Towards an efficient relational deductive system for propositional non-classical logics[⋆]

Andrea Formisano[⋆⋆] and Marianna Nicolosi Asmundo[⋆⋆⋆]

Abstract. We describe a relational framework that uniformly supports formalization and automated reasoning in various propositional modal logics. The proof system we propose is a relational variant of the classical Rasiowa-Sikorski proof system. We introduce a compact graph-based representation of formulae and proofs supporting an efficient implementation of the basic inference engine, as well as of a number of refinements.

1 Introduction

In the last decades, relational formalization of many non-classical propositional logics has been systematically and rigorously studied (see, for instance, [26, 28]). Long-standing research and well established results indicate that standard relational structures can be considered as a common core supporting representations of many non-classical propositional logics. In such an algebraic framework several alternatives for automation of (relational) reasoning can be considered as viable. For instance, to mention some of the approaches proposed in literature, we have tableaux systems [17], Gentzen-style systems [23], systems *à la* Rasiowa-Sikorski [29], display calculus [13], and equational proof systems [10].

Once a relational rendering of a modal theorem is obtained through a translation process, possibly together with the relational counterparts of the modal axioms, a relational proof system can be exploited in order to mechanize modal reasoning. This approach can be seen as alternative and complementary to the common *ad hoc* direct inference methods (cf., e.g., [24]).

This paper mainly focuses on Rasiowa-Sikorski systems. The basic constituent of a Rasiowa-Sikorski system is a collection of inference rules. Similarly to the case of tableaux systems [5], given a theorem to be proved, (the search for) a proof is developed through repeated applications of a set of *decomposition rules.* Through these rules the solution of a problem is reduced to the solutions of (syntactically) simpler sub-problems. The proof is completed whenever a success condition becomes satisfied. Success situations are detected by means of so called *closure rules* that, together with the decomposition rules, characterize the proof system. Dually with respect to tableaux systems, in a Rasiowa-Sikorski system the goal consists in proving a formula to be tautological, instead

[⋆] This work is partially supported by INTAS Project — *Algebraic and deduction methods in non-classical logic and their applications to computer science* and by TARSKI Project — *Theory and Applications of Relational Structures as Knowledge Instruments COST Action 274.* (An extended version of this paper will appear in J. Appl. Non-Classical Logics.)

[⋆⋆] Università di L'Aquila, `formisano@di.univaq.it`

[⋆⋆⋆] Università di Catania, `nicolosi@dmi.unict.it`

of unsatisfiable. For this reason Rasiowa-Sikorski systems are sometimes called *dual-tableaux* systems.

Let us consider any non-classical logic $\mathcal{L}$. Whenever a translation method for formulae of $\mathcal{L}$ into the relational calculus is known, proving a modal theorem φ of $\mathcal{L}$ amounts to prove validity of its relational formulation $t_{\mathcal{L}}(\varphi)$. Different translations are needed to deal with different logics since the proper (modal) axioms characterizing the specific logic have to be reflected in the relational framework. This usually corresponds to identify a collection $\mathcal{C}$ of constant relations to be interpreted as relational counterparts of modal accessibility relations. Proper modal axioms are then rendered by imposing relational axioms which restrain the admitted interpretations of constants in $\mathcal{C}$. Moreover, in the context of a Rasiowa-Sikorski system, properties of relations in $\mathcal{C}$ can also be dealt with by enriching the collection of inference rules of the system. Specific decomposition rules and refined closure rules are then added to the basic inference system.

A goal of the research described here consists in the realization of a relational Rasiowa-Sikorski system suitable to uniformly support modal reasoning for any relationally expressible propositional logic. The duality results linking tableaux and dual-tableaux constitute one of the starting points of this research. In developing the system we are going to describe, we fruitfully adapted and combined several techniques and proof-strategies independently developed for tableaux systems, as well as a compact representation for relational expressions akin to Decision Diagrams. Such techniques, to the best of our knowledge, have never been combined together in the realization of a (relational) deductive framework.

2 A relational logic for propositional non-classical logics

In this section we briefly describe a generic relational logic based on algebras of relations constituting a common framework in which the relational rendering of many non-classical propositional logics can be embedded and homogeneously treated (several examples of such logics are given in Sec. 2.2).

In such a homogeneous framework, all these renderings share basic features: formulae and accessibility relations are represented as binary relations; relational constructs represent extensional and intensional operations; and the deductive apparatus corresponds to a relational calculus.

Let us start by recalling some preliminary notions. Let $\boldsymbol{\mathcal{D}}$ be a non empty set. The full algebra of binary relations over $\boldsymbol{\mathcal{D}}$ is the structure $Re(\boldsymbol{\mathcal{D}}) = (\wp(\boldsymbol{\mathcal{D}} \times \boldsymbol{\mathcal{D}}), \overline{}, \cap, \cup, U, Z, ;, \dagger, \smile, I, D)$. The elements of $Re(\boldsymbol{\mathcal{D}})$ are the subsets of $\boldsymbol{\mathcal{D}} \times \boldsymbol{\mathcal{D}}$. Further, $(\wp(\boldsymbol{\mathcal{D}} \times \boldsymbol{\mathcal{D}}), \overline{}, \cap, \cup, U, Z)$ is a Boolean algebra and $(\wp(\boldsymbol{\mathcal{D}} \times \boldsymbol{\mathcal{D}}), ;, \smile, I)$ is a monoid with involution [1]. Notice that, for the sake of simplicity, we are considering as primitive, or *basic*, all the relational constructs $\overline{}, \cap, \cup, ;, \dagger$, and $\smile$ as well as the constants U, Z, I, D. An alternative possibility could consist in introducing a minimal set of primitive constructs (such as $\overline{}, \cap, ;, \smile, I$, for instance) and in defining the remaining ones in term of these.

Occasionally, we will enrich such a basic structure by considering a collection of relational constants (representing as we will see, accessibility relations) and with a set of *non-standard* relational constructs not definable in terms of the basic ones. A distinction between basic and non-standard relational constructs can be established by classifying a construct as basic if its semantics can be expressed through a first-order sentence in three variables [34]. We will call *non-standard* those constructs that are not expressible in three variables. Recall that establishing the 3-variable expressibility of a sentence is, in general, an undecidable problem [34, 20]. Nevertheless, techniques can be designed to treat particular classes of formulas [3]. Let $\mathcal{L}$ be a non-classical logic and *Rel*$\mathcal{L}$ its relational rendering. In what follows we introduce a common syntax and semantics for *Rel*$\mathcal{L}$ whereas in Sec. 3 we develop the corresponding deductive apparatus.

2.1 Syntax and semantics of *Rel*$\mathcal{L}$

Let $\mathcal{V}$ be a set of individual variables (denoted by u, w, x, y, z, or occasionally by e, t, possibly subscribed), $\mathcal{R}$ a set of relational variables (P,Q,R,...), and $\mathcal{C}$ a set of relational constants. A *relational expression* is any term generated from the symbols in $\mathcal{R} \cup \mathcal{C} \cup \{U, Z, I, D\}$ and the relational constructs. The collection of all the relational expressions is denoted by $\mathcal{E}$. As usual, given a binary relational construct $\circ$, its dual $\diamond$ is defined as $P \diamond Q =_{\mathrm{Def}} \overline{\overline{P} \circ \overline{Q}}$ (and similarly for monadic constructs different from complementation).

A *relational equation* is a writing of the form $R{=}Q$, with $R, Q \in \mathcal{E}$. Shorthand notations for equalities of special kind are also possible, e.g.: $P{\sqsubseteq}Q \leftrightarrow_{\mathrm{Def}} P{-}Q{=}Z$.

A *relational formula* is a writing of the form $x\varphi y$ where $\varphi \in \mathcal{E}$ is a relational expression and $x, y \in \mathcal{V}$ are individual variables. If $\varphi \in \mathcal{R} \cup \mathcal{C} \cup \{U, Z, I, D\}$, then $x\varphi y$ is said to be an *atomic* relational formula. Any atomic relational formula $x\varphi y$ and its complement $x\overline{\varphi}y$ are *literals*. A formula is *compound* if it is not a literal. The *leading* construct of a compound formula $x\varphi y$ is the dual of $\circ$ if $\varphi = \overline{\psi \circ \phi}$ or $\varphi = \overline{\circ\psi}$. While, it is $\circ$ if $\varphi = \psi \circ \phi$ or $\varphi = \circ\psi$.

