Two Proof Systems for Peirce Algebras*

Renate A. Schmidt[1], Ewa Orlowska[2], and Ullrich Hustadt[3]

[1] University of Manchester, UK, schmidt@cs.man.ac.uk
[2] National Institute of Telecommunications, Warsaw, orlowska@itl.waw.pl
[3] University of Liverpool, UK, U.Hustadt@csc.liv.ac.uk

Abstract. This paper develops and compares two tableaux-style proof systems for Peirce algebras. One is a tableau refutation proof system, the other is a proof system in the style of Rasiowa-Sikorski.

1 Introduction

The purpose of this paper is twofold. First, we develop two proof systems for the class of Peirce algebras, namely a Rasiowa-Sikorski-style system and tableau system. Second, we present in a formal way a principle of duality between these proof systems.

Procedurally, the two systems are very similar. They both use a top-down approach. Their rules are of the form

$$\frac{X}{X_1 \mid \ldots \mid X_n} \tag{1}$$

where both the numerator X and the denominators $X_1, \ldots, X_n$ ($n \geq 1$) are finite sets of formulae. Given a formula of a logic, the decomposition rules of the systems enable us to decompose it into simpler formulae, or the specific rules enable us to modify a set of formulae. Some sets of formulae have the status of axioms and are used as closure rules. Applying the rules to a given formula we form a tree (or tableau) whose nodes consist of finite sets of formulae. We stop applying the rules to a node of the tree (i.e., we close the corresponding branch) whenever we eventually obtain an axiomatic set of formulae.

The main difference between the two systems is in their underlying semantics. Rasiowa-Sikorski systems are validity checkers. The rules preserve and reflect validity of the sets of formulae which are their premises and conclusions (i.e. branching is interpreted as conjunction), and the axiomatic sets are valid. Validity of a set means first-order validity of the disjunction of its formulae (i.e., comma is interpreted as disjunction). In order to verify validity of a formula, we place it at the root of a tree and we apply the rules until all the branches

* This work was supported by EU COST Action 274, and research grants GR/M88761 and GR/R92035 from the UK Engineering and Physical Sciences Research Council. Part of the work by the first author was done while on sabbatical leave at the Max-Planck-Institut für Informatik, Germany, in 2002.

close (i.e. we reached a valid set of formulae) or there is an open branch that is complete. Completeness of a branch means that all the rules that can be applied have been applied. A formula is valid if there is a proof tree for it, that is, a tree where all the branches close.

Tableau systems are unsatisfiability checkers. In tableau systems the rules preserve and reflect unsatisfiability of sets of formulae which are their premises and conclusions, axiomatic sets are unsatisfiable, and satisfiability of a set means first order satisfiability of the conjunction of its formulae (i.e., comma is interpreted as conjunction). Equivalently, a rule of the form as above is admissible whenever X is satisfiable iff either of X_i, $(1 \leq i \leq n)$, is satisfiable (i.e. branching is interpreted as disjunction). In order to verify unsatisfiability of a formula, we place it at the root of a tree and we apply the rules until all the branches of the tree close (i.e. we reached an unsatisfiable set of formulae) or there is an open branch that is complete. As before, completeness of a branch means that all the rules that can be applied have been applied. A formula is unsatisfiable if there is a proof tree for it, where all the branches close. Clearly, applying the tableau proof procedure to the negation of a formula we can check validity of the formula itself.

We formalise a duality between Rasiowa-Sikorski and tableau systems in terms of a classification of the rules. It follows that the duality can be observed at the syntactic level. Clearly, its justification is based on semantic features of the two systems.

Since Peirce algebras are, in a sense, a join of a Boolean algebra and a relation algebra, the proof systems presented in the paper inherit many features, on the one hand, from the proof systems for first-order logic [18, 23, 7] and, on the other hand, from the proof systems for relation algebras [25, 10, 22, 14, 9, 4, 17, 6, 13].

2 Peirce algebra

Peirce algebra, introduced in Britz [3] and refined in Brink, Britz and Schmidt [2], formalises the properties of binary relations and sets, and their interactions. Although these can also be formalised in relation algbera (see remarks at the end of the section), our interest in Peirce algebra is motivated by practical considerations. Peirce algebra provides a natural algebraic framework for various branches of computer science. Applications include the modelling of programming constructs [2], natural language analysis [19, 20], and the interpretation of description logics which are used for knowledge representation and reasoning [2, 19]. Also, the algebraic semantics of modal logics and extended modal logics, such as Boolean modal logic [8] and dynamic modal logic [5] can be studied in the framework of Peirce algebra. It is not difficult to see that all these applications can be modelled in relation algebra. Of particular interest to these applications is the fact that Peirce algebra has more interesting and well-behaved reducts than relation algebra. For instance, many decidable description logics and extended modal logics (see [21] for an overview of the latter) correspond directly to reducts of Peirce algebra. By following the ideas of [17] and exploiting results

in [2], problems in these description or modal logics can of course be translated into relation algebra statements. However, experiments with the first-order logic theorem prover MSPASS [11] show that the performance is superior on the first-order encodings of Peirce algebra statements than relation algebra statements corresponding to problems belonging to decidable description and modal logics. This kind of behaviour is not particular to MSPASS; it is also expected from other first-order logic theorem provers. A simple explanation is that generally, in first-order logic provers, unary literals, which correspond to the Boolean elements, can be handled much more effectively than binary predicates.

