

A Dual Tableau-based Decision Procedure for a Relational Logic with the Universal Relation (Extended Version)

Domenico Cantone¹, Marianna Nicolosi Asmundo¹, and
Ewa Orłowska²

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: orłowska@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 $\text{RL}(\mathbf{1})$, called $(\{\mathbf{1}, \cup, \cap\}; _)$, which admits a restricted form of composition, $(R; S)$ or $(R; \mathbf{1})$, where the left subterm R of $(R; S)$ is only allowed to be either the constant $\mathbf{1}$, or a Boolean term neither containing the complement operator nor the constant $\mathbf{1}$, while in the case of $(R; \mathbf{1})$, R can only be a Boolean term involving relational variables and the operators of intersection and of union. 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 rules to decompose compositional formulae are modified so 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 \mathbf{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 [11]. A uniform relational framework based on the logic of binary relations $\text{RL}(\mathbf{1})$, presented in [10] 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 $\text{RL}(\mathbf{1})$ is based on the fact that once the Kripke-style semantics of the considered logic 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 $\text{RL}(\mathbf{1})$ is undecidable, it contains several decidable fragments. In many cases, however, dual tableau proof systems are not decision procedures for decidable fragments of $\text{RL}(\mathbf{1})$. This is mainly due to the way decomposition and specific rules are defined and 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 [5], for example, an optimized relational dual tableau for $\text{RL}(\mathbf{1})$, 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 [11] for fragments of $\text{RL}(\mathbf{1})$ corresponding to the class of first-order formulae in prenex normal form with universal quantifiers only, in [7,8] for the relational logic corresponding to the modal logic K , in [2,3] for fragments of RL characterized by some restrictions in terms of type $(R; S)$, in [6] for a class of relational logics admitting a single relational constant with the properties of reflexivity, transitivity, and heredity, and in [1] for a class of relational fragments extending the ones introduced in [6] by allowing a countable infinity of relational constants with the properties of reflexivity, transitivity, and heredity.

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

In some cases, like in [6] and in [1], 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 because termination of the proof procedure must be guaranteed. 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 each other.

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

Introduction of entailment inside relational proof systems permits to eliminate specific rules and, consequently, to keep the set of decomposition rules small. 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. We introduce a fragment of $\text{RL}(\mathbf{1})$, 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 neither containing the complement operator nor the constant $\mathbf{1}$.

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.

This fragment properly includes the logics presented in [2] and, therefore, it can express the multi-modal logic \mathbf{K} with union and intersection of accessibility relations, and the description logic \mathcal{ALC} with union and intersection of roles. Furthermore it can express via entailment properties of the form ' $r \subseteq \neg(s_1 \cup s_2)$ ' and ' $(s_1 \cup s_2) \subseteq \neg r$ ', where r , s_1 , and s_2 are relational variables.

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

2 The Relational Logic $\text{RL}(\mathbf{1})$ and its Dual Tableau

In this section we review the logic $\text{RL}(\mathbf{1})$ and its dual tableau in full extent (see also [3] and [11]).

Let \mathbb{RV} be a countably infinite set of *relational variables* p, q, r, s, \dots and let $\mathbf{1}$ be a *relational constant*. Then, the set \mathbb{RT} of *relational terms* of $\text{RL}(\mathbf{1})$ is the smallest set of terms (with respect to inclusion) containing all relational variables and the relational constant $\mathbf{1}$, and which is closed with respect to the *relational operators* ' \cap ', ' \cup ', ' $;$ ' (binary) and ' \neg ', ' \sim ' (unary).

Let \mathbb{OV} be a countably infinite set of *object (individual) variables* x, y, z, w, \dots . Then, $\text{RL}(\mathbf{1})$ -formulae have the form xRy , where $x, y \in \mathbb{OV}$ and $R \in \mathbb{RT}$. $\text{RL}(\mathbf{1})$ -formulae of type $x\mathbf{1}y$ and xry , with $r \in \mathbb{RV}$, are called *atomic* $\text{RL}(\mathbf{1})$ -formulae. A *literal* is either an atomic formula or its complementation (namely a formula

of type $x(-\mathbf{1})y$ or $x(-r)y$. For a relational operator ‘ $\#$ ’ other than ‘ \neg ’, by a ($\#$)-*term* we mean a relational term whose main operator is ‘ $\#$ ’, and by a ($-\#$)-*term* we indicate a relational term having ‘ \neg ’ followed by ‘ $\#$ ’ as its main operator. A ($\#$)-*formula* (resp., ($-\#$)-*formula*) is a formula whose relational term is a ($\#$)-*term* (resp., ($-\#$)-*term*). A *Boolean term* is a relational term involving only the Boolean operators ‘ \neg ’, ‘ \cup ’, and ‘ \cap ’.

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

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

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

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

An RL($\mathbf{1}$)-formula xRy is *true* in a model $\mathcal{M} = (U, m)$ if $\mathcal{M}, v \models xRy$, for every valuation v in \mathcal{M} . An RL($\mathbf{1}$)-formula xRy is said to be *valid* if it is *true* in all RL($\mathbf{1}$)-models. An RL($\mathbf{1}$)-formula xRy is *falsified* by a model $\mathcal{M} = (U, m)$ and by a valuation v in \mathcal{M} if $\mathcal{M}, v \not\models xRy$. It is *falsifiable* if there are a model \mathcal{M} and a valuation v in \mathcal{M} such that $\mathcal{M}, v \not\models xRy$. An RL($\mathbf{1}$)-set is a finite set of RL($\mathbf{1}$)-formulae $\{\varphi_1, \dots, \varphi_n\}$ such that for every RL($\mathbf{1}$)-model \mathcal{M} and for every valuation v in \mathcal{M} there exists an $i \in \{1, \dots, n\}$ such that φ_i is satisfied by v in \mathcal{M} . Plainly, the first-order disjunction of the formulae in an 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). Such an analytic approach is similar to Beth’s tableau method, with the difference that the two systems work in a dual manner. Duality between tableaux and dual tableaux has been analyzed in depth in [9].

