A Dual Tableau-based Decision Procedure for a Relational Logic with the Universal Relation

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 present a first result towards the use of entailment inside relational dual tableau-based decision procedures. To this end, we introduce a fragment of $\mathsf{RL}(\mathbf{1})$, called $(\{\mathbf{1}, \cup, \cap\}\,;\_)$, which admits a restricted form of composition. We prove the decidability of the $(\{\mathbf{1}, \cup, \cap\}\,;\_)$-fragment by defining a dual tableau-based decision procedure with a suitable blocking mechanism and where the decomposition rules for compositional formulae are modified so as to deal with the constant $\mathbf{1}$ while preserving termination.
The $(\{\mathbf{1}, \cup, \cap\}\,;\_)$-fragment properly includes the logics presented in previous work and, therefore, it allows one to express, among others, the multi-modal logic K with union and intersection of accessibility relations and the description logic $\mathcal{ALC}$ with union and intersection of roles.

1 Introduction

The relational representation of various non-classical propositional logics has been systematically analyzed in the last decades [16]. A uniform relational framework based on the logic of binary relations $\mathsf{RL}(\mathbf{1})$, presented in [15] and called *relational dual tableau*, showed to be an effective logical means to represent in a modular way three fundamental components of a formal system: its syntax, semantics, and deduction system. Relational systems have been defined for modal and intuitionistic logics, for relevant and many-valued logics, for reasoning in logics of information and data analysis, for reasoning about time and space, etc.

The formalization of non-classical logics in $\mathsf{RL}(\mathbf{1})$ is based on the fact that once the Kripke-style semantics of the logic under consideration is known, formulae can be treated as relations. In particular, since in Kripke-style semantics formulae are interpreted as collections of objects, in their relational representation they are seen as right ideal relations. In the case of binary relations this means that $(R\,;\mathbf{1}) = R$ is satisfied, where '$;$' is the composition operation on binary relations and '$\mathbf{1}$' is the universal relation.

One of the most useful features of the relational methodology is that, given a logic with a relational formalization, we can construct its relational dual tableau in a systematic and modular way.

Though the relational logic $\mathsf{RL}(\mathbf{1})$ is undecidable, it contains several decidable fragments. In many cases, however, dual tableau proof systems are not decision procedures for decidable fragments of $\mathsf{RL}(\mathbf{1})$. This is mainly due to the way decomposition and specific rules are defined and also to the strategy of proof construction.

Over the years, great efforts have been spent to construct dual tableau proof systems for various logics known to be decidable; little care has been taken, however, to design dual tableau-based decision procedures for them. On the other hand, it is well known that when a proof system is designed and implemented, it is important to have decision procedures for decidable logics. In [10], for example, an optimized relational dual tableau for $\mathsf{RL}(\mathbf{1})$, based on Binary Decision Graphs, has been implemented. However, such an implementation turns out not to be effective for decidable fragments.

As far as we know, relational dual tableau-based decision procedures can be found in [16] for fragments of $\mathsf{RL}(\mathbf{1})$ corresponding to the class of first-order formulae in prenex normal form with universal quantifiers only; in [12, 13] for the relational logic corresponding to the modal logic K; in [4, 5] for fragments of RL characterized by some restrictions in terms of type $(R\,;S)$; in [11] for a class of relational logics admitting a single relational constant with the properties of reflexivity, transitivity, and heredity; and in [3] for a class of relational fragments extending the ones introduced in [11] by allowing a countable infinity of relational constants with the properties of reflexivity, transitivity, and heredity.

Throughout the paper terms of type $(R\,;S)$ will be referred to as *compositional terms*. Similarly, formulae with compositional terms will be referred to as *compositional formulae*.

In some cases, like in [11] and in [3], fragments with relational constants satisfying some fixed properties are considered. Therefore, dual tableau-based decision procedures are endowed with specific rules to treat relational constants and their properties. The design of specific rules often needs much care in order to guarantee termination of the related proof procedure. This task is delicate especially when the proof system provides several specific rules for different relational constants, and when the relational constants are related to one another.

An alternative way to treat properties of relational constants and variables, and of relations between them, is to use *relational entailment*. Relational entailment can be formalized in the logic $\mathsf{RL}(\mathbf{1})$ as follows. Given relations $R, R_1, \ldots, R_n$, with $n \geq 1$, one has that $R_1{=}\mathbf{1}, \ldots, R_n{=}\mathbf{1}$ imply $R{=}\mathbf{1}$ in a model if and only if $(\mathbf{1}\,;(-(R_1 \cap \ldots \cap R_n))\,;\mathbf{1}) \cup R{=}\mathbf{1}$ holds. It means that entailment is expressible as a term of the language of logic $\mathsf{RL}(\mathbf{1})$ and, as a consequence, any validity checker for $\mathsf{RL}(\mathbf{1})$ can also be applied to entailment verification.

Introduction of entailment inside relational proof systems allows one to eliminate specific rules and, consequently, to keep the set of decomposition rules small. This approach can be convenient in implementations of automated theorem provers provided that decomposition rules and their application strategy are designed in a suitable way. In its relational formalization, however, entailment involves the universal constant $\mathbf{1}$ on the left hand side and on the right hand side

of compositional terms. Thus, the design of a relational dual tableau-based decision procedure where entailment is admitted is a challenging task that requires special care.

In this paper we present a first result towards the use of entailment inside relational dual tableau-based decision procedures. To this purpose, we introduce a fragment of $\mathsf{RL}(\mathbf{1})$, called $(\{\mathbf{1}, \cup, \cap\}\ ;\ \_)$, admitting a restricted form of composition where the left subterm R of any term of type $(R;S)$ is allowed to be either the constant $\mathbf{1}$ or any term constructed from the relational variables by applying only the operators of relational intersection and union. Similarly, terms of type $(R\ ;\ \mathbf{1})$ are admitted only if R is a Boolean term involving relational variables and the operators of intersection and union.

We prove that the $(\{\mathbf{1}, \cup, \cap\}\ ;\ \_)$-fragment is decidable by defining a dual tableau-based decision procedure where a suitable blocking mechanism has been introduced and rules for compositional and complemented compositional formulae have been appropriately modified to deal with the constant $\mathbf{1}$ while preserving termination.

Such fragment properly includes the logics presented in [4] and, therefore, it can express the multi-modal logic K with union and intersection of accessibility relations and the description logic $\mathcal{ALC}$ with union and intersection of roles. Furthemore it can also express, via entailment, properties of the form '$\mathsf{r} \subseteq -(\mathsf{s}_1 \cup \mathsf{s}_2)$' and '$(\mathsf{s}_1 \cup \mathsf{s}_2) \subseteq -\mathsf{r}$', where r, s_1, and s_2 are relational variables.

