Relational Characterisations of Paths

Rudolf Berghammer[a], Hitoshi Furusawa[b], Walter Guttmann[c], Peter Höfner[d,e,*]

[a] Institut für Informatik, Christian-Albrechts-Universität zu Kiel, Germany
[b] Department of Science, Kagoshima University, Japan
[c] Department of Computer Science and Software Engineering, University of Canterbury, New Zealand
[d] Data61, CSIRO, Sydney, Australia
[e] Research School of Computer Science, Australian National University, Australia

Abstract

Binary relations are one of the standard ways to encode, characterise and reason about graphs. Relation algebras provide equational axioms for a large fragment of the calculus of binary relations. Although relations are standard tools in many areas of mathematics and computing, researchers usually fall back to point-wise reasoning when it comes to arguments about paths in a graph. We present a purely algebraic way to specify different kinds of paths in Kleene relation algebras, which are relation algebras equipped with an operation for reflexive transitive closure. We study the relationship between paths with a designated root vertex and paths without such a vertex. Since we stay in first-order logic this development helps with mechanising proofs. To demonstrate the applicability of the algebraic framework we verify the correctness of three basic graph algorithms. All results of this paper are formally verified using the interactive proof assistant Isabelle/HOL.

Keywords: algorithms, cycles, graphs, Kleene algebras, paths, relation algebras, verification

1. Introduction

Paths are a fundamental concept in graph theory and its applications. Many textbooks define a path in a directed graph to be a sequence of vertices such that successive vertices in the sequence are connected by an edge of the graph; see [4, 16, 31]. A simple path is one whose vertices are distinct; a cycle is one whose first and last vertices are identical. There are variations in terminology, but definitions of paths are mostly based on sequences.

An alternative approach is to define a path as a connected subgraph of edges such that every vertex has at most one incoming edge and at most one outgoing edge; this is considered, for example, in [22, 68]. The aim of the present paper is to derive a theory of paths in directed graphs based on this alternative definition and to show how it can be used to verify the correctness of graph algorithms. Paths, according to this definition, correspond to simple paths when considered as sequences of vertices; this paper is not concerned with more general kinds of paths.

The main motivation for using subgraphs instead of sequences is that the former allow us to reason about paths as special kinds of graphs using the well-established framework of relation algebra. Specifically, the edge set E of a directed graph is a subset of the Cartesian product $A \times A$, where A is the set of vertices, and therefore E is a (homogeneous) relation on the set A. Hence both paths and graphs correspond to relations, and relational concepts and methods can be used to work with them in a unified setting. We are particularly interested in exploiting the algebraic structure of relations, in order to concisely express properties of graphs and reason about graphs using equational, calculational proofs. Such proofs can be easily mechanised and formally verified using automated theorem provers and interactive proof assistants [1, 10, 13, 24, 34, 58]. To express reachability and connectivity in graphs, we add to relation algebras the well-known Kleene star operation for reflexive transitive closure, using the axioms of Kleene algebras [40].

Relation algebras have been proposed by De Morgan, Peirce, Schröder and Tarski to express a rich fragment of first-order logic algebraically without using variables and quantifiers [67]; recent textbooks are [32, 46]. Besides for

[*] Corresponding author: Tel.: +61 2 6125 0159, Email: Peter.Hoefner@anu.edu.au

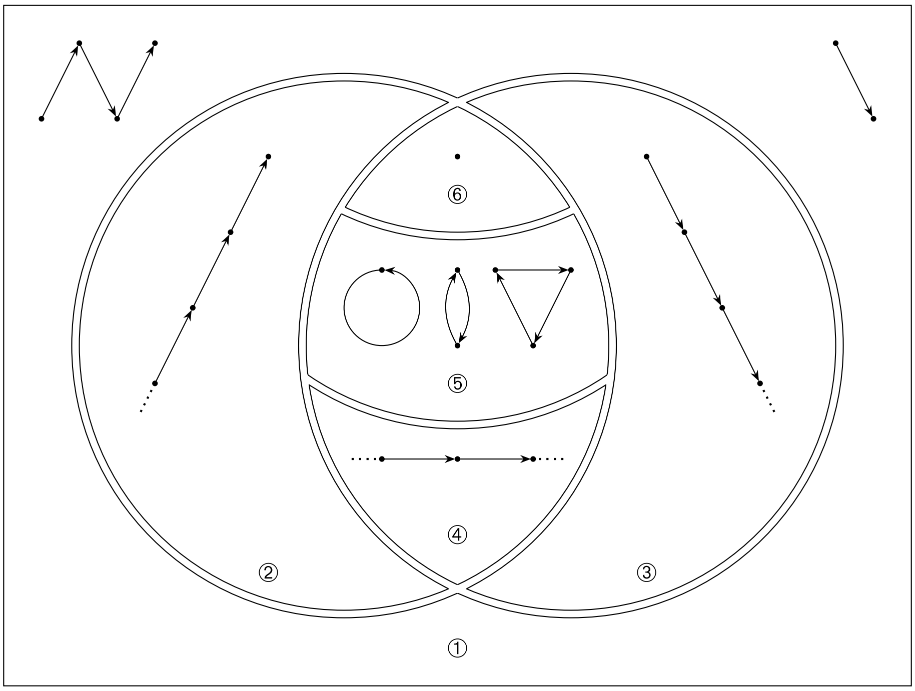


Figure 1: Six disjoint classes of paths

logical foundations, relational methods have been used for program analysis [21, 27], refinement [72], databases [56], preference modelling [50, 62], algorithm development [2, 5, 25, 28] and many other applications; for example, see [19, 20, 49, 51, 61, 66].

Relation algebras have been extended with an operation for transitive closure in [52]. We follow an alternative approach which simply combines relation algebras with Kleene algebras. We call the resulting structures Kleene relation algebras. The term 'Kleene relation algebra' was previously used by Crvenković and Madarász for the algebra of relations with reflexive transitive closure in [17].

Relations and Kleene relation algebras have been used for a wide range of topics in graph theory [8, 11, 26, 60, 64]. When it comes to reasoning about paths in a graph, however, researchers usually fall back to point-wise reasoning. By defining paths as subgraphs, we can reuse the well-developed theories of Kleene algebras and relation algebras and reason about paths without resorting to variables and quantifiers.

This paper studies paths using algebraic means, specifically, using Kleene relation algebras. Its contributions are:

- Equational characterisations of various classes of paths including cycles, finite paths, one-sided and two-sided infinite paths in Sections 3–5. An overview of the classification is shown in Figure 1 and discussed below.
- Algebraic specifications and correctness proofs of a number of basic graph algorithms that rely on paths in Section 6.
- Algebraic characterisations of paths with designated roots and their equivalence to paths without roots in Section 7.

All concepts, theorems and algorithms described in this paper have been implemented in Isabelle/HOL [55]. All results have been formally verified in this system making heavy use of its integrated automated theorem provers and SMT solvers [15, 57]. We omit most of the proofs, which can be found in the theory files [30].

Figure 1 shows the classes of paths discussed in this paper:

① Finite paths, which have both a start vertex and an end vertex, and finitely many edges.

② One-sided infinite paths that have an end vertex but no start vertex.

③ One-sided infinite paths that have a start vertex but no end vertex.

④ Two-sided infinite paths, which have neither a start vertex nor an end vertex, but infinitely many edges.

⑤ Cycles, which have neither a start vertex nor an end vertex, and at least one but finitely many edges.

⑥ The empty path, which has neither a start vertex nor an end vertex, and no edges.

We refer to unions of these disjoint classes by listing the associated labels; for example, [①③] are the paths that have a start vertex. The left circle of Figure 1 contains the paths without a start vertex [②④⑤⑥] and the right circle those without an end vertex [③④⑤⑥]. Paths in this setting are specified by edges, from which start vertex and end vertex are derived. It is not possible to represent a path with one vertex and no edge. In particular, [⑥] shows the path with no edges (the empty relation) as a subgraph of an overall graph with one vertex (a one-element base set). The shown vertex belongs to the base set, not to the path.

2. Kleene Relation Algebras

In this section we give basic definitions, operations and properties of relations. Their structure is captured abstractly by relation algebras, which let us characterise important properties of relations in a compact way using equations and inequalities. We explain how to represent edges, vertices and sets of vertices of a graph as relations. To describe reachability in graphs, we recall first-order axioms of Kleene algebras, which we combine with relation algebras. Finally, we discuss tool support for mechanising the presented concepts.

2.1. Relations

A (concrete) binary relation R on a set A is a subset of the Cartesian product $A \times A$, that is, a set of ordered pairs of elements of A. Using the language of graph theory, A is the set of vertices of the graph and R is the set of directed edges. In this context, we also call R a graph. Binary relations can furthermore be understood as Boolean matrices with rows and columns indexed by A, which corresponds to the adjacency matrix representation of graphs.

Let R and S be binary relations on a set A. Then the union $R \cup S$, the intersection $R \cap S$ and the complement $\overline{R}$ (relative to $A \times A$) are binary relations on A. The *empty relation* O on A is the empty set and the *universal relation* L on A is the full Cartesian product $A \times A$. The set of all binary relations on A with the operations $\cup$, $\cap$, $\overline{}$, O and L forms a Boolean algebra. The *composition* $R \,;\, S$ of two relations R and S is the set of all pairs $(a, c) \in A \times A$ such that $(a, b) \in R$ and $(b, c) \in S$ for some $b \in A$. The *converse* or *transpose* R^{T} of a relation R is the set of all pairs (a, b) with $(b, a) \in R$. The *identity relation* I on A is the set of all pairs (a, a) with $a \in A$.

The structure $(2^{A \times A}, \cup, ;, \overline{}, ^{\mathsf{T}}, \mathsf{I})$ is called the concrete relation algebra of all binary relations over A. The operations $\cap$, O and L can be defined in terms of the other operations. To discuss the algebraic structure of relations in a more abstract way, binary relations are replaced by arbitrary elements of a carrier set B, operations on B are introduced and they are axiomatised by equations as follows.