Relational formulae are interpreted in semantic structures of the kind $\boldsymbol{\mathcal{M}} = \langle \boldsymbol{\mathcal{D}}, \boldsymbol{\mathcal{I}} \rangle$ where $\boldsymbol{\mathcal{D}}$ is a non-empty set and $\boldsymbol{\mathcal{I}}$ is a function assigning:

- A binary relation over $\boldsymbol{\mathcal{D}}$ to each relation symbol in $\mathcal{R} \cup \mathcal{C} \cup \{U, Z, I, D\}$. In particular, U is interpreted as the universal relation, I as the identity, Z as $\overline{U}$, and D as $\overline{I}$. Relational variables are interpreted as right-ideal relations. (A relation R is right-ideal if $R;U = R$.)
- The intended interpretation to the primitive and defined constructs. (Considering, as usual, $\cup$ and $\dagger$ to be the duals of $\cap$ and ;, respectively.)

We call such structures, models of *Rel*$\mathcal{L}$. An evaluation (or assignment) A in $\boldsymbol{\mathcal{M}}$ is a function $A : \mathcal{V} \to \boldsymbol{\mathcal{D}}$. A relational formula $x\varphi y$ of *Rel*$\mathcal{L}$ is satisfied by $\boldsymbol{\mathcal{M}}$ and A if $(x^A, y^A) \in \varphi^{\boldsymbol{\mathcal{I}}}$. In this case we write $\boldsymbol{\mathcal{M}}, A \models x\varphi y$. The formula $x\varphi y$ is true in $\boldsymbol{\mathcal{M}}$ if and only if for every evaluation A in $\boldsymbol{\mathcal{M}}$, it holds that $\boldsymbol{\mathcal{M}}, A \models x\varphi y$. A formula $x\varphi y$ of *Rel*$\mathcal{L}$ is valid if it is satisfied in every model of *Rel*$\mathcal{L}$.

2.2 Relational formalizations of modal logics

In this section we recall the relational formalizations of various propositional logics [26]. For each logic we outline a syntax-directed translation into the algebra

Table 1. Lattice-based modalities: translations and axioms for constant relations

possibility $\Diamond$	$t(\Diamond\chi) =_{\mathrm{Def}} \overline{\leqslant_1;\overline{S_\Diamond;\leqslant_2;t(\chi)}}$	$\leqslant_1^{\smile};R_\Diamond;\leqslant_1^{\smile} \sqsubseteq R_\Diamond$ $\leqslant_2;S_\Diamond;\leqslant_2 \sqsubseteq S_\Diamond$	$R_\Diamond \sqsubseteq S_\Diamond;\leqslant_1^{\smile}$ $S_\Diamond \sqsubseteq \leqslant_2;R_\Diamond$
necessity $\Box$	$t(\Box\chi) =_{\mathrm{Def}} \overline{R_\Box;\overline{t(\chi)}}$	$\leqslant_1;R_\Box;\leqslant_1 \sqsubseteq R_\Box$ $\leqslant_2^{\smile};S_\Box;\leqslant_2^{\smile} \sqsubseteq S_\Box$	$R_\Box \sqsubseteq \leqslant_1;S_\Box$ $S_\Box \sqsubseteq R_\Box;\leqslant_2^{\smile}$
sufficiency $\boxbox$	$t(\boxbox\chi) =_{\mathrm{Def}} \overline{R_\boxbox;\leqslant_2;t(\chi)}$	$\leqslant_1;R_\boxbox;\leqslant_2 \sqsubseteq R_\boxbox$ $\leqslant_2^{\smile};S_\boxbox;\leqslant_1^{\smile} \sqsubseteq S_\boxbox$	$R_\boxbox \sqsubseteq \leqslant_1;S_\boxbox$ $S_\boxbox \sqsubseteq R_\boxbox;\leqslant_1^{\smile}$
dual sufficiency $\diamonddiamond$	$t(\diamonddiamond\chi) =_{\mathrm{Def}} \overline{\leqslant_1;\overline{S_\diamonddiamond;\overline{t(\chi)}}}$	$\leqslant_1^{\smile};R_\diamonddiamond;\leqslant_2^{\smile} \sqsubseteq R_\diamonddiamond$ $\leqslant_2;S_\diamonddiamond;\leqslant_1 \sqsubseteq S_\diamonddiamond$	$R_\diamonddiamond \sqsubseteq S_\diamonddiamond;\leqslant_2^{\smile}$ $S_\diamonddiamond \sqsubseteq \leqslant_2;R_\diamonddiamond$

of relations enriched with a specific collection $\mathcal{C}$ of constants. Such constants are often subject to a set of axioms restraining their admitted interpretations.

Mono-modal logics. This is the basic translation of (propositional) modal formulae into relational terms. In this case $\mathcal{C} = \{\mathsf{r}\}$. The propositional connectives and the necessity operator are so translated:

$$t(\neg\,\psi) =_{\mathrm{Def}} \overline{t(\psi)} \qquad t(\psi \,\&\, \chi) =_{\mathrm{Def}} t(\psi) \cap t(\chi)$$
$$t(\Diamond\,\psi) =_{\mathrm{Def}} \mathsf{r}\,;\, t(\psi) \qquad t(p_i) =_{\mathrm{Def}} p'_i$$

where $p'_i \in \mathcal{R}$ uniquely corresponding to the propositional variable p_i (similar rules are introduced for the other customary propositional connectives).

Multi-modal logic. These logics correspond to multi-modal frames consisting of a relational system where $\mathcal{C}$ enjoys closure properties with respect to relational constructs. Modalities are then of the form $[R]$ and $\langle R\rangle$, where $R \in \mathcal{E}$ [26]. The translation of modal operators is the same as in the case of mono-modal logic. The differences between operators are articulated in terms of the properties of the corresponding accessibility relations.

Lattice-based modal logics. Lattice-based modal logics have the operations of disjunction and conjunction and, moreover, each of them includes a modal operator which can be either a possibility $\Diamond$, or necessity $\Box$, or sufficiency $\boxbox$, or dual sufficiency operator $\diamonddiamond$ (see [8]). Notice that, since negation is not available in these logics, both in the possibility–necessity and in the sufficiency–dual-sufficiency pair, neither operator is expressible in terms of the other. We can also consider mixed languages with any subset of these operators. For all of these logics we have $\{\leqslant_1, \leqslant_2\} \subseteq \mathcal{C}$. Moreover, $\leqslant_1$ and $\leqslant_2$ are assumed to be reflexive and transitive and such that $\leqslant_1 \cap \leqslant_2 = I$. The translations of disjunction and conjunction are:

$$t(\psi \vee \chi) =_{\mathrm{Def}} \overline{\leqslant_1;\overline{\leqslant_2;(t(\psi) \cup t(\chi))}} \qquad t(\psi \,\&\, \chi) =_{\mathrm{Def}} t(\psi) \cap t(\chi).$$

As regards the alternative modalities, Table 1 summarizes their translations. Each modality is modeled in the relational framework by means of a pair of constant relations subject to suitable axioms (also displayed in Table 1).

