increase the number of formulae with left variable x. Thus the number of formulae in $\bigcup \theta_S$ with left variable x is finite.
- **Inductive step.** By inductive hypothesis, the number of formulae in $\bigcup \theta_S$ with left variable z is finite, for $z <_{\theta_S} w$. We prove that this holds for w as well.
  The variable w has been introduced by the application of the $(-\,;)$-decomposition rule to a formula $z(-(\mathsf{r}\,;S))y$. The decomposition of $w(-S)y$ by means of the Boolean rules can introduce in $\bigcup \theta_S$ a finite number of subformulae with left variable w. Application of the $(-\,;)$-rule to each of these formulae only adds a literal with left variable w.
  Formulae with left variable w can also be obtained by applying the $(\,;)$-decomposition rule to every formula of type $z(\mathsf{r}\,;Q)y$ (notice that by the $(-\,;)$-decomposition of $z(-(\mathsf{r}\,;S))y$, the literal $z(-\mathsf{r})w$ occurs in θ_S). By inductive hypothesis the number of such $z(\mathsf{r}\,;Q)y$ has to be finite, thus the number of the wQy formulae resulting from the $(\,;)$-decomposition is also finite. Finally, applying the Boolean rules and the $(-\,;)$-rule to each of the wQy formulae obtained before, we get a finite number of formulae with left variable w. Summing up, the number of formulae with left variable w is finite. □

Lemma 2. *Any $(\,;)$-formula in $\bigcup \theta_S$ can be decomposed a finite number of times.*

Proof: Let $z(\mathsf{r}\,;Q)y$ be a $(\,;)$-formula in $\bigcup \theta_S$. Clearly, it can be decomposed as many times as the number of literals $z(-\mathsf{r})w$ in $\bigcup \theta_S$, for any $w \in W_{\theta_S}$. This number is in turn bounded by the number of $(-\,;)$-formulae $z(-(\mathsf{r}\,;P))y$ in $\bigcup \theta_S$, for any relational term P. Since by Lemma 1 this number is finite, the lemma follows. □

Let us define recursively the *weight* of a formula as follows:

- $\mathit{weight}(x\mathsf{r}y) = \mathit{weight}(x(-\mathsf{r})y) = \mathit{weight}(x\mathbf{1}y) = 0$

- $weight(x(A \cap P)y) = weight(xAy) + weight(xPy) + 1$
- $weight(x(-(A \cap P))y) = weight(x(-A)y) + weight(x(-P)y) + 1$
- $weight(x(- - P)y) = weight(P) + 1$
- $weight(x(-(\mathsf{r} \,;\, P))y) = weight(z(-P)y) + 1$
- $weight(x(\mathsf{r} \,;\, P)y) = weight(zPy) + 1$.

We define the *weight* of a node N as the sum of the *weight*s of the formulae in N. In particular, the *weight* of the (**;**)-formulae that cannot be decomposed anymore in N is set to 0. Analogously we set to 0 the *weight*s of those non literal formulae in N that are not of type (**;**) which have been already decomposed in a previous step because they also occur in some ancestors of N. It is easy to check that the *weight* of a node N is 0 if and only if it contains only literals, (**;**)-formulae that cannot be expanded anymore, and non literal formulae that are not of type (**;**) already decomposed by some previous inference steps.

Lemma 3. *Let $\mathcal{T}_0$ be an initial deduction tree for xPy. After a finite number of steps a proof tree $\mathcal{T}$ can be constructed such that each of its leaf nodes have all weight* 0.

Sketch of the proof: Each time a rule $(\cap), (\cup), (--)$, or $(- \,;\,)$ is applied to a formula on a leaf node of a deduction tree, the new nodes have a lower weight. If a decomposition step yields a non literal formula that is not of type (**;**), that already occurs in some ancestor nodes and that has been decomposed in a previous step, the weight of that formula is set to 0 and by the strictness hypotheses it is not decomposed anymore. Each time a (**;**)-formula is expanded, the weight of the node is incremented. However, by Lemma 2 this may happen only a finite number of times. After that, the (**;**)-formula gets the weight 0 for ever. Notice also that every (**;**)-decomposition introduces a formula of a lower weight. □

Clearly each branch of the proof tree $\mathcal{T}$ of Lemma 3 is saturated and finite. Thus every proof tree for xPy, constructed according to the procedure described in Section 3.1, is finite. Hence we can state the following theorem.