The rest of the paper is organized as follows. In Sect. 2 we briefly review the syntax and semantics of the relational logic $\mathsf{RL}(\mathbf{1})$ together with its dual tableau and in Sect. 3 we introduce some useful notions which will be used throughout the paper. Then in Sect. 4 we present the $(\{\mathbf{1}, \cup, \cap\}\ ;\ \_)$-fragment and its dual tableau-based decision procedure. Finally, in Sect. 5, we draw our conclusions and give some hints for future work.

2 The Relational Logic RL(1) and its Dual Tableau

In this section we review the logic $\mathsf{RL}(\mathbf{1})$ and its dual tableau in full extent (see also [5] and [16]).

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 a *relational constant*. Then, the set $\mathbb{RT}$ of *relational terms* of $\mathsf{RL}(\mathbf{1})$ is the smallest set of terms (with respect to inclusion) built from relational variables and the relational constant $\mathbf{1}$ with the *relational operators* '$\cap$', '$\cup$', '$;$' (binary) and '$-$', '$^\smile$' (unary).

Let $\mathbb{OV}$ be a countably infinite set of *object (individual) variables* $x, y, z, w, \ldots$. Then, $\mathsf{RL}(\mathbf{1})$-formulae have the form xRy, where $x, y \in \mathbb{OV}$ and $R \in \mathbb{RT}$. $\mathsf{RL}(\mathbf{1})$-formulae of type $x\mathbf{1}y$ and $x\mathsf{r}y$, with $\mathsf{r} \in \mathbb{RV}$, are called *atomic* $\mathsf{RL}(\mathbf{1})$-formulae. A *literal* is either an atomic formula or its complementation (namely a formula of type $x(-\mathbf{1})y$ or $x(-\mathsf{r})y$). For a relational operator '$\sharp$' other than '$-$', by a $(\sharp)$-*term* we mean a relational term whose lead operator is '$\sharp$', and by a $(-\sharp)$-*term* we denote a complemented $(\sharp)$-term. For example, the term $(\mathsf{r}_1 \cup \mathsf{s}) \cap (-\mathsf{r}_2\ ;\ \mathsf{s})$ is a $(\cap)$-term and has '$\cap$' as its lead operator, whereas $-((\mathsf{r}_1 \cup \mathsf{s}) \cap (-\mathsf{r}_2\ ;\ \mathsf{s}))$ is

a $(-\cap)$-term. A $(\sharp)$-*formula* (resp., $(-\sharp)$-*formula*) is a formula whose relational term is a $(\sharp)$-*term* (resp., $(-\sharp)$-*term*). A *Boolean term* is a relational term built from relational variables with the Boolean operators '$-$', '$\cup$', and '$\cap$'. A *positive Boolean term* is a Boolean term in which the operator $-$ does not occur.

$\mathsf{RL}(\mathbf{1})$-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 $m : \mathbb{RV} \to \wp(U \times U)$ is a given map which is homomorphically extended to the whole collection $\mathbb{RT}$ of relational terms as follows:

- $m(\mathbf{1}) = U \times U$; $\quad m(-R) = (U \times U) \setminus m(R)$;
- $m(R \cup S) = m(R) \cup m(S)$; $\quad m(R \cap S) = m(R) \cap m(S)$;
- $m(R \,;\, S) = m(R) \,;\, m(S)$
  $= \{(a, b) \in U \times U : (a, c) \in m(R) \text{ and } (c, b) \in m(S), \text{ for some } c \in U\}$;
- $m(R^{\smile}) = (m(R))^{\smile} = \{(b, a) \in U \times U : (a, b) \in m(R)\}$.

Let $\mathcal{M} = (U, m)$ be an $\mathsf{RL}(\mathbf{1})$-model. A *valuation* in $\mathcal{M}$ is any function $v : \mathbb{OV} \to U$. An $\mathsf{RL}(\mathbf{1})$-formula xRy is *satisfied* by an $\mathsf{RL}(\mathbf{1})$-model $\mathcal{M} = (U, m)$ and by a valuation v in $\mathcal{M}$ (in which case we write $\mathcal{M}, v \models xRy$) provided that $(v(x), v(y)) \in m(R)$. An $\mathsf{RL}(\mathbf{1})$-formula xRy is (a) *true* in a model $\mathcal{M} = (U, m)$, if $\mathcal{M}, v \models xRy$, for every valuation v in $\mathcal{M}$; (b) *valid*, if it is *true* in all $\mathsf{RL}(\mathbf{1})$-models; (c) *falsified* by a model $\mathcal{M} = (U, m)$ and by a valuation v in $\mathcal{M}$, if $\mathcal{M}, v \not\models xRy$; (d) *falsifiable*, if there exist a model $\mathcal{M}$ and a valuation v in $\mathcal{M}$ such that $\mathcal{M}, v \not\models xRy$. An $\mathsf{RL}(\mathbf{1})$-*set* is a finite set $\{\varphi_1, \ldots, \varphi_n\}$ of $\mathsf{RL}(\mathbf{1})$-formulae such that, for every $\mathsf{RL}(\mathbf{1})$-model $\mathcal{M}$ and for every valuation v in $\mathcal{M}$, we have $\mathcal{M}, v \models \varphi_i$, for some $i \in \{1, \ldots, n\}$. Clearly, the first-order disjunction of the formulae in an $\mathsf{RL}(\mathbf{1})$-set is valid in first-order logic.

Proof development in dual tableaux proceeds by systematically decomposing the (disjunction of the) formula(e) to be proved till a validity condition is detected, expressed in terms of axiomatic sets (see below). The method originated in [17] (see also [18]). Such an analytic approach is similar to Beth's tableau method [1], with the difference that the two systems work in a dual manner. Duality between tableaux and dual tableaux has been analyzed in depth in [14].

$\mathsf{RL}(\mathbf{1})$-dual tableaux consist of *decomposition rules*, which allow one to analyze the structure of the formula to be proved valid, and of *axiomatic sets*, which specify closure conditions. The decomposition rules for $\mathsf{RL}(\mathbf{1})$ are listed in Table 1. In these rules, ',' and '|' are interpreted respectively as disjunction and conjunction. A rule is $\mathsf{RL}(\mathbf{1})$-*correct* provided that its premise is an $\mathsf{RL}(\mathbf{1})$-set if and only if each of its consequents is an $\mathsf{RL}(\mathbf{1})$-set. The rules presented in Table 1 have been proved $\mathsf{RL}(\mathbf{1})$-correct in [16].

An $\mathsf{RL}(\mathbf{1})$-axiomatic set is any set of $\mathsf{RL}(\mathbf{1})$-formulae containing a subset of one of the following two forms: (**Ax 1**)$\{xRy, x(-R)y\}$, (**Ax 2**) $\{x\mathbf{1}y\}$.