Temporal logics. We consider here the relational formalization of temporal logics given in [27]. The modalities referring to states in the future are: $\mathtt{G}\phi$ (interpreted as "ϕ will be always true in the future"); $\mathtt{F}\phi$ ("ϕ will be true sometime in the future"); $\phi\,\mathtt{U}\,\chi$ (Until: "there will be an instant in the future when χ is

Table 2. Classification of basic relational formulae

Conjunctive formulae,	$\alpha = xR{\cap}Sy$	$\alpha_1 = xRy$	$\alpha_2 = xSy$	$(\wedge)$
α-formulae	$\alpha = x\overline{R{\cup}S}y$	$\alpha_1 = x\overline{R}y$	$\alpha_2 = x\overline{S}y$	$(\neg\vee)$
Disjunctive formulae,	$\beta = xR{\cup}Sy$	$\beta_1 = xRy$	$\beta_2 = xSy$	$(\vee)$
β-formulae	$\beta = x\overline{R{\cap}S}y$	$\beta_1 = x\overline{R}y$	$\beta_2 = x\overline{S}y$	$(\neg\wedge)$
	$\beta = x\overline{\overline{R}}y$	$\beta_1 = xRy$		$(\neg\neg)$
δ^α-formulae:	$\delta^\alpha = xR;Sy$	$\delta_0^{\alpha_1} = xRz$	$\delta_0^{\alpha_2} = zSy$	$(\exists\wedge)$
	$\delta^\alpha = x\overline{R \dagger S}y$	$\delta_0^{\alpha_1} = x\overline{R}z$	$\delta_0^{\alpha_2} = z\overline{S}y$	$(\neg\forall\vee)$
	where z is an existentially quantified variable $(\delta^\alpha \equiv (\exists z)(\delta_0^{\alpha_1}(z) \wedge \delta_0^{\alpha_2}(z)))$			
γ^β-formulae	$\gamma^\beta = x\overline{R;S}y$	$\gamma_0^{\beta_1} = x\overline{R}z$	$\gamma_0^{\beta_2} = z\overline{S}y$	$(\neg\exists\wedge)$
	$\gamma^\beta = xR \dagger Sy$	$\gamma_0^{\beta_1} = xRz$	$\gamma_0^{\beta_2} = zSy$	$(\forall\vee)$
	where z is a universally quantified variable $(\gamma^\beta \equiv (\forall z)(\gamma_0^{\beta_1}(z) \vee \gamma_0^{\beta_2}(z)))$			
κ-formulae	$\kappa = xR^{\smile}y$	$\kappa_1 = yRx$		
	$\kappa = x\overline{R^{\smile}}y$	$\kappa_1 = y\overline{R}x$		

true and from now *until* then ϕ will be true"); $\mathtt{X}\phi$ ("ϕ will be true in the *next* instant in time"). Analogous modalities are introduced for states in the past.

Relational translations of temporal formulae are expressed by considering an accessibility relation r that links time instants:

$$t(\mathtt{G}\phi) =_{\mathrm{Def}} \overline{\mathsf{r};\overline{t(\phi)}} \qquad t(\mathtt{F}\phi) =_{\mathrm{Def}} \mathsf{r};t(\phi)$$
$$t(\phi\,\mathtt{U}\,\chi) =_{\mathrm{Def}} t(\phi)\,\mathtt{U}\,t(\chi) \qquad t(\mathtt{X}\phi) =_{\mathrm{Def}} t((\phi\,\&\,\phi)\,\mathtt{U}\,\phi)$$

Notice that in translating the modal operator $\mathtt{U}$ we introduced a new relational construct (denoted, for simplicity, by the same symbol). Observe that this construct is *non-standard* (in the sense of Sec. 2, page 3) since it cannot be defined in terms of the basic relational constructs [27]. This is the intended interpretation of $\mathtt{U}$: $P\mathtt{U}Q$ designates the binary relation consisting of all pairs $\langle u, v\rangle$ such that there exists t such that $\langle u, t\rangle$ belongs to the accessibility relation $\mathsf{r}^{\Im}$, $\langle t, v\rangle$ belongs to $Q^{\Im}$, and for all w, if $\langle u, w\rangle \in \mathsf{r}^{\Im}$ and $\langle w, t\rangle \in \mathsf{r}^{\Im}$ then $\langle w, v\rangle \in P^{\Im}$. We will discuss more on this aspect in Sec. 4.3.

Other propositional modal logics. Other modal logics for which it is possible to give a relational formalization are, among others: the logics of knowledge and information described in [6], the logics with specification operators [25], the logics with Humberstone operators [18], the logics with sufficiency operators [7].

2.3 Classification of relational formulae

In order to illustrate the deductive system and the related proof techniques and heuristics in a more concise and clear way, we introduce a classification of relational formulae w.r.t. their leading construct. A *basic* (resp. *non-standard*) relational formula is a formula having a leading construct which is basic (resp. non-standard). Basic relational formulae can be further classified by taking inspiration from Smullyan's uniform notation for first-order logic [33]. In fact, they can be grouped into five categories according to the first-order characterization of the relational constructs (cf. Table. 2).

Table 3. Basic decomposition rules

$$\frac{x\alpha y}{x\alpha_1 y \mid x\alpha_2 y} \qquad \frac{x\beta y}{\begin{array}{c} x\beta_1 y \\ {[x\beta_2 y]} \end{array}} \qquad \frac{x\delta^{\alpha} y}{x\delta_0^{\alpha_1} z, x\delta^{\alpha} y \mid z\delta_0^{\alpha_2} y, x\delta^{\alpha} y} \qquad \frac{x\gamma^{\beta} y}{\begin{array}{c} x\gamma_0^{\beta_1} w \\ w\gamma_0^{\beta_2} y \end{array}} \qquad \frac{x\kappa y}{x\kappa_1 y}$$

If the leading construct is extensional (i.e., $\cup$, $\cap$, $\overline{}$) then, in analogy with Smullyan's notation, the relational formula is classified as an α-formula if it involves intersection as leading construct (or an analogous construct of "conjunctive nature", such as difference or complemented union). Conversely, a formula is classified as β-formula if its leading construct is union (or another one of "disjunctive nature"). Formulae of the form $x\overline{\overline{R}}y$ are classified as β-formulae.

Observe that, the first-order formulation of a formula with leading operator ; (such as $xR; Sy$) is of the form $(\exists z)(xRz \wedge zSy)$, where a conjunction occurs under the scope of an existential quantifier. Dually, formulae with † as leading operator (such as $xR \dagger Sy$) present first-order equipollents of the form $(\forall z)(xRz \vee zSy)$, where a disjunction is universally quantified. From such perspective, ; and † could be seen as compound operators (existential quantification+conjunction, universal quantification+disjunction). Thus, in analogy with Smullyan's uniform notation, where existentially (resp. universally) quantified formulae are classified as δ-formulae (resp. γ-formulae), we classify relational formulae with leading operator ; (resp. †) as δ^{α}-formulae (resp. γ^{β}-formulae).[1] Complemented Peircean constructs are classified analogously. Formulae of kind κ deal with conversion $\smile$.

3 A Rasiowa-Sikorski proof system for relational logics

Proof development in usual Rasiowa-Sikorski systems proceeds by systematically decomposing the (disjunction of) formula(e) to be proved till a tautological condition is detected through a closure rule (examples of such systems are described in [31, 28]). Analogously to [28], the relational proof system we present relies upon a collection of decomposition rules for basic relational formulae and upon a closure rule which takes care of axiomatic sequences such as xRy, $x\overline{R}y$, and xUy (described in Sec. 3.2). The basic decomposition rules of Table 3 have been defined according to the classification of formulae given in Sec. 2.3.

The α- and β-rules are used to decompose conjunctive and disjunctive formulae, respectively. (The square brackets in the β-rule indicate that the second component, namely $x\beta_2 y$ may or may not be present.) The δ^{α}-rule decomposes δ^{α}-formulae, whereas γ^{β}-formulae are expanded by the γ^{β}-rule. The individual variable z introduced in the δ^{α}-rule is chosen among the individual variables occurring in the proof. On the other hand, w in the γ^{β}-rule is an individual

[1] In what follows we feel free to use notations as δ-formula in place of δ^{α}-formula (and similarly for δ-expression, δ-rule, and so on), to denote formulae, expressions, etc., having ; as leading construct.

variable new to (the current branch of) the derivation. Note that such decomposition rules reflect the duality of Rasiowa-Sikorski systems with respect to tableaux systems [12].

The decomposition rules in Table 3 constitute a common core of any relational Rasiowa-Sikorski system. As mentioned, there are logics whose relational translation may involve intensional operators not expressible by means of basic relational constructs. For instance, in temporal logics we introduced a new construct, namely $\mathtt{U}$ as counterpart of the Until modal operators (pag. 4). In cases such that, the decomposition rules from Table 3 do not suffice. In order to manipulate these new constructs some *ad hoc* decomposition rules have to be introduced. In particular, [27] proposes the following rule for $\mathtt{U}$:

$$\frac{xP\mathtt{U}Qy}{x\mathsf{r}t, xP\mathtt{U}Qy \mid tPy, xP\mathtt{U}Qy \mid x\bar{\mathsf{r}}u, u\bar{\mathsf{r}}t, uPy, xP\mathtt{U}Qy}$$

where t is chosen among the individual variables occurring in the proof and u is an individual variable new to the current branch of the derivation.[2]

The introduction of this rule can be justified by considering its frame-based semantics. In fact, we have this (binary) first-order formulation:

$$(\forall x\, y)(x\, P\mathtt{U}Q\, y) \equiv (\forall x\, y)(\exists t)(x\mathsf{r}t \wedge tQy \wedge (\forall u)(x\mathsf{r}u \wedge u\mathsf{r}t) \supset uPy).$$

which translates in the above decomposition rule.

