Journal of Universal Computer Science, vol. 7, no. 2 (2001), 107-123
submitted: 1/9/00, accepted: 13/10/00, appeared: 28/2/01 © *Springer Pub. Co.*

Modeling Sequences within the RelView System

Rudolf Berghammer
(Christian-Albrechts-Universität Kiel, Germany
rub@informatik.uni-kiel.de)

Thorsten Hoffmann
(Christian-Albrechts-Universität Kiel, Germany
tho@informatik.uni-kiel.de)

Abstract: We use a relational characterization of binary direct sums to model sequences within the relation-algebraic manipulation and prototyping system RelView in a simple way. As an application we formally derive a RelView program for computing equivalence classes of an equivalence relation, where we combine relation-algebraic calculations with the so-called Dijkstra-Gries program development method. Also a refinement of the simple modeling is presented, which leads to the classical datatype of stacks, and a further application is sketched.

Key Words: Relational algebra, relational modelling and programming, formal program derivation, equivalence classes, RelView system

Category: D.1.4 — Sequential Programming, D.2.4 — Program Verification, D.2.2 — Tools and Techniques, G.2.2 — Graph Theory

1 Introduction

For many years, Tarski's axiomatic relational algebra (cf. [Tar41, ChT51]) has been used very successfully for formal problem specifications and program derivations. Relations are well suited for modeling and reasoning about many discrete structures and computations on them. This holds in particular for those graph algorithms which manipulate sets of arcs or vertices since sets of arcs and relations are essentially the same and there are many simple and elegant ways to model sets of vertices by specific relations like vectors, partial identities, and injective embedding mappings. For details see [ScS93].

Sets are not the only datatype used by graph algorithms. Sequences are also very important since many specifications and algorithmic solutions of fundamental graph-theoretic problems require different types of them. For instance, to calculate a path – a task which is part of many graph algorithms – means to compute a sequence of vertices and to enumerate the strongly connected components means to compute a sequence of sets of vertices. We claim that in many cases relations can also be used for modeling (finite) sequences in a way which is well suited for calculations and formal program derivations. To prove our point we present in this paper a simple relation-algebraic model for sequences via binary direct sums which especially works for the relation-algebraic manipulation and prototyping system RelView and show some typical applications.

The paper is organized as follows: To make it self-contained, in Section 2 we introduce the relation-algebraic preliminaries which will be used in its remainder and in Section 3 we give a short overview of RelView. Then, in Section 4 we present a relation-algebraic characterization of binary direct sums and use it to model non-empty sequences of sets or elements with a concatenation operation within RelView in a very simple way. This section also contains some algebraic laws of concatenation. They are used in Section 5, which contains an application of our approach, viz. a formal derivation of a RelView program for computing equivalence classes. A refinement of the simple approach of Section 4 to obtain the classical datatype of stacks is presented in Section 6 and in this section also a further application is sketched. The last section contains some concluding remarks.

2 Relation-algebraic Preliminaries

In this section we collect some basic concepts of relational algebra; for more details see [ScS93, BKS97], for example. We denote the set (or type) of all (binary) relations with domain X and range Y by $[X \leftrightarrow Y]$ and write $R : X \leftrightarrow Y$ instead of $R \in [X \leftrightarrow Y]$. If the carrier sets X and Y of a relation $R : X \leftrightarrow Y$ are finite and of cardinality m resp. n, then we may consider R as a Boolean matrix with m rows and n columns. Since this Boolean matrix interpretation is well suited for many purposes and also used within the RelView system, in the following we often use matrix terminology and matrix notation. The latter means that we write R_{xy} instead of $(x,y) \in R$.

We assume the reader to be familiar with the basic operations on relations, viz. R^{T} (transposition), $\overline{R}$ (negation, complement), $R \cup S$ (join, union), $R \cap S$ (meet, intersection), $R \cdot S$ (composition, multiplication; in this paper abbreviated as RS), $R \subseteq S$ (inclusion), and the special relations O (empty relation), L (universal relation), and I (identity relation). The set-theoretic operations $\overline{}$, $\cup$, $\cap$, the ordering $\subseteq$, and the constants O and L form a complete Boolean lattice. Some further well-known rules of relations are, for instance,

$$\begin{array}{ll} R^{\mathsf{T}^{\mathsf{T}}} = R & R \subseteq S \Longrightarrow R^{\mathsf{T}} \subseteq S^{\mathsf{T}} \\ (RS)^{\mathsf{T}} = S^{\mathsf{T}} R^{\mathsf{T}} & \overline{R^{\mathsf{T}}} = \overline{R}^{\mathsf{T}} \\ R \subseteq S \Longrightarrow QR \subseteq QS & R \subseteq S \Longrightarrow RQ \subseteq SQ \\ Q(R \cap S) \subseteq QR \cap QS & Q(R \cup S) = QR \cup QS \\ (R \cap S)^{\mathsf{T}} = R^{\mathsf{T}} \cap S^{\mathsf{T}} & (R \cup S)^{\mathsf{T}} = R^{\mathsf{T}} \cup S^{\mathsf{T}} . \end{array} \tag{1}$$

The theoretical framework for all rules of (1) and many others to hold is that of a relational algebra. As constants and operations of this abstract algebraic structure we have those of the set-theoretic relations. The axioms of a relational algebra are the axioms of a complete Boolean lattice for negation, join, meet, ordering, empty and universal relation, the axioms of a monoid for composition and identity relation, the so-called *Schröder equivalences*

$$QR \subseteq S \iff Q^{\mathsf{T}}\,\overline{S} \subseteq \overline{R} \iff \overline{S}\,R^{\mathsf{T}} \subseteq \overline{Q}\,, \tag{2}$$

and the so-called *Tarski rule*

$$R \neq \mathsf{O} \iff \mathsf{L}R\mathsf{L} = \mathsf{L}\,. \tag{3}$$

An immediate consequence of (3) is $\mathsf{O} \neq \mathsf{L}$ which in turn implies that domain and range of each relation are non-empty. This agrees exactly with the use of relations in practice.

The above basic operations and constants are very helpful for defining properties of relations in an algebraic way; see [ScS93] for example. In this paper we are concerned with the following specific classes of relations.

A relation R is said to be *reflexive* if $\mathsf{I} \subseteq R$, *transitive* if $RR \subseteq R$, and *symmetric* if $R \subseteq R^{\mathsf{T}}$. By an *equivalence relation* we mean a reflexive, transitive, and symmetric relation. For all these types of relations, domain and range coincide, i.e., they are *homogeneous*. In the Boolean matrix model of relations a homogeneous relation is quadratic.

An arbitrary (also called *heterogeneous*) relation R is said to be *univalent* (or *functional*) if $R^{\mathsf{T}}R \subseteq \mathsf{I}$ and *total* if $R\mathsf{L} = \mathsf{L}$. As usual, an univalent and total relation is said to be a (total) *mapping*. Relation R is called *injective* if R^{T} is univalent and *surjective* if R^{T} is total. An injective and surjective relation is said to be *bijective*.

Another important class of heterogeneous relations are *(relational) vectors*, which are defined by the equation $v = v\mathsf{L}$ and can be used to describe subsets of a given set. If $[X \leftrightarrow Y]$ is the type of v, then the vector condition $v = v\mathsf{L}$ means that whatever set Z, universal relation $\mathsf{L} : Y \leftrightarrow Z$, and element x from X we choose either $(v\mathsf{L})_{xz}$ holds for all elements z of Z or for none element z of Z. Consequently, for a vector the range is irrelevant. Therefore, in the following we only consider vectors $v : X \leftrightarrow \mathbf{1}$ with a specific singleton set $\mathbf{1} = \{\perp\}$ as range and omit the second subscript, i.e., write v_x instead of $v_{x\perp}$. Then v can be considered as a Boolean column vector and describes the subset $\{x \in X : v_x\}$ of X. A vector is said to be a *(relational) point* if it is injective and surjective. For $v : X \leftrightarrow \mathbf{1}$ these properties mean that it describes a singleton set, i.e., an element of X if we identify a singleton set $\{x\}$ with its only element x. In the Boolean matrix model a point is a Boolean column vector in which exactly one component is true.