An *(abstract) relation algebra* is a structure $(B, \cup, ;, \overline{}, ^{\mathsf{T}}, \mathsf{I})$ satisfying the axioms [46, 67]

$$
\begin{array}{ccc}
(Q \cup R) \cup S = Q \cup (R \cup S) & R \cup S = S \cup R & R = \overline{\overline{R} \cup \overline{S}} \cup \overline{\overline{R} \cup S} \\
(Q \,;\, R) \,;\, S = Q \,;\, (R \,;\, S) & (Q \cup R) \,;\, S = Q \,;\, S \cup R \,;\, S & R \,;\, \mathsf{I} = R \\
R^{\mathsf{T}\mathsf{T}} = R & (R \cup S)^{\mathsf{T}} = R^{\mathsf{T}} \cup S^{\mathsf{T}} & (R \,;\, S)^{\mathsf{T}} = S^{\mathsf{T}} \,;\, R^{\mathsf{T}} \\
& R^{\mathsf{T}} \,;\, \overline{R \,;\, S} \cup \overline{S} = \overline{S} &
\end{array}
$$

The first line contains Huntington's axioms for Boolean algebras [35, 36]. The join operation is denoted by $\cup$, based on which the meet can be defined as $R \cap S = \overline{\overline{R} \cup \overline{S}}$. Since every relation algebra is a Boolean algebra, the set B is partially ordered by $R \subseteq S \Leftrightarrow R \cup S = S$ with greatest element $\mathsf{L} = R \cup \overline{R}$ and least element $\mathsf{O} = R \cap \overline{R}$. In the graph model, O is a graph with no edges and L is a complete graph. The axioms in the second line give properties of composition ;. It follows that every relation algebra is a semiring with the two operations $\cup$ and ;. The axioms in the last two lines specify the operation of converse. We assume that composition has higher precedence than join and meet and that complement and converse have higher precedence than composition.

It follows that the operations $\cup$, $\cap$, $;$ and $^\mathsf{T}$ preserve the order $\subseteq$ and the operation $\overline{}$ reverses the order $\subseteq$. As further examples, we discuss two properties which follow from the above axioms of relation algebras:

$$R \subseteq R \mathbin{;} R^\mathsf{T} \mathbin{;} R \tag{A}$$

$$R \cap \mathsf{I} = R^\mathsf{T} \cap \mathsf{I} \tag{B}$$

We interpret these properties in the graph model. Inequality (A) holds because any edge $(a, b) \in R$ in the graph can be traversed backwards in R^T, since $(b, a) \in R^\mathsf{T}$, so the edge (a, b) is also contained in the composition $R \mathbin{;} R^\mathsf{T} \mathbin{;} R$ by going forward, backward and forward again from a via b and a to b. Equality (B) contains an intersection with the identity relation on both sides; it therefore considers only loops in the graph, that is, edges from a vertex to itself. It states that such edges are not changed if a graph is transposed, that is, if all its edges are reversed.

Many further properties of relation algebras can be found in textbooks such as [46, 60, 64].

2.2. Relational Properties

Compact equational characterisations of special classes of relations can be given in relation algebras. We use the following properties of functions and orders.

A relation R is *univalent* if $R^\mathsf{T} \mathbin{;} R \subseteq \mathsf{I}$ and *total* if $\mathsf{I} \subseteq R \mathbin{;} R^\mathsf{T}$. A relation R is *injective* if R^T is univalent, *surjective* if R^T is total and *bijective* if R is injective and surjective. A relation R is *irreflexive* if $R \subseteq \overline{\mathsf{I}}$ and *symmetric* if $R = R^\mathsf{T}$.

For concrete relations, the equivalence of these relation-algebraic specifications and the common logical specifications can be easily derived. For example, $R \subseteq A \times A$ is univalent if and only if for each $a \in A$ there is at most one $b \in A$ such that $(a, b) \in R$. If R is interpreted as the edge set of a directed graph with vertex set A, this means that the out-degree of every vertex is at most 1. Similarly, R is total if and only if the out-degree of every vertex is at least 1. Conversely, injective and surjective state the same requirements for the in-degree of vertices instead of their out-degree. Irreflexivity specifies that a graph contains no loops. Symmetric relations are sometimes used to represent undirected graphs by containing both (a, b) and (b, a) if there is an edge between a and b.

A relation R is surjective if and only if $\mathsf{L} \mathbin{;} R = \mathsf{L}$. As another consequence of the above properties, we present the following result about injective and surjective relations. For this, and following results, we assume that variables range over a given relation algebra.

Lemma 2.1. *If P is surjective and R is injective, $P \subseteq Q \mathbin{;} R$ implies $R \subseteq Q^\mathsf{T} \mathbin{;} P$. These two inequalities are equivalent if P and R are bijective.*

2.3. Vectors, Points and Singletons

We now discuss three particular properties that are useful to represent sets of elements as relations. A relation v is a *vector* if $v = v \mathbin{;} \mathsf{L}$. A *point* is a bijective vector. A relation x is a *singleton* if both $x \mathbin{;} \mathsf{L}$ and $x^\mathsf{T} \mathbin{;} \mathsf{L}$ are points.

In the matrix model, a vector corresponds to a row-constant matrix. That is, $v \subseteq A \times A$ is a vector if and only if for every $a \in A$, the pair (a, b) is in v either for all $b \in A$ or for none. Such a relation is used to model the subset of elements of A that are related by v to all elements of A. In the graph model, this can be used to represent sets of vertices.

A point is a vector that is additionally injective and surjective. In the graph model, this means that the in-degree of every vertex is exactly 1, so the adjacency matrix contains exactly one row with 1-entries. In other words, a point represents a set of vertices that contains exactly one element. Such sets obviously correspond to elements of A and can therefore be used to represent individual vertices of graphs.

A singleton is a relation consisting of a single pair. Specifically, if R is a singleton and $(a, b) \in R$, then $R \mathbin{;} \mathsf{L}$ is the point representing the element a, and similarly $R^\mathsf{T} \mathbin{;} \mathsf{L}$ represents the element b. Hence a singleton corresponds to a single edge in a graph.

The composition $R \mathbin{;} \mathsf{L}$ is a vector for any relation R. More generally, the composition $R \mathbin{;} v$ of a relation R and a vector v is again a vector. It represents the set of vertices from which there are transitions under R into the set represented by v. In the graph model this amounts to the predecessors of the vertices in the set represented by v. Similarly, $R^\mathsf{T} \mathbin{;} v$ is the set of successors of the vertices in the set represented by v.

The following result gives an example of a property of vectors.

Lemma 2.2. *For vectors v, w, we have $v ; w^{\mathsf{T}} = v \cap w^{\mathsf{T}}$. In particular $R ; \mathsf{L} ; S = R ; \mathsf{L} \cap \mathsf{L} ; S$ for all relations R, S.*

In the remainder of the paper we will use two further properties of relations, which are valid for concrete relations, but do not follow from the axioms of relation algebras given in Section 2.1. The first one is the *Tarski rule*:

$$R \neq \mathsf{O} \Leftrightarrow \mathsf{L} ; R ; \mathsf{L} = \mathsf{L}$$

It can be interpreted as follows. If $R \subseteq A \times A$ is not empty, it contains a pair (a, b) for some $a, b \in A$. Then the vector $R ; \mathsf{L}$ represents a set that contains a; in particular, $R ; \mathsf{L}$ contains a point. Hence $\mathsf{L} ; R ; \mathsf{L}$ is the universal relation since points are surjective and composition preserves the order $\subseteq$. The backward implication of the Tarski rule is equivalent to $\mathsf{O} \neq \mathsf{L}$, which holds if and only if the set of vertices of a graph is not empty.

Using the Tarski rule it can be shown that a relation R is a point if and only if R is a vector, R is injective and $R \neq \mathsf{O}$, which is the definition given in [63]. It can then also be shown that R is a singleton if and only if $R \neq \mathsf{O}$ and $R^{\mathsf{T}} ; \mathsf{L} ; R \cup R ; \mathsf{L} ; R^{\mathsf{T}} \subseteq \mathsf{I}$, which is the definition used in [45].

The second property is the *point axiom*. It states that for each $R \neq \mathsf{O}$ there are points p and q such that $p ; q^{\mathsf{T}} \subseteq R$. Again, by the assumption there are $a, b \in A$ such that $(a, b) \in R$. Then points p and q can be chosen so as to represent the vertices a and b, respectively. By Lemma 2.2 we have $p ; q^{\mathsf{T}} = p \cap q^{\mathsf{T}}$, whence this expression intersects row a and column b of the matrix to yield the entry (a, b). In particular, it follows that $p ; q^{\mathsf{T}}$ is a singleton. To summarise, the point axiom states that every non-empty graph contains an edge, which is obvious in the graph model but an independent property in relation algebras.

In relation algebras satisfying the Tarski rule and the point axiom, a singleton is an atom in the lattice-theoretical sense, taking $\subseteq$ as the underlying partial order. Due to this, singletons are sometimes called (relational) atoms, for example, in [5]. In such algebras there is also a non-zero univalent element below every non-zero element. According to [43] this implies that such a relation algebra is representable, that is, isomorphic to an algebra of (concrete) binary relations and relational operations. Similar conditions for representability have been discussed in [37]. The point axiom has been studied in [63]; see [44] for the similar notion of pair-dense relation algebras.

2.4. *Reflexive Transitive Closure*

To describe reachability in graphs we expand relation algebras with an operation for the reflexive transitive closure. Relation algebras have been extended with an operation for transitive closure in [52]. We follow an alternative approach which simply combines relation algebras with Kleene algebras.