Clearly, an $\mathsf{RL}(\mathbf{1})$-axiomatic set is also an $\mathsf{RL}(\mathbf{1})$-set.

An $\mathsf{RL}(\mathbf{1})$-*proof tree* for an $\mathsf{RL}(\mathbf{1})$-formula xPy is an ordered tree whose nodes are labelled by disjunctive sets of formulae such that the following properties are satisfied:

- the root is labelled with $\{xPy\}$;

Table 1. $\mathsf{RL}(\mathbf{1})$ decomposition rules.

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

- each node, except the root, is obtained from its predecessor node by applying a decomposition rule in Table 1 to one of the formulae labelling it;
- 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 *branch* θ of a proof tree is any of its maximal paths; we denote with $\bigcup \theta$ the set of all the formulae contained in the nodes of θ, and with W_θ the collection of the object variables occurring in the formulae contained in the nodes of θ. A node of an $\mathsf{RL}(\mathbf{1})$-proof tree is *closed* if its associated set of formulae is 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 $\mathsf{RL}(\mathbf{1})$-formula is $\mathsf{RL}(\mathbf{1})$-*provable* if there is a closed $\mathsf{RL}(\mathbf{1})$-proof tree for it, referred to as an $\mathsf{RL}(\mathbf{1})$-*proof*.

A node of an $\mathsf{RL}(\mathbf{1})$-proof tree is *falsified* by a model $\mathcal{M} = (U, m)$ and a valuation v in $\mathcal{M}$ if every formula xRy in its set of formulae is falsified by $\mathcal{M}$ and v. A node is *falsifiable* if there exist a model $\mathcal{M}$ and a valuation v in $\mathcal{M}$ which falsify it.

Correctness and completeness of the $\mathsf{RL}(\mathbf{1})$-dual tableau are proved in [16]. However, the logic $\mathsf{RL}(\mathbf{1})$ is undecidable. This follows from the undecidability of the equational theory of representable relation algebras discussed in [19].

3 Useful Notions and Properties

We introduce some useful notions and properties which are needed for the presentation of the results of the paper.
Let P be any relational term in $\mathsf{RL}(\mathbf{1})$. The following identities hold:

$(\mathbf{1} \cup P) \equiv (P \cup \mathbf{1}) \equiv \mathbf{1}$	$((-\mathbf{1}) \cup P) \equiv (P \cup (-\mathbf{1})) \equiv P$
$(\mathbf{1} \cap P) \equiv (P \cap \mathbf{1}) \equiv P$	$((-\mathbf{1}) \cap P) \equiv (P \cap (-\mathbf{1})) \equiv (-\mathbf{1})$
$(-(-\mathbf{1})) \equiv \mathbf{1}$	

Let H be a relational term in $\mathsf{RL}(\mathbf{1})$ and let H' be obtained from H by systematically simplifying H by means of the above identities. If the simplification is carried out in an inside-out way, the computational complexity of the transformation of H into H' is linear in the length of H. Moreover, the following lemma holds (proof of Lemma 1 can be found in [6]).

Lemma 1. *Let H be a relational term and let H' be constructed as outlined above. Then every Boolean subterm P of H' either is equal to $\mathbf{1}$, or is equal to $-\mathbf{1}$, or it does not contain $\mathbf{1}$.*

It is easy to check that $m(H) = m(H')$ holds for every $\mathsf{RL}(\mathbf{1})$-model $\mathcal{M} = (U, m)$ and for every $H \in \mathsf{RL}(\mathbf{1})$. Therefore we can restrict ourselves to relational terms simplified as described above.

Parsing trees. It is possible to associate a *parsing tree* S_P to each relational term P of $\mathsf{RL}(\mathbf{1})$, as with formulae of standard first-order logic (see [9] and [8] for details on the construction of parsing trees in first-order logic). Let S_P be the parsing tree for P, and let ν be a node of S_P. We say that a relational term Q *occurs* within P at position ν if the subtree of S_P rooted at ν is identical to S_Q. In this case we refer to ν as an *occurrence* of Q in P and to the path from the root of S_P to ν as its *occurrence path*.

An occurrence of a relational term Q within a relational term P is *positive* if its occurrence path deprived of its last node contains an even number of nodes labelled with $\{-\}$. Otherwise, the occurrence is said to be *negative*.

Normal forms and term components. Next we introduce the notion of complement normal form for Boolean relational terms, the notions of Bool_N-formula, of Bool-construction from N, where N is a set of formulae, and of set of components of a relational term.

The *complement normal form* of a term R is a term $\mathsf{nf}_-(R)$ obtained by successive applications of the De Morgan laws and of the law of double negation to R.

A term is said to be in complement normal form whenever each occurrence of the complement operator in it acts only on relational variables or constants.

Clearly, for every Boolean relational term R, the formulae $xR\,y$ and $x\ \mathsf{nf}_-(R)\ y$ are *logically equivalent*, that is $\mathcal{M}, v \models xRy$ if and only if $\mathcal{M}, v \models x\ \mathsf{nf}_-(R)y$, for every model $\mathcal{M} = (U, m)$ and every valuation v in $\mathcal{M}$.

Let N be a set of formulae, and let R, S be two Boolean relational terms. We define the notion of Bool_N-formulae as follows:

- every literal xRy in N is a Bool_N-formula;
- every formula of the form $x(R \cap S)y$ is a Bool_N-formula, provided that either xRy is a Bool_N-formula and S is in complement normal form, or xSy is a Bool_N-formula and R is in complement normal form;
- every formula of the form $x(R \cup S)y$ is a Bool_N-formula if both xRy and xSy are Bool_N-formulae.

Clearly, if xSy is a Bool_N-formula, then xSy is syntactically equal to $x\ \mathsf{nf}_{-}(S)\ y$ and we write $xSy = x\ \mathsf{nf}_{-}(S)\ y$. We say that a formula xRy has a Bool-construction from N if $x\ \mathsf{nf}_{-}(R)\ y$ is a Bool_N-formula.