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 the closure conditions. The decomposition rules for RL($\mathbf{1}$) are listed in Table 1. In these rules, ‘ $,$ ’ is interpreted as disjunction and ‘ \wedge ’ as conjunction. A rule is RL($\mathbf{1}$)-correct whenever the premise is an RL($\mathbf{1}$)-set if and only if each

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

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

of its consequents is an RL(**1**)-set. The rules presented in Table 1 are proved RL(**1**)-correct in [11].

An RL(**1**)-axiomatic set is any set of RL(**1**)-formulae containing a subset of one of the following forms:

- (**Ax 1**) $\{xRy, x(-R)y\}$
(**Ax 2**) $\{x1y\}$.

Clearly, an RL(**1**)-axiomatic set is also an RL(**1**)-set.

Let xPy be an RL(**1**)-formula. An RL(**1**)-*proof tree* for 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 the formula xPy ;
- each node, except the root, is obtained from its predecessor node by an application of 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 object variables occurring in the formulae contained in the nodes of θ . A node of an RL(**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 RL(**1**)-formula is RL(**1**)-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 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 $\text{RL}(\mathbf{1})$ -dual tableau are proved in [11]. However, the logic $\text{RL}(\mathbf{1})$ is undecidable. This follows from the undecidability of the equational theory of representable relation algebras discussed in [12].

3 Useful Notions and Properties

In this section we introduce constructions and notions needed for the presentation of the results of the paper.

Let P be any relational term in $\text{RL}(\mathbf{1})$. The following identities hold:

$$\boxed{\begin{array}{ll} (\mathbf{1} \cup P) \equiv (P \cup \mathbf{1}) \equiv \mathbf{1} & ((-\mathbf{1}) \cup P) \equiv (P \cup (-\mathbf{1})) \equiv P \\ (\mathbf{1} \cap P) \equiv (P \cap \mathbf{1}) \equiv P & ((-\mathbf{1}) \cap P) \equiv (P \cap (-\mathbf{1})) \equiv (-\mathbf{1}) \\ (-(-\mathbf{1})) \equiv \mathbf{1} & \end{array}}$$

Let H be a relational term in $\text{RL}(\mathbf{1})$ 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.

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 it is equal to $-\mathbf{1}$, or it does not contain $\mathbf{1}$.*

Proof. Let P be a Boolean subterm of H' . The proof is by induction over the structure of P . We distinguish the following cases:

- If $P \in (\{\mathbf{1}, -\mathbf{1}\} \cup \mathbb{R}\mathbb{V})$, then the thesis is trivially satisfied.
- If $P = (Q \cup S)$ or $P = (Q \cap S)$, then, by the construction of H' , both Q and S must be distinct from $\mathbf{1}$ and $-\mathbf{1}$. Moreover, by the inductive hypothesis, they cannot contain $\mathbf{1}$ and, consequently, P cannot contain $\mathbf{1}$ too.
- If $P = (-Q)$, where $Q \neq \mathbf{1}$, then $Q \neq -\mathbf{1}$ by the construction of H' and the thesis follows by the inductive hypothesis. \square

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

Parsing trees. As with formulae of standard first-order logic, it is possible to associate to each relational term P of $\text{RL}(\mathbf{1})$ a *parsing tree* S_P , which is an ordered tree constructed in the usual way. 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 define a complement normal form for Boolean relational terms, the notions of \mathbf{Bool}_N -formula, of \mathbf{Bool} -construction from N , where N is a set of formulae, and of set of components of a relational term.

To begin with, we define recursively the function \mathbf{nf}_- (*complement normal form*) on the set of Boolean relational terms as follows:

- $\mathbf{nf}_-(\mathbf{1}) = \mathbf{1}$, and $\mathbf{nf}_-(s) = s$, for any relational variable s ;
- $\mathbf{nf}_-(S \# H) = \mathbf{nf}_-(S) \# \mathbf{nf}_-(H)$, for $\# \in \{\cap, \cup\}$;
- $\mathbf{nf}_-(\neg(S \cap H)) = \mathbf{nf}_-(\neg S) \cup \mathbf{nf}_-(\neg H)$;
- $\mathbf{nf}_-(\neg(S \cup H)) = \mathbf{nf}_-(\neg S) \cap \mathbf{nf}_-(\neg H)$;
- $\mathbf{nf}_-(\neg\neg S) = \mathbf{nf}_-(S)$.

Observe that the complement normal form of a term is obtained by successive applications of the De Morgan laws and of the law of double negation.

A term is in complement normal form whenever the occurrences of the complement operator in it act only on relational variables or constants.

Plainly, for every Boolean relational term R , the formulae xRy and $x \mathbf{nf}_-(R) y$ are *logically equivalent*, that is $\mathcal{M}, v \models xRy$ if and only if $\mathcal{M}, v \models x \mathbf{nf}_-(R) y$, 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 \mathbf{Bool}_N -formulae as follows:

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

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

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

Given a term R in \mathbb{RT} , an object variable x , and a set of formulae N , 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 $\text{cp}(P)$ of *the components of the term P* as follows:

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

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

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

Formulae of the fragment $(\{\mathbf{1}, \cup, \cap\}; \neg)$ of $\text{RL}(\mathbf{1})$ are characterized by the fact that the left subterm R of any term of type $(R; S)$ in them is 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 $\text{RL}(\mathbf{1})$ but the converse operator ‘ \smile ’.

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

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

then $(\neg P), (P \cup Q), (P \cap Q), (B; S), (\mathbf{1}; S) \in \mathbb{RT}_{(\{\mathbf{1}, \cup, \cap\}; \neg)}$.