A *Kleene algebra* is a structure $(K, \cup, ;, {}^*, \mathsf{O}, \mathsf{I})$ satisfying the axioms [40]

$$\begin{array}{cccc}
(Q \cup R) \cup S = Q \cup (R \cup S) & (Q \cup R) ; S = Q ; S \cup R ; S & (Q ; R) ; S = Q ; (R ; S) & \mathsf{I} \cup R ; R^* \subseteq R^* \\
R \cup S = S \cup R & Q ; (R \cup S) = Q ; R \cup Q ; S & \mathsf{I} ; R = R & \mathsf{I} \cup R^* ; R \subseteq R^* \\
R \cup R = R & \mathsf{O} ; R = \mathsf{O} & R ; \mathsf{I} = R & Q \cup R ; S \subseteq S \Rightarrow R^* ; Q \subseteq S \\
R \cup \mathsf{O} = R & R ; \mathsf{O} = \mathsf{O} & & Q \cup S ; R \subseteq S \Rightarrow Q ; R^* \subseteq S
\end{array}$$

where $\subseteq$ is the partial order defined by $R \subseteq S \Leftrightarrow R \cup S = S$. The axioms listed in the first three columns also hold in relation algebras, which makes it easy to combine them with Kleene algebras.

A *Kleene relation algebra* is a structure $(M, \cup, ;, \overline{}, {}^{\mathsf{T}}, {}^*, \mathsf{O}, \mathsf{I})$ such that $(M, \cup, ;, \overline{}, {}^{\mathsf{T}}, \mathsf{I})$ is a relation algebra and $(M, \cup, ;, {}^*, \mathsf{O}, \mathsf{I})$ is a Kleene algebra. In fact, it would be sufficient to add to relation algebras the operation * with the axioms $\mathsf{I} \cup R ; R^* \subseteq R^*$ and $Q \cup R ; S \subseteq S \Rightarrow R^* ; Q \subseteq S$ as the others can then be derived.

The operation * models reflexive transitive closure, from which the transitive closure is easily obtained by $R^+ = R ; R^*$. In concrete relation algebras $R^* = \bigcup_{i \geq 0} R^i$ and $R^+ = \bigcup_{i \geq 1} R^i$.

An edge (a, b) is in R^* if and only if there is a path from vertex a to vertex b in the graph R. If $(a, b) \in R^*$ we say vertex b *is reachable from* vertex a. A similar statement holds for R^+ and non-empty paths. A relation R is *acyclic* if R^+ is irreflexive, that is, there is no non-empty path from a vertex to itself.

For example, we obtain the following consequences:

$$R^* \cup S^* = R^+ \cup S^*$$
$$R^{\mathsf{T}*} = R^{*\mathsf{T}}$$

In the remainder of this paper we work in Kleene relation algebras that satisfy the Tarski rule and the point axiom. We note, however, that some of the following results do not require the Tarski rule and/or the point axiom but hold in more general structures, as can be seen in our Isabelle/HOL implementation.

2.5. Formalisation and Mechanisation

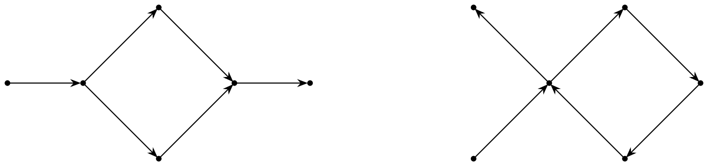
As shown above, Kleene relation algebras facilitate compact equational specifications of relational properties and support first-order axioms for reachability, which is a key notion for dealing with paths. All relation-algebraic statements can be expressed in logic, but doing so would introduce complex formulas with nested quantifiers. For example, a relation $R \subseteq A \times A$ is surjective if and only if $\forall y \in A : \exists x \in A : (x, y) \in R$ and R is injective if and only if $\forall y \in A : \forall x \in A : \forall w \in A : (x, y) \in R \wedge (w, y) \in R \Rightarrow x = w$. In relation algebras, these properties are expressed as $\mathsf{I} \subseteq R^{\mathsf{T}} ; R$ and $R ; R^{\mathsf{T}} \subseteq \mathsf{I}$, respectively. Like here, relation algebras often yield simpler, more modular and more concise specifications and proofs. This is one reason why we formalise our work using relation algebras rather than first-order logic over elements of the base set.

Due to their compact quantifier-free form, statements given in relation algebras can be more easily tackled by automated and interactive theorem provers. Off-the-shelf automated theorem provers such as E [65] and Prover9 [47] are performing well when proving statements containing relational expressions [18, 34]. There are also special-purpose first-order proof systems for relation algebras [41, 42]. Automated theorem provers, however, usually fail when proving complex properties of relation algebras since the search space becomes too large.

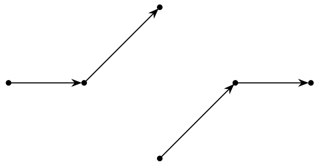
To overcome this deficiency, interactive proof assistants such as Coq [14] and Isabelle/HOL [55] can be used. Both systems have been successfully used in the context of Kleene algebras and relation algebras [1, 58]. Key differences are that Coq has a more expressive type system whereas Isabelle/HOL has better automation support. In particular, the Sledgehammer tool [48, 57] integrates first-order automatic theorem provers and SMT solvers to discharge goals arising in interactive Isabelle/HOL proofs. Since our work does not require an elaborate type system but greatly benefits from automation we have implemented all results of this paper in Isabelle/HOL. Proofs are omitted and can be found in the accompanying Isabelle theories. To improve readability, most of our proofs are written in Isabelle/Isar (Intelligible semi-automated reasoning) [70]. In Appendix A we illustrate that Isar proofs are indeed human-readable by comparing them with how they would be written in a textbook.

3. Paths

In this section we start our endeavour to characterise classes of paths. Observe that a path neither forks into edges to more than one successor nor merges edges from more than one predecessor. For example, neither of the two graphs depicted in the following diagram should be considered a path.



To prevent forking and merging we assume that a path is a univalent and injective relation. However, these two restrictions are not sufficient to characterise paths since they allow, for instance, the following graph which contains two components:



We additionally have to guarantee that all edges are connected, which can be expressed by

$$R ; \mathsf{L} ; R \subseteq R^* \cup R^{\mathsf{T}*} \tag{C}$$

The expression $R ; \mathsf{L} ; R$ is equivalent to $R ; \mathsf{L} \cap \mathsf{L} ; R$ by Lemma 2.2. In the graph model, it describes all pairs (a, b) such that vertex a has an outgoing edge in R and vertex b has an incoming edge in R. The inequality requires that there must be a path from a to b in R or a path from b to a in R. This ensures that the graph R has at most one non-trivial component. We therefore define paths as follows.

Definition 3.1. *A relation R is a* path *if R is injective, univalent and satisfies* (C)*, that is, $R ; \mathsf{L} ; R \subseteq R^* \cup R^{\mathsf{T}*}$.*

Paths in this sense encompass all classes shown in Figure 1 [①②③④⑤⑥], and nothing else. In particular, a path can be finite, one-sided infinite, two-sided infinite, a cycle or even the empty relation. In the remainder of the paper we will look at each class separately.

Theorem 3.2. *R satisfies* (C) *if and only if R^{T} does. R is a path if and only if R^{T} is a path.*

The following theorem shows equivalent formalisations of inequality (C).

Theorem 3.3. *For a univalent and injective relation R, the following properties are equivalent:*

(1) $R ; \mathsf{L} ; R \subseteq R^* \cup R^{\mathsf{T}*}$
(2) $R ; \mathsf{L} ; R^{\mathsf{T}} \subseteq R^* \cup R^{\mathsf{T}*}$
(3) $R^{\mathsf{T}} ; \mathsf{L} ; R \subseteq R^* \cup R^{\mathsf{T}*}$
(4) $R^{\mathsf{T}} ; \mathsf{L} ; R^{\mathsf{T}} \subseteq R^* \cup R^{\mathsf{T}*}$
(5) $R^+ ; \mathsf{L} ; R^+ \subseteq R^* \cup R^{\mathsf{T}*}$
(6) $R^+ ; \mathsf{L} ; R^{\mathsf{T}+} \subseteq R^* \cup R^{\mathsf{T}*}$
(7) $R^{\mathsf{T}+} ; \mathsf{L} ; R^+ \subseteq R^* \cup R^{\mathsf{T}*}$
(8) $R^{\mathsf{T}+} ; \mathsf{L} ; R^{\mathsf{T}+} \subseteq R^* \cup R^{\mathsf{T}*}$

4. Start Points and End Points

Many paths have special vertices, namely a start point (root) and an end point (sink). A start point is a vertex without predecessor, and an end point is a vertex without successor. This is captured by the following definition.

Definition 4.1. *The* start points *of a relation R are given by $\mathsf{start}(R) = R ; \mathsf{L} \cap \overline{R^{\mathsf{T}} ; \mathsf{L}}$ and its* end points *are given by $\mathsf{end}(R) = R^{\mathsf{T}} ; \mathsf{L} \cap \overline{R ; \mathsf{L}}$.*

As above, $R ; \mathsf{L}$ describes all vertices having a successor, and $R^{\mathsf{T}} ; \mathsf{L}$ those vertices with a predecessor. Hence, $R ; \mathsf{L} \cap \overline{R^{\mathsf{T}} ; \mathsf{L}}$ characterises the vertices with at least one successor and no predecessors, which are just the start points of a path.

Immediate consequences are $\mathsf{start}(R) = \mathsf{end}(R^{\mathsf{T}})$ and $R ; \mathsf{start}(R) = R^{\mathsf{T}} ; \mathsf{end}(R) = \mathsf{O}$. Of course, not every path has such distinguished elements. In Figure 1, the classes [①③] comprise the paths with start points. Similarly, classes [①②] are the paths with end points. A cycle or a two-sided infinite path has neither start points nor end points, in which case $\mathsf{start}(R) = \mathsf{end}(R) = \mathsf{O}$. The following result shows that every path has at most one start point and at most one end point.

Theorem 4.2. *Let R be a path. Then $\mathsf{start}(R)$ and $\mathsf{end}(R)$ are injective.*

It follows that start points and end points are almost points in the relation-algebraic sense. The only exception is the empty relation as the following result states.

Corollary 4.3. *Let R be a path. Then $\mathsf{start}(R) = \mathsf{O}$ or $\mathsf{start}(R)$ is a point and, similarly, $\mathsf{end}(R) = \mathsf{O}$ or $\mathsf{end}(R)$ is a point.*

The following consequences characterise the existence of start and end points by (in)equalities. We also give the corresponding classes in Figure 1.