Now we consider some special functions (in the everyday's sense) from relations to relations which are also called *operators*. The operators we will present in the following are introduced in terms of the above basic operations and in most cases they are only partially defined.

The least reflexive and transitive relation containing R is called the *reflexive-transitive closure* of R and is denoted by R^*.

Since we only deal with set-theoretic relations, the so-called point axiom of [ScS93] holds. It says that for every $R \neq \mathsf{O}$ there exist points p and q such that $pq^{\mathsf{T}} \subseteq R$. As a consequence, for each vector $v \neq \mathsf{O}$ there exists a point p fulfilling $p \subseteq v$. The choice of such a point is fundamental for relational programming since it corresponds to the choice of an element from a non-empty set. For a non-empty vector v, a relation-algebraic axiomatization of the *choice* $\mathsf{point}(v)$ is

$$\mathsf{point}(v) \subseteq v \qquad\qquad \mathsf{point}(v) \text{ is a point}\,. \tag{4}$$

At this place it should be emphasized that point is a (deterministic, partial) function in the usual mathematical sense. In particular, each call $\mathsf{point}(v)$ yields the same point in v so that, for example, $\mathsf{point}(v) = \mathsf{point}(v)$ holds. Of course, the axioms (4) allow different realizations. E.g., the specific implementation of the operator point in RelView uses that the system deals only with finite and enumerated carrier sets. A call $\mathsf{point}(v)$ then chooses that point which describes the first element of the set described by v.

The symmetric quotient $\mathsf{syq}(R,S)$ of relations R and S is defined as the greatest relation X such that $RX \subseteq S$ and $XS^\mathsf{T} \subseteq R^\mathsf{T}$. From this we get

$$\mathsf{syq}(R,S) = \overline{R^\mathsf{T}\,\overline{S}} \cap \overline{\overline{R}^\mathsf{T} S} \tag{5}$$

as explicit description. The right-hand side of (5) shows that $\mathsf{syq}(R,S)$ is only defined if R and S have the same domain. Then the domain of $\mathsf{syq}(R,S)$ is the range of R and the range of $\mathsf{syq}(R,S)$ is the range of S. Many properties of the symmetric quotient can be found in [ScS93]. In the following lemma we collect some further properties. Its proof can be found in the appendix.

Lemma 2.1 *Let relations R, S, p, and v be given. Then we have:*

(i) *If p is a point and $p^\mathsf{T}p = \mathsf{I}$, then $\mathsf{syq}(Rp, Rp) = \mathsf{I}$.*

(ii) *If p is a point, then $\mathsf{syq}(R,S)p = \mathsf{syq}(R,Sp)$.*

(iii) *If v is a vector, then $\mathsf{syq}(R,v)$ is a vector.*

(iv) *If R is an equivalence relation, then $\mathsf{syq}(S,R)R = \mathsf{syq}(S,R)$.*

(v) *If v is a non-empty vector and $R\mathsf{L} \subseteq \overline{v}$, then $\mathsf{syq}(R,v) = \mathsf{O}$.*

At the end of this section we show how symmetric quotients are related to powersets. To this end we translate (5) into predicate logic notation using the set-theoretic definitions of the basic operations. The result is

$$\mathsf{syq}(R,S)_{xy} \iff \forall\, z : R_{zx} \leftrightarrow S_{zy}\,. \tag{6}$$

Now we consider this equivalence for the special case of R being a membership relation $\in\, : X \leftrightarrow 2^X$ and S being a vector $v : X \leftrightarrow \mathbf{1}$. Then $\mathsf{syq}(\in, v)$ is of type $[2^X \leftrightarrow \mathbf{1}]$ and for each set Y from 2^X we have $\mathsf{syq}(\in, v)_Y$ if and only if $\forall\, z : z \in Y \leftrightarrow v_z$ holds. The latter shows that $\mathsf{syq}(\in, v)$ describes exactly the same set as the vector v but as an element of the powerset 2^X instead of a subset of the original set X as v does.

3 The RelView System

In the following we want to give an impression of the tool RelView for calculating with relations and relational programming. More details and applications can e.g., be found in [BKU96, BBS97, BBM97, Beh98, BBH99, BHL99].

In RelView all data are represented as binary relations, which the system visualizes in different ways. It offers several different algorithms for pretty-printing a homogeneous relation as a directed graph. Alternatively, a relation may be displayed as a Boolean matrix which is very useful for visual editing and also for discovering various structural properties that are not evident from a graphical presentation. Because RelView often is used on large input data, in the last two years we have developed a very efficient implementation of relations using binary decision diagrams.

The RelView system can manage as many relations simultaneously as memory allows and the user may manipulate and analyse them by pre-defined operations and tests, relational functions, and relational programs. The pre-defined operations include e.g., `^`, `-`, `|`, `&`, and `*` for transposition, negation, join, meet, and composition; the tests include e.g., `incl` and `eq` for testing inclusion resp. equality of relations. All that can be accessed through simple mouse-clicks. A declaration of a relational function is of the form $F(X_1, \ldots, X_n) = t$, where F is the function name, the X_i, $1 \leq i \leq n$, are the formal parameters (standing for relations), and t is a relational term over the relations of the system's workspace that can additionally contain the formal parameters X_i. A relational program in RelView essentially is a while-program based on the datatype of binary relations. Such a program has many similarities with a function procedure in the programming languages Pascal or Modula-2. It starts with a head line containing the program's name and a list of formal parameters. Then the declaration part follows, which consists of the declarations of local relational domains, local relational functions, and local variables. The third part of a relational program is its body, a sequence of statements which are separated by semicolons and terminated by the return-clause.

We renounce at this place examples for relational functions and programs since many of them can be found in Sections 5 and 6.

As especially shown in [BBH99], the RelView system can be used to solve many different tasks while working with relations. But its main application domain is the formal development of relational programs for solving problems on relation-based discrete structures like graphs, ordered sets, and Petri nets. Here the system supports the main validation tasks within nearly all stages of a development. For example, it can be applied to check the formal relational problem specification against the informal fixed requirements. Toying with relations and relation-algebraic properties, the system can also help to find loop invariants or other decisive properties necessary for a correctness proof of a program. As a third application, the execution of a relational program or a piece of it via RelView in the course of a derivation can reveal alternative development steps and possibilities for optimizations.

4 A Simple Relation-algebraic Model for Sequences

Our modeling of sequences within RelView is based upon a relation-algebraic characterization of binary direct sums. Within the relation-algebraic framework it is natural to do the latter by means of the natural injective embedding mappings. This leads to the following relation-algebraic specification (see also [Zie91]): A pair $\imath_1$ and $\imath_2$ of relations is called a *binary direct sum* if

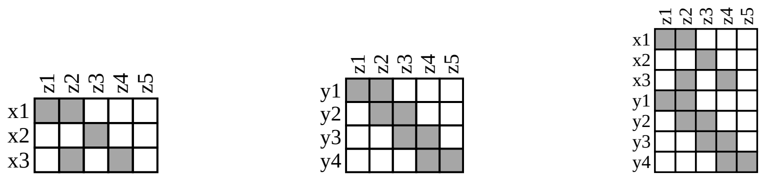
$$\imath_1 \imath_1^{\mathsf{T}} = \mathsf{I} \qquad \imath_2 \imath_2^{\mathsf{T}} = \mathsf{I} \qquad \imath_1^{\mathsf{T}} \imath_1 \cup \imath_2^{\mathsf{T}} \imath_2 = \mathsf{I} \qquad \imath_1 \imath_2^{\mathsf{T}} = \mathsf{O}\,. \quad (7)$$

Given two sets X_1 and X_2 it is easy to verify that the injective embedding mappings from these sets to the set-theoretic direct sum $X_1 + X_2$ are a model of the formulae (7). Furthermore, by purely relation-algebraic reasoning it can be shown that the binary direct sum is uniquely characterized up to isomorphism by (7).