Examples of formulae of the $(\{\mathbf{1}, \cup, \cap\}; \neg)$ -fragment are: $x(\neg((r_1 \cup s); (p; \mathbf{1})))y$, $x(\mathbf{1}; ((r_1 \cup s); \neg(((q \cup p) \cap r_1); \mathbf{1})))y$, and $x(\mathbf{1}; (((r_1 \cup s) \cap r_2); \mathbf{1}))y$. The latter formula can be rewritten as $x(\mathbf{1}; (\neg(\neg(r_1 \cup s) \cup \neg r_2); \mathbf{1}))y$, where $(\neg(r_1 \cup s) \cup \neg r_2)$ is a relational term formalizing the property ‘ $(r_1) \cup s \subseteq \neg r_2$ ’.

The decomposition rules for Boolean formulae of our dual tableau-based decision procedure are just the ones in Table 1. Concerning the rule to decompose $(;)$ -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 $V(\neg B, x, \bigcup \theta)$. 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 $(;)_b$ depicted 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, $x(\mathbf{1}; S)y$ is not decomposed with z . If $S = \mathbf{1}$, the consideration made with rule $(;)_a$ for the node resulting from the decomposition step holds here as well.

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

$(;)_a$	$\frac{x(B; S)y}{zSy, x(B; S)y}$	$(;)_b$	$\frac{x(\mathbf{1}; S)y}{zSy, x(\mathbf{1}; S)y}$
	$(z \text{ object variable in } V(-B, x, \cup \theta))$		$(z \text{ is any object variable})$
$(-;)_a$	$\frac{x-(B; \mathbf{1})y}{x(-B)z}$	$(-;)_b$	$\frac{x-(\mathbf{1}; S)y}{z(-S)y}$
	$(z \text{ is a new object variable})$		$(z \text{ is a new object variable})$

For what concerns $(-;)$ -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 for the current node. The rule is 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 $(;)_b$, $(-;)_a$, and $(-;)_b$ of Table 2 are in order. Observe that for every RL($\mathbf{1}$)-model $\mathcal{M} = (U, m)$, $m(\mathbf{1}) = U \times U$ and thus, for every valuation v and object variables x and z , we have $\mathcal{M}, v \models x\mathbf{1}z$ and $\mathcal{M}, v \not\models x(-\mathbf{1})z$. 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. However, we shall prove that such a restriction preserves the completeness of the procedure.

It is convenient to introduce the notion of *deduction tree* for 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 it 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*

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

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, \dots, 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 $z_1(-(B_1;S_1))y, \dots, z_n$ is generated by a $(-;)$ -formula $z_{n-1}(-(B_{n-1};S_{n-1}))y$, where $w(-(B_0;S_0))y, z_1(-(B_1;S_1))y, \dots, z_{n-1}(-(B_{n-1};S_{n-1}))y$ 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 introduce a total order among object variables in W_θ , denoted $<_\theta$, such that:

- $x <_\theta w$, for every $w \in W_\theta \setminus \{x\}$,
- $x_1 <_\theta x_2$, for every $x_1, x_2 \in W_\theta \setminus \{x, y\}$ such that x_1 has been 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 .

Remark 1. Notice that the relationship ancestor/descendant is based on the literals of type $x'(-r)z$ that are generated by applying either the $(-;)$ -rule of Table 1 or the $(-;)_a$ -rule of Table 2, and some consequent Boolean decompositions. 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.

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 object 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 that result from the Boolean decomposition of $z(-B)w$, the $(;)$ -formula $z'(B_1;Q)y$ occurs in θ as well.

4.1 The Decision Procedure

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 till saturation. We require that:
 - 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 $z'(-r)w \in \bigcup \theta$ (obtained from the application of the Boolean rules to $z'(-B)w$) we store the literal $z(-r)w$ 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 the model \mathcal{M}_θ (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 m_θ by putting $(x', y') \notin m_\theta(R)$ if and only if $x'Ry' \in (Lit_\theta \cup Lit_{(-;)})$. Let $v_\theta : \mathbb{O}\mathbb{V} \rightarrow U_\theta$ be a valuation such that $v_\theta(x) =_{Def} x$, for every $x \in U_\theta$. We terminate returning θ , \mathcal{M}_θ , and v_θ .

The next lemma states two useful properties of the formulae occurring on the deduction trees constructed by the proof procedure above.

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) $R \in \text{cp}(P)$,
- (ii) if R contains the composition operator, then $x'' = y$.

Proof. Let $\mathcal{T}_0, \dots, \mathcal{T}_n$ be a sequence of deduction trees constructed by an execution of the proof procedure illustrated above, where $\mathcal{T}_0 = \{xPy\}$ and each \mathcal{T}_{i+1} is obtained from \mathcal{T}_i , for $i = 0, 1, \dots, n-1$ by the application of a decomposition rule and $\mathcal{T}_n = \mathcal{T}$.

We prove by induction on i that, for every $x'Rx''$ occurring on \mathcal{T}_i , (i) and (ii) hold. If $i = 0$, $\mathcal{T}_0 = \{xPy\}$ and therefore (i) and (ii) are trivially verified. By

the inductive step, we assume that (i) and (ii) are true for every formula of \mathcal{T}_i . Since \mathcal{T}_{i+1} is obtained from \mathcal{T}_i by the application of a decomposition rule to the leaf node of one of its branches $\bar{\theta}$, all the formulae on \mathcal{T}_{i+1} , with the exception of the new formulae originated by the decomposition step, occur also in \mathcal{T}_i and thus satisfy (i) and (ii). Then we have to prove that (i) and (ii) hold also for the formulae originating from the decomposition step.

Let us consider $\bar{\theta}$, its leaf node \bar{N} , and the formula $x'Rx''$ chosen to be decomposed. We analyze the possible cases. If $x'Rx'' \equiv x'(Q_1 \cap Q_2)x''$, then we append $\bar{N}' = (\bar{N} \setminus \{x'Rx''\}) \cup \{x'Q_1x''\}$ as the left successor and $\bar{N}'' = (\bar{N} \setminus \{x'Rx''\}) \cup \{x'Q_2x''\}$ as the right successor of \bar{N} . Since $Q_1, Q_2 \in \text{cp}(R)$ and $R \in \text{cp}(P)$, we have that $Q_1, Q_2 \in \text{cp}(P)$ and thus (i) is satisfied by $x'Q_1x''$ and $x'Q_2x''$. If R does not contain the composition operator, (ii) is trivially verified. Otherwise, $x'' = y$, and thus (ii) holds for $x'Q_1x''$ and for $x'Q_2x''$ too. The remaining Boolean cases can be proved in an analogous way.