3.1 The proof construction

A Rasiowa-Sikorski derivation $\mathcal{D}$ for a disjunction of relational formulae S is represented as a binary ordered tree whose nodes are labeled by disjunctions of formulae.[3] We call *branch* of $\mathcal{D}$ any maximal path in $\mathcal{D}$. More formally:

Definition 1. *Let S be a disjunction of relational formulae of Rel$\mathcal{L}$. A Rasiowa-Sikorski derivation $\mathcal{D}$ for S is recursively defined as follows.*

The tree with only one node labeled with S, is a derivation for S. Let $\mathcal{D}$ be a derivation for S, θ a branch of $\mathcal{D}$, N the leaf-node of θ. Then, the tree obtained from $\mathcal{D}$ by applying a decomposition rule, as illustrated by items 1-6 below, is a derivation for S. Let φ be a formula in N.

1. *If φ is a β-formula $x\beta y$, add $(N \setminus \{x\beta y\}) \cup \{x\beta_1 y, x\beta_2 y\}$ as a successor of N;*
2. *If φ is a κ-formula, $x\kappa y$, add $(N \setminus \{x\kappa y\}) \cup \{y\kappa_1 x\}$ as successor of N;*
3. *If φ is an α-formula $x\alpha y$, add $(N \setminus \{x\alpha y\}) \cup \{x\alpha_1 y\}$ and $(N \setminus \{x\alpha y\}) \cup \{x\alpha_2 y\}$ as left and right successors of N, respectively;*
4. *If φ is a γ^β-formula $x\gamma^\beta y$, add $(N \setminus \{x\gamma^\beta y\}) \cup \{x\gamma_0^{\beta_1} w, w\gamma_0^{\beta_2} y\}$ as successor of N, where w is a variable new to θ;*
5. *If φ is a δ^α-formula $x\delta^\alpha y$, add $N \cup \{x\delta_0^{\alpha_1} z\}$ and $N \cup \{z\delta_0^{\alpha_2} y\}$ as left and right successors of N, where z is chosen among the variables occurring in $\mathcal{D}$;*
6. *If φ is a non-standard formula, extend θ according to the corresponding specific decomposition rule.*

[2] Notice that the above decomposition rules originate three branches. Nonetheless, it is easy to surrogate such triple-branching by a binary tree.

[3] By abuse of notation, we often identify a node N with the disjuncts of its label.

$$\begin{tikzcd}
 & \text{(1.) } \{x(Q\cup(P;R;S))\cup(\overline{P} \dagger \overline{R;S})y\} \arrow[d, dash, "\beta"] & & \\
 & \text{(2.) } \{xQy,\ xP;R;Sy,\ x\overline{P} \dagger \overline{R;S}y\} \arrow[d, dash, "\gamma^{\beta}"] & & \\
 & \text{(3.) } \{xQy,\ xP;R;Sy,\ x\overline{P}e_1,\ e_1\overline{R}e_2,\ e_2\overline{S}y\} \arrow[dl, dash, "\delta^{\alpha}"] \arrow[dr, dash, "\delta^{\alpha}"] & & \\
\text{(4.) } \{xQy, xPe_1, xP;R;Sy, x\overline{P}e_1, e_1\overline{R}e_2, e_2\overline{S}y\} & & \text{(5.) } \{xQy, e_1R;Sy, xP;R;Sy, x\overline{P}e_1, e_1\overline{R}e_2, e_2\overline{S}y\} \arrow[dl, dash, "\delta^{\alpha}"] \arrow[dr, dash, "\delta^{\alpha}"] & \\
 & \text{(6.) } \{xQy, e_1Re_2, e_1R;Sy, xP;R;Sy, x\overline{P}e_1, e_1\overline{R}e_2, e_2\overline{S}y\} & & \text{(7.) } \{xQy, e_2Sy, e_1R;Sy, xP;R;Sy, x\overline{P}e_1, e_1\overline{R}e_2, e_2\overline{S}y\}
\end{tikzcd}$$

Fig. 1. Rasiowa-Sikorski proof for Example 1

3.2 Closure phase

The closure of a branch is determined by applying a closure rule. For instance, we mentioned that a branch is declared closed whenever its leaf-node contains a pair of the form xRy and $x\overline{R}y$. The following definition summarizes, in more generality, such closure condition by handling the properties of equality.

Definition 2. *A node N is* closed *if one of the following conditions holds:*

1. *N contains the formulae $x_1Dx_2, \ldots, x_{h-1}Dx_h$, $y_1Dy_2, \ldots, y_{\ell-1}Dy_\ell$, $x_1\overline{R}z$, x_hRy_ℓ, with $z = y_\ell$ (for some $h, \ell \geq 1$). In case R is a right-ideal atomic relation, then it is not required that $z = y_\ell$ holds.*
2. *N contains the formulae $x_1Ix_h, x_1Dx_2, \ldots, x_{h-1}Dx_h$ (for some $h \geq 1$).*

Moreover, in order to take care of symmetry of equality, the distinction between x_iDx_j and x_jDx_i in these conditions is considered immaterial. A node is atomically closed *if R is an atomic relation.*

Notice that the basic closure conditions of usual relational Rasiowa-Sikorski systems, cf. [27], are instances of the above ones when putting $h = \ell = 1$.

A derivation $\mathcal{D}$ for a disjunction of formulae S of $Rel\mathcal{L}$ is *satisfied* by a model $\boldsymbol{\mathcal{M}} = \langle \boldsymbol{\mathcal{D}}, \boldsymbol{\mathcal{I}} \rangle$ of $Rel\mathcal{L}$ if, for every variable assignment A and branch θ of $\mathcal{D}$, $\boldsymbol{\mathcal{M}}$ and A satisfy θ (and we write $(\boldsymbol{\mathcal{M}}, A) \models \theta$). A model $\boldsymbol{\mathcal{M}}$ and an assignment A satisfy a branch θ if they satisfy each node of θ. A node N is satisfied by $\boldsymbol{\mathcal{M}}$ and A if at least one formula φ in N is satisfied by $\boldsymbol{\mathcal{M}}$ and A. A branch θ in a derivation is said to be (*atomically*) *closed* if its leaf-node is (atomically) closed. A derivation $\mathcal{D}$ is (atomically) closed if all its branches are (atomically) closed. A closed derivation for S is a *proof* of S.

Example 1. Consider the valid formula $x(Q\cup(P;R;S))\cup(\overline{P} \dagger \overline{R;S})y$. Fig. 1 illustrates a Rasiowa-Sikorski proof for it. The labels of the arcs indicate the decomposition rules applied. In particular, node (2.) has been obtained from node (1.) by applying the β-rule twice. Similarly, to obtain (3.), the γ^{β}-rule has been applied twice. Nodes (4.) and (6.) are closed because of the pairs of literals

xPe_1 and $x\overline{P}e_1$, and e_1Re_2 and $e_1\overline{R}e_2$, respectively. Finally, the pair e_2Sy and $e_2\overline{S}y$ closes node (7.).

4 An efficient representation of formulae and proofs

Trees are the most natural structure to represent proofs in analytic systems such as tableaux and Rasiowa-Sikorski (cf. Fig. 1). Indeed, they reflect the idea of recursively decompose the formula to be proved till elementary contradictions (in case of tableaux) or tautologies (in case of Rasiowa-Sikorski) are found. Nevertheless the frequent presence of redundant parts in the trees, makes the proof search procedure highly inefficient in practice [4]. Therefore, the use of more suitable data structures is required to construct efficient concrete provers.