Based upon a binary direct sum $X_1 + X_2$ with the two injective embedding mappings $\imath_1 : X_1 \leftrightarrow X_1{+}X_2$ and $\imath_2 : X_2 \leftrightarrow X_1{+}X_2$, we now define for relations $R : X_1 \leftrightarrow Y$ and $S : X_2 \leftrightarrow Y$ their *relational sum* $R + S : X_1{+}X_2 \leftrightarrow Y$ by

$$R + S = \imath_1{}^{\mathsf{T}} R \cup \imath_2{}^{\mathsf{T}} S\,. \tag{8}$$

This construction behaves like the relation R for all elements from the set $X_1{+}X_2$ which come from its first part X_1 and like the relation S for all elements from $X_1{+}X_2$ which come from its second part X_2. The following three Boolean matrices – produced by the RelView system in which the relational sum operator is pre-defined – show a small example; for a better understanding of the construction of (8) we have instructed the RelView system to label the rows and columns of each Boolean matrix with the elements of the domain and range of the relation it depicts.



R $\qquad$ S $\qquad$ $R + S$

Having defined the relational sum, now we are in a position to model non-empty (and finite) sequences of sets and elements in a very simple way. In Section 2 we have already shown how a subset (resp. an element) of a set X can be modeled by a vector (resp. a point) $v : X \leftrightarrow \mathbf{1}$. Therefore, it suffices to model sequences $\mathbf{v} = (v_i)_{1 \le i \le n}$ of vectors of type $[X \leftrightarrow \mathbf{1}]$ with relation-algebraic means. As we are interested in algorithms, especially executable through the RelView system, in the following we restrict ourselves to relations $R : X \leftrightarrow Y$ with finite and enumerated carrier sets X and Y. Hence, we are allowed to consider R as a Boolean $m \times n$ matrix with m being the cardinality of X and n that of Y. In this Boolean matrix interpretation, which is also the standard way of RelView to depict a relation on its screen, R models the sequence $\mathbf{v} = (v_i)_{1 \le i \le n}$ of vectors, where $v_i : X \leftrightarrow \mathbf{1}$ corresponds to its i^{th} column ($1 \le i \le n$). Considering the transpositions of the above three pictures, now it becomes obvious how to define in general the concatenation $R \,@\, S : X \leftrightarrow Y_1{+}Y_2$ of relations $R : X \leftrightarrow Y_1$ and $S : X \leftrightarrow Y_2$ such that it models the concatenation of sequences $\mathbf{v} = (v_i)_{1 \le i \le m}$ and $\mathbf{w} = (w_i)_{1 \le i \le n}$ of vectors of type $[X \leftrightarrow \mathbf{1}]$ if R models $\mathbf{v}$ and S models $\mathbf{w}$:

$$R \,@\, S = (R^{\mathsf{T}} + S^{\mathsf{T}})^{\mathsf{T}}\,. \tag{9}$$

However, it must be pointed out that this concatenation of relations models the usual concatenation of sequences only within RelView because of the specific definition of the relational sum of the system which forms the Boolean matrix of $R + S$ by putting the Boolean matrices of R and S one upon another. Independent of the system, for a correct modeling of sequences via relation-algebraic binary direct sums and (9) we have to demand that the direct sum $X{+}Y$ of two disjoint and finite sets X and Y, the elements of which are enumerated as $x_1, \ldots, x_m$ resp. $y_1, \ldots, y_n$, is enumerated as $x_1, \ldots, x_m, y_1, \ldots, y_n$. Without this property only non-empty multisets and their unions are modeled.

The following lemma states some facts of the concatenation operator of (9) which are used in the program derivation of the next section. Its proof can again be found in the appendix.

Lemma 4.1 *Let relations R, S, and T be given. Then we have:*

(i) $\mathsf{syq}(R,R) = \mathsf{I}$, $\mathsf{syq}(S,S) = \mathsf{I}$, $\mathsf{syq}(R,S) = \mathsf{O}$ *imply* $\mathsf{syq}(R \,@\, S, R \,@\, S) = \mathsf{I}$.

(ii) $\mathsf{syq}(R, S \,@\, T) = \mathsf{syq}(R,S) \,@\, \mathsf{syq}(R,T)$.

(iii) $(R \,@\, S)\mathsf{L} = R\mathsf{L} \cup S\mathsf{L}$.

It should be pointed out that this lemma does not fall out of the blue but relation-algebraically formalizes rather clear properties of sequences. We demonstrate this by means of implication (i). Using matrix terminology, from the equivalence (6) we obtain that $\mathsf{syq}(R,R) = \mathsf{I}$ holds if and only if the columns of R are pair-wise distinct and $\mathsf{syq}(R,S) = \mathsf{O}$ holds if and only if R and S have no column in common. Changing to sequence terminology, hence Lemma 4.1.i says: If the elements of a sequence $\mathbf{v}$ as well as of a sequence $\mathbf{w}$ are pair-wise distinct and $\mathbf{v}$ and $\mathbf{w}$ have no element in common, then also the elements of their concatenation are pair-wise distinct.

5 An Application: Computing Equivalence Classes

Now we use the model of the last section to solve a concrete problem. We assume $R : X \leftrightarrow X$ to be an equivalence relation on a finite set X. Our goal is to combine relational algebra and the so-called Dijkstra-Gries program development method (see [Dij76, Gri81]) to derive formally a RelView program which enumerates the set $\mathcal{C}$ of all equivalence classes of R column-wise as a relation $C : X \leftrightarrow \mathcal{C}$.

First we have to specify the problem by relation-algebraic pre- and postconditions. As we assume the relation R to be the input of the program we want to derive, the precondition $pre(R)$ is obvious. From Section 2 we get

$$pre(R) \;\stackrel{\triangle}{=}\; \mathsf{I} \subseteq R \;\wedge\; RR \subseteq R \;\wedge\; R \subseteq R^{\mathsf{T}}.$$

The relation C is the output of the program. Using matrix terminology, therefore we have to specify that its columns are pair-wise distinct and describe all equivalence classes of R. At the end of Section 4 we have already shown how to formalize the first of these properties with relation-algebraic means. This leads to $\mathsf{syq}(C,C) = \mathsf{I}$ as first part of the postcondition $post(R,C)$. To formalize also the second of the above properties within relational algebra we calculate

$$\begin{array}{rcll} c \text{ is equivalence class} & \Longleftrightarrow & \exists\, x\; \forall\, y : y \in c \leftrightarrow R_{yx} & \text{where } \in : X \leftrightarrow 2^X \\ & \Longleftrightarrow & \exists\, x : \mathsf{syq}(\in, R)_{cx} & (6) \\ & \Longleftrightarrow & (\mathsf{syq}(\in, R)\mathsf{L})_c & \text{where } \mathsf{L} : X \leftrightarrow \mathbf{1} \end{array}$$

and get $\mathsf{syq}(\in, R)\mathsf{L} : 2^X \leftrightarrow \mathbf{1}$ as vector for describing the equivalence classes of R as elements of the powerset 2^X. This leads to $\mathsf{syq}(\in, R)\mathsf{L} = \mathsf{syq}(\in, C)\mathsf{L}$ as second part of $post(R,C)$. In words it says that each column of C describes an equivalence class of R and, conversely, each equivalence class of R is described by a column of C. Altogether, we have

$$post(R,C) \;\stackrel{\wedge}{=}\; \mathsf{syq}(C,C) = \mathsf{I} \,\wedge\, \mathsf{syq}(\in,R)\mathsf{L} = \mathsf{syq}(\in,C)\mathsf{L}\,.$$