For example, given the set of formulae $N = \{x(-\mathsf{r})z, x\mathsf{s}z, x(-\mathsf{p})y, z(\mathsf{p} \cup \mathsf{s})y\}$, we have that the formula $x(((-\mathsf{r}) \cup \mathsf{s}) \cap \mathsf{q})z$ is a Bool_N-formula because $x((-\mathsf{r}) \cup \mathsf{s})z$ is a Bool_N-formula and $x\mathsf{q}z$ is in complement normal form. On the other hand the formula $x(\mathsf{s} \cap (-(\mathsf{q} \cup \mathsf{p})))z$ is not a Bool_N-formula because although $x\mathsf{s}z$ is a Bool_N-formula, $x(-(\mathsf{q} \cap \mathsf{p}))z$ is not in complement normal form. Both formulae, however, have a Bool-construction from N because $x(((-\mathsf{r}) \cup \mathsf{s}) \cap \mathsf{q})z$ is a Bool_N-formula and $x\ \mathsf{nf}_{-}(\mathsf{s} \cap (-(\mathsf{q} \cup \mathsf{p})))z = x(\mathsf{s} \cap ((-\mathsf{q}) \cap (-\mathsf{p})))z$ is a Bool_N-formula. In this latter case, specifically, $x\mathsf{s}z$ is a Bool_N-formula and $x((-\mathsf{q}) \cap (-\mathsf{p}))z$ is in complement normal form, although it is not a Bool_N-formula.

Given a term R in $\mathbb{RT}$, an object variable x, and a set N of formulae, we define $V(R, x, N)$ as the set of object variables z such that xRz has a Bool-construction from N.

Let P be a term in $\mathbb{RT}$. We define recursively the set $\mathsf{cp}(P)$ of *the components of the term* P as follows:

- if P is the relational constant $\mathbf{1}$, or a relational variable, or their complements, then $\mathsf{cp}(P) = \{P\}$;
- if $P = --B$, then $\mathsf{cp}(P) = \{P\} \cup \mathsf{cp}(B)$;
- if $P = B^{\smile}$, then $\mathsf{cp}(P) = \{P\} \cup \mathsf{cp}(B)$;
- if $P = B \sharp C$ (resp., $P = -(B \sharp C)$), then $\mathsf{cp}(P) = \{P\} \cup \mathsf{cp}(B) \cup \mathsf{cp}(C)$ (resp., $\mathsf{cp}(P) = \{P\} \cup \mathsf{cp}(-B) \cup \mathsf{cp}(-C)$), for every binary relational operator $\sharp$.

Clearly $\mathsf{cp}(P)$ is finite, for any relational term P.

4 The Fragment $(\{\mathbf{1}, \cup, \cap\}\ ;\ \_)$ and its Decision Procedure

4.1 The Fragment $(\{\mathbf{1}, \cup, \cap\}\ ;\ \_)$

Formulae of the fragment $(\{\mathbf{1}, \cup, \cap\}\ ;\ \_)$ of $\mathsf{RL}(\mathbf{1})$ are characterized by the fact that the left subterm R of any term of type $(R\ ;\ S)$ in them is only allowed to be either the constant $\mathbf{1}$ or a term constructed from the relational variables of $\mathbb{RV}$ by applying only the '$\cup$' and '$\cap$' operators, whereas the right subterm S of $(R;S)$ can involve all the relational operators of $\mathsf{RL}(\mathbf{1})$ but the converse operator '$\smile$'.

Formally, the set $\mathbb{RT}_{(\{\mathbf{1},\cup,\cap\};\_)}$ of the terms allowed in $(\{\mathbf{1}, \cup, \cap\}\ ;\ \_)$-formulae is the smallest set of terms containing the constant $\mathbf{1}$ and the variables in $\mathbb{RV}$, and such that if $P, Q, B, H \in \mathbb{RT}_{(\{\mathbf{1},\cup,\cap\};\_)}$ and $S \in \{H, \mathbf{1}\}$, with

- B a Boolean term containing neither $\mathbf{1}$ nor the complement operator, and
- H containing the constant $\mathbf{1}$ only inside terms of type $(B\ ;\ \mathbf{1})$,

then $(-P), (P \cup Q), (P \cap Q), (B\ ;\ S), (\mathbf{1}\ ;\ S) \in \mathbb{RT}_{(\{\mathbf{1},\cup,\cap\};\_)}$.
Examples of formulae of the $(\{\mathbf{1}, \cup, \cap\}\ ;\ \_)$-fragment are: $x(-((\mathsf{r}_1 \cup \mathsf{s})\ ;\ (\mathsf{p}\ ;\ \mathbf{1})))y$, $x(\mathbf{1}\ ;\ ((\mathsf{r}_1 \cup \mathsf{s})\ ;\ -(((\mathsf{q} \cup \mathsf{p}) \cap \mathsf{r}_1)\ ;\ \mathbf{1})))y$, and $x(\mathbf{1}\ ;\ (((\mathsf{r}_1 \cup \mathsf{s}) \cap \mathsf{r}_2)\ ;\ \mathbf{1}))y$. The latter formula can be rewritten as $x(\mathbf{1}\ ;\ (-(-(\mathsf{r}_1 \cup \mathsf{s}) \cup -\mathsf{r}_2)\ ;\ \mathbf{1}))y$, where $(-(\mathsf{r}_1 \cup \mathsf{s}) \cup -\mathsf{r}_2)$ is a relational term formalizing the property '$(\mathsf{r}_1 \cup \mathsf{s}) \subseteq -\mathsf{r}_2$'.

Table 2. Decomposition rules proper of the $(\{\mathbf{1}, \cup, \cap\}\,;\,\_)$-fragment.

$$(;)_a \quad \frac{x(B\,;\,S)y}{zSy,\, x(B\,;\,S)y,} \qquad (z \text{ is an object variable in } V(-B, x, N))$$

$$(;)_b \quad \frac{x(\mathbf{1}\,;\,S)y}{zSy,\, x(\mathbf{1}\,;\,S)y} \qquad (z \text{ is any object variable})$$

$$(-;)_a \quad \frac{x\,{-}(B\,;\,\mathbf{1})y}{x(-B)z} \qquad (z \text{ is a new object variable})$$

$$(-;)_b \quad \frac{x\,{-}(\mathbf{1}\,;\,S)y}{z(-S)y} \qquad (z \text{ is a new object variable})$$

4.2 A Dual Tableau Calculus for $(\{\mathbf{1}, \cup, \cap\}\,;\,\_)$

The decomposition rules for Boolean formulae of our dual tableau-based calculus are just the ones in Table 1. Concerning the decomposition rule for $(;)$-formulae, it is convenient to distinguish between $(;)$-formulae of type $x(B\,;S)y$ and of type $x(\mathbf{1}\,;S)y$. The rule for $(;)$-formulae of type $x(B\,;S)y$ is the $(;)_a$ -rule of Table 2. There, z is an object variable belonging to $V(-B, x, N)$, where N stands for the current node. Notice that if $S = \mathbf{1}$, the node resulting from the decomposition step is axiomatic. In case of $(;)$-formulae of type $x(\mathbf{1}\,;\,S)y$, we apply the rule $(;)_b$ in Table 2. The variable z used in rule $(;)_b$ is any variable on the current node, provided that the current branch does not already contain the formula zSy. Otherwise, $x(\mathbf{1}\,;\,S)y$ cannot be decomposed with z. If $S = \mathbf{1}$, the same remark made for rule $(;)_a$, for the node resulting from the decomposition step, holds here as well.