In what follows we show how to represent relational formulae and Rasiowa-Sikorski proofs by means of labeled acyclic graphs and describe a series of refinements on such basic framework. Such a representation is akin to Binary Decision Diagrams and has several desirable properties when Boolean formulae are processed: it maximizes structure sharing since common sub-formulae are represented (and processed) once; it has unique (up to the ordering imposed on labels of nodes) reduced canonical forms; and satisfiability and tautology checking are easily performed on reduced canonical forms. As we will see, in order to treat Peircean constructs (which involve implicit use of quantification), we need to circumvent the limited expressive power of basic BDD-style representations.

4.1 From trees to graphs: structure sharing

Let us start by considering a simplified scenario where only Boolean constructs (union, intersection, complement, etc) are involved. Thus, only α- and β-rules can be applied to construct a proof for a given relational formula xRy: proving that $R{=}U$ amounts to fully decompose xRy by α- and β-steps to obtain a fully expanded proof $\mathcal{D}$ and then to close each of its leaves by applying the closure rule of Sec. 3.2. A similar situation arises while using BDDs in checking for satisfiability of Boolean functions [2]. For this reason, the use of graph-based representations (such as BDDs or Shannon graphs) have been proposed in the context of tableaux systems (see, among others, [30, 32, 15]). We adopt a similar representation for relational expressions.

Given a relational formula involving only Boolean constructs, by α- and β-decompositions, we obtain its representation as a (labeled) rooted directed acyclic graph. Let us call such graph RDG (standing for relational decision graph). Each non-leaf node n of an RDG has two outgoing edges labeled $-$ and $+$, respectively. Let us denote the corresponding sub-graphs by n^- and n^+, resp. Moreover, let $r(n)$ denote the formula labeling n. An example of RDG of an atomic formula (xSy, in this case) is depicted in Fig. 2 (where $\mathbf{0}$ and $\mathbf{1}$ are new symbols labeling the only two leaf-nodes). In general, an RDG of a given relational formula is built up in syntax directed bottom-up process, from the RDGs of its sub-formulae, by "replacement of leaves". For instance, given the

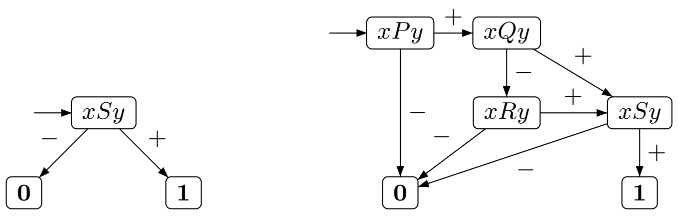


Fig. 2. The RDGs for xSy and $xP\cup((Q\cap R)\cup S)y$

RDGs for xRy and xQy, an RDG for $x(Q\cap R)y$ can be obtained by replacing the **0**-leaf of the former with the root of the latter and merging the two **1**-leaves. This corresponds to apply α-decomposition. Similarly, an RDG for $x(Q\cap R)\cup Sy$ is obtained replacing the **1**-leaf of the RDG for $x(Q\cap R)y$, with the root node of the RDG for xSy (and merging the remaining identical leaves). This corresponds to apply β-decomposition.

The major advantage of adopting such a representation is the maximal structure sharing it intrinsically provides. Indeed, in building up the RDG of a formula, whenever a sub-formula occurs multiply, its sub-RDG is built only once.

An RDG obtained in this way fulfills these conditions: *a)* there are two leaves only (i.e., **0** and **1**); *b)* there are not two distinct non-leaf nodes n_1, n_2 such that the sub-graphs rooted at n_1 and n_2 are isomorphic. On the other hand, at this stage, we do not preclude nodes n such that $n^- = n^+$. (Hence, such RDG are not guaranteed to be *reduced* in the sense of [2].) Moreover, we do not impose any order on the (formulae labeling the) nodes of the RDG. We will deal with these aspects in the sequel, while introducing a normalization procedure for RDGs.

We are left to deal with complementation of relations: given an RDG for xRy, an RDG for $x\overline{R}y$ can be obtained by exchanging the two leaf-nodes.

Given an RDG, a **1**-path is a path from the root node to the **1**-leaf. Any **1**-path p can be denoted as a sequence $n_1, s_1, n_2, s_2, \ldots, n_k, s_k, \mathbf{1}$ (for $k \geq 0$), where each symbol s_i is the label of the i^{th} edge of p. The set $r(p)$ of relational formulae occurring on a **1**-path $p = n_1, s_1, n_2, s_2, \ldots, n_k, s_k, \mathbf{1}$ is defined as:

$$r(p) = \{r(n_i) : s_i = +\} \cup \{\overline{r(n_j)} : s_j = -\}.$$

A **1**-path p is closed if $r(p)$ is closed in the sense of Def. 2. Checking if a relational formula xRy, involving only Boolean constructs, is valid amounts to verifying that every **1**-path p of an RDG for xRy is closed.

4.2 Dealing with Peircean constructs

The procedure described so far does not handle Peircean constructs. As mentioned, γ- and δ-decomposition rules correspond to universal and existential quantifications. Hence dealing with them imposes going beyond the expressive power of basic BDD-like representations.

Among the various approaches proposed in literature, we refine and adapt to our context an interesting extension of un-ordered BDD proposed, for instance, in [30, 32]. In particular, in [30] the authors propose a variant of a free-variable tableaux system where universal quantification is handled by means of

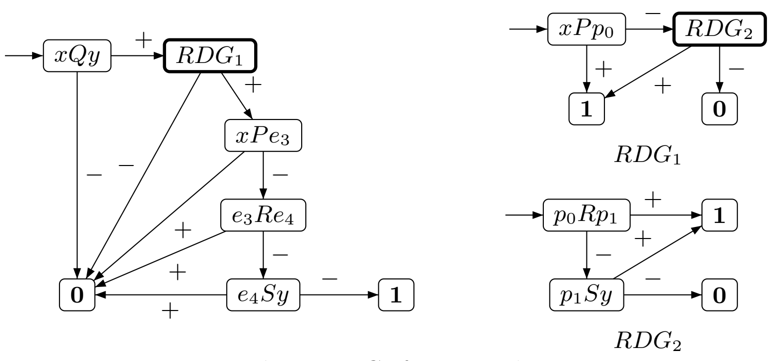


Fig. 3. RDGs for Example 2

a graph-nesting mechanism, while existential quantification is handled through a preliminary transformation into Skolemized negative normal form.

Since, w.r.t. tableaux systems, we are facing a dual problem, (i.e., to prove validity in place of unsatisfiability), we use such nesting mechanism to process δ-formulae instead of γ-formulae. Notice also that we do not need to manipulate first-order formulae in their full generality, this because relational equalities characterize a proper sub-language of first-order logic obeying significant restrictions [34]. This simplifies the treatment and originates a more profitable and efficient usage of the graph-nesting technique. The basic idea consists in allowing an RDG to label a node in another RDG. Whenever a δ-decomposition is applied during the construction of an RDG, another RDG is constructed for the δ-formula at hand. Such an ancillary RDG will be "nested" as a single node (let us call δ-node a node of this kind) in the main RDG.

Example 2. Consider the formula of Example 1. A (nested) RDG for it is depicted in Fig. 3 (where for the sake of readability, we duplicated the **0**-leaf and the **1**-leaf. Thicker lines indicate δ-nodes). Notice how the nesting mechanism is exploited to translate the sub-expression $P; R; S$: the formula $x(P; (R; S))y$ corresponds to RDG_1, while RDG_2 is the auxiliary graph for $p_0(R; S)y$. In the main graph, two γ-decompositions (of the formula $x(\overline{P} \dagger \overline{R; S})y$) introduce the fresh individual variables e_3 and e_4. (Compare this RDG with the tree in Fig. 1.)

In order to reflect the extension mechanism illustrated in Def. 1, any application of a δ-rule must leave open the possibility of further δ-decompositions of the same δ-formula. In each of these decompositions any of the individual variables occurring in the branch being extended may be used. Intuitively speaking, in trying to close a **1**-path p, we proceed by extending p with an instantiation of one of its nested RDGs. Such an instantiation involves one of the individual variables occurring in p. To prepare for this instantiation+extension step, when a δ-node is created, a template for the corresponding (nested) RDG is built and a "placeholder" is used as parameter (cf., p_0 and p_1 in Fig. 3). Such a placeholder will be replaced during the extension step, by the individual variable selected from those occurring in the path.