Formally, a *Peirce algebra* (as defined in Brink et al [2]) is a two-sorted algebra $(\mathfrak{B}, \mathfrak{R}, :, ^c)$ defined equationally by:

1. $\mathfrak{B} = (B, +, \cdot, -, 0, 1)$ is a Boolean algebra,
2. $\mathfrak{R} = (R, +, \cdot, -, 0, 1, ;, ^\smile, e)$ is a relation algebra [24],
3. $:$ is a mapping $R \times B \longrightarrow B$ satisfying ($r, s \in R$ and $a, b \in B$):
   M1 $r : (a + b) = r : a + r : b$
   M2 $(r + s) : a = r : a + s : a$
   M3 $r : (s : a) = (r ; s) : a$
   M4 $e : a = a$
   M5 $0 : a = 0$
   M6 $r^\smile : -(r : a) \leq -a$
4. ^{c} is a mapping $B \longrightarrow R$ satisfying ($a \in B$ and $r \in R$):
   P1 $a^c : 1 = a$
   P2 $(r : 1)^c = r ; 1$

The operation $:$ is the *Peirce product* of Brink's Boolean modules [1]; in fact, M1–M6 are the axioms of Boolean modules which capture the interrelationship of the operators, except ^{c} with the Peirce product. A Peirce algebra is therefore an extension of a Boolean module $(\mathfrak{B}, \mathfrak{R}, :)$ with an operation ^{c} defined by 4. The operation ^{c} is called the *left cylindrification* operation.

On the set-theoretic level the Peirce product multiplies a binary relation R with a set A to give the set $R : A = \{x \mid \exists y\ (x, y) \in R \text{ and } y \in A\}$. This gives an algebraic interpretation of the multi-modal diamond operator. The intuitive definition of the left cylindrification of a set A is $A^c = \{(x, y) \mid x \in A\}$, i.e. the relation with domain A and the range consisting of all the elements of a corresponding universe. There are other ways to formalise the relationship from sets to relations. The test operator of propositional dynamic logic (*PDL*), domain restriction and the cross product would be alternatives to the left cylindrification operator [2].

Peirce algebras are expressively equivalent to relation algebras. This follows from the observation in Brink et al [2] that the Boolean elements in a Peirce algebra can be modelled as either right ideal elements or identity elements in the underlying relation algebra. In a Peirce algebra $(\mathfrak{B}, \mathfrak{R}, :, ^c)$, the Boolean algebra of right ideal elements in the underlying relation algebra $\mathfrak{R}$ and the Boolean algebra of identity elements is isomorphic to the Boolean algebra $\mathfrak{B}$ underlying the Peirce algebra [2]. Peirce algebras therefore inherit various properties of relation algebras. For example, it follows that the class of Peirce algebras is not

representable. This is a consequence of a well-known result for relation algebras by Lyndon [12]. Consequently, there are properties of binary relations that cannot be proved in the framework of Peirce algebra. Monk [15] has proved that the class of representable relation algebras is not finitely axiomatisable by a set of equational axioms. This means there is no finite equational proof system for reasoning about the (equational) properties of relations. From an applications perspective this is a disadvantage. Therefore, in order to express and derive every property of sets and binary relations we restrict our attention to the reasoning problem of the elementary theory of Peirce algebra as formalised in Peirce logic which is defined next. That is, Peirce logic is intended to be the logic of the class of representable Peirce algebras.

3 Peirce logic

There are different ways of defining Peirce logic. One is just as first-order logic over one-place or two-place predicates augmented with the operations of Peirce algebra, cf. Nellas [16]. Our definition is similar in style to the logical formalisation of Peirce algebras given by de Rijke [5].

The language $\mathcal{L}$ of Peirce logic consists of two syntactic types: (i) countably many Boolean symbols, called *atomic Boolean formulae* and denoted by A_i, and (ii) countably many relational symbols, called *atomic relational formulae* and denoted by R_i. The logical connectives are the classical connectives, negation, intersection and falsum (or bottom), here denoted by $-$, $\cap$ and 0, respectively, the standard connectives of relational logics, $;$ (composition), $^\smile$ (converse), Id (identity), the logical version of Peirce product $:$ and left cylindrification ^{c}.

An *atomic formula* defined over the language $\mathcal{L}$ is any atomic Boolean or relational formula in $\mathcal{L}$. The set of *Boolean formulae* over $\mathcal{L}$ is very similar to Boolean terms in Peirce algebras, similarly for *relational formulae*. In particular, the Boolean and relational formulae are defined inductively by the following BNF production rules.

$$\text{Boolean formulae:} \quad A, B \longrightarrow A_i \mid 0 \mid -A \mid A \cap B \mid R : A$$

$$\text{Relational formulae:} \quad R, S \longrightarrow R_i \mid 0 \mid -R \mid R \cap S \mid R ; S \mid R^\smile \mid Id \mid A^c$$

The symbols 0 and Id are nullary connectives which are interpreted as the empty set (or relation) and the identity relation, respectively. The set of *formulae* of Peirce logic is the smallest set of Boolean and relational formulae defined over $\mathcal{L}$. We assume two defined connectives: *Peirce sum* $R \ddagger A = -((-R) : (-A))$ and *relational sum* $R \dagger S = -((-R) ; (-S))$.