Corollary 4.4. *Let R be a path. Then*

(1) $\mathsf{start}(R) \neq \mathsf{O}$ *if and only if* $\mathsf{L} = \overline{\mathsf{L} ; R} ; R ; \mathsf{L}$ [①③]
(2) $\mathsf{start}(R) = \mathsf{O}$ *if and only if* $R ; \mathsf{L} \subseteq R^{\mathsf{T}} ; \mathsf{L}$ [②④⑤⑥]
(3) $\mathsf{end}(R) \neq \mathsf{O}$ *if and only if* $\mathsf{L} = \mathsf{L} ; R ; \overline{R ; \mathsf{L}}$ [①②]

(4) $\mathsf{end}(R) = \mathsf{O}$ *if and only if* $R^{\mathsf{T}} ; \mathsf{L} \subseteq R ; \mathsf{L}$ [③④⑤⑥]

(5) $\mathsf{start}(R) \neq \mathsf{O}$ *and* $\mathsf{end}(R) \neq \mathsf{O}$ *if and only if* $\mathsf{L} = \overline{\mathsf{L} ; R} ; R ; \mathsf{L} \cap \mathsf{L} ; R ; \overline{R ; \mathsf{L}}$ [①]

(6) $\mathsf{start}(R) = \mathsf{end}(R) = \mathsf{O}$ *if and only if* $R ; \mathsf{L} = R^{\mathsf{T}} ; \mathsf{L}$ [④⑤⑥]

It follows that a path has a start point if and only if the converse path has an end point.

The simplest kind of path that has both a start point and an end point is a single edge. It can be constructed from two (relational) points as shown by the following result.

Lemma 4.5. *Let p and q be points. Then $p ; q^{\mathsf{T}}$ is a path. If $p \neq q$, then $\mathsf{start}(p ; q^{\mathsf{T}}) = p$ and $\mathsf{end}(p ; q^{\mathsf{T}}) = q$.*

The start point of the constructed edge is the vertex represented by the point p; its end point is the one represented by q. If p and q coincide, $\mathsf{start}(p ; q^{\mathsf{T}}) = \mathsf{end}(p ; q^{\mathsf{T}}) = \mathsf{O}$ since the result is a loop.

A path with no start point and no end point [④⑤⑥] need not be infinite; it can also be the empty relation or a cycle. We will elaborate on this distinction in Section 5.

Sometimes we wish to include the empty relation in our reasoning. If a path has either a start point or is empty, we call it *backward terminating* [①③⑥]. Symmetrically, we call a path *forward terminating* if it has an end point or is empty [①②⑥]. If a path is both forward terminating and backward terminating, we call it *terminating* [①⑥]. The following result shows how to express each of these properties as an inequality.

Theorem 4.6. *Let R be a path. Then the following properties are equivalent and each characterises backward termination.*

(1) $\mathsf{L} = \overline{\mathsf{L} ; R} ; R ; \mathsf{L}$ *or* $R = \mathsf{O}$

(2) $R \subseteq \overline{\mathsf{L} ; R} ; R ; \mathsf{L}$

(3) $R \subseteq \overline{\mathsf{L} ; R} ; R^*$

(4) $R \subseteq R^{\mathsf{T}*} ; \overline{R^{\mathsf{T}} ; \mathsf{L}}$

Moreover, each of the following properties is equivalent to forward termination.

(5) $\mathsf{L} = \mathsf{L} ; R ; \overline{R ; \mathsf{L}}$ *or* $R = \mathsf{O}$

(6) $R \subseteq \mathsf{L} ; R ; \overline{R ; \mathsf{L}}$

(7) $R \subseteq R^* ; \overline{R ; \mathsf{L}}$

(8) $R \subseteq \overline{\mathsf{L} ; R^{\mathsf{T}}} ; R^{\mathsf{T}*}$

Finally, each of the following properties is equivalent to termination.

(9) $\mathsf{L} = \overline{\mathsf{L} ; R} ; R ; \mathsf{L} \cap \mathsf{L} ; R ; \overline{R ; \mathsf{L}}$ *or* $R = \mathsf{O}$

(10) $R \subseteq \overline{\mathsf{L} ; R} ; R ; \mathsf{L} \cap \mathsf{L} ; R ; \overline{R ; \mathsf{L}}$

(11) $R \subseteq \overline{\mathsf{L} ; R} ; R^* \cap R^* ; \overline{R ; \mathsf{L}}$

(12) $R \subseteq R^{\mathsf{T}*} ; \overline{R^{\mathsf{T}} ; \mathsf{L}} \cap \overline{\mathsf{L} ; R^{\mathsf{T}}} ; R^{\mathsf{T}*}$

It follows that a path is backward terminating if and only if its converse is forward terminating. Moreover a path is terminating if and only if its converse is terminating.

If the end point of a path R and the start point of a path S coincide, the paths can be concatenated. It has to be guaranteed, however, that the two paths do not cross each other. The condition used in the following result allows that the start point of R and the end point of S coincide, if they exist, but requires all other vertices of R and S to be distinct.

Theorem 4.7. *Let R be a forward terminating path and let S be a backward terminating path with $\mathsf{end}(R) = \mathsf{start}(S)$ and $R;\mathsf{L}\cap(R^{\mathsf{T}};\mathsf{L}\cup S;\mathsf{L})\cap S^{\mathsf{T}};\mathsf{L} = \mathsf{O}$. Then $R\cup S$ is a path. Moreover, $\mathsf{start}(R\cup S) \subseteq \mathsf{start}(R)$ and $\mathsf{end}(R\cup S) \subseteq \mathsf{end}(S)$.*

If additionally $\mathsf{start}(R) = \mathsf{end}(S)$ are non-empty and hence points, path concatenation creates a cycle. As a consequence we only obtain inequalities in the second part of the previous theorem. By strengthening its assumption to $R ; \mathsf{L} \cap S^{\mathsf{T}} ; \mathsf{L} = \mathsf{O}$ we can exclude the creation of a cycle. This means that R and S only meet at the end point of R and the start point of S.

Lemma 4.8. *The strengthened assumption $R ; \mathsf{L} \cap S^{\mathsf{T}} ; \mathsf{L} = \mathsf{O}$ is equivalent to the conjunction of the assumption $R ; \mathsf{L} \cap (R^{\mathsf{T}} ; \mathsf{L} \cup S ; \mathsf{L}) \cap S^{\mathsf{T}} ; \mathsf{L} = \mathsf{O}$ of Theorem 4.7 and $\mathsf{start}(R) \cap \mathsf{end}(S) = \mathsf{O}$.*

Under this assumption, it is possible to determine the start and end points of the composed path $R \cup S$ according to the following result.

Theorem 4.9. *Let R be a forward terminating path and let S be a backward terminating path such that* $\mathsf{end}(R) = \mathsf{start}(S)$ *and* $R \mathbin{;} \mathsf{L} \cap S^{\mathsf{T}} \mathbin{;} \mathsf{L} = \mathsf{O}$. *Then* $\mathsf{start}(R \cup S) = \mathsf{start}(R)$ *and* $\mathsf{end}(R \cup S) = \mathsf{end}(S)$.

Rather than looking at path concatenation, one can also consider path restriction. Given a path R and an arbitrary point p, we construct a path with start point p by following the edges of R.

Theorem 4.10. *Let p be a point and let R be a path. Then $R^{\mathsf{T}*} \mathbin{;} p \cap R$ is a path. Moreover,* $\mathsf{start}(R^{\mathsf{T}*} \mathbin{;} p \cap R) \subseteq p$, *and* $\mathsf{end}(R^{\mathsf{T}*} \mathbin{;} p \cap R) \subseteq \mathsf{end}(R)$.

The second statement shows that either p is the start point of the path, or there is no start point at all, which means that the new path is a cycle or empty. The expression $R^{\mathsf{T}*} \mathbin{;} p$ represents the set of predecessors of the point p under the relation $R^{\mathsf{T}*}$, which are the successors of p under R^*, which is the set of vertices reachable from p by a path in R. Intersecting with R keeps the edges of R that start in such a vertex.

The additional assumption $p \subseteq R \mathbin{;} \mathsf{L}$ assures that the point lies on the path R and has at least one successor. In that case $\mathsf{start}(R^{\mathsf{T}*} \mathbin{;} p \cap R) = p$, and $\mathsf{end}(R^{\mathsf{T}*} \mathbin{;} p \cap R) = \mathsf{end}(R)$.

5. Cycles

By Corollary 4.4 a path R has no start point and no end point if and only if $R \mathbin{;} \mathsf{L} = R^{\mathsf{T}} \mathbin{;} \mathsf{L}$. In this section we distinguish three kinds of paths satisfying this property: the empty path, cycles and two-sided infinite paths. A cycle is obtained by assuming a strong connectivity requirement, given in the following definition.

Definition 5.1. *A path R is a* cycle *if $R^* = R^{\mathsf{T}*}$.*

Cycles in this sense encompass classes [⑤⑥] of Figure 1. An immediate consequence is that a relation is a cycle if and only if its converse is a cycle.

The additional property $R^* = R^{\mathsf{T}*}$ specifies that if a vertex b is reachable from a vertex a in R, also a is reachable from b in R. In this case, a and b belong to the same strongly connected component of R. The following result gives equivalent ways to express this property and some consequences. For example, $R \subseteq R^{\mathsf{T}+}$ states that for every edge $(a, b) \in R$ vertex b is reachable from vertex a by a non-empty path backwards in R.