Having completed the problem specification, now our goal is to calculate a loop invariant and a guard from $post(R,C)$. Here we follow the most common approach of generalizing a postcondition by introducing new variables. In the present case it seems to be a good idea to compute the equivalence classes of R, i.e., the sequence of vectors modeled by C, one after the other and to use a vector for describing the elements yet to be checked. If we use $v : X \leftrightarrow \mathbf{1}$ for the latter purpose, then we arrive at

$$inv(R,C,v) \;\stackrel{\wedge}{=}\; \mathsf{syq}(C,C) = \mathsf{I} \,\wedge\, \mathsf{syq}(\in,R)\,\overline{v} = \mathsf{syq}(\in,C)\mathsf{L} \,\wedge\, Rv \cap C\mathsf{L} = \mathsf{O}$$

as loop invariant $inv(R,C,v)$. In words its second equation says that the columns of C describe the equivalence classes of the already checked elements (which again are described by $\overline{v}$) and its third equation says that no element of an equivalence class still to be computed occurs in an equivalence class which has already been computed. It is obvious that $inv(R,C,v)$ implies $post(R,C)$ when v is empty. This leads to $v \neq \mathsf{O}$ as the guard of the loop. Hence, it remains to develop an initialization of C and v which establishes $inv(R,C,v)$ and a body of the loop which maintains $inv(R,C,v)$ and ensures termination.

Let's start with the initialization. Here it seems to be a good idea to choose arbitrarily an equivalence class, i.e., to initialize C with the vector $R\,\mathsf{point}(\mathsf{L})$, where $\mathsf{L} : X \leftrightarrow \mathbf{1}$. Consequently, we have to initialize v with $\overline{C}$. This initialization establishes the loop invariant. In the subsequent proof of this fact we abbreviate the choice $\mathsf{point}(\mathsf{L})$ as p.

The first equation $\mathsf{syq}(Rp,Rp) = \mathsf{I}$ of $inv(R,Rp,\overline{Rp})$ is an immediate consequence of Lemma 2.1.i since p is a point due to (4) and from its type $[X \leftrightarrow \mathbf{1}]$ we get $p^{\mathsf{T}}p = \mathsf{I}$. A proof of the second equation $\mathsf{syq}(\in,R)\,\overline{\overline{Rp}} = \mathsf{syq}(\in,Rp)\mathsf{L}$ of $inv(R,Rp,\overline{Rp})$ is given by

$$\mathsf{syq}(\in,R)\,\overline{\overline{Rp}} = \mathsf{syq}(\in,R)Rp = \mathsf{syq}(\in,R)p = \mathsf{syq}(\in,Rp) = \mathsf{syq}(\in,Rp)\mathsf{L}\,,$$

where we successively have applied lattice theory, Lemma 2.1.iv in combination with the precondition, Lemma 2.1.ii, and Lemma 2.1.iii. For a proof of the third equation $R\,\overline{Rp} \cap Rp\mathsf{L} = \mathsf{O}$ of $inv(R,Rp,\overline{Rp})$ we use that it is equivalent to $R\,\overline{Rp} \subseteq \overline{Rp}$ and prove this inclusion. Symmetry and transitivity of R show $R^{\mathsf{T}}R \subseteq R$ which in turn implies $R^{\mathsf{T}}Rp \subseteq Rp$. Now we apply the Schröder equivalences (2) and are done.

To complete the derivation, we have to work out a loop body and to verify that it maintains $inv(R,C,v)$ and ensures termination. Since C finally shall enumerate the equivalence classes of R and v describes those elements of X whose classes still have to be added to C, it seems to be promising to explore the effect of the assignments $C := C\,@\,Rp$ and $v := v \cap \overline{Rp}$ with p being a point contained in v. By that a new class is added to C and v is changed accordingly. In the syntax of RelView (the basic constants and operations as well as `point` and the emptiness test `empty` are pre-defined; see [BBS97]) we then get:

```
conc(A,B) = (A^ + B^)^.

classes(R)
 DECL C, v
 BEG C = R * point(Ln1(R));
     v = -C;
     WHILE -empty(v) DO
       C = conc(C, R * point(v));
       v = v & -(R * point(v)) OD
     RETURN C
 END.
```

For the proof that the loop body of `classes` maintains the loop invariant we assume $v \neq \mathsf{O}$ and $inv(R, C, v)$. Furthermore, we abbreviate $\mathsf{point}(v)$ as p.

We apply Lemma 4.1.i to prove the first equation $\mathsf{syq}(C @ Rp, C @ Rp) = \mathsf{I}$ of $inv(R, C @ Rp, v \cap \overline{Rp})$ and only have to verify its three premises. Equation $\mathsf{syq}(C, C) = \mathsf{I}$ is part of the supposed invariant $inv(R, C, v)$. A proof of $\mathsf{syq}(Rp, Rp) = \mathsf{I}$ directly follows from Lemma 2.1.i since p is again a point with $p^\mathsf{T} p = \mathsf{I}$. Finally, to prove $\mathsf{syq}(C, Rp) = \mathsf{O}$ we use $Rv \cap C\mathsf{L} = \mathsf{O}$ since this part of $inv(R, C, v)$ in combination with $p \subseteq v$ shows $C\mathsf{L} \subseteq \overline{Rp}$ which in turn implies the desired result due to the vector property of Rp, the inequation $\mathsf{O} \neq Rp$ (which follows from surjectivity of p and reflexivity of R), and Lemma 2.1.v. Subsequent is the proof of the second equation $\mathsf{syq}(\in, R)\,\overline{v \cap \overline{Rp}} = \mathsf{syq}(\in, C @ Rp)\mathsf{L}$ of $inv(R, C @ Rp, v \cap \overline{Rp})$:

$$
\begin{array}{ll}
\ \mathsf{syq}(\in, R)\,\overline{v \cap \overline{Rp}} & \\
= \mathsf{syq}(\in, R)\,\overline{v} \cup \mathsf{syq}(\in, R)Rp & \\
= \mathsf{syq}(\in, R)\,\overline{v} \cup \mathsf{syq}(\in, R)p\mathsf{L} & pre(R),\ \text{Lemma 2.1.iv},\ p = p\mathsf{L} \\
= \mathsf{syq}(\in, R)\,\overline{v} \cup \mathsf{syq}(\in, Rp)\mathsf{L} & p \text{ point, Lemma 2.1.ii} \\
= \mathsf{syq}(\in, C)\mathsf{L} \cup \mathsf{syq}(\in, Rp)\mathsf{L} & inv(R, C, v) \\
= (\mathsf{syq}(\in, C) @ \mathsf{syq}(\in, Rp))\mathsf{L} & \text{Lemma 4.1.iii} \\
= \mathsf{syq}(\in, C @ Rp)\mathsf{L} & \text{Lemma 4.1.ii}\,.
\end{array}
$$

For a proof of the third equation $R(v \cap \overline{Rp}) \cap (C @ Rp)\mathsf{L} = \mathsf{O}$ of the property $inv(R, C @ Rp, v \cap \overline{Rp})$ we start with the calculation

$$
\begin{array}{ll}
\ R(v \cap \overline{Rp}) \cap (C @ Rp)\mathsf{L} & \\
= R(v \cap \overline{Rp}) \cap (C\mathsf{L} \cup Rp\mathsf{L}) & \text{Lemma 4.1.iii} \\
\subseteq Rv \cap R\,\overline{Rp} \cap (C\mathsf{L} \cup Rp) & p = p\mathsf{L} \\
= R\,\overline{Rp} \cap ((Rv \cap C\mathsf{L}) \cup (Rv \cap Rp)) & \\
= R\,\overline{Rp} \cap Rp & inv(R, C, v),\ p \subseteq v\,.
\end{array}
$$

Hence, it remains to show the equation $R\,\overline{Rv} \cap Rp = \mathsf{O}$. But it follows, as in the proof of $inv(R, Rp, \overline{Rp})$, from the inclusion $R^{\mathsf{T}}Rp \subseteq Rp$ and an application of the Schröder equivalences (2).