We now define the semantics of Peirce logic. A *model* for Peirce logic is a system of the form $M = (U, m)$, where U is a non-empty set, and m is a meaning function subject to the following conditions:

1. If A is a Boolean symbol then $m(A) \subseteq U$ and m extends to all the Boolean formulae as follows, where A and B are arbitrary Boolean formulae over $\mathcal{L}$.

$$m(0) = \emptyset \qquad m(-A) = U \setminus m(A) \qquad m(A \cap B) = m(A) \cap m(B)$$

2. If R is a relational symbol then $m(R) \subseteq U \times U$ and m extends to all the relational formulae as follows, for all relational formulae R and S over $\mathcal{L}$.

$$\begin{array}{lll} m(0) = \emptyset & m(Id) = Id_U & m(R \cap S) = m(R) \cap m(S) \\ m(-R) = U \times U \setminus m(R) & m(R^{\smile}) = m(R)^{\smile} & m(R\,;S) = m(R)\,;m(S) \end{array}$$

3. If A is a Boolean formula and R a relational formula then

$$\begin{array}{l} m(R\!:\!A) = \{x \in U \mid \text{there is a } y \in m(A) \text{ such that } (a,b) \in m(R)\} \\ m(A^c) = \{(x,y) \in U \times U \mid x \in m(A)\}. \end{array}$$

A formula F in $\mathcal{L}$ is said to be *satisfiable* in a model M iff one of the following is true: (i) there is an element s in U such that $s \in m(F)$, if F is a Boolean formula, and (ii) there is a pair of elements s and t in U such that $(s,t) \in m(F)$, if F is a relational formula. In these cases we write $M, s \models F$ or $M, (s,t) \models F$, respectively. A formula F in $\mathcal{L}$ is *valid* in a model M iff F is satisfiable with respect to arbitrary elements in U or pairs of elements in U. In this case we write $M \models F$. Observe that if F is valid in M then $m(F) = U$, if F is a Boolean formula, and $m(F) = U \times U$, if F is a relational formula. A Peirce logic formula is said to be *satisfiable* if there is a model M in which F is satisfiable. A Peirce logic formula F is said to be *valid*, and we write $\models F$, if it is valid in all models of Peirce logic. Let Γ be a set of Peirce logic formulae. A formula F is *semantically entailed* by Γ, written $\Gamma \models F$, iff for all models M, whenever $M \models G$ for all $G \in \Gamma$ then $M \models F$. In Peirce logic semantic entailment can be reduced to validity:

Lemma 1. *Let $\Gamma \cup \{F\}$ be a set of Peirce logic formulae. Suppose Γ is partitioned into two sets Γ_b and Γ_r of the Boolean and relational formulae in Γ, respectively. Then*

$$\begin{array}{lll} \Gamma \models F & \textit{iff} & -((1\ddagger \cap \Gamma_b) \cap (1\dagger \cap \Gamma_r \dagger 1) \cap -F) \textit{ is valid} \\ & \textit{iff} & (1\ddagger \cap \Gamma_b) \cap (1\dagger \cap \Gamma_r \dagger 1) \cap -F \textit{ is unsatisfiable.} \end{array}$$

The proof systems we are going to describe for proving or refuting formulae of Peirce logic manipulate labelled formulae defined over two extended languages. One is the language tailored for refutation proofs using tableau and the other is tailored for proofs of validity in the style Rasiowa-Sikorski. In the tableau system the labels are constants and in the Rasiowa-Sikorski system the labels are variables. Therefore, let the tableau language $\mathcal{L}^{\mathsf{T}}$ be an extension of the language $\mathcal{L}$ with a countable set Con of individual constants, denoted by a_i, and the connective $\bot$. Further, let the Rasiowa-Sikorski language $\mathcal{L}^{\mathsf{RS}}$ be an extension of $\mathcal{L}$ with a countable set Var of individual variables, denoted by x_i, and the connective $\top$. The set of formulae over $\mathcal{L}^{\mathsf{T}}$, respectively over $\mathcal{L}^{\mathsf{RS}}$, is defined by

$$\begin{array}{lll} \text{Formulae over } \mathcal{L}^{\mathsf{T}}\text{:} & \psi & \longrightarrow \quad \bot \mid a\,A \mid a\,R\,b \\ \text{Formulae over } \mathcal{L}^{\mathsf{RS}}\text{:} & \psi & \longrightarrow \quad \top \mid x\,A \mid x\,R\,y \end{array}$$

where a and b denote individual constants in Con, and x and y denote individual variables in Var. Subsequently, we use the notation s and t for either both constants or both variables, which will be clear from the context. An *atomic (labelled) formula* over $\mathcal{L}^{\mathsf{T}}$ or $\mathcal{L}^{\mathsf{RS}}$ is a formula of the form $s\,A$ or $s\,R\,t$ are either both individual constants of variables, in which the formula A or R is primitive, i.e. is a Boolean or relational symbols or a nullary connective (0 or Id).