Theorem 1 (Termination). *The dual tableau proof procedure for the* $(\mathsf{r} \,;\, \_)$*-fragment described in Section 3.1 always terminates.*

4 The $(\cup, \cap \,;\, \_)$-fragment of $\mathsf{RL}(\mathbf{1})$ and its decision procedure

The $(\cup, \cap \,;\, \_)$-fragment of $\mathsf{RL}(\mathbf{1})$ is an extension of the $(\mathsf{r} \,;\, \_)$-fragment in which the constraints on the composition operator " **;** " are more relaxed. In particular, the first argument in a term of type $R \,;\, S$ of the $(\cup, \cap \,;\, \_)$-fragment can be any term constructed from the relational variables of a proper nonempty subset of $\mathbb{RV}$, say $\mathbb{RV}_1$, by applying only the $\cup$ and $\cap$ operators. The restriction on the second argument is the same of the $(\mathsf{r} \,;\, \_)$-fragment: thus S can involve all the relational operators used in $\mathbb{RL}(\mathbf{1})$-formulae except the converse operator $^{-1}$.

More precisely, we put

$$\mathbb{RT}_{(\cup,\cap\,;\,\_)} =_{\mathrm{Def}} \mathbb{RT}_{(\cup,\cap\,;\,\_)_1} \cup \mathbb{RT}_{(\cup,\cap\,;\,\_)_2},$$

where $\mathbb{RT}_{(\cup,\cap\,;\,\_)_1}$ and $\mathbb{RT}_{(\cup,\cap\,;\,\_)_1}$ are defined as follows. $\mathbb{RT}_{(\cup,\cap\,;\,\_)_1}$ is the smallest set of terms which contains the relational variables of $\mathbb{RV}_1$ and is closed with respect to the operators $-$, $\cup$, and $\cap$, whereas $\mathbb{RT}_{(\cup,\cap\,;\,\_)_2}$ is the smallest set of terms involving only the constant $\mathbf{1}$ and the relational variables $\mathbb{RV} \setminus \mathbb{RV}_1$ and such that if $P, S \in \mathbb{RT}_{(\cup,\cap\,;\,\_)_2}$ and $R \in \mathbb{PRT}_{(\cup,\cap\,;\,\_)_1}$, where $\mathbb{PRT}_{(\cup,\cap\,;\,\_)_1}$ is the subset of $\mathbb{RT}_{(\cup,\cap\,;\,\_)_1}$ whose elements do not contain complemented relational terms, then $-P, P \cup S, P \cap S$, and $R\,;P \in \mathbb{RT}_{(\cup,\cap\,;\,\_)_2}$.

The $(\cup,\cap\,;\,\_)$-fragment of $\mathsf{RL}(\mathbf{1})$ can express the description logic $\mathcal{ALC}(\cup,\cap)$ [2]. Intuitively speaking, formulae of $\mathcal{ALC}(\cup,\cap)$ can be embedded into the relational framework by mapping role names into the variables in $\mathbb{RV}_1$, concept names into the variables in $\mathbb{RV} \setminus \mathbb{RV}_1$, and the operator of existential concept restriction "$\exists$", into the composition operator "$;$".

4.1 A dual tableau procedure for the $(\cup,\cap\,;\,\_)$-fragment

We define a dual tableau system for the $(\cup,\cap\,;\,\_)$-fragment of the relational logic $\mathsf{RL}(\mathbf{1})$ as follows. Axiomatic sets are defined as in Section 2.3. Concerning the decomposition rules for Boolean formulae and formulae of type $(-\,;)$, we adopt the ones displayed in Table 1.

The $(\,;)$-rule deserves a separate treatment. We begin by observing that the $(\,;)$-rule of Table 1 is too liberal in the choice of the object variable to be used in the $(\,;)$-decomposition and does not allow to define a terminating proof procedure for the $(\cup,\cap\,;\,\_)$-fragment. On the other hand the variant of $(\,;)$-rule of Table 2 turns out to be too restrictive to define a complete system for the $(\cup,\cap\,;\,\_)$-fragment.

In order to define a $(\,;)$-rule that is adequate for our purposes, it is convenient to introduce the following auxiliary notions.