Theorem 5.2. *The following properties are equivalent:*

(1) $R^* = R^{\mathsf{T}*}$

(2) $R^+ = R^{\mathsf{T}+}$

(3) $R^{\mathsf{T}} \subseteq R^*$

(4) $R^{\mathsf{T}} \subseteq R^+$

(5) $R \subseteq R^{\mathsf{T}*}$

(6) $R \subseteq R^{\mathsf{T}+}$

(7) $R^* \mathbin{;} R^{\mathsf{T}} \subseteq R^+$

(8) $R^{\mathsf{T}} \mathbin{;} R^* \subseteq R^+$

Each of them implies the following properties:

(9) $R \mathbin{;} \mathsf{L} = R^{\mathsf{T}} \mathbin{;} \mathsf{L}$

(10) $R \mathbin{;} R^{\mathsf{T}} \subseteq R^+$

(11) $R^{\mathsf{T}} \mathbin{;} R \subseteq R^+$

(12) $R \mathbin{;} R^{\mathsf{T}} \mathbin{;} R^* \subseteq R^+$

(13) $R^* \mathbin{;} R^{\mathsf{T}} \mathbin{;} R \subseteq R^+$

(14) $R^* \mathbin{;} R \mathbin{;} R^{\mathsf{T}} \subseteq R^+$

(15) $R^{\mathsf{T}} \mathbin{;} R \mathbin{;} R^* \subseteq R^+$

If R is univalent, each of (10), (12) and (14) is equivalent to property (1). If R is injective, each of (11), (13) and (15) is equivalent to property (1).

Theorem 5.2 remains valid if any of the inequalities in (7), (8), (12)–(15) are replaced with equalities.

As witnessed by the identity relation I, the property $R^* = R^{\mathsf{T}*}$ alone does not imply that the relation R contains just one strongly connected component. This results from the combination with the connectivity requirement (C). The following result expresses the combination of these two properties as a single (in)equality.

Theorem 5.3. *The following properties are equivalent:*

(1) *R satisfies $R \mathbin{;} \mathsf{L} \mathbin{;} R \subseteq R^* \cup R^{\mathsf{T}*}$ (C) and $R^* = R^{\mathsf{T}*}$*

(2) $R^{\mathsf{T}} \mathbin{;} \mathsf{L} \mathbin{;} R^{\mathsf{T}} \subseteq R^*$

(3) $R^{\mathsf{T}} \mathbin{;} \mathsf{L} \mathbin{;} R^{\mathsf{T}} \subseteq R^+$

(4) $R^{\mathsf{T}} \mathbin{;} \mathsf{L} \mathbin{;} R^{\mathsf{T}} = R^+$

(5) $R^{\mathsf{T}} \mathbin{;} \mathsf{L} \mathbin{;} R = R^+$

(6) $R \mathbin{;} \mathsf{L} \mathbin{;} R^{\mathsf{T}} = R^+$

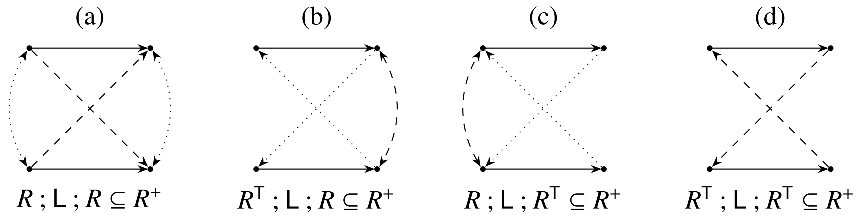

(a)
(b)
(c)
(d)
$R ; \mathsf{L} ; R \subseteq R^+$
$R^\mathsf{T} ; \mathsf{L} ; R \subseteq R^+$
$R ; \mathsf{L} ; R^\mathsf{T} \subseteq R^+$
$R^\mathsf{T} ; \mathsf{L} ; R^\mathsf{T} \subseteq R^+$

Figure 2: R is strongly connected in (b)–(d), but not strongly connected in (a)

They imply each of the following properties:

(7) $R^\mathsf{T} ; \mathsf{L} ; R \subseteq R^+$
(8) $R ; \mathsf{L} ; R^\mathsf{T} \subseteq R^+$
(9) $R ; \mathsf{L} ; R \subseteq R ; R^+$
(10) $R ; \mathsf{L} ; R \subseteq R^+$

If R is injective, property (7) is equivalent to property (1). If R is univalent, property (8) is equivalent to property (1). If R is both injective and univalent, property (9) is equivalent to property (1).

Furthermore, property (9) of Theorem 5.3 is equivalent to the corresponding equality $R ; \mathsf{L} ; R = R ; R^+$ and property (10) is equivalent to the corresponding equality $R ; \mathsf{L} ; R = R^+$.

In [60], a relation R is called a *simple circuit* if it is univalent, injective and satisfies property (6) of the previous theorem and the condition $R ; R \subseteq \overline{\mathsf{I}}$. By the previous theorem, this is equivalent to R being a cycle except for the additional condition that $R ; R$ is irreflexive. The latter states that no two vertices are mutually connected by an edge; this excludes cycles of length 2. The condition also implies that R is irreflexive, which excludes cycles of length 1, that is, loops. When undirected graphs are represented as symmetric relations, cycles of length 2 cannot be distinguished from edges and loops are typically not considered (simple graphs). For directed graphs, however, we wish to include cycles of length 1 or 2 in our reasoning, so we do not require the additional condition.

Note that properties (3), (7), (8) and (10) of Theorem 5.3 have identical right-hand sides and similar left-hand sides. Despite this, property (10) is weaker than the others, as illustrated in Figure 2. It shows, for each of the four properties, a pair of edges of R as solid lines. We know that these edges exist by considering the left-hand sides of the four properties; the edges may be identical or have the same source vertex or target vertex. Dashed lines indicate non-empty paths whose existence follows by applying the respective property to these edges. We know that these paths are non-empty because the right-hand side of each property is R^+. From this, the existence of (possibly empty) paths indicated by dotted lines follows if R is univalent (c) or injective (b) or both (a). The existing paths in (b)–(d) allow us to show that the two edges are in the same strongly connected component of R thereby creating a cycle. This does not work in (a) even if R is both injective and univalent. However, with the stronger assumption $R ; \mathsf{L} ; R \subseteq R ; R^+$, which is property (9) of the previous theorem, the dashed paths in (a) would have length 2 or more. Hence the dotted paths in (a) would be non-empty, so the argument of (b) or (c) could be applied to show that the edges belong to a cycle.

Sometimes we wish to include cycles in reasoning about paths with start or end points. A path is *backward finite* if it is a cycle or backward terminating [①③⑤⑥]. A path is *forward finite* if it is a cycle or forward terminating [①②⑤⑥]. A path is *finite* if it is a cycle or terminating [①⑤⑥]. The following result shows how to express each of these properties as an inequality.

Theorem 5.4. *Let R be a relation. Then the following properties are equivalent and, if R is a path, characterise backward finiteness.*

(1) $R^* = R^{\mathsf{T}*}$ *or* $R \subseteq \overline{\mathsf{L} ; R} ; R ; \mathsf{L}$
(2) $R \subseteq \overline{\mathsf{L} ; R} ; R ; \mathsf{L} \cup R^{\mathsf{T}*}$

Moreover, the following properties are equivalent and, if R is a path, characterise forward finiteness.

(3) $R^* = R^{\mathsf{T}^*}$ *or* $R \subseteq \mathsf{L} ; R ; \overline{R ; \mathsf{L}}$

(4) $R \subseteq \mathsf{L} ; R ; \overline{R ; \mathsf{L}} \cup R^{\mathsf{T}^*}$

Finally, the following properties are equivalent and, if R is a path, characterise finiteness.

(5) $R^* = R^{\mathsf{T}^*}$ *or* $R \subseteq \overline{\mathsf{L} ; R} ; R ; \mathsf{L} \cap \mathsf{L} ; R ; \overline{R ; \mathsf{L}}$

(6) $R \subseteq (\overline{\mathsf{L} ; R} ; R ; \mathsf{L} \cap \mathsf{L} ; R ; \overline{R ; \mathsf{L}}) \cup R^{\mathsf{T}^*}$

It follows that a path is backward finite if and only if its converse is forward finite. Moreover a path is finite if and only if its converse is finite.

We conclude this section with a number of useful facts about cycles. The first result shows that any terminating path can be extended to a cycle by connecting its end point with its start point.

Theorem 5.5. *Let R be a terminating path. Then* $R \cup \mathsf{end}(R) ; \mathsf{start}(R)^\mathsf{T}$ *is a cycle.*

Conversely, the next result shows that any non-empty cycle becomes a terminating path if an edge is removed.

Theorem 5.6. *Let R be a cycle and let* s, e *be points with* $e ; s^\mathsf{T} \subseteq R$*. Then* $R \cap \overline{e ; s^\mathsf{T}}$ *is a terminating path with* $\mathsf{start}(R \cap \overline{e ; s^\mathsf{T}}) \subseteq s$ *and* $\mathsf{end}(R \cap \overline{e ; s^\mathsf{T}}) \subseteq e$*. If* $s \neq e$*, the last two inequalities can be strengthened to equalities.*

The final result of this section shows how to join two paths with suitable start and end points to a cycle. It uses the assumption of Theorem 4.7 to require that the two paths do not overlap.

Theorem 5.7. *Let* R, S *be terminating paths with* $\mathsf{start}(R) = \mathsf{end}(S)$ *and* $\mathsf{start}(S) = \mathsf{end}(R)$*. Then* $R \cup S$ *is a cycle if* $R ; \mathsf{L} \cap (R^\mathsf{T} ; \mathsf{L} \cup S ; \mathsf{L}) \cap S^\mathsf{T} ; \mathsf{L} = \mathsf{O}$.

6. Application: Verifying the Correctness of Graph Algorithms

Using the Kleene-relation-algebraic characterisations of different kinds of paths derived in the previous sections, we now verify the correctness of three basic graph algorithms. Previous work has shown that relation-algebraic reasoning can be applied for program transformation and verification in general and in the context of graph algorithms; for example, see [2, 5, 10, 13, 25, 28]. The aim of this section is to show that our theory of paths integrates well with such arguments.

Algorithms at the presented level are executable and can serve prototyping purposes. For example, the tool RelView [59], which is an interactive tool for computer-supported manipulation of relations, accepts the presented code nearly verbatim. Relations are implemented in RelView using ROBDDs rather than Boolean matrices to improve efficiency. Data refinement can be carried out to move from such algorithms to more efficient programs; for example, see [7].