Now we define the semantics of formulae over the extended languages $\mathcal{L}^{\mathsf{T}}$ and $\mathcal{L}^{\mathsf{RS}}$. Assume $M = (U, m)$ is a model defined as above with the meaning function m extended so that it provides also an interpretation of individual constants, that is, for each a in Con, $m(a) \in U$. A *valuation* in M is a mapping v from the set of variables and constants of the language to U such that if a is a constant then $v(a) = m(a)$. The *satisfiability* of formulae in a model $M = (U, m)$ by a valuation v in M is defined by (s and t either both denote constants or variables):

$$
\begin{array}{ll}
M, v \models \top & M, v \not\models \bot \\
M, v \models s\,A & \text{iff} \quad v(s) \in m(A) \text{ for any Boolean formula } A \\
M, v \models s\,R\,t & \text{iff} \quad (v(s), v(t)) \in m(R) \text{ for any relational formula } R.
\end{array}
$$

Let ψ be any formula over $\mathcal{L}^{\mathsf{T}}$ (or $\mathcal{L}^{\mathsf{RS}}$). ψ is *satisfiable* whenever there is a model M and a valuation v in M such that $M, v \models \psi$. ψ is said to be *unsatisfiable* whenever it is not satisfiable. Validity of a formula in a model is defined by: A formula ψ is *valid* in a model M whenever $M, v \models \psi$ for every v in M. A formula is *valid* whenever it is true in all the models defined over the extended languages.

Lemma 2. *Let x and y be variables in* Var*, and let a and b be constants in* Con*. (i) If F is a Boolean formula of Peirce logic then: F is valid iff $x\,F$ is valid iff $\neg F$ is unsatisfiable iff $a\,F$ is unsatisfiable. (ii) If F is a relational formula of Peirce logic then: F is valid iff $x\,F\,y$ is valid iff $\neg F$ is unsatisfiable iff $a\,F\,b$ is unsatisfiable.*

In the definition of the inference rules we will be using the symbol $\sim$ which is defined as follows: If F denotes a Boolean or relational formula then $\sim F$ denotes G, if $F = \neg G$, and $\neg F$ otherwise.

4 A tableau refutation system

A tableau is a finitely branching tree whose nodes are sets of formulae. Given a formula F of Peirce logic to be tested for satisfiability the root node is the set $\{a\,F\}$, when F is a Boolean formula and $\{a\,F\,b\}$, when F is a relational formula. Successor nodes are constructed in accordance with a set of expansion rules. An expansion rule has the form (1), where X, X_i are sets of formulae over $\mathcal{L}^{\mathsf{T}}$ ($1 \leq i \leq n$). The formulae in X are called premises and the formulae in X_i are called conclusions.

Decomposition rules:

$$(\cap)\ \frac{a\,A\cap B}{a\,A,\ a\,B} \qquad (-\cap)\ \frac{a\,-(A\cap B)}{a\sim A\ \mid\ a\sim B} \qquad (--)\ \frac{a\,--A}{a\,A}$$

$$(:)\ \frac{a\,R:A}{a\,R\,c,\ c\,A}\ \text{where } c \text{ is a new constant}$$

$$(-:)\ \frac{a\,-(R:A)}{a\sim R\,c\ \mid\ c\sim A}\ \text{where } c \text{ is any constant}$$

$$(\cap)\ \frac{a\,R\cap S\,b}{a\,R\,b,\ a\,S\,b} \qquad (-\cap)\ \frac{a\,-(R\cap S)\,b}{a\sim R\,b\ \mid\ a\sim S\,b} \qquad (--)\ \frac{a\,--R\,b}{a\,R\,b}$$

$$(^\smile)\ \frac{a\,R^\smile\,b}{b\,R\,a} \qquad (-^\smile)\ \frac{a\,-(R^\smile)\,b}{b\sim R\,a}$$

$$(^c)\ \frac{a\,A^c\,b}{a\,A} \qquad (-^c)\ \frac{a\,-(A^c)\,b}{a\sim A}$$

$$(;)\ \frac{a\,R;S\,b}{a\,R\,c,\ c\,S\,b}\ \text{where } c \text{ is a new constant}$$

$$(-;)\ \frac{a\,-(R;S)\,b}{a\sim R\,c\ \mid\ c\sim S\,b}\ \text{where } c \text{ is any constant}$$

Specific rules:

$$(\text{sym})\ \frac{a\,Id\,b}{b\,Id\,a}$$

$$(\mathrm{id}_1)\ \frac{b\,A,\ a\,Id\,b}{a\,A} \qquad (\mathrm{id}_2)\ \frac{b\,R\,c,\ a\,Id\,b}{a\,R\,c} \qquad (\mathrm{id}_3)\ \frac{c\,R\,a,\ a\,Id\,b}{c\,R\,b}$$

Closure rules:

$$(\mathrm{cl}_1)\ \frac{a\,A,\ a\,-A}{\bot} \qquad (\mathrm{cl}_2)\ \frac{a\,0}{\bot}$$

$$(\mathrm{cl}_3)\ \frac{a\,R\,b,\ a\,-R\,b}{\bot} \qquad (\mathrm{cl}_4)\ \frac{a\,0\,b}{\bot} \qquad (\text{refl})\ \frac{a\,-Id\,a}{\bot}$$

Fig. 1. Tableau rules for Peirce logic.