It remains to verify that the body of the loop ensures termination. Let p again be given as $p = \mathsf{point}(v)$. Since X is a finite set and $Rp \neq \mathsf{O}$ because $p \neq \mathsf{O}$ and $\mathsf{I} \subseteq R$, the above program obviously terminates if $Rp \cap v \neq \mathsf{O}$. We prove this property by contradiction: Assume $Rp \subseteq \overline{v}$. Then we get $R^{\mathsf{T}}v \subseteq \overline{p}$ due to the Schröder equivalences. Now $pre(R)$ implies $v \subseteq Rv = R^{\mathsf{T}}v \subseteq \overline{p}$, i.e., $p \subseteq \overline{v}$, and combining this latter inclusion with $p \subseteq v$ yields the contradiction $p = \mathsf{O}$.

6 A Refinement of the Approach

Hitherto we can only model non-empty sequences and their concatenation within RelView. Now we refine the simple relation-algebraic model of Section 4 to obtain the classical datatype of stacks with the operations first and rest such that, using matrix terminology, $\mathsf{first}(R)$ yields the first column of R and $\mathsf{rest}(R)$ yields the remaining columns of R after removal of the first one. For this purpose, first we introduce two new operators on relations.

We start with the representation of sets via embedding mappings which already has been mentioned in the introduction. Given an injective mapping $\imath : Y \leftrightarrow X$ we may regard Y as a subset of X. Then the vector $\imath^{\mathsf{T}}\mathsf{L} : X \leftrightarrow \mathbf{1}$ describes Y in the sense introduced in Section 2. A transition in the other direction, i.e., the construction of an injective mapping $\mathsf{inj}(v) : Y \leftrightarrow X$ from a given non-empty vector $v : X \leftrightarrow \mathbf{1}$ describing Y in such a way that $\mathsf{inj}(v)_{yx}$ if and only if $y = x$, is also possible. Using matrix terminology, one only has to remove from the identity matrix those rows which don't correspond to an element of Y. We call $\mathsf{inj}(v)$ the *injective mapping generated by* v. The following relation-algebraic axiomatization of the construction originates from [BeH00]:

$$\begin{array}{l} \mathsf{inj}(v) \text{ is an injective mapping} \\ v = \mathsf{inj}(v)^{\mathsf{T}}\mathsf{L} \\ \mathsf{inj}(\mathsf{inj}(v)^{\mathsf{T}}w) = \mathsf{inj}(w)\mathsf{inj}(v) \text{ for all non-empty vectors } w. \end{array} \tag{10}$$

Note that $\mathsf{inj}(v)$ is only defined if $v \neq \mathsf{O}$ as $v = \mathsf{O}$ would imply the domain of $\mathsf{inj}(v)$ to be empty. At this place also a few words should be said about the third axiom of (10). If the vectors $v : X \leftrightarrow \mathbf{1}$ and $w : Z \leftrightarrow \mathbf{1}$ describe the subset Z of X and W of Z, then $\mathsf{inj}(v)^{\mathsf{T}}w : X \leftrightarrow \mathbf{1}$ obviously describes W as a subset of X. Hence, the axiom says that the embedding of W into X is obtained by first embedding W into Y and then embedding Y into X.

As we have already mentioned in Section 4, we restrict ourselves to relations on finite and enumerated carrier sets. Such an enumeration $x_1, \ldots, x_n$ of a set X leads to the specific initial element x_1 and a partial successor function which maps x_i to x_{i+1} for $1 \leq i \leq n-1$. In RelView, the *point describing the initial element* x_1 is computed by the call $\mathsf{init}(v)$, where the argument v of the operator init may be any vector of type $[X \leftrightarrow \mathbf{1}]$ and is used only for typing reasons. A relation-algebraic axiomatization of $\mathsf{init}(v) : X \leftrightarrow \mathbf{1}$ is possible using a further relation $\mathsf{succ}(v) : X \leftrightarrow X$ for the above mentioned partial successor function.

We demand the following properties:

$$\begin{aligned}&\mathsf{succ}(v) \text{ is univalent and injective}\\&\mathsf{succ}(v)^{\mathsf{T}}\mathsf{L} \subseteq \overline{\mathsf{init}(v)}\\&(\mathsf{succ}(v)^{\mathsf{T}})^{*}\mathsf{init}(v) = \mathsf{L}\,.\end{aligned} \tag{11}$$

Here the second formula says that the point $\mathsf{init}(v)$ is not a successor and the third axiom expresses the fact that every element can be obtained from the initial one by finitely many applications of the partial successor function. The axiomatization (11) is a variant of the relation-algebraic version of the well-known Peano axioms for natural numbers given in [BeZ86].

With the help of the two new operators inj and init on relations, now it is rather trivial to model the two operations first and rest on sequences. Assume a relation $R : X \leftrightarrow Y$ and a vector $v : Y \leftrightarrow \mathbf{1}$ with the set Y enumerated as $y_1, \ldots, y_n$. In matrix terminology then $\mathsf{inj}(\mathsf{init}(v)) : \{y_1\} \leftrightarrow Y$ is a Boolean row vector in which exactly the first component is true and hence the vector $R\,\mathsf{inj}(\mathsf{init}(v))^{\mathsf{T}} : X \leftrightarrow \{y_1\}$ is the first column of R. Choosing v for instance as empty vector leads to the definition $\mathsf{first}(R) = R\,\mathsf{inj}(\mathsf{init}(\mathsf{O}))^{\mathsf{T}}$ which simplifies to

$$\mathsf{first}(R) = R\,\mathsf{init}(\mathsf{O}) \tag{12}$$

as $\mathsf{inj}(p) = p^{\mathsf{T}}$ for all points $p : X \leftrightarrow \mathbf{1}$. Applying the same ideas we obtain

$$\mathsf{rest}(R) = R\,\mathsf{inj}(\,\overline{\mathsf{init}(\mathsf{O})}\,)^{\mathsf{T}} \tag{13}$$

for rest since, using again matrix terminology, $\mathsf{inj}(\,\overline{\mathsf{init}(v)}\,) : \{y_2, \ldots, y_n\} \leftrightarrow Y$ is obtained from $\mathsf{I} : Y \leftrightarrow Y$ by removing the first row.

Having developed the operators first and rest, now we deal with a concrete application. We assume a finite directed graph g with vertex set X and arc relation $R : X \leftrightarrow X$. Our goal is to develop a RelView program which computes a vector $b : X \leftrightarrow \mathbf{1}$ describing a vertex basis of g, i.e., a $\subseteq$-minimal subset of X such that every vertex can be reached from a vertex of this set.