Concerning $(-;)$-formulae, we consider first the case of formulae of type $x\,{-}(B\,;\,S)y$. If $S \neq \mathbf{1}$, such formulae are decomposed by means of the $(-;)$-rule in Table 1. Otherwise, when $S = \mathbf{1}$, we use the rule $(-;)_a$ of Table 2. In the case of formulae of type $x\,{-}(\mathbf{1}\,;\,S)y$, with $S \neq \mathbf{1}$, we use instead the rule $(-;)_b$ of Table 2, with z an object variable new to the current node. The rule can be applied provided that the current branch does not contain any formula of the form $z'(-S)y$, for any 'new' variable z' (otherwise, the formula $x\,{-}(\mathbf{1}\,;S)y$ cannot be decomposed). The formula $x(-(\mathbf{1}\,;\,\mathbf{1}))y$ is not decomposed.

Some remarks on the rules $(;)_a$, $(;)_b$, $(-;)_a$, and $(-;)_b$ in Table 2 are in order. The idea behind the definition of the side condition of the rule $(;)_a$ takes inspiration from the side condition of expansion rules for universally quantified formulae present in various well known tableau-based proof systems (see for instance [2]). The introduction of the set $V(-B, x, N)$ is motivated by the fact that our relational fragment admits compositional terms $(B\,;\,S)$, where B may be a compound term. Observe that for every $\mathsf{RL}(\mathbf{1})$-model $\mathcal{M} = (U, m)$, $m(\mathbf{1}) = U \times U$ so that, $\mathcal{M}, v \models x\mathbf{1}z$ and $\mathcal{M}, v \not\models x(-\mathbf{1})z$ hold, for every valuation v and object variables x and z. Thus, we shall assume without loss of generality that each node of any dual tableau for formulae of the $(\{\mathbf{1}, \cup, \cap\}\,;\,\_)$-fragment contains implicitly all literals of type $x(-\mathbf{1})z$. This accounts for the fact that the decomposition rules $(-;)_a$ and $(-;)_b$ do not introduce $z(-\mathbf{1})y$ and $x'(-\mathbf{1})z$, respectively, on the new node, and rule $(;)_b$ restricts z to be any variable on the

current node, rather than any possible variable. At any rate, we shall prove that such a restriction preserves the completeness of the procedure.

It is convenient to introduce the notion of *deduction tree* for $\mathsf{RL}(\mathbf{1})$-formulae to give a step-by-step description of the proof tree construction process.

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 will be 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 $(\{\mathbf{1}, \cup, \cap\}\,;\_)$-formula. 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.[3] *The tree obtained from $\mathcal{T}$ by applying to N either one of the decomposition rules in Table 1 (for Boolean formulae and for $(-;)$-formulae of type $x' \,{-}(B\,;S)y$, with $S \neq \mathbf{1}$), or one of the decomposition rules in Table 2 (for $(;)$-formulae and for $(-;)$-formulae of type $x' \,{-}(B;\mathbf{1})y$ and of type $x \,{-}(\mathbf{1};S)y$) is a deduction tree for xPy. More precisely, rules applications are described as follows:*

- *if a formula $x'Qy$ occurs in N and a rule with a single conclusion set of formulae Γ (resp., a branching rule with the conclusion sets Γ_1 and Γ_2) is applicable to $x'Qy$, then we append the node $N' = (N \setminus \{x'Qy\}) \cup \Gamma$ as the successor of N in θ (resp., the node $N'_1 = (N \setminus \{x'Qy\}) \cup \Gamma_1$ as the left successor of N and the node $N'_2 = (N \setminus \{x'Qy\}) \cup \Gamma_2$ as the right successor of N in θ).*

Given a branch θ of a deduction tree, each object variable in $W_\theta \setminus \{x, y\}$ is generated by an application of a $(-;)$-decomposition rule. We say that a variable w is an *ancestor of degree n* of a variable $z \in W_\theta \setminus \{x, y\}$ if there is a sequence $z_1, \ldots, z_n$ of variables in $W_\theta \setminus \{x, y\}$, with $z_n = z$ and $n \geq 1$, such that z_1 is generated by a $(-;)$-formula $w(-(B_0\,;S_0))y$, z_2 is generated by a $(-;)$-formula $z_1(-(B_1;S_1))y$,..., z_n is generated by a $(-;)$-formula $z_{n-1}(-(B_{n-1};S_{n-1}))y$, where $w(-(B_0\,;S_0))y$, $z_1(-(B_1\,;S_1))y$,..., $z_{n-1}(-(B_{n-1}\,;S_{n-1}))y$ are formulae of θ. In such a case, we say that z_1 is a *descendant of degree* 1 of w and that $z_n = z$ is a descendant of degree n of w.

It is useful to define a total order $<_\theta$ among variables in W_θ such that:

- $x <_\theta w$, for every $w \in W_\theta \setminus \{x\}$,
- $x_1 <_\theta x_2$, for every $x_1, x_2 \in W_\theta \setminus \{x, y\}$, with x_1 introduced in θ before x_2,
- $y <_\theta z$, for every z descendant of y,
- $w <_\theta y$, for every w that is not a descendant of y.

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

Remark 1. Notice that the relationship ancestor/descendant is based on the literals of type $x'(-\mathsf{r})z$ that are generated by applying either the $(-;)$-rule of Table 1 or the $(-;)_a$ -rule of Table 2, and, possibly, the Boolean decomposition rules.

Remark 2. By the construction of $\mathbb{RT}_{(\{\mathbf{1},\cup,\cap\};\_)}$, a deduction tree for a formula xPy may contain formulae of type $x(\mathbf{1};S_1)y$ and of type $x(-(\mathbf{1};S_2))y$ only if their left variable is x. This is motivated by the fact that each of these formulae can be obtained only by the Boolean decomposition of xPy. Any variable z resulting from the decomposition of a $(-;)$-formula of type $x(-(\mathbf{1};S))y$ is not a descendant of x. However, according to the definition of the order $<_\theta$, $x <_\theta z$ holds.

The following notions will be used in the next section to turn our tableau calculus for $(\{\mathbf{1},\cup,\cap\};\_)$ into a terminating proof tree construction procedure.