- Let xRy be a Boolean formula of the $(\cup,\cap\,;\,\_)$-fragment. We define $\mathsf{nnf}(xRy)$ to be the formula obtained from xRy by moving all the occurrences of the complement operator in R as inward as possible. Formally we put $\mathsf{nnf}(xRy) = x\ \mathsf{nnt}(R)y$, where:
  - if R is an atomic formula or its complementation, then $\mathsf{nnt}(R) = R$;
  - if $R = (S \cap H)$, then $\mathsf{nnt}((S \cap H)) = (\mathsf{nnt}(S) \cap \mathsf{nnt}(H))$;
  - if $R = (S \cup H)$, then $\mathsf{nnt}((S \cup H)) = (\mathsf{nnt}(S) \cup \mathsf{nnt}(H))$;
  - if $R = (-(S \cap H))$, then $\mathsf{nnt}((-(S \cap H))) = \mathsf{nnt}((-S)) \cup \mathsf{nnt}((-H))$;
  - if $R = (-(S \cup H))$, then $\mathsf{nnt}((-(S \cup H))) = \mathsf{nnt}((-S)) \cap \mathsf{nnt}((-H))$;
  - if $R = (--S)$, then $\mathsf{nnt}((--S)) = \mathsf{nnt}(S)$.

  Clearly xRy and $\mathsf{nnf}(xRy)$ are *logically equivalent*, that is for every model $\mathcal{M} = (U, m)$, and every evaluation v, $\mathcal{M}, v \models xRy$ if and only if $\mathcal{M}, v \models \mathsf{nnf}(xRy)$.
- Let N be a set of formulae. We characterize the notion of Bool_N-*formulae* as follows:

- every literal in N is a Bool_N-formula;
- every formula of type $x(R \cap S)y$ is a Bool_N-formula if either xRy or xSy is a Bool_N-formula;
- every formula of type $x(R \cup S)y$ is a Bool_N-formula if both xRy and xSy are Bool_N-formulae.

It easy to check that if xSy is a Bool_N-formula, then $xSy = \mathsf{nnf}(xSy)$.

We say that a formula xRy has a *Boolean construction* from N if there is a Bool_N-formula xSy such that $xSy = \mathsf{nnf}(xRy)$.

– Let R be a Boolean term of $\mathbb{RT}_{(\cup,\cap;\_)}$, x an object variable, F a set of formulae. Then we define $V(R, x, F)$ to be the set of object variables z such that xRz has a Boolean construction from F.

Our variant of the (;)-rule is formalized as follows:

$$\frac{x(R\,;P)y}{zPy, x(R\,;P)y} ,$$

where:

– $x(R\,;P)y$ is a formula of the $(\cup, \cap ; \_)$-fragment occurring on the leaf node N of a branch θ of a deduction tree, and
– z is an object variable belonging to $V(-R, x, \bigcup\theta)$.

It is easy to see that $V(-R, x, \bigcup\theta) = V(-R, x, N)$ (such identity will be helpful below). Indeed, since N is the leaf node of θ, the set of literals in N is the same as the set of literals in $\bigcup\theta$, so that a formula is a Bool_N-formula if and only if it is a $\mathsf{Bool}_{\bigcup\theta_S}$-formula. Hence, the set of formulae that have a Boolean construction from N is identical to the set of formulae having a Boolean construction from $\bigcup\theta$, and the identity $V(-R, x, \bigcup\theta) = V(-R, x, N)$ follows.

If $x(-R)z$ is a literal, then $V(-R, x, \bigcup\theta)$ is the collection of object variables z such that $x(-R)z$ is in N, and therefore, in this case, such variant of the (;)-rule coincides with the version presented in Section 3.1.

The (;)-rule given above can be obtained from the (;)-rule in Table 1 by requiring that the variable z used to decompose $x(R\,;P)y$ on the leaf node N of a branch θ can only be selected from the set $V(-R, x, \bigcup\theta)$ (that is from the set $V(-R, x, N)$).

In fact, let us assume that we are using the (;)-rule of Table 1 to decompose $x(R\,;P)y$: we construct the proof tree by adding as a left successor of N the node $N' = N \cup \{xRz\}$ and as a right successor of N the node $N'' = N \cup \{zPy\}$. Since $x(-R)z$ has a Boolean construction from the literals of N' (notice that N' contains all the literals in N and recall also that $z \in V(-R, x, N)$), the subproof tree originated from N' (which contains xRz) is closed. Consequently we can get rid of the subtree proof originated from N' and concentrate on the subtree proof originated from N'' only.