It is easy to show that for cyclefree g its sources form the only vertex basis; see [BHL99] for a relation-algebraic proof of this fact. In principle, with that the problem of computing a vertex basis is also solved for the general case. Let $\mathcal{C}$ be the set of strongly connected components of g and $g^{\#}$ the (cyclefree) reduced graph. I.e., $g^{\#}$ has $\mathcal{C}$ as vertex set and its arc relation $R^{\#} : \mathcal{C} \leftrightarrow \mathcal{C}$ given by

$$R^{\#}_{cd} \iff c \neq d \wedge \exists\, x,y : x \in c \wedge y \in d \wedge R_{xy}\,. \tag{14}$$

The sources of $g^{\#}$ are exactly the predecessor-closed strongly connected components of g. Let $\mathcal{S}$ denote the set of these specific components. From the above we know that $\mathcal{S}$ is a vertex basis of $g^{\#}$. Taking from each set of $\mathcal{S}$ exactly one vertex, now obviously leads to a vertex basis of g.

It is rather trivial to realize the procedure just described as a RelView program. The classes of the equivalence relation $R^{*} \cap (R^{\mathsf{T}})^{*}$ coincide with the strongly connected components of g and hence are computed by the following function (with a pre-defined operation `rtc` for reflexive-transitive closure):

```
sccs(R) = classes(rtc(R) & rtc(R^)).
```

Assume $C : X \leftrightarrow \mathcal{C}$ to be the result when applying the relational program sccs to R. Then the relation C is the canonical epimorphism from X to $\mathcal{C}$ wrt. the equivalence relation $R^* \cap (R^\mathsf{T})^*$, i.e., we have C_{xc} if and only if x is an element of c. Combining this with (14) leads to $R^\# = \bar{\mathsf{I}} \cap C^\mathsf{T} R C$ as relation-algebraic description of $R^\#$ and as a consequence we get the following vector $v : \mathcal{C} \leftrightarrow \mathbf{1}$ for the description of the sources of $g^\#$, i.e., the subset $\mathcal{S}$ of $\mathcal{C}$:

$$v = \overline{(\bar{\mathsf{I}} \cap C^\mathsf{T} R^\mathsf{T} C)\mathsf{L}} \ .$$

Now we are almost done. We consider the injective mapping $\mathsf{inj}(v) : \mathcal{S} \leftrightarrow \mathcal{C}$ generated by v and get a column-wise modeling of $\mathcal{S}$ by $C\,\mathsf{inj}(v)^\mathsf{T} : X \leftrightarrow \mathcal{S}$. As a RelView program the computation of this relation looks as follows, where the pre-defined operation dom yields for a relation A the vector $A\mathsf{L}$ and the function strict(A) = -I(A) & A is part of the system's start-up file and automatically loaded at startup time:

```
initsccs(R)
 DECL C, v
 BEG C = sccs(R);
     v = -dom(strict(C^ * R^ * C))
     RETURN C * inj(v)^
 END.
```

Using matrix terminology, it remains as a last step to take from each column of the result of this program a point and to join all these points to a vector describing a vertex basis. If we suppose that also the definitions (12) and (13) are part of the start-up file, then we obtain the following obvious solution:

```
basis(R)
 DECL b, S
 BEG b = On1(R);
     S = initsccs(R);
     WHILE -eq(S,S * L(S^ * S)) DO
        b = b | point(first(S));
        S = rest(S) OD;
     b = b | point(S)
     RETURN b
 END.
```

In this program the vertical bar | is the RelView operation for union and the guard tests S to be not a vector, i.e., to contain as a matrix at least two columns.

Having sketched an application, let us return to the original task of this section. Until now we have presented only a model for non-empty stacks with operations @, first, and rest. However, it is an easy exercise to modify it in such a way that arbitrary stacks are modeled. The technique we usually apply within RelView is to add to each Boolean matrix which column-wise represents a sequence of vectors an additional specific column from the left or the right, which then stands for the empty stack, and to modify the definitions (9), (12), and (13) of @, first, and rest accordingly. At this place it should also be mentioned that our approach works for all objects which can be represented by vectors. Especially we can deal with sequences of relations (which e.g., occur when computing all fundamental cycles if these are regarded as sets of arcs) using the correspondence between the types $[X \leftrightarrow Y]$ and $[X \times Y \leftrightarrow \mathbf{1}]$ given in [ScS93].

7 Conclusion

In this paper we have presented a relation-algebraic model of sequences within the RelView system. We have then combined relation-algebraic calculations with the Dijkstra-Gries program development method and formally derived a RelView program for the column-wise enumeration of equivalence classes. Finally we have presented a refinement of the simple model to obtain the classical datatype of stacks and have sketched the computation of a vertex basis of a graph as a further application.

Besides the programs presented in this paper we have applied our relation-algebraic approach to sequences to solve many other problems with RelView. Here are some examples. We have used sequences within RelView for prototyping relational specifications which are constructed following the approach of [BGS94, BKU96]. This allows, e.g., to compute the cut completion of an ordered set or to enumerate column-wise all kernels of a graph resp. all deadlocks of a Petri net with the help of the system. We have also used RelView for implementing Fleury's algorithm. The resulting program computes for the relation of an undirected, loop-free, and connected graph, in which every vertex degree is even, a so-called Eulerian cycle as the sequence of columns of the resulting matrix. As a final example, we have transformed Behnke's RelView program of [Beh98] for solving the reachability problem of condition-event Petri nets into a version which uses sequences of vectors of type $[C \leftrightarrow \mathbf{1}]$ instead of only vectors of type $[2^C \leftrightarrow \mathbf{1}]$, where C is the set of conditions. To give an impression for the speed-up obtained by this, in the case of the well-known five dining philosophers net (15 conditions, 10 events) executing Behnke's program with RelView on a Sun Ultra I workstation takes 7 seconds but the version with sequences needs only 1 second. For the seven philosophers net (21 conditions, 14 events) we even obtained a speed-up from 3034 to 11 seconds.

Of course, the idea of a relational approach to datatypes is rather old and has been studied by many researchers. See [DGB97, BiM97, Moe91] just to cite some important contributions of three different groups working in this domain. Our approach contrasts with all of this work since it is specifically tailored to RelView without touching the system's present state. Therefore, it is legitimate to question its adequacy. Why not simple extend the programming language of RelView by finite sequences and use, besides relational algebra, their well-known axiomatization as second algebraic framework for formal program derivation?

It is certainly fair to say that in some points an approach with sequences as pre-defined datatype is superior. Especially, it is more general and also more easily accessible for an "average" user of relation-algebraic methods than ours. Therefore, we plan to extend the programming language of RelView by finite sequences and some other standard datatypes in the near future.

But also our approach has some advantages. First, using higher-order constructs it allows often to specify a problem involving sequences not only very precise and concise but also executable with RelView. Such a specification then additionally may be used for prototyping purposes as already mentioned above. At Kiel University we have found it also very attractive to use executable specifications for producing good examples in teaching.