Our correctness proofs use the well-known assertion-based program verification technique based on Hoare logic [33]. The algorithms are expressed as while-programs with variables whose values are elements of a Kleene relation algebra. Correctness of the algorithms is stated by preconditions and postconditions, which are relation-algebraic formulas. The first task in proving the correctness of a while-program is to provide for each while-loop an invariant that holds throughout the execution of the loop; invariants are again relation-algebraic formulas. We then need to prove that each loop invariant

(1) is established from the precondition before the while-loop,

(2) is maintained by each iteration of the while-loop, and

(3) implies the required postcondition after the while-loop.

This proves partial correctness of the program: if it starts in a state satisfying the precondition and terminates, the postcondition will hold in the final state.

For all our examples, we briefly describe the algorithm itself, and present the pre- and postconditions as well as the loop invariants. The verification conditions listed above are automatically generated from this information by Isabelle/HOL's Hoare logic library [53, 54]. The proofs of these obligations use relation-algebraic reasoning similar to the proofs of other results in this paper; they can be found in the Isabelle/HOL theory files.

We have also proved termination of the algorithms using a total-correctness Hoare logic library discussed in [29]. To this end, each while-loop has to be annotated with a variant or bound function; see the Isabelle/HOL theories for details. We only discuss partial correctness below; the termination proofs use the additional assumption that graphs have finitely many vertices. For this, the underlying algebra is assumed to be finite; in the model this means there are only finitely many graphs (over any particular vertex set) which is equivalent to having finitely many vertices.

6.1. Construction of a Path

Our first example is a simple Greedy algorithm that constructs a path from a vertex x to a different vertex y of a directed acyclic graph D. We assume that a path between these vertices exists and, moreover, conditions that ensure the Greedy algorithm will find one without searching. See [8, 9] for relational implementations of depth-first search and breadth-first search.

We use the predicate $\mathsf{point}(p)$ to specify that the relation p is a point in the relation-algebraic sense as defined in Section 2.3.

The algorithm maintains a relation W, which is the path that is constructed backwards from y towards x. As soon as W forms a path from x to y the algorithm terminates with result W (Line 9 in Algorithm 1). The algorithm works as follows: W is initialised as the empty relation for no path has been constructed yet (Line 2). The point q is the start point of W if W is not empty; it is initialised with y (W's final destination) (Line 3). As long as the start point q of W is different from x (Line 4), the algorithm chooses a predecessor p of q (Line 5) and extends the relation W by the edge from p to q (Line 6).

For the selection of predecessors we use an operation $\mathsf{choosePoint}(v)$ that (deterministically) chooses a point contained in a non-empty vector v. The existence of such a point follows from the point axiom. To reason about the operation $\mathsf{choosePoint}$, we assume it satisfies the following axioms:

$$\mathsf{choosePoint}(v) \subseteq v \qquad v \neq \mathsf{O} \Rightarrow \mathsf{point}(\mathsf{choosePoint}(v))$$

The inequality states that the chosen relation is contained in v, and the implication states that the chosen relation is a point. The operation $\mathsf{choosePoint}$ is deterministic in the sense that it always produces the same result for a given argument and hence may be modelled as a function in a relation algebra. This does not mean that the operation is uniquely determined. There may be several different implementations of a deterministic operation satisfying these axioms; any one works fine since our reasoning only uses the properties stated by or following from the axioms.

Using $\mathsf{choosePoint}$ we formally describe the construction of a path in Algorithm 1.

Algorithm 1 Constructing a path

```
1 input $D, x, y$
2 $W \leftarrow \mathsf{O}$
3 $q \leftarrow y$
4 while $q \neq x$ do
5     $p \leftarrow \mathsf{choosePoint}(D ; q)$
6     $W \leftarrow W \cup p ; q^{\mathsf{T}}$
7     $q \leftarrow p$
8 end
9 output $W$
```

We show correctness of Algorithm 1 if the input satisfies the following preconditions, whose conjunction is denoted by $\mathsf{Pre}(D, x, y)$:

$$D^+ \subseteq \overline{\mathsf{I}} \qquad \mathsf{point}(x) \qquad \mathsf{point}(y) \qquad D^* ; y \subseteq D^{\mathsf{T}^*} ; x$$

The first inequality states that D is acyclic. The following two conditions specify that x and y are vertices. The last inequality uses two vectors: $D^* ; y$ contains the transitive predecessors of y, which are the vertices from which y is reachable, and $D^{\mathsf{T}^*} ; x$ contains the transitive successors of x, which are the vertices reachable from x. The inequality expresses that every transitive predecessor of y is a transitive successor of x, which implies that there is a path from

x to y. Moreover, it follows that there is a path from x to any transitive predecessor of y, which is the reason why the above Greedy algorithm works without searching.

The postcondition $\mathsf{Post}(D, x, y, W)$ of the algorithm is the conjunction of the following relation-algebraic formulas:

$$W \subseteq D \qquad \mathsf{termPath}(W) \qquad W = \mathsf{O} \Leftrightarrow x = y \qquad W \neq \mathsf{O} \Leftrightarrow (x = \mathsf{start}(W) \wedge y = \mathsf{end}(W))$$

The first condition guarantees that the output W of Algorithm 1 is contained in the graph D. The second condition states that W is a terminating path as defined in Section 4. The last condition ensures that the constructed path starts in x and ends in y. However, this is not the case if W is empty as such a relation has neither start nor end points. If the result of the algorithm is empty then $x = y$ (third condition).

The loop invariant $\mathsf{Inv}(D, x, y, W, q)$ used to prove partial correctness of Algorithm 1 is the conjunction of the following formulas:

$$\begin{array}{lllll} D^{+} \subseteq \overline{\mathsf{I}} & \mathsf{point}(x) & \mathsf{point}(y) & \mathsf{point}(q) & D^{*} ; q \subseteq D^{\mathsf{T}^{*}} ; x \\ W \subseteq D & \mathsf{termPath}(W) & W = \mathsf{O} \Leftrightarrow x = y & W \neq \mathsf{O} \Leftrightarrow (q = \mathsf{start}(W) \wedge y = \mathsf{end}(W)) & \end{array}$$

The first line shows that the invariant contains most of $\mathsf{Pre}(D, x, q)$. Maintaining the precondition throughout the while-loop is trivial since it uses only input variables, whose values do not change. Nevertheless these conditions need to be formally part of the loop invariant since they are not only necessary for establishing the rest of the invariant but also to maintain it. Moreover, the invariant contains $\mathsf{Post}(D, q, y, W)$ as shown in the second line.

From this invariant, $\mathsf{Post}(D, x, y, W)$ immediately follows using the negated loop condition $q = x$, which holds after the while-loop terminates.

If we require the additional precondition that the two vertices x and y are distinct, the invariant, the postcondition and the verification become simpler as the case distinction on W is not needed.

6.2. *Topological Sorting*

In our second example we look at topological sorting: given a directed acyclic graph R, the problem is to construct a linear order of its vertices that contains x before y for each edge (x, y) of the graph. If the input graph models dependencies between tasks, the output is a linear schedule of the tasks that respects all dependencies. We represent the linear order of vertices as a path, in which the sequence of vertices gives the schedule.

A simple algorithm based on [38] for finding a topological sort W works as follows. It first picks a vertex without a predecessor (Line 3 in Algorithm 2), which exists since the input R is acyclic. This vertex is marked (Line 4) and used as the start point of the path to be constructed. As long as there are unmarked vertices (Line 5), the algorithm picks one that does not have an unmarked predecessor (Line 6). As before, such a vertex exists since any subgraph of R is acyclic, too. The edge from the current end point q of W to the chosen vertex is then added to the path (Line 7). The selected vertex is marked (Line 9).

The relational program given in Algorithm 2 implements this procedure. It represents the set of marked vertices as the vector v; a vertex is marked if and only if the corresponding (relational) point is contained in v. Hence $\overline{v}$ contains the unmarked vertices, and the condition $v \neq \mathsf{L}$ in Line 5 checks if there is still an unmarked vertex.

To find vertices without unmarked predecessors, Algorithm 2 uses the operation $\mathsf{min}(S, w) = w \cap \overline{S^{\mathsf{T}} ; w}$, where S is a relation and w is a vector [64]. It yields the S-minimal elements of the set represented by w. Hence $\mathsf{min}(R, \mathsf{L})$ in Line 3 gives the vertices without predecessors in the whole graph, and $\mathsf{min}(R, \overline{v})$ in Line 6 gives the unmarked vertices without any unmarked predecessors. As in Algorithm 1, we use the $\mathsf{choosePoint}$ operation to select one of these vertices.

For the selection in Line 6 to succeed, the argument of $\mathsf{choosePoint}$ must be different from O. This is established from the condition $v \neq \mathsf{L}$ of the while-loop. Formally, we require $v \neq \mathsf{L} \Rightarrow \mathsf{min}(R, \overline{v}) \neq \mathsf{O}$, which is equivalent to $\overline{v} \cap \overline{R^{\mathsf{T}} ; \overline{v}} = \mathsf{O} \Rightarrow v = \mathsf{O}$. By the shunting law of Boolean algebra this is equivalent to $\overline{v} \subseteq R^{\mathsf{T}} ; \overline{v} \Rightarrow v = \mathsf{O}$. We therefore require that the input relation R satisfies

$$w \subseteq R^{\mathsf{T}} ; w \Rightarrow w = \mathsf{O}$$

for all vectors w. A relation R satisfying this property is also called *well-founded* or *regressively finite* [3, 23, 64]. This property is the only precondition $\mathsf{Pre}(R)$ of the algorithm. Every well-founded relation is acyclic and the converse

Algorithm 2 Topological sorting

```
input $R$
$W \leftarrow \mathsf{O}$
$q \leftarrow \mathsf{choosePoint}(\mathsf{min}(R, \mathsf{L}))$
$v \leftarrow q$
while $v \neq \mathsf{L}$ do
    $p \leftarrow \mathsf{choosePoint}(\mathsf{min}(R, \overline{v}))$
    $W \leftarrow W \cup q \,;\, p^{\mathsf{T}}$
    $q \leftarrow p$
    $v \leftarrow v \cup p$
end
output $W$
```

implication holds for relations representing graphs with finitely many vertices. In the case of a total-correctness proof, which additionally requires finiteness, the precondition simply states that R is acyclic.