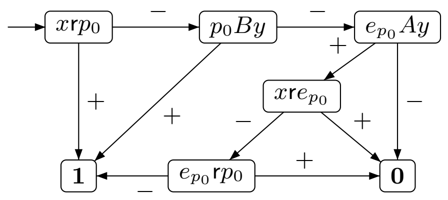
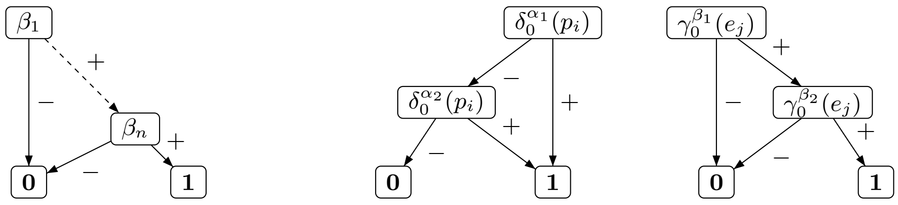


Fig. 4. RDG for the construct U



β_1
β_n
$\delta_0^{\alpha_1}(p_i)$
$\delta_0^{\alpha_2}(p_i)$
$\gamma_0^{\beta_1}(e_j)$
$\gamma_0^{\beta_2}(e_j)$
0
1
+
−

Fig. 5. Initial graph proof for $S = \beta_1 \cup \ldots \cup \beta_n$ and δ^α- and γ^β-decompositions

As regards γ-formulae, we proceed similarly to β-decomposition, with the only difference that a freshly generated individual-variable symbol is introduced (cf, Def. 1). This clearly corresponds to Skolemization and if a γ-decomposition occurs within nested RDGs (i.e. within the scope of one or more δ-expressions), the Skolem term will be parametric in all the corresponding placeholders.

Recall that, the term to be introduced by a γ-decomposition has to be chosen among a finite set of individual variables. (In the case of tableaux systems such term may be selected among any of the compound terms of the language of the current branch. Such a language could be infinite.)

The remaining primitive Peircean construct, conversion, is eliminated during the generation of the RDG by rewriting any formula of the form $xR^{\smile}y$ as yRx.

4.3 Non-standard relational constructs

Thanks to the use of nesting, it is possible to generalize RDGs in order to treat any kind of construct, provided that its semantics can be given in terms of a formula of binary predicate logic. As an example, let us consider the Until operator of temporal logic and the corresponding relational construct U. It can be characterized by the first-order formula $x\, A\mathtt{U}B\, y \equiv \exists t(x\mathsf{r}t \wedge tBy \wedge (\forall u)(uAy \vee x\bar{\mathsf{r}}u \vee u\bar{\mathsf{r}}t))$, where r denotes a constant relation modeling world accessibility. The structure of the formula suggests the structure of an RDG: we treat disjunctions (resp. conjunctions) similarly to α-decompositions (resp. β-decompositions). The RDG sought for is shown in Fig. 4. Whenever, in building up an RDG for a given formula, the construct U has to be decomposed, a δ-node having the graph in Fig. 4 as nested RDG is created.

Such a mechanism corresponds to use a sort of "macro expansion" of non-standard constructs. In fact, the part of the graph related to the existentially quantified variable t presents a structure similar to the graph generated by a δ-decomposition (the only difference is in the occurrences of individual variables).

Conversely, the part of the graph related to the universally quantified variable u presents a structure similar to the graph generated by a γ-decomposition. These analogies allows us to reuse the very same machinery developed for basic Peircean constructs to handle nested RDG originated by non-standard constructs.

5 Soundness and Completeness

In this section we briefly outline the ideas behind the proofs of soundness and completeness, limiting ourselves to consider the graph-based version of the system with basic rules only. A detailed treatment, including the missing proofs, can be found in [9]. In Sec. 4 a syntax directed bottom-up process is outlined to construct graph-based representations of Rasiowa-Sikorski proofs. This choice is basically due to efficiency reasons, since it allows maximization of structure sharing. Here, in order to make proofs of the soundness and completeness statements simpler, we introduce a top-down recursive construction of the very same graph-based proof.

Definition 3. *Let S be a disjunction of formulae of $Rel\mathcal{L}$. A derivation $\mathcal{G}$ for S is recursively defined as follows. The graph in Fig. 5 (with $\beta_1, \ldots, \beta_n$ the disjuncts in S) is a derivation for S. Moreover, let $\mathcal{G}$ be a derivation for S, then the graph $\mathcal{G}'$, obtained from $\mathcal{G}$ by applying one of the decomposition steps described in Sec. 4.1 and 4.2, as shown below, is a derivation for S. Let n be any node in $\mathcal{G}$ labeled by φ.*

1. *If φ is a β-formula, replace n with its β-decomposition;*
2. *If φ is an α-formula, replace n with its α-decomposition;*
3. *If φ is a κ-formula, substitute φ with its component $y\kappa_1 x$;*
4. *If φ is a γ^β-formula, replace n with its γ^β-decomposition, with e_j a variable new to every **1**-path containing n;*
5. *If φ is a δ^α-formula, add its δ^α-decomposition as successor of n, with the placeholder p_i replaced by any e_i, chosen among the variables in $\mathcal{G}$.*

Soundness. The relational Rasiowa-Sikorski system with graph-based proof representation is sound if every disjunction of relational formulae with a closed derivation (graph) is valid. The statement can be proved by showing that preservation of validity is an invariant property throughout the recursive graph construction illustrated in Def. 3, and that the closure rule from Def. 2 "closes" only **1**-paths p with tautological $r(p)$.

Completeness. The system is complete if for every valid disjunction S of relational formulae it is able to produce a finite closed RDG for S. As usual, the statement is proved by exhibiting a fair proof-search procedure (such as the one introduced in Sec. 6) that, in a deterministic way, builds a *saturated* RDG for a disjunction of relational formulae S. A derivation $\mathcal{G}$ for S is *propositionally saturated* if it is constructed out of the initial graph-proof for S by applying only decomposition steps $1-4$ from Def. 3, till all its nodes are labeled by either atomic formulae or by δ^α-formulae. A δ^α-formula occurrence $x\delta^\alpha y$ in $\mathcal{G}$

Table 4. Rewriting system for the normalization process

1)	$RDG(r(n), n^-, n^+) \rightsquigarrow n^-$	if $n^- = n^+$
2)	$RDG(r(n_1), RDG(r(n_2), n_2^-, n_2^+), n_1^+) \rightsquigarrow RDG(r(n_1), n_2^-, n_1^+)$	if $r(n_1) = r(n_2)$
3)	$RDG(r(n_1), n_1^-, RDG(r(n_2), n_2^-, n_2^+)) \rightsquigarrow RDG(r(n_1), n_1^-, n_2^+)$	if $r(n_1) = r(n_2)$
4)	$RDG(r(n_1), RDG(r(n_2), n_2^-, n_2^+), n_1^+) \rightsquigarrow$ $RDG(r(n_2), RDG(r(n_1), n_2^-, n_1^+), RDG(r(n_1), n_2^+, n_1^+))$	if $r(n_1) \succ r(n_2)$
5)	$RDG(r(n_1), n_1^-, RDG(r(n_2), n_2^-, n_2^+)) \rightsquigarrow$ $RDG(r(n_2), RDG(r(n_1), n_1^-, n_2^-), RDG(r(n_1), n_1^-, n_2^+))$	if $r(n_1) \succ r(n_2)$
6)	$RDG(r(n), n^-, n^+) \rightsquigarrow n^-$	if $r(n)=xUy$
7)	$RDG(r(n), n^-, n^+) \rightsquigarrow n^+$	if $r(n)=xZy$
8)	$RDG(r(n), n^-, n^+) \rightsquigarrow n^-$	if $r(n)=xIx$
9)	$RDG(r(n), n^-, n^+) \rightsquigarrow n^+$	if $r(n)=xDx$

is *fully expanded* if, for every variable e_i in the **1**-paths of $\mathcal{G}$ containing $x\delta^\alpha y$ with a positive sign, the corresponding δ^α-decomposition graph (see Fig. 5) instantiated to e_i, is propositionally saturated and attached as a right successor to $x\delta^\alpha y$. A derivation $\mathcal{G}$ for S is saturated if it is propositionally saturated and each δ^α-formula occurring in it is fully expanded. Every **1**-path on a saturated derivation is said to be saturated. It can be proved that if p is a saturated open **1**-path, then there exists a model not satisfying $r(p)$ [9].