If $x'Rx'' = x'(B; S)x''$ and $z \in V(-B, x', \bigcup \theta)$, we append $\bar{N}' = \bar{N} \cup \{zSx''\}$ as the successor of \bar{N} . Since $S \in \text{cp}(R)$ and $R \in \text{cp}(P)$, it follows that $S \in \text{cp}(P)$ and thus (i) holds also for zSx'' . Since $x'' = y$, (ii) holds for zSx'' as well. The case in which $x'Rx'' = x'(\mathbf{1}; S)x''$ can be treated in an analogous way.

If $x'Rx'' = x'(-B; S)x''$, with $S \neq \mathbf{1}$, we append $\bar{N}' = (\bar{N} \setminus \{x'(-B; S)x''\}) \cup \{x'(-B)z, z(-S)x''\}$ as the successor of \bar{N} , where z is new for \bar{N} . Since $-B, -S \in \text{cp}(R)$ and $R \in \text{cp}(P)$, $-B, -S \in \text{cp}(P)$ and thus (i) holds for both $x'(-B)z$ and $z(-S)x''$. Since $x'' = y$ and $(-B)$ does not contain the complement operator, (ii) is trivially satisfied for both $x'(-B)z$ and $z(-S)x''$. The cases $x'(-\mathbf{1}; S)x''$ and $S = \mathbf{1}$ can be handled in a similar way. \square

4.2 Termination of the procedure

Let \mathcal{T} be a deduction tree for a formula xPy of the $(\{\mathbf{1}, \cup, \cap\}; _)$ -fragment constructed according to the procedure described in Sect. 4.1. To prove that the procedure always terminates it is useful to give the following characterization of a non-axiomatic not further expandable branch θ of \mathcal{T} :

- $x'\mathbf{1}y' \notin N$, for every node $N \in \theta$;
- if $x'Ry'$ (resp., $x'(-R)y'$) occurs in a node $N \in \theta$, then $x'(-R)y'$ (resp., $x'Ry'$) does not occur in any node $N' \in \theta$;
- if $x'(-Q)y'$ occurs in a node $N \in \theta$, then there is a node $N' \in \theta$, successor of N , such that $x'Qy' \in N'$;
- if $x'(Q_1 \cap Q_2)y'$ occurs in a node $N \in \theta$, then there is a node $N' \in \theta$, successor of N , such that either $x'Q_1y' \in N'$ or $x'Q_2y' \in N'$;
- if $x'(Q_1 \cup Q_2)y'$ occurs in a node $N \in \theta$, then there is a node $N' \in \theta$, successor of N , such that $x'Q_1y' \in N'$ and $x'Q_2y' \in N'$;
- if $x'(-(Q_1 \cap Q_2))y'$ occurs in a node $N \in \theta$, then there is a node $N' \in \theta$, successor of N , such that $x'(-Q_1)y', x'(-Q_2)y' \in N'$;
- if $x'(-(Q_1 \cup Q_2))y'$ occurs in a node $N \in \theta$, then there is a node $N' \in \theta$, successor of N , such that either $x'(-Q_1)y'$ or $x'(-Q_2)y' \in N'$;

- if $x'(B ; S)y$, with $S \neq \mathbf{1}$, occurs in a node $N \in \theta$, then, for every $z \in V(-B, x', S)$, there is a node $N' \in \theta$, successor of N , such that $zSy \in N'$;
- if $x(\mathbf{1} ; S)y$, with $S \neq \mathbf{1}$, occurs in a node $N \in \theta$, then, for every $w \in W_\theta$, there is a node $N' \in \theta$, successor of N , such that $wSy \in N'$;
- if $x'(-B ; S)y \in N$ and is not blocked by any $z'(-B ; S)y \in N'$, with N' predecessor of N , then there is a node N'' , successor of N , and an object variable u such that $x'(-B)u, u(-S)y \in N''$, if $S \neq \mathbf{1}$, and $x'(-B)u \in N''$ otherwise;
- if $x(-\mathbf{1} ; S)y$ occurs in a node $N \in \theta$ and $S \neq \mathbf{1}$, then there is a node $N' \in \theta$ such that $zSy \in N'$, for some object variable z .

Next we state and prove some preliminary lemmas and make some remarks.

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

Proof. The lemma is immediate for formulae $x'Rx''$ in θ , with $x'' = y$, because x' is fixed and $\text{cp}(P)$ is a finite set. On the other hand, if $x'Rx''$ is in θ and $x'' \neq y$, then, by Lemma 2 (ii), R is a Boolean term neither containing the complement operator nor the constant $\mathbf{1}$, obviously with $R \in \text{cp}(P)$ by Lemma 2 (i). Thus, by systematic Boolean decomposition, $x'Rx''$ generates inside θ , a finite number of formulae with left variable x' and right variable x'' . Since x' is fixed and $\text{cp}(P)$ is a finite set, in order to prove that, for any variable $x'' \in W_\theta \setminus \{x, y\}$, the number of formulae of type $x'Rx''$ in θ is finite, we have to show that the number of such x'' 's is finite. But this follows from the fact that each right variable of a formula of type $x'Rx''$, with $x'' \neq y$, is generated by the decomposition of a $(-;)$ -formula $x'(-B ; S)y$ and, possibly, by a finite number of Boolean decompositions. By the first part of this lemma, the number of $(-;)$ -formulae $x'(-B ; S)y$ in θ is finite, moreover by conditions (a) and (b) on rules application stated in step 2 of the procedure of Sect. 4.1, each of these formulae can be decomposed at most once. \square

Remark 2. Variables generated by $(-;)$ -formulae with left variable y are finitely many because $(-;)$ -formulae of type $y(-B ; S)y$ are finitely many as well. Moreover these variables are distinct from all the variables generated by the other $(-;)$ -formulae because the $(-;)$ -rule always introduces a new variable when it is applied.