Let T be the *calculus for Peirce logic* defined by the rules of Figure 1. The specific rules express properties of the identity relation. (sym) expresses the symmetry of Id and (id_1)–(id_3) express that $Id : A \subseteq A$, $R\,;Id \subseteq R$ and $Id\,;R \subseteq R$. Reflexivity of Id is ensured by the reflexivity rule (refl), classified here as a closure rule. The other closure rules reduce elementary contradictions to $\bot$. (Observe that the transitivity rule for identity is redundant in T because it is an instance of both identity rules (id_2) and (id_3).)

Concerning the rules for negated main connectives, consider for example the rule $(-\cap)$. By the use of $\sim$ (defined above), we have chosen to eliminate immediately the double negations normally introduced for a formula like

$a-(-A \cap B)$, where one of the conjuncts is a negated formula, had we used the rule $a-(A \cap B)\,/\,a-A \,|\, a-B$ instead. This does not make the $(--)$ rules superfluous however.

A *tableau derivation* from a set N of formulae over $\mathcal{L}^{\mathsf{T}}$ is a finitely branching, ordered tree T with root N and nodes which are sets of $\mathcal{L}^{\mathsf{T}}$-formulae. The tree is constructed by applications of the expansion rules to the leaves. Let N be a leaf node in a (partially constructed) tableau derivation. A rule (1) is *applicable* to N, if N contains formulae in the form X. Then an application of the rule creates a new tree T' which is the same as T except that the node N has n successor nodes N_i which are extensions of N with the formulae in X_i. That is, $N_i = N \cup X_i$ $(1 \leq i \leq n)$. It is assumed that on a branch in any tableau derivation no instance of a rule is applied twice to the same instance of the numerator.

For each rule application to a node N if the following is true, then the rule is said to be *(satisfiability) admissible*.

$$\exists \overline{a}\,(\bigwedge X \wedge \bigwedge N) \text{ is satisfiable} \quad \text{iff} \quad \bigvee_i \exists \overline{a}\,(\bigwedge X_i \wedge \bigwedge N) \text{ is satisfiable,}$$

where $\overline{a}$ denotes the sequence of constants occurring in the corresponding matrix.

Lemma 3. *Each rule in* T *is (satisfiability) admissible.*

That is, in the tableau system (as usual), sets of formulae are interpreted conjunctively and the vertical bar is interpreted disjunctively.

Any path $N_0, N_1, \ldots$ in a derivation T, where N_0 denotes the root node of T, is called a *closed branch* in T iff the set $\bigcup_{j\geq 0} N_j$ contains $\bot$ (a contradiction has occurred), otherwise it is called an *open branch*. We call a branch B in a derivation tree *complete* (with respect to T) iff no new successor nodes can be added to the endpoint of B by T, otherwise it is called an *incomplete branch*. A derivation T is *closed* iff every path $N(= N_0), N_1, \ldots$ in it is a closed branch, otherwise it is called an *open derivation*. A closed derivation tree is also called a *refutation (tree)*.

A derivation T from N is called *fair* iff for any path $N(= N_0), N_1, \ldots$ in T, with *limit* $N_\infty = \bigcup_{j\geq 0} N_j$, it is the case that each formula ψ which can be deduced from premises in N_∞ is contained in some N_j. Intuitively, fairness means that no possible application of an inference rule is delayed indefinitely. It also means that the γ rules, i.e. the rules $(-:)$ and $(-;)$, are applied infinitely often. For a finite complete branch $N(= N_0), N_1, \ldots N_n$, the limit N_∞ is equal to N_n.

Theorem 1 (Soundness and completeness of tableau). *Let T be a fair T derivation from a set N of formulae in $\mathcal{L}^{\mathsf{T}}$. Then: (i) If $N(= N_0), N_1, \ldots$ is a path with limit N_∞, then N_∞ is closed under the rules of T. (ii) N is satisfiable iff there exists a path in T with limit N_∞ such that N_∞ is satisfiable. (iii) N is unsatisfiable iff for every path $N(= N_0), N_1, \ldots$ the limit N_∞ contains $\bot$.*

This result follows immediately from the corresponding result for ground tableau of first-order logic, cf. Fitting [7] for a cut-free tableau calculus and a completeness proof. The reason is that the rules of T mirror the rules of the first-order logic

ground tableau calculus (cf. Nellas [16]). By ground first-order logic tableau we mean a Smullyan-style tableau calculus [23], as opposed to free-variable tableau.

Corollary 1. *A Peirce logic formula is unsatisfiable iff the rules of* T *can be used to construct a closed tableau.*

The decomposition rules very clearly reflect the semantics of the top most connective in the premises. Because the decomposition rules are based on semantic equivalences, the following is immediate.

Lemma 4. *The decomposition rules of* T *are invertible.*

Recall, a rule of the form (1) is *invertible*, if the following is satisfied: there is a closed derivation for X iff there are closed derivations for each X_i $(1 \leq i \leq n)$.