Let θ be a branch of a deduction tree, and let $z(-(B;S))y$ and $z'(-(B;S))y$ be two $(-;)$-formulae occurring in θ. We say that $z'(-(B;S))y$ *blocks* $z(-(B;S))y$ (and that $z(-(B;S))y$ *is blocked by* $z'(-(B;S))y$), if the following conditions are satisfied:

- $z(-(B;S))y$ and $z'(-(B;S))y$ are identical with the exception of the left object variable,
- $z'(-(B;S))y$ has been already decomposed in θ using the variable w,
- for every $(;)$-formula $z(B_1;Q)y$ occurring in θ such that $z(-B_1)w$ has a Bool-construction from the set of literals resulting from the Boolean decomposition of $z(-B)w$, the $(;)$-formula $z'(B_1;Q)y$ occurs in θ as well.

4.3 A Proof Tree Construction Procedure for $(\{\mathbf{1},\cup,\cap\};\_)$

Starting with an initial deduction tree $\mathcal{T}_0$ for a given formula xPy, the following procedure constructs a proof tree for xPy.

1. For every non-axiomatic branch θ of the current deduction tree,
2. while θ is non-axiomatic and is further expandable, let z be the smallest variable w.r.t. $<_\theta$ such that formulae on θ with left variable z have not been decomposed in θ. Apply to the formulae on θ having left variable z the decomposition rules in the following order: Boolean rules, $(-;)$-rules, rule $(;)_a$, and then apply rule $(;)_b$ to decompose the $(;)$-formulae of type $x(\mathbf{1};S)y$ in θ with the variable z in a systematic way under the following restrictions:
   a. all the rules can be applied at most once with the same premise;
   b. every formula of type $(-;)$, $z(-(B;S))y$ is not decomposed provided that it is blocked by a $(-;)$-formula $z'(-(B;S))y$ occurring in θ.
   If $z'(-(B;S))y$ was decomposed in θ with the variable w, then for every literal $z'(-\mathsf{r})w \in \bigcup\theta$ (obtained from the application of the Boolean rules to $z'(-B)w$) we store the literal $z(-\mathsf{r})w$ in $Lit_{(-;)}$, a set (empty at the beginning of the execution of the procedure) collecting literals not explicitly occurring in θ that are needed to construct $\mathcal{M}_\theta$ (see step 4).

3. If the branch θ is axiomatic and all the other branches on the current deduction tree are axiomatic, then the current deduction tree is a proof tree for xPy and we terminate. Otherwise, if the branch θ is axiomatic and there are still non-axiomatic branches on the current deduction tree, return to step 1.
4. Otherwise, if θ is non-axiomatic, namely it is a non-axiomatic not further expandable branch, we construct from θ the model $\mathcal{M}_\theta = (U_\theta, m_\theta)$ defined as follows. We put $U_\theta = W_\theta$. Next, let Lit_θ be the set of all literals occurring in θ, and let $Lit_{(-;)}$ be defined as in step 2. We define the interpretation $\mathcal{M}_\theta$ by putting $(x', y') \notin m_\theta(R)$ if and only if $x'Ry' \in (Lit_\theta \cup Lit_{(-;)})$. Let $v_\theta : \mathbb{OV} \to U_\theta$ be a valuation such that $v_\theta(x) =_{Def} x$, for every $x \in U_\theta$. We terminate returning θ, $\mathcal{M}_\theta$, and v_θ.

The next lemma states two useful properties of the formulae occurring on the deduction trees constructed by the proof procedure above. Its proof can be carried out by induction on proof construction and by case distinction on the structure of $x'Rx''$.

Lemma 2. *Let $\mathcal{T}$ be a deduction tree for xPy constructed by an execution of the procedure described above. If $x'Rx''$ is a formula of a branch θ of $\mathcal{T}$, then (i) $R \in \mathsf{cp}(P)$, and (ii) if R contains the composition operator, then $x'' = y$.*

Termination of the procedure. Let $\mathcal{T}$ be a proof tree for a formula xPy of the $(\{\mathbf{1}, \cup, \cap\}\,;\_)$-fragment constructed according to our proof-tree construction procedure. To prove that our procedure always terminates, we show that any branch of $\mathcal{T}$ can be constructed in a finite number of steps. We mainly focus on non-axiomatic not further expandable branches, since in the case of axiomatic branches the proof is straightforward. To begin with, we characterize a non-axiomatic not further expandable branch θ of $\mathcal{T}$ as a non-axiomatic branch such that all the rules applicable to the formulas occurring on its nodes have been applied following the steps of the given decision procedure.

Next we state some preliminary lemmas and remarks useful to show that θ contains a finite number of formulae. Lemma 3 is a technical lemma used to prove Lemmas 4 and 5 which, in their turn, are used in Lemma 6 to show that (;)-formulae, the only formulae that can be decomposed more than once, are decomposed a finite number of times. Lemma 4 is proved by showing that the set $V(-B, w, N)$ is finite, where N is the leaf node of θ. The proof of Lemma 5 uses the fact that $\mathsf{cp}(P)$ is finite (Lemma 2), the fact that for every $x' \in W_\theta$, $\bigcup\theta$ contains a finite number of formulae of type $x'Rx''$ (Lemma 3), and the blocking mechanism introduced in Sect. 4.2. The interested reader may find the proofs of Lemmas 3, 4, and 5 in [6].

Lemma 3. *Let θ be a non-axiomatic not further expandable branch of a proof tree $\mathcal{T}$ for a formula xPy. Then, for every $x' \in W_\theta$, $\bigcup\theta$ contains a finite number of formulae of type $x'Rx''$.*

Remark 3. Variables generated by $(-;)$-formulae with left variable y are finitely many because $(-;)$-formulae of type $y(-(B;S))y$ are finitely many too. Moreover

these variables are distinct from all the variables generated by the other $(-;)$-formulae because each application of the $(-;)$-rule introduces a new variable.

Remark 4. If a variable w is generated by a $(-;)$-formula $x'(-(B\,;S))y$ with $x' \neq y$, then no literal of the form $y(-\mathsf{r})w$ is in θ. In fact, by Lemma 3 we know that literals of type $y(-\mathsf{r})z$, with $z \neq y$, are introduced in θ only after the decomposition of a $(-;)$-formula with left variable y. But then z cannot be the same variable introduced by a $(-;)$-formula $x'(-(B\,;S))y$ with $x' \neq y$.

Remark 5. Every $(;)$-formula $w(B;S)y$ is decomposed only with the variables introduced by the decomposition of $(-;)$-formulae with left variable w and possibly with the variable y.

Lemma 4. *Every formula $w(B;S)y$ in θ is decomposed a finite number of times.*

Lemma 5. *W_θ is finite.*

Next, we define recursively the *weight* of a term by putting:

- $\mathit{weight}(\mathsf{r}) = \mathit{weight}(-\mathsf{r}) = \mathit{weight}(\mathbf{1}) = \mathit{weight}(-\mathbf{1}) = 0$;
- $\mathit{weight}(A \sharp P) = \mathit{weight}(A) + \mathit{weight}(P) + 1$, for $\sharp \in \{\cup, \cap, ;\}$;
- $\mathit{weight}(-(A \sharp P)) = \mathit{weight}(-A) + \mathit{weight}(-P) + 1$, for $\sharp \in \{\cup, \cap, ;\}$;
- $\mathit{weight}(--P) = \mathit{weight}(P) + 1$.