We now consider the postcondition. The relation W stores the constructed topological sort as a terminating path with a start point and an end point. To express that W preserves the dependencies in R we use the condition $R \subseteq W^+$, that is, there must be a path in W from the source x to the target y of every edge $(x, y) \in R$. Finally, the path W has to contain all vertices of R. If R has two or more vertices this can be specified as $(W \cup W^{\mathsf{T}}) \,;\, \mathsf{L} = \mathsf{L}$, where the left-hand side is the vector of all points in W. Alternative expressions for this vector are $W^{\mathsf{T}} \,;\, \mathsf{L} \cup \mathsf{start}(W)$ and $W \,;\, \mathsf{L} \cup \mathsf{end}(W)$. If R contains only a single vertex, W is the empty path, so this specification does not work. We can either add a precondition $\mathsf{I} \neq \mathsf{L}$ stating that R has at least two vertices or modify the condition by replacing L with $\overline{\mathsf{I}} \,;\, \mathsf{L}$ to handle this special case. Using the latter option, we obtain the postcondition $\mathsf{Post}(R, W)$ as the conjunction of the following formulas:

$$R \subseteq W^+ \qquad \mathsf{termPath}(W) \qquad (W \cup W^{\mathsf{T}}) \,;\, \mathsf{L} = \overline{\mathsf{I}} \,;\, \mathsf{L}$$

The loop invariant that allows us to prove partial correctness with respect to the precondition $\mathsf{Pre}(R)$ and the postcondition $\mathsf{Post}(R, W)$ is the conjunction of the following formulas:

$$\begin{array}{lll} \mathsf{Pre}(R) & R \cap v \,;\, v^{\mathsf{T}} \subseteq W^+ & \mathsf{termPath}(W) \\ \mathsf{point}(q) & q \subseteq v & W = \mathsf{O} \vee q = \mathsf{end}(W) \\ v = v \,;\, \mathsf{L} & W \,;\, \mathsf{L} = v \cap \overline{q} & R \,;\, v \subseteq v \end{array}$$

The first line contains the precondition, which is maintained trivially since R never changes, and the first two parts of the postcondition; note that the formula $R \cap v \,;\, v^{\mathsf{T}} \subseteq W^+$ is a generalisation of $R \subseteq W^+$ that restricts R to the subgraph induced by the set of marked vertices. The remaining properties (second and third lines) are auxiliary properties necessary to maintain the first three formulas of the loop invariant and to ensure the postcondition $(W \cup W^{\mathsf{T}}) \,;\, \mathsf{L} = \overline{\mathsf{I}} \,;\, \mathsf{L}$. The second line characterises the relation q as a marked vertex, which is the end point of W except when W is empty at the beginning of Algorithm 2. The third line characterises the relation v as the set of vertices of W and ensures that all predecessors of these marked vertices are marked, too.

6.3. *Construction of a Non-empty Cycle*

Our last application is a correctness proof of an algorithm that constructs a non-empty cycle for a given directed graph R. As precondition $\mathsf{Pre}(R)$ we assume that R is not acyclic, that is, it contains at least one cycle:

$$R^+ \cap \mathsf{I} \neq \mathsf{O}$$

This is the only condition needed for input R.

Algorithm 3 shows the relational program. It starts by picking two points, using the operation $\mathsf{choosePoint}$ again. The first point y is selected in Line 2 as a vertex lying on an arbitrary cycle of R. The existence of such a point is guaranteed by $\mathsf{Pre}(R)$. The second point x is selected in Line 3 as a direct successor of y (since $x \subseteq R^{\mathsf{T}} \,;\, y$) that is also a transitive predecessor of y (since $x \subseteq R^* \,;\, y$). These conditions ensure that x and y lie on a cycle of R.

The algorithm then progresses in three steps. Lines 4–10 construct a directed tree $D \subseteq R$ with root x in which y is a leaf. Lines 11–17 select a path W in D that connects x with y. This path W together with the edge from y to x gives the required cycle in Line 18.

In the first step, the directed tree D is constructed as follows. Line 4 initialises the relation D as the tree without any edges. The while-loop maintains the vector v that contains all vertices of the tree. At the start of the loop, v contains only the point x which is the root of D; see Line 5. As long as the vertex y has not been reached, which is checked in Line 6, the algorithm chooses an edge e of R that goes from a vertex in v to a vertex outside v in Line 7. The edge is added to the tree in Line 8 and its end point is added to v in Line 9.

An edge is a relation consisting of a single pair and can therefore be represented by a singleton. We use the predicate $\mathsf{singleton}(a)$ to specify that the relation a is a singleton in the relation-algebraic sense as defined in Section 2.3. Similarly to points, we use an operation $\mathsf{chooseSingleton}(x)$ that (deterministically) chooses a singleton (an edge) contained in a non-empty relation x. To reason about the operation $\mathsf{chooseSingleton}$, we assume it satisfies the following axioms:

$$\mathsf{chooseSingleton}(x) \subseteq x \qquad x \neq \mathsf{O} \Rightarrow \mathsf{singleton}(\mathsf{chooseSingleton}(x))$$

Since the while-loop adds edges leaving the set v and since it is known that y is reachable from x, the vertex y will eventually be included in v, so D will contain a path from x to y.

By construction, D is acyclic and satisfies $D^* ; y \subseteq D^{\mathsf{T}*} ; x$. As a consequence we can use Algorithm 1 to determine a path from x to y; Lines 11–17 of Algorithm 3 are identical to Lines 2–8 of Algorithm 1. When the second while-loop of Algorithm 3 terminates, W contains a terminating path from x to y. Line 18 adds to this path the edge from y to x to obtain a cycle C. By the choice of x and y in Lines 2 and 3 this edge is contained in R.

Algorithm 3 Constructing a cycle

```
 1 input $R$
 2 $y \leftarrow \mathsf{choosePoint}((R^+ \cap \mathsf{I}) ; \mathsf{L})$
 3 $x \leftarrow \mathsf{choosePoint}(R^* ; y \cap R^\mathsf{T} ; y)$
 4 $D \leftarrow \mathsf{O}$
 5 $v \leftarrow x$
 6 while $\neg(y \subseteq v)$ do
 7     $e \leftarrow \mathsf{chooseSingleton}(v ; \overline{v}^\mathsf{T} \cap R)$
 8     $D \leftarrow D \cup e$
 9     $v \leftarrow v \cup e^\mathsf{T} ; \mathsf{L}$
10 end
11 $W \leftarrow \mathsf{O}$
12 $q \leftarrow y$
13 while $q \neq x$ do
14     $p \leftarrow \mathsf{choosePoint}(D ; q)$
15     $W \leftarrow W \cup p ; q^\mathsf{T}$
16     $q \leftarrow p$
17 end
18 $C \leftarrow W \cup y ; x^\mathsf{T}$
19 output $C$
```

We now discuss the postcondition of Algorithm 3. It constructs a non-empty cycle C that is also a subgraph of R. These conditions immediately translate into the following formulas, whose conjunction is the postcondition $\mathsf{Post}(R, C)$:

$$C \neq \mathsf{O} \qquad \mathsf{cycle}(C) \qquad C \subseteq R$$

Here the condition $\mathsf{cycle}(C)$ specifies that C is a cycle as defined in Section 5.

We conclude this section by discussing the invariant used to prove partial correctness of Algorithm 3 with respect to $\mathsf{Pre}(R)$ and $\mathsf{Post}(R, C)$. As before, Isabelle/HOL's Hoare logic tactic splits the correctness proof into a number of verification conditions.

To reuse Algorithm 1 we have to show that its precondition $\mathsf{Pre}(D, x, y)$ holds at the beginning of Line 11. Recall that $\mathsf{Pre}(D, x, y)$ is the conjunction of the following formulas:

$$D^+ \subseteq \bar{\mathsf{I}} \qquad \mathsf{point}(x) \qquad \mathsf{point}(y) \qquad D^* \,;\, y \subseteq D^{\mathsf{T}*} \,;\, x$$

To establish these conditions we use the conjunction of the following formulas as the invariant for the first while-loop in Lines 6–10:

$$\begin{array}{llll} \mathsf{point}(x) & \mathsf{point}(y) & y \subseteq R^{\mathsf{T}*} \,;\, x & y \,;\, x^{\mathsf{T}} \subseteq R \\ D \subseteq R & D^+ \subseteq \bar{\mathsf{I}} & D \,;\, D^{\mathsf{T}} \subseteq \mathsf{I} & D \,;\, x = \mathsf{O} \\ v = v \,;\, \mathsf{L} & D \subseteq v \,;\, v^{\mathsf{T}} & x \,;\, v^{\mathsf{T}} \subseteq D^* & v = x \cup D^{\mathsf{T}} \,;\, \mathsf{L} \end{array}$$

The first line lists simple conditions that follow immediately from the selection of x and y in Lines 2 and 3 of Algorithm 3. These properties are needed later and have to be maintained throughout the entire proof, which is not difficult since R, x and y are not modified in Lines 4–18.

The first three inequalities of the second line state that D is an injective acyclic subgraph of R, that is, a forest in R. The last equality of this line specifies that vertex x has no predecessors in D.

The third line contains conditions involving v. The first two conditions state that v is a vector and that D contains only edges between vertices in the set represented by v. The third condition in this line specifies that all vertices in v are reachable from x in D. This implies that D is a tree with root x. Finally, $v = x \cup D^{\mathsf{T}} \,;\, \mathsf{L}$ states that v contains x and the target vertices of all edges in D.