Dual tableaux for the $(\cup, \cap ; \_)$-fragment are provided with a procedure for constructing proof trees along the lines described in Section 3.1.

4.2 Soundness

The proof of soundness of the dual tableaux system for the $(\cup, \cap\,;\,\_)$-fragment can be carried out by showing that each step of the construction process of a proof tree for a formula xPy of the $(\cup, \cap\,;\,\_)$-fragment preserves falsifiability.

Lemma 4. *Let $\mathcal{T}$ be a falsifiable deduction tree and let $\mathcal{T}'$ be obtained from $\mathcal{T}$ by a step of the proof procedure described in Section 4.1. Then $\mathcal{T}'$ is a falsifiable deduction tree.*

Proof. Since $\mathcal{T}$ is falsifiable, there is a branch θ of $\mathcal{T}$ that is falsifiable. Let $\mathcal{M} = (U, m)$ and v be respectively a model and an evaluation falsifying each node of θ. If $\mathcal{T}'$ is obtained from $\mathcal{T}$ by expanding a branch different from θ, we are done. Otherwise, suppose that $\mathcal{T}'$ is obtained from $\mathcal{T}$ by decomposing a non-literal formula $x'Ex''$ occurring on the leaf node N of θ. The proof that $\mathcal{T}'$ is falsifiable can be carried out according to the type of the formula $x'Ex''$. We consider in detail only the case in which $x'Ex''$ is a $(\,;\,)$-formula. Thus, suppose that $x'Ex'' = x'(R\,;P)x''$ occurs on the leaf node N of a branch θ and that $z \in V(-R, x', \bigcup\theta)$. Then $\mathcal{T}'$ contains the branch $\theta' = \theta N'$, with $N' = N \cup \{zPx''\}$. Since $\mathcal{M}, v \not\models x'(R\,;P)x''$, we can write $\mathcal{M}, v \models x'(-(R\,;P))x''$. That is, for every $u \in U$ either $(v(x'), u) \in m(-R)$ or $(u, v(x'')) \in m(-P)$. This holds true in particular for the element $\bar{u} \in U$ such that $\bar{u} = v(z)$ and therefore either $\mathcal{M}, v \models x'(-R)z$ or $\mathcal{M}, v \models z(-P)x''$ holds.

We now show that $\mathcal{M}, v \not\models x'(-R)z$. Since $\mathcal{M}$ and v falsify N, they falsify each literal in it and, in particular, the literals used to construct $x'(-R)z$. We show by induction over the structure of $x'(-R)z$ that, if a model $\mathcal{M}$ and an evaluation v falsify all the literals in N employed for the Boolean construction of $x'(-R)z$, then $\mathcal{M}$ and v falsify $x'(-R)z$. If $x'(-R)z$ is itself a literal, then it is clearly falsified by $\mathcal{M}$ and v. Next, suppose that $x'(-R)z$ is such that $\mathsf{nnf}(x'(-R)z) = x'(S \cup T)z$, where $x'(S \cup T)z$ is a Bool_N-formula. Then, by definition of Bool_N-formula, $x'Sz$ and $x'Tz$ are Bool_N-formulae too. Thus they trivially have a Boolean construction from the literals in N and, by inductive hypothesis they are falsified by $\mathcal{M}$ and v. Consequently, $\mathcal{M}$ and v falsify $x'(S \cup T)z$ and $x'(-R)z$. Finally, let $x'(-R)z$ be such that $\mathsf{nnf}(x'(-R)z) = x'(S \cap T)z$, with $x'(S \cap T)z$ a Bool_N-formula. Then, by definition of Bool_N-formula, either $x'Sz$ or $x'Tz$ is a Bool_N-formula. Thus, either $x'Sz$ or $x'Tz$ has a Boolean construction from the literals in N and, by inductive hypothesis, either $x'Sz$ or $x'Tz$ is falsified by $\mathcal{M}$ and v. This is enough to deduce that $\mathcal{M}$ and v falsify $x'(-R)z$ as well.

Thus, $\mathcal{M}, v \not\models zPx''$ holds and hence $\mathcal{M}, v \not\models N'$, $\mathcal{M}, v \not\models \theta'$, and $\mathcal{M}, v \not\models \mathcal{T}'$. □

The preceding lemma yields immediately the soundness of our dual tableau system.