Second, if a sequence immediately induces a relation, then our approach of identifying both allows to exploit this for further relational computations. We have applied this technique already in Section 6 and used $C : X \leftrightarrow \mathcal{C}$ (the sequence of strongly connected components / the canonical epimorphism) for computing the relation of the reduced graph. Another example for this is the computation of the so-called block-articulation graph with the sketch of which we will conclude. Given an undirected graph $g = (X, R)$, it is very easy to formulate the well-known depth-first search algorithm as a RelView-program and, based on it, a program which computes the well-known *lowpoint*-function as a mapping $\mathit{lowpoint} : X \leftrightarrow X$ in the relational sense; see [BHL99]. This immediately leads to relational programs for the vector $v : X \leftrightarrow \mathbf{1}$ describing the set A of articulations (or: cut vertices) and the relation $B : X \leftrightarrow \mathcal{B}$ columnwise enumerating the blocks (or: biconnected components) of g[1]. Hence, $\mathsf{inj}(v)B : A \leftrightarrow \mathcal{B}$ associates an articulation a to a block b if and only if $a \in b$. The (undirected) block-articulation graph has the binary direct sum $A + \mathcal{B}$ as vertex set and connects two vertices a and b if and only if a is an articulation, b is a block, and $a \in b$. Using the injective embedding mappings $\imath_1 : A \leftrightarrow A{+}\mathcal{B}$ and $\imath_2 : \mathcal{B} \leftrightarrow A{+}\mathcal{B}$, it is obvious that its relation can be computed by ${\imath_1}^{\mathsf{T}}\mathsf{inj}(v)B\imath_2 \cup {\imath_2}^{\mathsf{T}}B^{\mathsf{T}}\mathsf{inj}(v)^{\mathsf{T}}\imath_2$.

References

[Beh98] Behnke R.: Prototyping relational specifications and programs with RelView. In: Buth B., Berghammer R., Peleska J. (eds.): Tools for system development and verification. BISS Monographs 1, Shaker Verlag, 22-42 (1998)

[BBS97] Behnke R., Berghammer R., Schneider P.: Machine support of relational computations. The Kiel RelView system. Bericht 9711, Institut für Informatik und Praktische Mathematik, Univ. Kiel (1997)

[BBM97] Behnke R., Berghammer R., Meyer E., Schneider P.: RelView – A system for calculating with relations and relational programming. In: Astesiano E. (ed.): Proc. Conference on Fundamental Approaches to Software Engineering (FASE '98), LNCS 1382, Springer Verlag, 318-321 (1998)

[BBH99] Behnke R., Berghammer R., Hoffmann T., Leoniuk B., Schneider P.: Applications of the RelView system. In: Berghammer R., Lakhnech Y. (eds.): Tool support for system specification, development and verification. Advances in Computing, 33-47 (1999)

[BeZ86] Berghammer R., Zierer H.: Relational algebraic semantics of deterministic and nondeterministic programs. Theoretical Computer Science 43, 123-147 (1986)

[BGS94] Berghammer R., Gritzner T., Schmidt G.: Prototyping relational specifications using higher-order objects. In: Heering, J. et al. (eds.): Proc. Workshop

[1] An articulation is a vertex whose removal strictly increases the number of components of a graph and a block is a maximal subset of pair-wise reachable vertices without an articulation.

on Higher Order Algebra, Logic and Term Rewriting (HOA '93), LNCS 816, Springer Verlag, 56-75 (1994)

[BKU96] Berghammer R., Karger B. von, Ulke C.: Relation-algebraic analysis of Petri nets with RelView. In: Margaria T., Steffen B. (eds.): Proc. Workshop on Tools and Applications for the Construction and Analysis of Systems (TACAS '96), LNCS 1055, Springer Verlag, 49-69 (1996)

[BHL99] Berghammer R., Hoffmann T., Leoniuk B.: Rechnergestützte Erstellung von Prototypen für Programme auf relationalen Strukturen. Bericht Nr. 9905, Institut für Informatik und Praktische Mathematik, Univ. Kiel (1999)

[BeH00] Berghammer R., Hoffmann T.: Deriving relational programs for computing kernels by reconstructing a proof of Richardson's theorem. Science of Computer Programming 38, 1-25 (2000)

[BiM97] Bird R., de Moor O.: Algebra of programming. Int. Series in Computer Science, Prentice Hall (1997)

[BKS97] Brink C., Kahl W., Schmidt G. (eds.): Relational methods in Computer Science. Advances in Computing Science, Springer Verlag (1997)

[ChT51] Chin L.H., Tarski A.: Distributive and modular laws in the arithmetic of relation algebras. Univ. of California Publ. in Math. (new series) 1, 341-384 (1951)

[Dij76] Dijkstra E.W.: A discipline of programming. Prentice-Hall (1976)

[DGB97] Doornbos H., van Gasteren N., Backhouse R.: Programs and datatypes. In: Brink C. et al. (eds.): Relational methods in Computer Science. Advances in Computing Science, Springer Verlag, 150-165 (1997)

[Gri81] Gries D.: The science of computer programming. Springer Verlag (1981)

[Moe91] Möller B.: Relations as a program development language. In: Möller B. (ed.): Constructing programs from specifications. North-Holland, 373-397 (1991)

[ScS93] Schmidt G., Ströhlein T.: Relations and graphs. Discrete Mathematics for Computer Scientists, EATCS Monographs on Theoretical Computer Science, Springer Verlag (1993)

[Tar41] Tarski A.: On the calculus of relations. Journal of Symbolic Logic 6, 73-89 (1941).

[Zie91] Zierer H.: Relation algebraic domain constructions. Theoretical Computer Science 87, 163-188 (1991)

Appendix

Proof of Lemma 2.1: (i) We use the two Theorems 4.4.7.iii (note that Rp is a vector) and 2.4.5.ii of [ScS93] followed by the assumption $p^{\mathsf{T}}p = \mathsf{I}$ and get

$$\mathsf{syq}(Rp, Rp) = \mathsf{L} = p^{\mathsf{T}}p = \mathsf{I}\,.$$

(ii) The following derivation uses in the first step Theorem 4.4.1.ii of [ScS93] and in the last two steps Theorem 4.4.1.vi of [ScS93] (in combination with the fact that p^{T} is a mapping) and Theorem 4.4.1.ii of [ScS93]:

$$\mathsf{syq}(R,S)p = \mathsf{syq}(S,R)^{\mathsf{T}}p = (p^{\mathsf{T}}\mathsf{syq}(S,R))^{\mathsf{T}} = \mathsf{syq}(Sp,R)^{\mathsf{T}} = \mathsf{syq}(R,Sp)\,.$$

(iii) The inclusion $\mathsf{syq}(R,v) \subseteq \mathsf{syq}(R,v)\mathsf{L}$ is trivial; the remaining inclusion is shown by the following calculation which essentially uses that the vector property is maintained by multiplication from left and negation:

$$\mathsf{syq}(R,v)\mathsf{L} \subseteq \overline{R^{\mathsf{T}}\,\overline{v}}\,\mathsf{L} \cap \overline{\overline{R}^{\mathsf{T}} v}\,\mathsf{L} = \overline{R^{\mathsf{T}}\,\overline{v}} \cap \overline{\overline{R}^{\mathsf{T}} v} = \mathsf{syq}(R,v)\,.$$

(iv) In the first two steps of the following calculation we use Theorem 4.4.5.i of [ScS93] (in combination with the assumption on R) and Theorem 4.4.3.i of [ScS93]; the last step is trivial:

$$\mathsf{syq}(S,R)R = \mathsf{syq}(S,R)\mathsf{syq}(R,R) = \mathsf{syq}(S,R) \cap \mathsf{syq}(S,R)\mathsf{L} = \mathsf{syq}(S,R)\,.$$

(v) The assumption $R\mathsf{L} \subseteq \overline{v}$ is equivalent to $v\mathsf{L} \subseteq \overline{R}$ due to the Schröder equivalences (2) and transposition yields $\mathsf{L}v^{\mathsf{T}} \subseteq \overline{R}^{\mathsf{T}}$. Since v is non-empty, the same holds for $v^{\mathsf{T}}v$. Hence, the vector property $v = v\mathsf{L}$ and the Tarski rule (3) show $v^{\mathsf{T}}v = \mathsf{L}$ and we obtain the desired result by

$$\mathsf{syq}(R,v) = \overline{R^{\mathsf{T}}\overline{v}} \cap \overline{\overline{R}^{\mathsf{T}}v} \subseteq \overline{\overline{R}^{\mathsf{T}}v} \subseteq \overline{\mathsf{L}v^{\mathsf{T}}v} = \overline{\mathsf{L}} = \mathsf{O}\,.$$