Then the weight of a formula xPy is defined as the weight of its term P and the weight of a node N is defined as the sum of the weights of the formulae in N. In particular, the weight of every $(;)$-formula and the weight of every $(-;)$-formula that cannot be decomposed in N, according to the decomposition rules and, in particular, to the conditions on rules application stated in step 2, is set to 0. It can be checked that the weight of a node N is 0 if and only if it contains only literals and formulae of types $(;)$ and $(-;)$ that cannot be further decomposed, according to the definition of the decomposition rules and of the requirements on rules application in step 2 of our proof-tree construction procedure. Thus, a branch with leaf node of weight 0 is not further expandable.

Lemma 6. *After a finite number of decomposition steps, a branch θ of a deduction tree for xPy is prolonged to a branch which can be either axiomatic or non-axiomatic and whose leaf node has weight equal to 0.*

Proof. Let $\theta = \theta_1, \theta_2, \ldots$ be such that θ_{i+1} is obtained from θ_i by an application of a decomposition rule to the leaf node N_i of θ_i, for $i = 1, \ldots$. If θ_i happens to be an axiomatic branch, then the thesis immediately follows. Otherwise, we reason as shown next. For every $(;)$-formula φ of N_i, of both types $x'(B\,;S)y$ and $x(\mathbf{1}\,;S)y$, let $\mathsf{dec}(\varphi, N_i)$ be the number of times φ has been decomposed on the branch to which N_i belongs. If $\varphi = x(\mathbf{1}\,;S)y$, then $\mathsf{dec}(\varphi, N_i) \leq |W_\theta|$. By Lemma 5, $|W_\theta|$ is finite and once $\mathsf{dec}(x(\mathbf{1}\,;S)y, N_i)$ reaches it, $\mathit{weight}(x(\mathbf{1}\,;S)y)$ is set to 0. If $\varphi = x'(B\,;S)y$, then $\mathsf{dec}(\varphi, N_i)$ is bounded as stated in Lemma 4. It turns out that, at each decomposition step, we have either

1. $weight(N_i) > weight(N_{i+1})$, or
2. $\Sigma_{\varphi \in N_i} \mathsf{dec}(\varphi, N_i) < \Sigma_{\varphi \in N_{i+1}} \mathsf{dec}(\varphi, N_{i+1})$.

The first condition holds when the decomposition rule applied to θ_i to produce θ_{i+1} is different from the (**;**)-rule. In fact the decomposed formula is not introduced in the new node and the components have smaller weights. Moreover, each (−**;**)-formula that is blocked gets weight 0. The second condition, on the other hand, holds when the (**;**)-rule is used. In this case, since the decomposed (**;**)-formula φ is introduced in the new node, the weight of the new node does not decrease (it could increase), but $\mathsf{dec}(\varphi, N_i)$ increases and since it is bounded, after a finite number of steps φ is not decomposed anymore getting weight 0.

Since each node contains a finite number of formulae, after a finite number of steps we obtain a branch θ_n whose leaf node has weight 0. This means that θ_n is not further expandable. Moreover, if θ_n is not closed, then it is a non-axiomatic not further expandable branch. In fact, all the Boolean formulae in θ_n have been decomposed, and, in view of the conditions of step 2 all the (−**;**)-formulae either have been decomposed into formulae of smaller weight or have not been decomposed and their weight has been set to 0. Finally, all the (**;**)-formulae in θ_n have been decomposed, each finitely many times according to condition (a) of step 2. □

Considering that our proof-tree construction procedure constructs any axiomatic branch and any non-axiomatic not further expandable branch of a proof tree for xPy in a finite number of decomposition steps and that each decomposition rule is finitely branching, we can state the following theorem.

Theorem 1 (Termination). *The dual tableau procedure for the* $(\{\mathbf{1}, \cup, \cap\}\,\mathbf{;}\,\_)$*-fragment always terminates.*

Soundness and completeness. Correctness of our proof-tree construction procedure is proved by showing that when the input formula xPy is valid, the procedure yields a closed (axiomatic) dual tableau for xPy, whereas if xPy is not valid, the procedure yields a non-axiomatic not further expandable branch θ of a dual tableau for xPy and a model $\mathcal{M}_\theta$ that falsifies every formula on θ (thus, in particular, xPy itself).

Lemma 7, stated below, is used in the proof of Theorem 2 to establish the first half of the correctness proof, and also later, in the proof of Theorem 3.

Lemma 7. *Let* $\mathcal{T}$ *be a deduction tree for a formula* xPy *of the* $(\{\mathbf{1}, \cup, \cap\}\,\mathbf{;}\,\_)$*-fragment, constructed as described in our proof-tree construction procedure. If the procedure terminates at step 4 yielding a non-axiomatic not further expandable branch* θ*, a model* $\mathcal{M}_\theta = (U_\theta, m_\theta)$*, and a valuation* v_θ*, then* $\mathcal{M}_\theta$ *and* v_θ *falsify* θ*.*

Theorem 2. *If* xPy *is a valid formula of the* $(\{\mathbf{1}, \cup, \cap\}\,\mathbf{;}\,\_)$*-fragment of* $\mathsf{RL}(\mathbf{1})$*, then our proof-tree construction procedure yields a closed proof tree for* xPy*.*

Lemma 8, presented next, states that each decomposition step performed by our proof-tree construction procedure preserves falsifiability. This result is needed later in the proof of Theorem 3, to establish the second half of the correctness proof. Proofs of Lemmas 7 and 8 and of Theorems 2 and 3 can be found in [6].

Lemma 8. *Let θ be a branch of a deduction tree for a formula xPy of the $(\{\mathbf{1}, \cup, \cap\}\ \mathbf{;}\ \_)$-fragment that is being constructed by our proof-tree construction procedure, and let θ' be obtained from θ by a decomposition step performed by the decision procedure. If θ is a falsifiable branch, then θ' is falsifiable too.*

Theorem 3. *Let xPy be a non valid relational formula of the $(\{\mathbf{1}, \cup, \cap\}\ \mathbf{;}\ \_)$-fragment. Then our proof-tree construction procedure yields a non-axiomatic not further expandable branch θ of a dual tableau for xPy and a model $\mathcal{M}_\theta$ that falsifies every formula on θ and, therefore, xPy itself.*

Summing up, Theorems 2 and 3 yield the following result.