6 Towards an efficient implementation

In this section we describe a number of techniques we adopted in implementing an automated Rasiowa-Sikorski deduction system. The system we are going to delineate has been implemented in SICStus Prolog.

Ordering. Given a relational formula, the procedure outlined in Sec. 4 produces an RDG without imposing any order on the (atomic formulae labeling the) nodes. Furthermore, it may be the case that the same formula labels two distinct nodes in a path. We introduce now a normalizing procedure that, by proceeding top-town, imposes a given order $\succ$ on the nodes of the RDG. Such a procedure is described as a term rewriting system (in the spirit of [16, 37]). The rewriting process will continues until no further rewriting is applicable. An useful piece of notation: let the term $RDG(r(n), n^-, n^+)$ denote the RDG rooted in the node n. The rewriting system is constituted of the rules $1) - 5)$ in Table 4, where $\succ$ is any total ordering on the collection of relational atomic formulae. Notice that all these rules preserve the closure property of the RDGs. Moreover, if all of the **1**-paths of the initial RDG can be closed because of pairs of complementary formulae, then the normalization process yields an RDG made of a single **0**-leaf. This immediately certifies the initial formula to be tautological. The described rewriting system can be enriched by adding additional rules (i.e. ,6) $-$ 9) in Table 4) to handle basic constants U, Z, I, D. In this way, the rewriting process simplifies the RDG by removing those **1**-paths which are tautological because of atomic formulae of the forms xUy, xIx, etc.

To impose node ordering and maximal structure-sharing (even between isomorphic sub-graphs of different nested graphs), the normalization process is recursively applied to each nested RDG.

Remark 1. Two refinements often adopted in tableaux systems are the use of lemmas and the imposition of some sort of regularity constraint on the application strategy for decomposition rules [21]. These techniques have as counterpart, in our system, the adoption of a graph-based representation where ordering and maximal structure sharing are imposed.

Equality and symmetric or reflexive relations. The adoption of a two-phases approach (i.e., a procedure to build-up the RDG coupled with a normalizing phase), instead of developing a procedure to obtain an ordered RDG directly from the formula, permits a simpler treatment of those relations subject to specific properties or axioms. Examples are reflexivity and symmetry.

Let us start by considering any relational expression R which is known to denote a symmetric binary relation, i.e., such that $\forall x \forall y(xRy \rightarrow yRx)$. In this case the label xRy be rewritten as yRx preserving the validity of the whole relational expression. This fact justifies the introduction of the following rewriting rule in the normalization procedure:

$$\mathit{RDG}(xRy, n^-, n^+) \rightsquigarrow \mathit{RDG}(yRx, n^-, n^+) \ \text{ if } x \succ y$$

where we consider the total order $\succ$ as extended to the collection of all individual variables. On the other hand, whenever a relation R is such that $\forall x(xRx)$ holds, we can apply the following rewriting rule:

$$\mathit{RDG}(xRy, n^-, n^+) \rightsquigarrow n^- \ \text{ if } x = y$$

Clearly, these rules applies in the case of the constants I and D, as well. Nevertheless, in the case of D more fruitful rewriting rules can be introduced:

$$\begin{aligned} \mathit{RDG}(xDy, n^-, n^+) &\rightsquigarrow \mathit{RDG}(yDx, n^-, n^+) \ \text{ if } x \succ y \\ \mathit{RDG}(xDy, n^-, n^+) &\rightsquigarrow \mathit{RDG}(xDy, n^-, n') \ \text{ if } y \succ x \end{aligned}$$

where n' is obtained from n^+ by substituting each occurrence of y with x. For instance, if $x_1 \succ \cdots \succ x_h \succ y_1 \succ \cdots \succ y_\ell$, such rules (in combination with the others) rewrite the set $\{x_1 D x_2, \ldots, x_{h-1} D x_h, y_1 D y_2, \ldots, y_{\ell-1} D y_\ell, x_1 \overline{R} y_1, x_h R y_\ell\}$ into $\{x_h \overline{R} y_\ell, x_h R y_\ell\}$.

Remark 2. It is important to notice that, in virtue of this refinement of the rewriting system, we can sensibly simplify the closure rule. Actually, we can modify the closing conditions in Def. 2 by imposing $h = \ell = 1$.

Proof-search procedure. Once an RDG is normalized, two situations may arise. If the RDG is made of a single node, then establishing validity of the initial formula xRy is immediate. Conversely, it might be the case that such RDG contains a number of non-trivial **1**-paths. In order to declare xRy valid, each of them has to be closed. Checking all the **1**-paths involves visiting the whole (finite) RDG. For any **1**-path $p : n_1, s_1, n_2, s_2, \ldots, n_k, s_k, \mathbf{1}$ let $l(p)$ be the set of formulae of $r(p)$ restricted to the nodes of p that are not δ-nodes. (Notice that, by the bottom-up procedure used to obtain the RDGs, $l(p)$ is a set of literals.) Moreover, let $\delta(p)$ be the set of δ-nodes n_i, in p, such that $s_i = +$. For any **1**-path two situation are possible: *i)* $l(p)$ satisfies a closure condition. In this case p is declared closed. *ii)* $l(p)$ does not satisfy any closure condition. In this case closure may still be achievable by performing some extensions using one or more nested RDG. Hence, the search procedure proceeds by selecting a δ-node m in $\delta(p)$ and an individual variable e among those occurring in the path. Let p_m be the placeholder of m. The **1**-path p is extended with a copy of the ancillary RDG of m, with e replaced for the placeholder p_m. The extension happens substituting the ending **1**-leaf of p with the root of the selected RDG. Such a step introduces a number of new **1**-paths (all of them having p as prefix). At this point the process (recursively) proceeds trying to close these new **1**-paths. The process continues, possibly by applying further extension steps, until, either all **1**-paths are declared closed; or no more extensions are possible; or some termination condition is verified.

Clearly, suitable strategies and heuristics should be exploited to guide the proof procedure. In particular, two questions must be answered each time an extension step has to be performed: *a)* which δ-node is selected? And *b)* which individual variable is selected? Since more than one δ-node may be necessary in order to close a **1**-path, any *fair* selection rule can be adopted. As regards the selection of individual variables for repeated use of the same δ-node, we rely once more on the order $\succ$ and each time an extension step is performed the next unused variable according to $\succ$ is used.

It must be noted that Skolem terms may occur in nested RDG. Such terms are parameterized by placeholders names. Hence each time an extension step is performed, new individual variables (may) be introduced in the (extended) **1**-path. Consequently, the search procedure is not guaranteed to terminate. This phenomenon cannot be avoided, since establishing the validity of a relational formula is, in general, undecidable. To avoid infinite computation we adopt bounded depth-first iterative deepening [19].

Given a specific RDG, an effective approach in efficiently implementing the proof-search procedure outlined so far consists in generating the Prolog code which encodes the proof-search itself. In particular, each node n of the RDG originates a Prolog clause which, during execution, calls the clauses corresponding to n^+ and n^-. Clearly, the clause related to the **1**-leaf of the RDG also encodes the extension mechanism. Hence the proof of a modal theorem follows these phases: *a)* the given formula is translated in relational form (Sec. 2.2); *b)* an RDG for such form is generated (Sec. 4); *c)* the RDG is normalized (Sec. 6); *d)* the normalized RDG is compiled into a Prolog program; *e)* the search for the proof is done by executing such program.

Some concluding remarks about the graph representation: we conclude this section by observing that our approach presents some differences w.r.t. the one in [30] in at least two further aspects that have significant impact in the realization of the proof-search procedure. First, we do not make use of free-variables, even if, in principle, this is not precluded. Hence our system turns out to be closer to ground tableaux systems that to free-variable ones. As a consequence, we do not use any unification procedure to implement closure rules. Second, we profitably impose ordering of nodes. This sensibly reduces the size of the generated RDGs to be processed by a subsequent proof-search procedure.