Theorem 2. *Let xPy be a relational formula of the $(\cup, \cap\,;\,\_)$-fragment. If there is a closed proof tree for xPy, then xPy is valid.*

4.3 Completeness

The notion of open saturated branch θ_S of a deduction tree $\mathcal{T}$ for a formula xPy is defined as in Section 3.2 with the exception of the item relative to (**;**)-formulae that here is formalized as follows:

- if $x'(R\,;P)y' \in N$, with N a node of θ_S, there is an $N' \in \theta_S$ such that $zPy' \in N'$, for every $z \in V(-R, x', \bigcup\theta_S)$.

Lemma 5. *Let $\mathcal{T}$ be a deduction tree for a formula xPy of the* $(\cup, \cap\,;\,\_)$*-fragment of* $\mathsf{RL}(\mathbf{1})$*. If θ_S is a saturated open branch of $\mathcal{T}$, then there exist a model $\mathcal{M} = (U, m)$ and an evaluation v that falsify θ_S.*

Proof: Let us construct a model $\mathcal{M} = (U, m)$ and an evaluation v falsifying every node of the branch θ_S. Let W_{θ_S} be the collection of all the variables occurring in the formulae of the nodes of θ_S. Then we put $U =_{\mathrm{Def}} W_{\theta_S}$ and $v(x) =_{\mathrm{Def}} x$, for every $x \in U$.

Let Lit_{θ_S} be the set of all literals occurring in the nodes of θ_S. The interpretation m is defined by $(x', y') \notin m(R)$ if and only if $x'Ry' \in Lit_{\theta_S}$. m is well defined since, by definition of open saturated branch, if $x'Ry'$ (resp., $x'(-R)y'$) occurs in a node of θ_S, then $x'(-R)y'$ (resp., $x'Ry'$) does not occur in any other node of θ_S. Next, we prove that $\mathcal{M}$ and v falsify each formula in the nodes of θ_S. For this purpose, it is convenient to introduce the set $\bigcup\theta_S$ of all the formulae contained in the nodes of θ_S, and show that $\mathcal{M}$ and v falsify each formula in $\bigcup\theta_S$. Then, since each node N of θ_S is a subset of $\bigcup\theta_S$, $\mathcal{M}$ and v falsify N as well.

Let φ be a formula of $\bigcup\theta_S$. The proof is carried out by induction over the structure of φ.

- **Base case.** φ is a literal. Clearly, by definition, $\mathcal{M}$ and v falsify all the literals in $\bigcup\theta_S$ (in fact they falsify all the literals in the nodes of θ_S).
- **Inductive step.** For simplicity, we report the proof only for the case $\varphi = x'(R\,;Q)y'$, in which case $x'(R\,;Q)y' \in N$, for some node N of θ_S. To prove that $\mathcal{M}, v \not\models x'(R\,;Q)y'$, we have to show that for every $z \in U$ (that is, $z \in W_{\theta_S}$)
$$\mathcal{M}, v \models x'(-R)z \text{ or } \mathcal{M}, v \models z(-Q)y' \tag{1}$$
holds (recall that $v(x) = x$, for every $x \in W_{\theta_S}$).
By a repeated application of the (**;**)-rule, all the formulae zQy', with $z \in V(-R, x', \theta_S)$ occur in $\bigcup\theta_S$. In particular, each of them belongs to a node of the branch and, by inductive hypothesis, $\mathcal{M}$ and v do not satisfy all of them. Thus (1) is satisfied for every $z \in V(-R, x', \bigcup\theta_S)$. We have to prove that it holds also for every $z \in W_{\theta_S} \setminus V(-R, x', \bigcup\theta_S)$. In fact we show that if $z \in W_{\theta_S} \setminus V(-R, x', \bigcup\theta_S)$, then $\mathcal{M}, v \models x'(-R)z$. The proof is by induction over the structure of $x'(-R)z$.
  - Base case: $x'(-R)z$ is a literal. Then, $x'(-R)z \notin \bigcup\theta_S$. Indeed, if $x'(-R)z \in \bigcup\theta_S$ then z has to be a member of $V(-R, x', \bigcup\theta_S)$ contradicting our hypothesis. Thus $\mathcal{M}, v \models x'(-R)z$.