Theorem 4. *The $(\{\mathbf{1}, \cup, \cap\}\ \mathbf{;}\ \_)$-fragment has a decidable validity problem.*

5 Conclusions and Future Work

Relational entailment allows one to deal with properties of relational constants and of relational variables in dual tableau proofs without adding any specific rule to the basic set of decomposition rules. Using entailment in dual tableau-based decision procedures, however, can be tricky because the constant $\mathbf{1}$ occurs both on the left-hand side and on the right-hand side of composition.

We have presented a dual tableau-based decision procedure for a fragment of the logic $\mathsf{RL}(\mathbf{1})$ which can express simple forms of inclusion between relations. Specifically, we admit inside entailment only positive occurrences of Boolean terms and thus we can express inclusion properties of the form '$(\mathsf{r}_1 \cup \mathsf{s}) \subseteq -\mathsf{r}_2$'.

We plan to extend the expressibility of our relational fragment in order to make entailment widely applicable in dual tableau-based decision procedures. As a first step, we intend to include negative occurrences of Boolean terms inside entailment. In this way we will be able to formulate terms of type $\mathbf{1}\,\mathbf{;}\,(-(-(\mathsf{r}_1 \cup \mathsf{s}) \cup \mathsf{r}_2)\,\mathbf{;}\,\mathbf{1})$ expressing the (positive) inclusion property '$(\mathsf{r}_1 \cup \mathsf{s}) \subseteq \mathsf{r}_2$'.

Our further aim is to add, inside entailment, some restricted forms of composition so as to be able to express terms of type $\mathbf{1}\,\mathbf{;}\,(-(-(\mathsf{s}\,\mathbf{;}\,\mathsf{s}) \cup \mathsf{s})\,\mathbf{;}\,\mathbf{1})$ and of type $\mathbf{1}\,\mathbf{;}\,(-(-(\mathsf{r}\,\mathbf{;}\,\mathsf{r}\,\mathbf{;}\,\mathsf{r}) \cup \mathsf{r})\,\mathbf{;}\,\mathbf{1})$, stating, respectively, that the relational variables s and r are transitive (i.e., '$(\mathsf{s}\,\mathbf{;}\,\mathsf{s}) \subseteq \mathsf{s}$') and three-transitive (i.e., '$(\mathsf{r}\,\mathbf{;}\,\mathsf{r}\,\mathbf{;}\,\mathsf{r}) \subseteq \mathsf{r}$'), respectively. Expressing these properties is important if one wants to use our dual tableau decision procedure with various non-classical logics such as, for instance, modal logics to reason with incomplete information [7].

We also intend to introduce the converse relation '$^\smile$' and the identity relation '$\mathbf{1}'$' inside entailment for the purpose of dealing with properties such as symmetry and reflexivity.

Acknowledgments. Thanks are due to three anonymous referees for their helpful suggestions. This work was supported by Indam-GNCS, Progetto di ricerca "Automi Reattivi e loro Simulazione nell'Ambito del Non-Standard Secure Text Processing".

References

1. W. E. Beth. Semantic entailment and formal derivability. *Mededelingen van de Koninklijke Nederlandse Akademie van Wetenschappen, Afdeling Letterkunde*, N.R. Vol 18, no 13, 1955, pp 30942. Reprinted in Jaakko Hintikka (ed.) *The Philosophy of Mathematics*, Oxford University Press, 1969.
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. Cantone, J. Golińska-Pilarek, M. Nicolosi-Asmundo. A Relational Dual Tableau Decision Procedure for Multimodal and Description Logics. To appear in: *Proceedings of the 9th International Conference on Hybrid Artificial Intelligence Systems*, Salamanca, Spain, 11th - 13th June 2014.
4. D. Cantone, M. Nicolosi Asmundo, E. Orłowska. Dual tableau-based decision procedures for some relational logics. In: *Proceedings of the 25th Italian Conference on Computational Logic*, Rende, Italy, July 7-9, 2010, pp. 1–16. CEUR Workshop Proceedings vol. 598.
5. D. Cantone, M. Nicolosi Asmundo, E. Orłowska. Dual tableau-based decision procedures for relational logics with restricted composition operator. *Journal of Applied Non-classical Logics* 21, No 2, 2011, 177-200.
6. D. Cantone, M. Nicolosi-Asmundo, E. Orłowska. *A Dual Tableau-based Decision Procedure for a Relational Logic with the Universal Relation (extended version)*. Available at `http://www.dmi.unict.it/~nicolosi/CNOCILC14ext.pdf`, 2014.
7. S. Demri, E. Orlowska, D. Vakarelov. Indiscernibility and complementarity relations in information systems. In: *J. Gerbrandy, M. Marx, M. de Rijke and Y. Venema (eds) JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*, Amsterdam University Press, 1999.
8. N. Dershowitz, J.-P. Jouannaud. Rewrite Systems. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B). Elsevier.* pp. 243-320, 1990.
9. M.C. Fitting. First-Order Logic and Automated Theorem Proving. Second edition. *Graduate Texts in Computer Science. Springer-Verlag. New York,* 1996.
10. A. Formisano and M. Nicolosi Asmundo. An efficient relational deductive system for propositional non-classical logics. *Journal of Applied Non-Classical Logics*, vol. 16(3-4), pp. 367-408 (2006).
11. J. Golińska-Pilarek, T. Huuskonen, E. Munoz-Velasco, Relational dual tableau decision procedures and their applications to modal and intuitionistic logics. *Annals of Pure and Applied Logics* vol. 165 (2), pp. 409-427 (2014).
12. J. Golińska-Pilarek, E. Munoz-Velasco, and A. Mora. Implementing a relational theorem prover for modal logic K. *International Journal of Computer Mathematics*, 88(9):1869–1884, 2011.
13. J. Golińska-Pilarek, E. Munoz-Velasco, and A. Mora. A new deduction system for deciding validity in modal logic K. *Logic Journal of IGPL* 19(2):425–434, 2011.

14. J. Golińska-Pilarek, E. Orłowska. Tableaux and dual tableaux: Transformation of proofs. *Studia Logica*, 85(3):283-302, 2007.
15. 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.
16. E. Orłowska, J. Golińska-Pilarek. Dual Tableaux: Foundations, Methodology, Case Studies. *Trends in Logic* vol. 36, Springer, 2011.
17. H. Rasiowa, R. Sikorski. On Gentzen theorem. *Fundamenta mathematicae* 48, 57-69, 1960.
18. H. Rasiowa, R. Sikorski. Mathematics of Metamathematics, *Polish Scientific Publishers PWN*, Warsaw 1963.
19. A. Tarski, S. Givant. A Formalization of Set Theory without Variables. *American Mathematical Society Colloquium Publications*, Providence, Rhode Island, 1987.