Remark 3. If a variable w is generated by a $(-;)$ -formula $x'(-B ; S)y$ with $x' \neq y$, then no literal of the form $y(-r)w$ is in θ . In fact, by Lemma 3 we know that literals of type $y(-r)z$, with $z \neq y$, are introduced in N 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 4. 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 can be decomposed a finite number of times.*

Proof. Every $(;)$ -formula $w(B ; S)y$ in θ can be decomposed a number of times equal to the cardinality of the set $V(-B, w, \bigcup \theta)$. By Lemmas 2 and 3, there are only finitely many $(-;)$ -formulae with left variable w . Thus, all the literals with left variable w and right variable $z \neq y$ are finitely many. By Lemma 3 it turns out that literals with left variable w and right variable y are finite too, and therefore the set $V(-B, w, \bigcup \theta)$ must be finite. \square

Lemma 5. *W_θ is finite.*

Proof. During the construction of the non-axiomatic not further expandable branch θ , W_θ is enlarged by decomposing formulae of type $(-;)$. By the conditions on the application of the decomposition rules, each $(-;)$ -formula can be decomposed at most once. Therefore, in order for W_θ to be infinite, θ must contain an infinite number of formulae of type $(-;)$. Since $\text{cp}(P)$ is finite, the number of terms of type $(-;)$ in θ is finite too. Thus, by Lemma 3 (ii), the only possibility for θ to contain an infinite number of formulae of type $(-;)$ is that there is at least one $(-;)$ -formula $x'(-B ; S)y$ that occurs in θ infinitely many times, each time with a different left variable. Since, by Lemma 4, every formula $w(B_1 ; S_1)y$ can be decomposed a finite number of times in θ , the formula $x'(-B ; S)y$, that appears in θ infinitely many times, each time with a different left variable, must originate from the decomposition of a $(;)$ -formula of type $x(\mathbf{1} ; S_2)y$. By condition (b) of step 2 of the decision procedure in Sect. 4.1, $x'(-B ; S)y$ is allowed to appear infinitely often with a different left variable only if there are infinite distinct sets of $(;)$ -terms belonging to $\text{cp}(P)$. But this is not possible because $\text{cp}(P)$ is finite. Thus, the conditions of application of decomposition rules introduced in step 2 guarantee that no $(-;)$ -formula $x'(-B ; S)y$ is allowed to occur in θ infinitely many times, each time with a different left variable. Therefore it follows that W_θ is finite. \square

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

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

Then we define the weight of a formula xPy as the weight of its term P and the weight of a node 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 the procedure of Sect. 4.1. 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 with the weight of the leaf node equal to 0.*

Proof. Let $\theta = \theta_1, \theta_2, \dots$ 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 $i = 1, \dots$. If θ_i results 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 $\text{dec}(\varphi, N_i)$ 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 $\text{dec}(\varphi, N_i) \leq |W_\theta|$. By Lemma 5, $|W_\theta|$ is a finite number and once $\text{dec}(x(\mathbf{1};S)y, N_i)$ reaches it, $\text{weight}(x(\mathbf{1};S)y)$ is set to 0. If $\varphi = x'(B;S)y$, then $\text{dec}(\varphi, N_i)$ is bounded as stated in Lemma 4. It turns out that, at each decomposition step, we have either

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

The first condition holds when the decomposition rule applied to obtain θ_{i+1} from θ_i is different from the (;)-rule, whereas the second condition holds when the (;)-rule is used.

Since each node contains a finite number of formulae, $\text{weight}(N_i)$ is a non-negative function and $\text{dec}(\varphi, N_i)$ is bounded for every (;)-formula φ , after a finite number of steps we obtain a branch θ_n with a leaf node of 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, all the (-;)-formulae, in view of the conditions of step 2, either have been decomposed into formulae of smaller weight or they have been not 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. \square

Considering that the procedure of Sect. 4.1 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, we can state the following theorem.

Theorem 1 (Termination). *The dual tableau procedure for the $(\{\mathbf{1}, \cup, \cap\}; -)$ -fragment described in Sect. 4.1 always terminates.*

4.3 Correctness of the Procedure

We show next that the procedure is correct in the sense 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}_θ that falsifies every formula on θ and, in particular, xPy itself.

Lemma 7. *Let \mathcal{T} be a deduction tree for a formula xPy of the $(\{\mathbf{1}, \cup, \cap\}; -)$ -fragment of $\text{RL}(\mathbf{1})$ constructed as described in the procedure of Sect. 4.1. If the*

procedure terminates at step 4 returning 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}_θ and v_θ falsify θ .

Proof. We have to prove that \mathcal{M}_θ and v_θ falsify all the formulae in the nodes of θ . For this purpose, it is convenient to consider the set $\bigcup\theta$ of all the formulae contained in the nodes of θ .

Let φ be a formula of $\bigcup\theta$. We prove by induction over the structure of φ that \mathcal{M}_θ and v_θ falsify φ . The base case is handled as in [3, Lemma 4]. Concerning the inductive step, for simplicity, we report the proof only for $(;)$ -formulae and for $(-;)$ -formulae. Let $\varphi = x'(B; S)y$, and let us consider first the case where $S \neq \mathbf{1}$. To prove that $\mathcal{M}_\theta, v_\theta \not\models x'(B; S)y$, we have to show that, for every $z \in W_\theta = U_\theta$,

$$\text{either } \mathcal{M}_\theta, v_\theta \models x'(-B)z \text{ or } \mathcal{M}_\theta, v_\theta \models z(-S)y. \quad (1)$$

By a repeated application of the $(;)$ -rule, all the formulae zSy , with $z \in V(-B, x', \bigcup\theta)$, occur in $\bigcup\theta$. In particular, each of them belongs to a node of the branch and, by inductive hypothesis, \mathcal{M} and v do not satisfy each of them. Thus, (1) is satisfied for every $z \in V(-B, x', \bigcup\theta)$. We have to prove that it is satisfied also for every $z \in (W_\theta \setminus V(-B, x', \bigcup\theta))$. We distinguish the case where $x'(-B)z$ has no **Bool**-construction from $Lit(-;)$, and the case where $x'(-B)z$ has a **Bool**-construction from $Lit(-;)$.