- Inductive step: we distinguish the following two cases.
  - ∗ Let $\mathsf{nnf}(x'(-R)z) = x'(S \cup H)z$. Then $x'(S \cup H)z$ has been obtained from the union of $x'Sz$ and of $x'Hz$. At least one of them, say $x'Sz$ (without loss of generality), is not a $\mathsf{Bool}_{\bigcup \theta_S}$-formula, because otherwise z would belong to $V(-R, x', \bigcup \theta_S)$. Thus $x'Sz$ does not have a Boolean construction from $\bigcup \theta_S$, $z \in W_{\theta_S} \setminus V(S, x', \bigcup \theta_S)$ and therefore, by inductive hypothesis, $\mathcal{M}, v \models x'Sz$. Thus $\mathcal{M}, v \models x'(R \cup S)z$, and hence $\mathcal{M}, v \models x'(-R)z$.
  - ∗ Let $\mathsf{nnf}(x'(-R)z) = x'(S \cap H)z$. Then none of $x'Sz$ and $x'Hz$ are $\mathsf{Bool}_{\bigcup \theta_S}$-formulae, because otherwise z would belong to $V(-R, x', \bigcup \theta_S)$. Thus, $x'Sz$ and $x'Hz$ do not have a Boolean construction from $\bigcup \theta_S$ and $z \in (W_{\theta_S} \setminus V(S, x', \bigcup \theta_S)) \cap (W_{\theta_S} \setminus V(H, x', \bigcup \theta_S))$. Therefore, by inductive hypothesis, $\mathcal{M}, v \models x'Sz$ and $\mathcal{M}, v \models x'Hz$, so that $\mathcal{M}, v \models x'(R \cap S)z$, and hence $\mathcal{M}, v \models x'(-R)z$.

We have shown that $\mathcal{M}, v \models x'(-R)z$, for every $z \in W_{\theta_S} \setminus V(-R, x', \bigcup \theta_S)$, and that $\mathcal{M}, v \models z(-Q)y$, for every $z \in V(-R, x', \bigcup \theta_S)$. Consequently, for every $z \in W_{\theta_S}$ either $\mathcal{M}, v \models x'(-R)z$ or $\mathcal{M}, v \models z(-Q)y'$ and therefore $\mathcal{M}, v \not\models x'(R\,;Q)y'$, as we wished to prove. □

Theorem 3 (Completeness). *If xPy is a valid formula of the $(\cup, \cap\,;\,\_)$-fragment of $\mathbb{RL}(\mathbf{1})$ then there is a closed proof tree for xPy.*

Proof: Suppose by way of contradiction that there is no closed proof tree for xPy. Let $\mathcal{T}_S$ a proof tree produced by the procedure described above, such that all leaves are closed or not further expandible. Since $\mathcal{T}_S$ is not closed, there must be a branch θ_S of $\mathcal{T}_S$ that is not closed. Thus θ_S is an open, saturated branch, since it is not further expandible. Thus, by Lemma 5, there is a model $\mathcal{M}$ and an evaluation v falsifying each node of θ_S. This holds in particular for the root $\{xPy\}$, thus contradicting the hypothesis. □

4.4 Termination

The proof of termination of the proof procedure described in Section 4.1 can be carried out as in Section 3.2. The proof of Lemma 1 can be easily adapted to this context by observing that:

(a) in formulae of type $x(-(R\,;S))y$, the term $-R$ is always a Boolean term. Consequently the formula $x(-R)z$ originated by the $(-\,;)$-decomposition of $x(-(R\,;S))y$ can be decomposed only a finite number of times.

(b) There is a finite number of formulae with left variable w that are obtained by applications of the $(\,;)$-decomposition rule: we observe that the variable w has been introduced by the $(-\,;)$-decomposition of a formula $z(-(R\,;H))y$. By the side conditions of the $(-\,;)$-decomposition rule, the literals on the branch θ_S with right variable w can only have z as the left variable. Thus, the formulae of type $(\,;)$ that can be decomposed using the variable w must have z as the left variable and therefore, by the inductive hypothesis they have to be finite in number. By the strictness hypotheses it follows that the number of formulae with left variable w originated from $(\,;)$-decomposition is finite.

5 Conclusions and future work

We have presented decision procedures based on the method of dual tableaux for two fragments of the relational logic $\mathsf{RL}(\mathbf{1})$. These fragments, called the $(\mathsf{r}\,;\_)$- and the $(\cup,\cap\,;\_)$-fragments, are characterized by the fact that they allow only a restricted application of the composition operator "$;$". In particular, in every term of type $R\,;S$, the left argument R can be either a relational variable (for the $(\mathsf{r}\,;\_)$-fragment) or a positive Boolean term (for the $(\cup,\cap\,;\_)$-fragment).