There are alternative ways of capturing the properties of the identity relation in the calculus. In the presence of (sym), the following rule combines the rules (id_1)–(id_3) and can be used instead.

$$(Id)\ \frac{\psi,\ a\,Id\,b}{\psi[b]_\lambda}\ \text{ if } \psi|_\lambda = a$$

This rule corresponds to the familiar substitution axiom of equality in sentence tableau for first-order logic [7]. If the formula $a\,Id\,b$ is in a leaf node then the substitution rule generates the formula ψ, in which the occurrence of the constant a at position λ is replaced by b.

5 A Rasiowa-Sikorski proof system

Now we turn to a different style of proof system. Rasiowa-Sikorski proof systems aim to prove validity. Given a candidate formula F they aim to prove its validity or, if it is not valid, the aim is to construct a counter-model (i.e. a model for the complement of the candidate formula). Starting with $\{x\,F\}$ (or $\{x\,F\,y\}$), this is done by systematic case analysis until fundamental validities are found. Rasiowa-Sikorski expansion rules have the same form (1) as for tableau and are also applied top-down. The definition of a Rasiowa-Sikorski derivations, and its construction by application of rules, is the same as a tableau derivation with the difference that the language is $\mathcal{L}^{\mathsf{RS}}$ instead of $\mathcal{L}^{\mathsf{T}}$. Crucially the interpretation of the rules is different. As above, X, X_i denote sets of formulae, but different from above sets of formulae are interpreted as disjunctions of formulae, whereas branching is interpreted conjunctively. A rule is *(validity) admissible*, if for any application of the rule to a node N,

$$\forall\overline{x}\,(\bigvee X \vee \bigvee N) \text{ is valid} \quad \text{iff} \quad \bigwedge_i \forall\overline{x}\,(\bigvee X_i \vee \bigvee N) \text{ is valid},$$

where $\overline{x}$ is the sequence of variables occurring in the corresponding matrix.

Any path $N_0, N_1, \ldots$ in a Rasiowa-Sikorski derivation T, where N_0 denotes the root node of T, is called a *closed branch* in T iff the set $\bigcup_{j\geq 0} N_j$ contains $\top$

(an axiomatic set was found), otherwise it is called an *open branch*. A Rasiowa-Sikorski derivation T is *closed* iff every path from the root in it is a closed branch, otherwise it is called an *open derivation*. A closed Rasiowa-Sikorski derivation is also called a *proof (tree)*. The concepts of *(in)complete branches*, *fairness* and *invertible rules* are the same as for tableau.

Let RS be the Rasiowa-Sikorski calculus for Peirce logic defined by the rules of Figure 2. As for tableau we distinguish between three kinds of deduction rules: decomposition rules, specific rules for identity and closure rules. The premises of the closure rules are commonly referred to as *axiomatic sets*.

Lemma 5. *Each rule in* RS *is (validity) admissible.*

Lemma 6. *Each decomposition rule in* RS *is invertible.*

Theorem 2 (Soundness and completeness of Rasiowa-Sikorski). *Let T be a fair* RS *derivation from a set N of formulae in $\mathcal{L}^{\mathsf{RS}}$. Then: (i) If $N(= N_0), N_1, \ldots$ is a path with limit N_∞, then N_∞ is closed under the rules of* RS*. (ii) N is valid iff there exists a path in T with limit N_∞ such that N_∞ is valid. (iii) N is valid iff for every path $N(= N_0), N_1, \ldots$ the limit N_∞ contains $\top$.*

Corollary 2. *A Peirce logic formula is valid iff the rules of* RS *can be used to construct a closed derivation tree.*

We conclude this section with remarks relating our presentation to presentations of Rasiowa-Sikorski systems usually found in the literature. We assume the rules are extension rules similar as for tableau which ignore the issue of repetition by assuming all main premises are retained during an inference step. Another difference is that we use sets instead of sequences of formulae. These differences are logically insignificant, however, and largely a matter of taste, although when developing an implementation of the calculus, the differences will need to be taken into account. Our presentation was chosen for reasons of uniformity.

6 Duality

The two systems presented are clearly dual to each other. This section is a formal discussion of this relationship between T and RS.

Suppose $\mathcal{R}_1$ and $\mathcal{R}_2$ are two expansion rules of the form (1). If $\mathcal{R}_2$ is obtained from $\mathcal{R}_1$ (or vice versa) by interchanging the logical connectives and symbols in accordance with the tables in Figure 3, then $\mathcal{R}_2$ is the *dual rule* to $\mathcal{R}_1$. I.e. all occurrences of $F_1 \cap F_2$ are replaced with $-(F_1 \cap F_2)$, all occurrences of $-(F_1 \cap F_2)$ are replaced with $F_1 \cap F_2$, etc. Notice we assume that $\wedge$ and $\vee$ refer to meta-level conjunction and disjunction, i.e. ',' and '|' for tableau and '|' and ',' for Rasiowa-Sikorski. Thus although the form of dual rules is the same the meta-level interpretation is interchanged.