Assume first that $z \in (W_\theta \setminus V(-B, x', \bigcup\theta))$ and that $x'(-B)z$ has no **Bool**-construction from $Lit(-;)$. We prove that $\mathcal{M}, v \models x'(-B)z$ by induction over the structure of $x'(-B)z$.

If $x'(-B)z$ is a literal, then $x'(-B)z \notin \bigcup\theta$. Indeed, if $x'(-B)z \in \bigcup\theta$, then z has to be a member of $V(-B, x', \bigcup\theta)$ contradicting our hypothesis. Moreover $Lit(-;)$ does not contain $x'(-B)z$ and thus $\mathcal{M}, v \models x'(-B)z$.

If $x' \text{nf}(-B)z = x'(Q_1 \cup Q_2)z$, then at least one of $x'Q_1z$ and $x'Q_2z$, say $x'Q_1z$ (without loss of generality), is not a **Bool** $_{\bigcup\theta}$ -formula, because otherwise z would belong to $V(-B, x', \bigcup\theta)$, and it is not a **Bool** $_{Lit(-;)}$ -formula either, because otherwise $x'(-B)z$ would have a **Bool**-construction from $Lit(-;)$. Since $x'Q_1z$ is in complement normal form and it is not a **Bool** $_{Lit(-;)}$ -formula, it does not have a **Bool**-construction from $Lit(-;)$. Moreover, $z \in (W_\theta \setminus V(-Q_1, x', \bigcup\theta))$, because $x'Q_1z$ is in complement normal form and it is not a **Bool** $_{\bigcup\theta}$ -formula. Thus, by inductive hypothesis, $\mathcal{M}, v \models x'Q_1z$. Hence, $\mathcal{M}, v \models x'(Q_1 \cup Q_2)z$, so that $\mathcal{M}, v \models x'(-B)z$. Finally, if $x' \text{nf}(-B)z = x'(Q_1 \cap Q_2)z$, then none of $x'Q_1z$ and $x'Q_2z$ is either a **Bool** $_{\bigcup\theta}$ -formula, because otherwise z would belong to $V(-B, x', \bigcup\theta)$, or a **Bool** $_{Lit(-;)}$ -formula. Both $x'Q_1z$ and $x'Q_2z$ do not have a **Bool**-construction from $\bigcup\theta$ and from $Lit(-;)$, and thus $z \in (W_\theta \setminus V(-Q_1, x', \bigcup\theta))$ and $z \in (W_\theta \setminus V(-Q_2, x', \bigcup\theta))$. Hence, by inductive hypothesis, $\mathcal{M}, v \models x'Q_1z$ and $\mathcal{M}, v \models x'Q_2z$. Thus $\mathcal{M}, v \models x'(Q_1 \cap Q_2)z$, so that $\mathcal{M}, v \models x'(-B)z$.

Finally, let us consider the case where $z \in (W_\theta \setminus V(-B, x', \bigcup\theta))$ and $x'(-B)z$ has a **Bool**-construction from $Lit(-;)$. By construction of $Lit(-;)$, $\bigcup\theta$ must contain two $(-;)$ -formulae: $x''(-B_1; Q)y$, decomposed in θ with the variable z and $x'(-B_1; Q)y$ not decomposed in θ . In addition, $\bigcup\theta$ contains literals of type

$x''(-r)z$ differing from the literals of type $x'(-r)z$ in $Lit(-;)$ only because they contain the variable x'' in place of the variable x' . The formulae $x''-(B_1; Q)y$ and $x'-(B_1; Q)y$ satisfy condition (b) of step 2 of the decision procedure of Sect. 4.1 (in the sense that $x'-(B_1; Q)y$ is blocked by $x''-(B_1; Q)y$) and, therefore, by the blocking condition, there must be in $\bigcup \theta$ a $(;)$ -formula $x''(B; S)y$. Clearly, $z \in V(-B, x'', \bigcup \theta)$, the formula zSy is in $\bigcup \theta$, and, by inductive hypothesis, it holds that $\mathcal{M}_\theta, v_\theta \not\models zSy$. Thus, $\mathcal{M}_\theta, v_\theta \models z(-S)y$, as we wished to prove.

Let $\varphi = x'(B; S)y$, with $S = \mathbf{1}$. To prove that $\mathcal{M}_\theta, v_\theta \not\models x'(B; \mathbf{1})y$, we have to show that, for every $z \in W_\theta$, $\mathcal{M}_\theta, v_\theta \models x'(-B)z$ holds. Since θ is a non-axiomatic not further expandable branch of \mathcal{T} , $V(-B, x', \bigcup \theta)$ must be empty. Moreover, $x'(-B)z$ cannot have a Bool-construction from $Lit(-;)$. Assume by contradiction that $x'(-B)z$ has a Bool-construction from $Lit(-;)$. Then, by reasoning as above, there must be a $(;)$ -formula $x''(B; S)y$ such that $z \in V(-B, x'', \bigcup \theta)$. Then $z\mathbf{1}y \in \bigcup \theta$, which is a contradiction, since θ is by hypothesis a non-axiomatic branch. Thus, $x'(-B)z$ does not have a Bool-construction from $Lit(-;)$. In addition, it can be shown that $\mathcal{M}_\theta, v_\theta \models x'(-B)z$ holds, for every $z \in W_\theta$. This can be done by induction on the structure of $x'(-B)z$ much along the same lines of a previous case of this proof, where condition (1) is verified for every $z \in (W_\theta \setminus V(-B, x', \bigcup \theta))$ under the hypothesis that $x'(-B)z$ does not have a Bool-construction from $Lit(-;)$.