Proof of Lemma 4.1. Let $\imath_1$ and $\imath_2$ be the two injective embedding mappings used in the definition of $R+S$. First we show the auxiliary equation

$$\overline{R \,@\, S} = \overline{R} \,@\, \overline{S}\,, \tag{15}$$

i.e., $\overline{R\imath_1 \cup S\imath_2} = \overline{R}\imath_1 \cup \overline{S}\imath_2$: The axioms (7) imply that $\imath_1$ and $\imath_2$ are surjective. Therefore, we get

$$\mathsf{L} = \mathsf{L}\imath_1 \cup \mathsf{L}\imath_2 = (R \cup \overline{R})\imath_1 \cup (S \cup \overline{S})\imath_2 = \overline{R}\imath_1 \cup \overline{S}\imath_2 \cup R\imath_1 \cup S\imath_2$$

which is equivalent to $\overline{R\imath_1 \cup S\imath_2} \subseteq \overline{R}\imath_1 \cup \overline{S}\imath_2$. To prove the remaining inclusion we start with the inclusions $(R\imath_1 \cup S\imath_2)\imath_1{}^{\mathsf{T}} \subseteq R$ and $(R\imath_1 \cup S\imath_2)\imath_2{}^{\mathsf{T}} \subseteq S$ which hold due to the axioms (7). Now, the Schröder equivalences yield $\overline{R}\imath_1 \subseteq \overline{R\imath_1 \cup S\imath_2}$ and $\overline{S}\imath_2 \subseteq \overline{R\imath_1 \cup S\imath_2}$ and these inclusions yield $\overline{R}\imath_1 \cup \overline{S}\imath_2 \subseteq \overline{R\imath_1 \cup S\imath_2}$.

(i) First we use the definition of the symmetric quotient, the above auxiliary equation (15), and a de Morgan law and obtain

$$\begin{aligned}
\mathsf{syq}(R \,@\, S, R \,@\, S) &= \overline{(R \,@\, S)^{\mathsf{T}}(\overline{R} \,@\, \overline{S})} \cap \overline{(\overline{R} \,@\, \overline{S})^{\mathsf{T}}(R \,@\, S)} \\
&= \overline{(R \,@\, S)^{\mathsf{T}}(\overline{R} \,@\, \overline{S}) \cup (\overline{R} \,@\, \overline{S})^{\mathsf{T}}(R \,@\, S)}\,.
\end{aligned}$$

From this equation we get

$$\begin{aligned}
&\mathsf{syq}(R \,@\, S, R \,@\, S) \\
= \;& \overline{(R\imath_1 \cup S\imath_2)^{\mathsf{T}}(\overline{R}\imath_1 \cup \overline{S}\imath_2) \cup (\overline{R}\imath_1 \cup \overline{S}\imath_2)^{\mathsf{T}}(R\imath_1 \cup S\imath_2)} \\
= \;& \overline{\imath_1{}^{\mathsf{T}}(R^{\mathsf{T}}\overline{R} \cup \overline{R}^{\mathsf{T}}R)\imath_1 \cup \imath_2{}^{\mathsf{T}}(S^{\mathsf{T}}\overline{S} \cup \overline{S}^{\mathsf{T}}S)\imath_2 \cup \imath_1{}^{\mathsf{T}}(\ldots)\imath_2 \cup \imath_2{}^{\mathsf{T}}(\ldots)\imath_1} \\
= \;& \overline{\imath_1{}^{\mathsf{T}}\,\overline{\mathsf{syq}(R,R)}\,\imath_1 \cup \imath_2{}^{\mathsf{T}}\,\overline{\mathsf{syq}(S,S)}\,\imath_2 \cup \imath_1{}^{\mathsf{T}}\,\overline{\mathsf{syq}(R,S)}\,\imath_2 \cup \imath_2{}^{\mathsf{T}}\,\overline{\mathsf{syq}(S,R)}\,\imath_1}
\end{aligned}$$

if we apply the definitions of the concatenation operation and of the symmetric quotient, distributivity, and de Morgan's laws. Now, the desired result follows from the subsequent calculation:

$$
\begin{aligned}
\mathsf{syq}(R \mathbin{@} S, R \mathbin{@} S) &= \overline{\imath_1{}^{\mathsf{T}} \overline{\mathsf{I}} \imath_1 \cup \imath_2{}^{\mathsf{T}} \overline{\mathsf{I}} \imath_2 \cup \imath_1{}^{\mathsf{T}} \mathsf{L} \imath_2 \cup \imath_2{}^{\mathsf{T}} \mathsf{L} \imath_1} && \text{assumptions} \\
&= \overline{(\imath_1{}^{\mathsf{T}} \overline{\mathsf{I}} \cup \imath_2{}^{\mathsf{T}} \mathsf{L})\imath_1 \cup (\imath_2{}^{\mathsf{T}} \overline{\mathsf{I}} \cup \imath_1{}^{\mathsf{T}} \mathsf{L})\imath_2} && \\
&= \overline{(\imath_1{}^{\mathsf{T}} \overline{\mathsf{I}} \cup \imath_2{}^{\mathsf{T}} \mathsf{L}) \mathbin{@} (\imath_2{}^{\mathsf{T}} \overline{\mathsf{I}} \cup \imath_1{}^{\mathsf{T}} \mathsf{L})} && \text{definition } @ \\
&= \overline{\imath_1{}^{\mathsf{T}} \overline{\mathsf{I}} \cup \imath_2{}^{\mathsf{T}} \mathsf{L}} \mathbin{@} \overline{\imath_2{}^{\mathsf{T}} \overline{\mathsf{I}} \cup \imath_1{}^{\mathsf{T}} \mathsf{L}} && (15) \\
&= \overline{\overline{\mathsf{I}} \imath_1 \cup \mathsf{L} \imath_2}^{\mathsf{T}} \mathbin{@} \overline{\overline{\mathsf{I}} \imath_2 \cup \mathsf{L} \imath_1}^{\mathsf{T}} && \\
&= \overline{\overline{\mathsf{I}} \mathbin{@} \mathsf{L}}^{\mathsf{T}} \mathbin{@} \overline{\mathsf{L} \mathbin{@} \overline{\mathsf{I}}}^{\mathsf{T}} && \text{definition } @ \\
&= (\mathsf{I} \mathbin{@} \mathsf{O})^{\mathsf{T}} \mathbin{@} (\mathsf{O} \mathbin{@} \mathsf{I})^{\mathsf{T}} && (15) \\
&= \imath_1{}^{\mathsf{T}} \mathbin{@} \imath_2{}^{\mathsf{T}} && \text{definition } @ \\
&= \imath_1{}^{\mathsf{T}} \imath_1 \cup \imath_2{}^{\mathsf{T}} \imath_2 && \text{definition } @ \\
&= \mathsf{I} && (7)\,.
\end{aligned}
$$

(ii) This proof is very similar to the proof of (i), i.e., uses also only the definitions of the operators syq and $@$, some elementary relation-algebraic transformations, and the above auxiliary equation (15). It is left to the reader.

(iii) From the axioms (7) we immediately get that $\imath_1$ and $\imath_2$ are total relations. Using this in the last step, the following calculation shows the desired equation:

$$(R \mathbin{@} S)\mathsf{L} = (R\imath_1 \cup S\imath_2)\mathsf{L} = R\imath_1\mathsf{L} \cup S\imath_2\mathsf{L} = R\mathsf{L} \cup S\mathsf{L}\,.$$