Lemma 7. *The pair of rules in* T *and* RS *in each column of the table in Figure 4 are dual rules.*

Decomposition rules:

$$(\cap)\ \frac{x\,A\cap B}{x\,A\ \mid\ x\,B} \qquad (-\cap)\ \frac{x\,-(A\cap B)}{x\sim A,\ x\sim B} \qquad (--)\ \frac{x\,--\,A}{x\,A}$$

$$(:)\ \frac{x\,R{:}\,A}{x\,R\,z\ \mid\ z\,A}\ \text{where } z \text{ is any variable}$$

$$(-:)\ \frac{x\,-(R:A)}{x\sim R\,z,\ z\sim A}\ \text{where } z \text{ is a new variable}$$

$$(\cap)\ \frac{x\,R\cap S\,y}{x\,R\,y\ \mid\ x\,S\,y} \qquad (-\cap)\ \frac{x\,-(R\cap S)\,y}{x\sim R\,y,\ x\sim S\,y} \qquad (--)\ \frac{x\,--\,R\,y}{x\,R\,y}$$

$$(^\smile)\ \frac{x\,R^\smile\,y}{y\,R\,x} \qquad (-^\smile)\ \frac{x\,-(R^\smile)\,y}{y\sim R\,x}$$

$$(^c)\ \frac{x\,A^c\,y}{x\,A} \qquad (-^c)\ \frac{x\,-(A^c)\,y}{x\sim A}$$

$$(;)\ \frac{x\,R\,;S\,y}{x\,R\,z\ \mid\ z\,S\,y}\ \text{where } z \text{ is any variable}$$

$$(-;)\ \frac{x\,-(R\,;S)\,y}{x\sim R\,z,\ z\sim S\,y}\ \text{where } z \text{ is new variable}$$

Specific rules:

$$(\text{sym})\ \frac{x\,-Id\,y}{y\,-Id\,x}$$

$$(\text{id}_1)\ \frac{y\,-A,\ x\,-Id\,y}{x\,-A} \qquad (\text{id}_2)\ \frac{y\,-R\,z,\ x\,-Id\,y}{x\,-R\,z} \qquad (\text{id}_3)\ \frac{z\,-R\,x,\ x\,-Id\,y}{z\,-R\,y}$$

Closure rules:

$$(\text{cl}_1)\ \frac{x\,A,\ x\,-A}{\top} \qquad (\text{cl}_2)\ \frac{x\,-0}{\top}$$

$$(\text{cl}_3)\ \frac{x\,R\,y,\ x\,-R\,y}{\top} \qquad (\text{cl}_4)\ \frac{x\,-0\,y}{\top} \qquad (\text{refl})\ \frac{x\,Id\,x}{\top}$$

Fig. 2. Rasiowa-Sikorski rules for Peirce logic.

$\cap$	$-\cap$	$:$	$-:$	$^\smile$	$-^\smile$	^{c}	$-^c$	$;$	$-;$	Id	$-Id$	A	$-A$	R	$-R$	0	-0
$-\cap$	$\cap$	$-:$	$:$	$-^\smile$	$^\smile$	$-^c$	^{c}	$-;$	$;$	$-Id$	Id	$-A$	A	$-R$	R	-0	0

$\bot$	$\top$	x, y, z	a, b, c
$\top$	$\bot$	a, b, c	x, y, z

$\wedge$	$\vee$
$\vee$	$\wedge$

Fig. 3. Dual connectives and symbols

T	$\cap$	$-\cap$	$:$	$-:$	$^\smile$	$-^\smile$	^{c}	$-^c$	$;$	$-;$	sym	id_i	cl_j	refl
RS	$-\cap$	$\cap$	$-:$	$:$	$-^\smile$	$^\smile$	$-^c$	^{c}	$-;$	$;$	sym	id_i	cl_j	refl

Fig. 4. Dual rules ($1 \leq i \leq 3$, $1 \leq j \leq 4$)

Note the double negation rules are the only rules which do not appear in Figure 4.

Lemma 8. *Let $\mathcal{R}$ be any satisfiability admissible rule in T. Then the dual of $\mathcal{R}$ is a validity admissible rule in RS. Let $\mathcal{R}$ be any validity admissible rule in RS. Then the dual of $\mathcal{R}$ is a satisfiability admissible rule in T.*

Theorem 3. *Let F be a Peirce logic formula. Then, starting with $x \sim F$ (or $x \sim F\, y$), every inference step $\mathcal{I}$ (i.e. every rule application) in a T derivation for $a\, F$ (or $a\, F\, b$) can be mimicked in RS by $\mathcal{I}$ itself, when $\mathcal{I}$ involves the application of $(--)$, or it can be mimicked by the application of the dual rule. Similarly, every RS inference step from $x\, F$ (or $x\, F\, y$) can be mimicked by a corresponding inference step in T starting from $a \sim F$ (or $a \sim F\, b$).*

It follows that the systems T and RS step-wise simulate each other in a dual sense. They also p-simulate each other both with respect to derivations and search in a dual sense. See Schmidt and Hustadt [21, §8] for definitions of the notions of step-wise simulation and p-simulation.

It also follows that any prover for one of the systems can be used as a prover for the other system; users only need to keep in mind the dual interpretation of the formulae and rules. Clearly, optimisations compatible with one system will also be compatible in the dual form with the other system. For example, the tableau system admits that the γ rules can be restricted to constants occurring on the current branch (and means that γ rules are not necessarily applied infinitely often on a branch). This property carries over from ground tableau for first-order logic with equality, cf. [7]. By duality this signature restriction of the γ rules is compatible with RS.