Let $\varphi = x(\mathbf{1}; S)y$. To prove that $\mathcal{M}_\theta, v_\theta \not\models x(\mathbf{1}; S)y$, we have to show that, for every $z \in W_\theta$,

$$\text{either } \mathcal{M}_\theta, v_\theta \models x(-\mathbf{1})z \text{ or } \mathcal{M}_\theta, v_\theta \models z(-S)y. \quad (2)$$

Clearly $\mathcal{M}_\theta, v_\theta \models x\mathbf{1}z$, for every $z \in W_\theta$. On the other hand, we know that $zSy \in \bigcup \theta$, for every $z \in W_\theta$, and thus, by inductive hypothesis, we have that $\mathcal{M}_\theta, v_\theta \not\models zSy$. Hence, $\mathcal{M}_\theta, v_\theta \models z(-S)y$ holds, as we wished to prove. Plainly $S \neq \mathbf{1}$ because, by hypothesis, the branch is non-axiomatic.

Let $\varphi = x'(-B; S)y$, with $S \neq \mathbf{1}$. To prove that $\mathcal{M}_\theta, v_\theta \not\models x'(-B; S)y$, we have to show that there exists a $z \in W_\theta$ such that

$$\mathcal{M}_\theta, v_\theta \models x'Bz \text{ and } \mathcal{M}_\theta, v_\theta \models zSy. \quad (3)$$

If there is no $(-;)$ -formula $x''(-B; S)y$ occurring in θ that blocks $x'(-B; S)y$, then $x'(-B; S)y$ is decomposed with an object variable z , and thus $x'(-B)z$ and $z(-S)y$ are in $\bigcup \theta$. By inductive hypothesis, $\mathcal{M}_\theta, v_\theta \not\models x'(-B)z$ and $\mathcal{M}_\theta, v_\theta \not\models z(-S)y$ hold. Hence, we have $\mathcal{M}_\theta, v_\theta \models x'Bz$ and $\mathcal{M}_\theta, v_\theta \models zSy$, as we wished to prove. Otherwise, let $x''(-B; S)y$ be a $(-;)$ -formula blocking $x'(-B; S)y$ and let w be the variable used to decompose $x''(-B; S)y$. Then, $x''(-B)w$ and $w(-S)y$ are in $\bigcup \theta$. By construction of \mathcal{M}_θ , we have $\mathcal{M}_\theta, v_\theta \models x'Bw$ and, since, by inductive hypothesis, $\mathcal{M}_\theta, v_\theta \not\models w(-S)y$, we have $\mathcal{M}_\theta, v_\theta \models wSy$ and thus the thesis follows. The case where $S = \mathbf{1}$ can be treated in a very similar way, considering that $\mathcal{M}_\theta, v_\theta \not\models x'(-B; \mathbf{1})y$ can be proved showing that there is a $z \in W_\theta$ such that $\mathcal{M}_\theta, v_\theta \models x'Bz$.

Let $\varphi = x(-\mathbf{1}; S)y$. To prove that $\mathcal{M}_\theta, v_\theta \not\models x(-\mathbf{1}; S)y$, we have to show that there is a $z \in W_\theta$ such that

$$\mathcal{M}_\theta, v_\theta \models x\mathbf{1}z \text{ and } \mathcal{M}_\theta, v_\theta \models zSy. \quad (4)$$

By construction, $\bar{z}(-S)y \in \bigcup \theta$, for some \bar{z} , and thus, by inductive hypothesis, $\mathcal{M}_\theta, v_\theta \not\models \bar{z}(-S)y$, and $\mathcal{M}_\theta, v_\theta \models \bar{z}Sy$ hold. Clearly, we also have $\mathcal{M}_\theta, v_\theta \models x\mathbf{1}\bar{z}$, as we wished to prove. If $S = \mathbf{1}$, the thesis is immediate. \square

Theorem 2. *If xPy is a valid formula of the $(\{\mathbf{1}, \cup, \cap\}; _)$ -fragment of $\text{RL}(\mathbf{1})$, then the procedure described in Sect. 4.1 yields a closed proof tree for xPy .*

Proof. Suppose by way of contradiction that the procedure described in Sect. 4.1 does not yield any closed proof tree for xPy . Then, step 4 is executed and the procedure yields a non-axiomatic not further expandable branch θ , a model \mathcal{M}_θ , and a valuation v_θ . By Lemma 7, \mathcal{M}_θ and v_θ falsify θ , namely they falsify each formula on the nodes of θ , and thus, in particular, $\mathcal{M}_\theta, v_\theta$ falsify xPy , thus contradicting the hypothesis. \square

Next we show that each decomposition step performed by the decision procedure of Sect. 4.1 preserves falsifiability. This result is needed later in the proof of Theorem 3.

Lemma 8. *Let θ be a branch of a deduction tree for a formula xPy of the $(\{\mathbf{1}, \cup, \cap\}; _)$ -fragment that is being constructed by the procedure of Sect. 4.1 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.*

Proof. Assume that θ is falsifiable and let $\mathcal{M} = (U, m)$ and v be, respectively, a model and a valuation falsifying each node of θ .