7 Conclusions and future works

The research we reported upon in this paper shows that the duality results holding between (ground) tableaux systems and Rasiowa-Sikorski systems, naturally provide strong support to cross-fertilization between the two streams of research. Actually, we explored just a small portion of the immense work and well established research done in the context of tableaux systems, considering a rather smooth application of such technology to the development of Rasiowa-Sikorski systems. Besides tableaux technology, techniques and heuristics can be borrowed from related fields such as automation of propositional logic, equational reasoning, term rewriting systems, semantic unification, theory reasoning, to mention some. In particular, in this initial work, we adopted a representation akin to Decision Diagrams, in origin designed to implement propositional logics. Moreover, we reported on the applicability of techniques developed for equality handling and based on term rewriting rules.

Several steps in this direction of research have to be yet completed. For instance, the theoretical basis of our system design are inspired to ground versions of tableaux, but an interesting theme for further study would consist in investigating the use of *free variables* in relational Rasiowa-Sikorski systems. In general, free-variable tableaux systems can be implemented so to ensure greater efficiency, w.r.t. ground tableaux (see [11] for an efficient treatment of the unification procedure). It is reasonable that the use of free variables in Rasiowa-Sikorski systems could be advantageous as well.

The system described in this paper can be seen as an alternative to other, more traditional methodologies such as the ones reviewed in [14]. Comparisons in term of proof complexity and efficiency are needed in order to identify those cases where our deduction method behaves better (or worse) than others.

Most of the modal logics of interest are decidable. Decidability results constitute a solid theoretical background for the development of attractive specific inference tools for such logics. A challenging theme of research consists in detecting under which conditions the decidability results for a non-classical logic can be reflected in some decidability property of its relational counterpart. A possibility could consist in identifying particular classes of deduction rules, together with a specific strategy in rule application, that ensures decidability in the relational framework.

Finally, an extensive comparison has to be done with alternative approaches and tools designed for relational and modal reasoning. Among the systems supporting various forms of relational manipulation and reasoning we would like to mention, RALF, RELVIEW, RALL, δRA, and ARA. References for most of such systems can be found in [36]. (Among the tools based on Rasiowa-Sikorski systems, we mention RelDT [22].) As regards systems expressively dedicated to modal reasoning, references for most of them can be found in [35].

Acknowledgements We thank Martin Giese, Joanna Golińska, Rajeev Goré, and Ewa Orłowska for useful discussions on the topics of this paper.

References

[1] C. Brink, W. Kahl, and G. Schmidt. *Relational Methods in Computer Science.* Advances in Computing Science. Springer, 1997.

[2] R. E. Bryant. Graph-based algorithms for boolean function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[3] D. Cantone, A. Formisano, E. Omodeo, and C. Zarba. Compiling dyadic first-order specifications into map algebra. *Theoretical Computer Science*, 293(2):447–475, 2003.

[4] M. D'Agostino. Are tableaux an improvement on truth tables? *Journal of Logic, Language and Information*, 1:235–252, 1992.

[5] M. D'Agostino, D. M. Gabbay, R. Hänle, and J. Posegga, editors. *Handbook of tableau methods.* Kluwer Academic Publishers, Dordrecht, 1999.

[6] S. Demri and E. Orłowska. *Incomplete Information: Structure, Inference, Complexity.* Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2002.

[7] I. Düntsch and E. Orłowska. Beyond modalities: Sufficiency and mixed algebras. In E. Orłowska and A. Szalas, editors, *Relational Methods for Computer Science Applications*, pages 277–299, Heidelberg, 2001. Physica-Verlag.

[8] I. Düntsch, E. Orłowska, A. M. Radzikowska, and D. Vakarelov. Relational representation theorems for some lattice-based structures. *Journal on Relational Methods in Computer Science*, 1:132–160, 2004.

[9] A. Formisano and M. Nicolosi Asmundo. An efficient relational deductive system for propositional non-classical logics. Technical Report 8/06, Dipartimento di Informatica, Università di L'Aquila, 2006.

[10] A. Formisano, E. G. Omodeo, and M. Temperini. Instructing equational set-reasoning with otter. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Automated reasoning: First International Joint Conference, IJCAR 2001. Proceedings*, volume 2083 of *LNCS*, pages 152–167. Springer, 2001.

[11] M. Giese. *Proof Search without Backtracking for Free Variable Tableaux.* PhD thesis, Fakultät für Informatik, Universität Karlsruhe, July 2002.

[12] J. Golińska-Pilarek and E. Orłowska. Tableaux and dual tableaux: Transformation of proofs. unpublished, 2006.

[13] R. Goré. Cut-free display calculi for relation algebras. In *CSL*, volume 1258 of *LNCS*, pages 198–210, 1996. Selected Papers of the Annual Conference of the Association for Computer Science Logic.

[14] R. Goré. Tableau methods for modal and temporal logics, 1999. In [5].

[15] J. Goubault. Proving with BDDs and control of information. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, volume 814 of *LNCS*, pages 499–513, Nancy, France, 1994. Springer.

[16] J. F. Groote and J. van de Pol. Equational binary decision diagrams. In M. Parigot and A. Voronkov, editors, *Logic for programming and automated reasoning: 7th International Conference, LPAR 2000*, volume 1955 of *LNCS*, pages 161–178. Springer, 2000.

[17] M. C. B. Hennessy. A proof system for the first-order relational calculus. *Journal of Computer and System Sciences*, 20(1):96–110, 1980.

[18] L. Humberstone. Inaccessible worlds. *Notre Dame Journal of Formal Logic*, 24:346–352, 1983.

[19] R. E. Korf. Depth-first iterative deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[20] M. Kwatinetz. *Problems of Expressibility in Finite Languages.* Doctoral diss., Univ. of California, Berkeley, 1981.

[21] R. Letz. First-order tableau methods, 1999. In [5].

[22] W. MacCaull and E. Orlowska. A logic of typed relations and its applications to relational databases. Technical report, Department of Mathematics, Statistics and Computer Science, St. Francis Xavier University, Antigonish, Canada, 2003.

[23] R. D. Maddux. A sequent calculus for relation algebras. *Annals of Pure and Applied Logic*, 25:73–101, 1983.

[24] H. J. Ohlbach, A. Nonnengart, M. de Rijke, and D. M. Gabbay. Encoding two-valued non-classical logics in classical logic. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 21, pages 1403–1486. Elsevier Science B.V., 2001.

[25] E. Orłowska. Proof system for weakest prespecification. *Information Processing Letters*, 27(6):309–313, 1988.

[26] E. Orłowska. Relational semantics for non-classical logics: Formulas are relations. In J. Wolenski, editor, *Philosophical Logic in Poland.*, pages 167–186. Kluwer, 1994.

[27] E. Orłowska. Temporal logics — in a relational framework. In L. Bolc and A. Szalas, editors, *Time and Logic — A Computational Approach.*, pages 249–277. Univ. College London Press, 1995.

[28] E. Orłowska. Relational proof systems for modal logics. In H. Wansing, editor, *Proof Theory of Modal Logic*, volume 2, pages 55–77. Kluwer, 1996.

[29] E. Orłowska. Relational formalisation of nonclassical logics. In C. Brink, W. Kahl, and G. Schmidt, editors, *Relational Methods in Computer Science*, Advances in Computing Science, chapter 6, pages 90–105. Springer, Wien, New York, 1997.

[30] J. Posegga and P. H. Schmitt. Implementing semantic tableaux, 1999. In [5].

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

[32] K. Schneider, R. Kumar, and T. Kropf. Accelerating tableaux proofs using compact representations. *Formal Methods in System Design*, 5(1-2):145–176, 1994.

[33] R. M. Smullyan. *First-Order Logic.* Dover Publications, New York, second corrected edition, 1995. First published 1968 by Springer.

[34] A. Tarski and S. Givant. *A Formalization of Set Theory without Variables*, volume 41 of *Colloquium Publications.* American Mathematical Society, 1987.

[35] Web site of AiML. `www.cs.man.ac.uk/~schmidt/tools`.

[36] Web site of RelMiCS. `www2-data.informatik.unibw-muenchen.de/relmics/html`.

[37] H. Zantema and J. van de Pol. A rewriting approach to binary decision diagrams. *Journal of Logic and Algebraic Programing*, 49(1-2):61–86, 2001.