The decision procedures have been drawn from the dual tableau system for $\mathsf{RL}(\mathbf{1})$ presented in [12] by strengthening the side conditions of the $(\,;)$-decomposition rule in such a way as to reduce the collection of object variables that can be used at each decomposition step.

In a forthcoming paper we present the detailed proofs of soundness and completeness of the $(\mathsf{r}\,;\_)$-fragment and decision procedures for some other fragments of $\mathsf{RL}(\mathbf{1})$, in particular for a fragment that admits terms of type $R;S$, where R can be any Boolean term with converse operation.

We plan to provide the complexity analysis for the decision procedures presented in the paper. Our aim is also to check the possibility of improving them by introducing, for instance, a more liberal application of the $(-\,;)$-decomposition rule, or by adding further strictness hypotheses to the proof tree construction process.

We also intend to investigate other extensions of the fragments considered here which allow relational terms containing constant relations with properties such as reflexivity, transitivity, symmetry, and so on. This will permit the definition of dual tableau-based decision procedures for the relational renderings of modal logics such as B, T, $\mathsf{S4}$, of intuitionistic logics, information logics, and context logics, such as the ones reported in [12]. We also plan to explore the possibility of importing into the relational context techniques and strategies used to prove and optimize decidability results in the field of computable set theory, such as the model checking technique introduced in [5] or the small model construction approach described in [6].

References

1. F. Baader. Description logics. In: *Reasoning Web: Semantic Technologies for Information Systems, 5th International Summer School*. Lecture Notes in Computer Science 5689, 2009, pp. 1–39.
2. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
3. D. Bresolin, private communication, 2007.
4. C. Brink, K. Britz, and R. A. Schmidt. Peirce algebras. *Formal Aspects of Computing*, 6(3):339-358, 1994.
5. D. Cantone. A fast saturation strategy for set-theoretic tableaux. In: *TABLEAUX '97: Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pp. 122–137, London, UK, 1997. Springer-Verlag.

6. D. Cantone, M. Nicolosi Asmundo. On the satisfiability problem for a 3-level quantified syllogistic. In: *Proceedings of CEDAR'08*, Sydney, Australia, 11 August, 2008, pp. 1–16.
7. A. Formisano, E. Omodeo, 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, pp. 1–10.
8. J. Golińska-Pilarek, E. Munoz-Velasco, and A. Mora. A new decision procedure for modal logic K. In: *Proceedings of the International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2009*, pp. 537–548.
9. J. Golińska-Pilarek, E. Orłowska. Tableaux and dual tableaux: Transformation of proofs. *Studia Logica*, 85(3):283-302, 2007.
10. R. Maddux. Relation algebras. In: *C. Brink, W. Kahl and G. Schmidt (eds.) Relational Methods in Computer Science. Advances in Computer Science.* Springer: Wien, New York (1997).
11. E. Orłowska. Relational interpretation of modal logics. In: *H. Andreka, D. Monk, and I. Nemeti eds., Algebraic Logic. Colloquia Mathematica Societatis Janos Bolyai*, vol. 54, pp. 443–471, North Holland, 1988.
12. E. Orłowska, J. Golińska-Pilarek. Dual Tableaux: Foundations, Methodology, Case Studies. *Book submitted*.
13. C. S. Peirce. Note B: the logic of relatives. In: *C. S. Peirce (ed) Studies in Logic by Members of the Johns Hopkins University, Little, Brown, and Co.*, Boston, pp. 187–203 (1883).
14. R. A. Schmidt, E. Orłowska, and U. Hustadt. Two proof systems for Peirce algebras. In: *R. Berghammer, B. Möller, and G. Struth, (eds.). Relational and Kleene-Algebraic Methods in Computer Science: 7th International Seminar on Relational Methods in Computer Science and 2nd International Workshop on Applications of Kleene Algebra, Bad Malente, Germany, May 12-17, 2003, Revised Selected Papers, Lecture Notes in Computer Sciencce 3051, Springer*, 2004, pp. 238–251.
15. A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6(3):73-89, 1941.
16. A. Tarski, S. Givant. A Formalization of Set Theory without Variables. *American Mathematical Society Colloquium Publications*, Providence, Rhode Island, 1987.