The branch θ' is obtained from θ by decomposing a non-literal formula φ occurring on the leaf node N of θ as illustrated in the procedure of Sect. 4.1 and the proof can be carried out according to the type of the formula φ . If φ is a Boolean formula, the thesis follows easily. If $\varphi = x'-(B; S)y$, with $S \neq \mathbf{1}$ and there is no formula of type $x''-(B; S)y$ in $\bigcup \theta$ blocking φ , we decompose $x'-(B; S)y$ using the $(-;)$ -rule illustrated in Table 1 and reason as in [3, Lemma 2(1)]. Otherwise, if there is a formula of type $x''-(B; S)y$ in $\bigcup \theta$ blocking $x'-(B; S)y$, we do not decompose $x'-(B; S)y$. Thus, $N' = N$, $\theta' = \theta$ and trivially $\mathcal{M}, v \not\models N'$ and $\mathcal{M}, v \not\models \theta'$. If $\varphi = x'-(B; \mathbf{1})y$ and there is no formula of type $x''-(B; \mathbf{1})y$ in $\bigcup \theta$ blocking $x'-(B; \mathbf{1})y$, we decompose $x'-(B; \mathbf{1})y$ using the rule $(-;)_b$ of Table 2: $\theta' = \theta N'$, where $N' = (N \setminus \{x'-(B; \mathbf{1})y\}) \cup \{x'(-B)z\}$, with z a new variable for θ . Since $\mathcal{M}, v \not\models x'-(B; \mathbf{1})y$, then $\mathcal{M}, v \models x'(B; \mathbf{1})y$, i.e., there is a $u \in U$ such that $(v(x'), u) \in m(B)$ and $(u, v(y)) \in m(\mathbf{1})$. Let us consider the z -variant v' of v , such that $v'(z) = u$. we have $\mathcal{M}, v' \models x'Bz$ and $\mathcal{M}, v' \models z\mathbf{1}y$. Thus $\mathcal{M}, v' \not\models z(-B)y$ and, since z is a new variable, $\mathcal{M}, v' \not\models N'$. Hence $\mathcal{M}, v' \not\models \theta'$, as we wished to prove. The case where there is a formula of type $x''-(B; \mathbf{1})y$ in $\bigcup \theta$ already decomposed, that satisfies together with $x'-(B; \mathbf{1})y$ the conditions on the applicability of the $(-;)$ -decomposition rule stated in step 2 of the procedure of Sect. 4.1, can be treated as the previous case.

If $\varphi = x-(\mathbf{1}; S)y$, then $\theta' = \theta N'$, where $N' = (N \setminus \{x-(\mathbf{1}; S)y\}) \cup \{z(-S)y\}$, with z a new variable for θ . Since $\mathcal{M}, v \not\models x-(\mathbf{1}; S)y$, then $\mathcal{M}, v \models x(\mathbf{1}; S)y$, i.e., there is a $u \in U$ such that $(v(x), u) \in m(\mathbf{1})$ and $(u, v(y)) \in m(S)$. Let us consider

the z -variant v' of v , such that $v'(z) = u$. Then $\mathcal{M}, v' \models x\mathbf{1}z$ and $\mathcal{M}, v' \models zSy$ hold. Thus, $\mathcal{M}, v' \not\models z(-S)y$ and since z is a new variable, we have $\mathcal{M}, v' \not\models N'$, so that $\mathcal{M}, v' \not\models \theta'$, as we wished to prove.

If $\varphi = x'(B; S)y$, the proof can be carried out along the same lines of [3, Lemma 2(2)].

If $\varphi = x(\mathbf{1}; S)y$ and z is any object variable on N such that zSy is already in θ , we put $\theta' = \theta N'$, with $N' = N \cup \{zSy\}$. The proof that $\mathcal{M}, v \not\models \theta'$ can be carried out as for the previous case, namely $\varphi = x'(B; S)y$. \square

We close the section with the following result.

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

Proof. Suppose, by way of contradiction, that the procedure yields a closed proof tree \mathcal{T}_c of xPy . By definition, all the branches of \mathcal{T}_c must be closed, namely they must be axiomatic. Let us consider the initial deduction tree $\mathcal{T}_0 = \{xPy\}$. By hypothesis, \mathcal{T}_c is falsifiable. Thus, by iteratively applying Lemma 8, \mathcal{T}_c must contain a falsifiable and closed branch θ which is a contradiction. Thus, the procedure constructs a non-axiomatic not further expandable branch θ (the first falsifiable branch it encounters) and then it constructs a model \mathcal{M}_θ that, by Lemma 7, falsifies each formula on θ . \square

5 Conclusions and Future Work

Relational entailment allows 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 $\text{RL}(\mathbf{1})$ allowing to express some simple forms of inclusion between relations. Specifically, we admit inside entailment only positive occurrences of Boolean terms. This permits to express inclusion properties of type ' $(r_1) \cup s \subseteq -r_2$ '.

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 $\mathbf{1}; (-(r_1 \cup s) \cup r_2); \mathbf{1}$ expressing the (positive) inclusion property ' $(r_1 \cup s) \subseteq r_2$ '.

Our further aim is to add, inside entailment, some restricted forms of composition so to be able to express terms of type $\mathbf{1}; (-(s; s) \cup s); \mathbf{1}$ and of type $\mathbf{1}; (-(r; r; r) \cup r); \mathbf{1}$, stating that the relational variables s and r are transitive (i.e., ' $s; s \subseteq s$ ') and tree-transitive (i.e., ' $r; r; r \subseteq r$ '), respectively. Being

able to express these properties is important if we want to use our dual tableau decision procedure for modal logics to reason with incomplete information [4].

We also aim at introducing the converse relation ‘ \sim ’ and the identity relation ‘1’ inside entailment for the purpose of dealing with properties such as symmetry and reflexivity.

Acknowledgments. This work was supported by the Polish National Science Centre research project DEC-2011/02/A/HS1/00395.

References

1. 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.
2. 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.
3. 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.
4. S. Demri, E. Orłowska, 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.
5. 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).
6. 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).
7. 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.
8. 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.
9. J. Golińska-Pilarek, E. Orłowska. Tableaux and dual tableaux: Transformation of proofs. *Studia Logica*, 85(3):283-302, 2007.
10. 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.
11. E. Orłowska, J. Golińska-Pilarek. Dual Tableaux: Foundations, Methodology, Case Studies. *Trends in Logic* vol. 36, Springer, 2011.
12. A. Tarski, S. Givant. A Formalization of Set Theory without Variables. *American Mathematical Society Colloquium Publications*, Providence, Rhode Island, 1987.