7 Concluding remarks

We have presented two proof systems for Peirce logic. Both are tableaux-style proof systems, with the difference that one is a refutation calculus and the other is a calculus for proving validities of relations and sets. It is not difficult to see that the duality between tableau and Rasiowa-Sikorski proof systems generalises quite naturally to other logics, especially first-order logic.

An implementation of the tableau calculus for Peirce logic was developed by Nellas [16]. By the duality result shown in this paper it can also be used as a prover for the Rasiowa-Sikorski calculus for Peirce logic.

Can the presented calculi be used to prove validities of Peirce algebra? We know that Maddux's sequent calculus of relational logic can be used to prove validities in relation algebra [14]. Maddux proved that an equation about relations is true in every relation algebra iff its three-variable translation has a four-variable proof in first-order logic. Because of the known connection between sequent calculi and tableau calculi, we expect this result to carry over to Peirce algebra and proofs or refutations constructed by the systems presented in this paper. This would provide a method to prove validities in Peirce algebra by considering the validity, or satisfiability, of an equation (represented as a suitable Peirce logic formula) and the proof, or refutation, of it in one of our systems. If the proof of a Peirce logic formula corresponding to a validity in Peirce algebra uses at most four variables then the equation would be valid in every Peirce algebra. (Dually for the refutation of a Peirce logic formula.)

References

1. C. Brink. Boolean modules. *J. Algebra*, 71(2):291–313, 1981.
2. C. Brink, K. Britz, and R. A. Schmidt. Peirce algebras. *Formal Aspects of Computing*, 6(3):339–358, 1994.
3. K. Britz. Relations and programs. Master's thesis, Univ. Stellenbosch, South Africa, 1988.
4. J. Dawson and R. Goré. A mechanised proof system for relation algebra using display logic. In *Proc. JELIA'98*, vol. 1489 of *LNAI*, pp. 264–278. Springer, 1998.
5. M. de Rijke. *Extending Modal Logic*. PhD thesis, Univ. Amsterdam, 1993.
6. I. Düntsch and E. Orlowska. A proof system for contact relation algebras. *J. Philos. Logic*, 29:241–262, 2000.
7. M. Fitting. *First-Order Logic and Automated Theorem Proving*. Texts and Monographs in Computer Science. Springer, 1990.
8. G. Gargov and S. Passy. A note on Boolean modal logic. In P. P. Petkov, editor, *Mathematical Logic: Proc. 1988 Heyting Summerschool*, pp. 299–309, New York, 1990. Plenum Press.
9. R. Goré. Cut-free display calculi for relation algebras. In *Selected Papers of CSL'96*, vol. 1258 of *LNCS*, pp. 198–210. Springer, 1996.
10. M. C. B. Hennessy. A proof-system for the first-order relational calculus. *J. Computer and System Sci.*, 20:96–110, 1980.
11. U. Hustadt and R. A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In R. Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, vol. 1847 of *LNAI*, pp. 67–71. Springer, 2000.

12. R. C. Lyndon. The representation of relational algebras. *Ann. Math.*, 51:707–729, 1950.
13. W. MacCaull and E. Orlowska. Correspondence results for relational proof systems with applications to the Lambek calculus. *Studia Logica*, 71:279–304, 2002.
14. R. D. Maddux. A sequent calculus for relation algebras. *Ann. Pure Applied Logic*, 25:73–101, 1983.
15. J. D. Monk. On representable relation algebras. *Michigan Math. J.*, 11:207–210, 1964.
16. K. Nellas. Reasoning about sets and relations: A tableaux-based automated theorem prover for Peirce logic. Master's thesis, Univ. Manchester, UK, 2001.
17. E. Orlowska. Relational formalisation of nonclassical logics. In C. Brink, W. Kahl, and G. Schmidt, editors, *Relational Methods in Computer Science*, Advances in Computing, pp. 90–105. Springer, Wien, 1997.
18. H. Rasiowa and R. Sikorski. *The Mathematics of Metamathematics*. Polish Scientific Publ., Warsaw, 1963.
19. R. A. Schmidt. Algebraic terminological representation. Master's thesis, Univ. Cape Town, South Africa, 1991. Available as Technical Report MPI-I-91-216, Max-Planck-Institut für Informatik, Saarbrücken, Germany.
20. R. A. Schmidt. Relational grammars for knowledge representation. In M. Böttner and W. Thümmel, editors, *Variable-Free Semantics*, vol. 3 of *Artikulation und Sprache*, pp. 162–180. Secolo Verlag, Osnabrück, Germany, 2000.
21. R. A. Schmidt and U. Hustadt. Mechanised reasoning and model generation for extended modal logics. In *Theory and Applications of Relational Structures as Knowledge Instruments*, pp. 38–67. Springer, 2003. To appear.
22. W. Schönfeld. Upper bounds for a proof-search in a sequent calculus for relational equations. *Z. Math. Logik Grundlagen Math.*, 28:239–246, 1982.
23. R. M. Smullyan. *First Order Logic*. Springer, Berlin, 1971.
24. A. Tarski. On the calculus of relations. *J. Symbolic Logic*, 6(3):73–89, 1941.
25. W. W. Wadge. A complete natural deduction system for the relational calculus. Theory of Computation Report 5, Univ. Warwick, 1975.