These invariants imply that $D^* \,;\, y \subseteq D^{\mathsf{T}*} \,;\, x$ holds after the first while-loop, so we can use Algorithm 1 afterwards. By the correctness of Algorithm 1 we know that its postcondition $\mathsf{Post}(D, x, y, W)$ holds after the second while-loop of Algorithm 3. This postcondition is:

$$W \subseteq D \qquad \mathsf{termPath}(W) \qquad W = \mathsf{O} \Leftrightarrow x = y \qquad W \neq \mathsf{O} \Leftrightarrow (x = \mathsf{start}(W) \wedge y = \mathsf{end}(W))$$

If $W = \mathsf{O}$ we need that the edge $x \,;\, x^{\mathsf{T}}$ is a cycle, which is easy to prove. If $W \neq \mathsf{O}$, Theorem 5.5 implies that C as constructed in Line 18 is a cycle. The remaining two postconditions of Algorithm 3, that is, $C \neq \mathsf{O}$ and $C \subseteq R$ are easy to prove using $W \subseteq D$ and $D \subseteq R$.

7. Paths with Roots

In the previous sections a path was described by a relation only; in Section 4 we have discussed conditions under which paths have a start point and/or an end point. In this section we describe paths together with a designated root. Characterising paths together with a designated root is important as often algorithms start with a single vertex, and then build up a path, a tree or another structure; examples are Dijkstra's shortest path algorithm and augmenting paths for constructing maximum bipartite matchings. A root is a vertex of the graph represented by a (relational) point. Our main results are equivalences between the definitions of paths with and without roots. Most of these equivalences only hold if the point axiom is assumed.

We represent a path with root by two relations R and p such that R is injective and univalent and p is a point. The relations R and p will satisfy further conditions depending on the kind of path that is represented.

7.1. Paths

We start with a result that characterises all paths [①②③④⑤⑥]. The condition $p \,;\, R \subseteq R^* \cup R^{\mathsf{T}*}$ states that any end vertex of an edge in R must be reachable from the vertex p by going forward in R or by going backward in R.

Theorem 7.1. *Let R be an injective and univalent relation. Then R is a path if and only if there exists a point p such that $p \,;\, R \subseteq R^* \cup R^{\mathsf{T}*}$.*

If the path is not empty, we can guarantee that the vertex p of the previous theorem is contained in the path. This is stated by the additional condition $p \subseteq (R \cup R^{\mathsf{T}}) \,;\, \mathsf{L}$ of the following characterisation [①②③④⑤].

Corollary 7.2. *Let R be an injective and univalent relation. Then R is a non-empty path if and only if there exists a point p such that $p \mathbin{;} R \subseteq R^* \cup R^{\mathsf{T}*}$ and $p \subseteq (R \cup R^{\mathsf{T}}) \mathbin{;} \mathsf{L}$.*

The following theorem shows that a path R is acyclic if p has no predecessors in R.

Theorem 7.3. *Let R be an injective and univalent relation and let p be a point such that $p \mathbin{;} R \subseteq R^* \cup R^{\mathsf{T}*}$ and $R \mathbin{;} p = \mathsf{O}$. Then R is acyclic.*

7.2. Backward Finite Paths

The following result characterises backward finite paths [①③⑤⑥]. The condition $p \mathbin{;} R \subseteq R^+$ states that any end vertex of an edge in R must be reachable from the vertex p by a non-empty path in R. It implies the previous condition $p \mathbin{;} R \subseteq R^* \cup R^{\mathsf{T}*}$ since $R^+ \subseteq R^* \subseteq R^* \cup R^{\mathsf{T}*}$.

Theorem 7.4. *Let R be an injective and univalent relation. Then R is a backward finite path if and only if there exists a point p such that $p \mathbin{;} R \subseteq R^+$.*

For a point p, the condition $p \mathbin{;} R \subseteq R^+$ is equivalent to each of $R^{\mathsf{T}} \mathbin{;} \mathsf{L} \subseteq R^{\mathsf{T}+} \mathbin{;} p$ and $R^{\mathsf{T}} \mathbin{;} \mathsf{L} = R^{\mathsf{T}+} \mathbin{;} p$.

Again, if the path is not empty, we obtain that it contains p, which can now be stated as $p \subseteq R \mathbin{;} \mathsf{L}$ in the following characterisation [①③⑤].

Corollary 7.5. *Let R be an injective and univalent relation. Then R is a non-empty backward finite path if and only if there exists a point p such that $p \mathbin{;} R \subseteq R^+$ and $p \subseteq R \mathbin{;} \mathsf{L}$.*

For backward finite paths, Theorem 7.3 can be extended to an equivalence as the following result shows.

Theorem 7.6. *Let R be an injective and univalent relation and let p be a point such that $p \mathbin{;} R \subseteq R^+$. Then $R \mathbin{;} p = \mathsf{O}$ if and only if R is acyclic.*

7.3. Cycles

The following result characterises non-empty cycles [⑤]. The additional condition $p \subseteq R^{\mathsf{T}} \mathbin{;} \mathsf{L}$ states that p is the end vertex of an edge in R. For a point p it is equivalent to $R \mathbin{;} p \neq \mathsf{O}$.

Theorem 7.7. *Let R be an injective and univalent relation. Then R is a non-empty cycle if and only if there exists a point p such that $p \mathbin{;} R \subseteq R^+$ and $p \subseteq R^{\mathsf{T}} \mathbin{;} \mathsf{L}$.*

In this case any point q on the cycle – that is, satisfying $q \subseteq R^* \mathbin{;} p$ – can take the place of p in the previous theorem – that is, satisfies $q \mathbin{;} R \subseteq R^+$ and $q \subseteq R^{\mathsf{T}} \mathbin{;} \mathsf{L}$. Moreover, $p \subseteq R^* \mathbin{;} q$ and $q \subseteq R^+ \mathbin{;} q = R^* \mathbin{;} q = R^* \mathbin{;} p = R \mathbin{;} \mathsf{L}$ follow. Finally, also $p \mathbin{;} R^{\mathsf{T}} \subseteq R^{\mathsf{T}+}$ and $p \subseteq R \mathbin{;} \mathsf{L}$ hold, so the previous theorem dualises to the converse cycle. The condition $q \subseteq R^* \mathbin{;} p$ to make all of this happen for non-empty cycles can equivalently be stated as each of $p \subseteq R^* \mathbin{;} q$ or $R \mathbin{;} q \neq \mathsf{O}$ or $R^{\mathsf{T}} \mathbin{;} q \neq \mathsf{O}$ or $q \subseteq R \mathbin{;} \mathsf{L}$ or $q \subseteq R^{\mathsf{T}} \mathbin{;} \mathsf{L}$.

Moreover, it follows that if R and S are non-empty cycles such that $R \subseteq S$ then $R = S$. Another consequence is that the two conditions of the previous theorem can be combined into one inequality if also the empty cycle is allowed [⑤⑥].

Corollary 7.8. *Let R be an injective and univalent relation. Then R is a cycle if and only if there exists a point p such that $p \mathbin{;} R \subseteq R^+ \cap R^{\mathsf{T}} \mathbin{;} \mathsf{L}$.*

7.4. Backward Terminating Paths

The following result characterises backward terminating paths [①③⑥]. The additional condition $R \mathbin{;} p = \mathsf{O}$ states that vertex p has no predecessors in R.

Theorem 7.9. *Let R be an injective and univalent relation. Then R is a backward terminating path if and only if there exists a point p such that $p \mathbin{;} R \subseteq R^+$ and $R \mathbin{;} p = \mathsf{O}$.*

If the relation is not empty, it follows that the root p of the previous theorem is the start point of R [①③].

Corollary 7.10. *Let R be an injective and univalent relation. Then R is a non-empty backward terminating path if and only if* $\mathsf{start}(R)$ *is a point such that* $\mathsf{start}(R) \mathbin{;} R \subseteq R^+$.

7.5. *Terminating Paths*

For characterising terminating paths we additionally need an end point [①⑥]. The condition $p \subseteq R^* ; q$ states that vertex q is reachable from vertex p in R. The condition $R^\mathsf{T} ; q = \mathsf{O}$ states that vertex q has no successors in R.

Theorem 7.11. *Let R be an injective and univalent relation. Then R is a terminating path if and only if there exist points p, q such that $p ; R \subseteq R^+$ and $p \subseteq R^* ; q$ and $R^\mathsf{T} ; q = \mathsf{O}$.*

In this case it follows that $R ; p = \mathsf{O}$. Moreover, also $q \subseteq R^{\mathsf{T}^*} ; p$ and $q ; R^\mathsf{T} \subseteq R^{\mathsf{T}^+}$ follow, so the previous theorem dualises by swapping p and q and taking the converse of R. Finally, $R = \mathsf{O}$ if and only if $p = q$.

If the relation is not empty, the point q of the previous theorem is the end point of R [①].

Corollary 7.12. *Let R be an injective and univalent relation. Then R is a non-empty terminating path if and only if $\mathsf{start}(R)$ and $\mathsf{end}(R)$ are points such that $\mathsf{start}(R) ; R \subseteq R^+$ and $\mathsf{start}(R) \subseteq R^* ; \mathsf{end}(R)$.*

8. Conclusion

We have shown how Kleene relation algebras can be used to compactly specify and reason about different kinds of paths in graphs. This avoids often tedious point-wise calculations about paths in a graph. To show applicability, we have used the developed formalism to verify the correctness of simple graph algorithms. Proofs of results use equational reasoning instead of point-wise arguments with quantified variables. This style of reasoning strongly benefits from support by automated and interactive theorem provers.

An aim of the paper was to develop a fundamental theory of paths based on relation algebras and Kleene algebras. There are several directions of extension. One is to verify more complex algorithms. Refinement-style proofs can be used to derive efficient algorithms. However, some algorithms cannot be verified with the presented framework as they may talk about cardinalities and/or weighted graphs.

Relation algebras can be extended to deal with cardinalities of relations [6, 12, 39]; for a path this would simply be the number of edges it contains. This facilitates, for example, proofs about the complexity of graph algorithms. Different combinations of the defining properties of paths and other properties yield different classes of graphs. For example, an injective acyclic relation represents a forest; adding connectivity yields trees. We expect that a number of results are common to the classes and can be derived from fewer properties making them more widely applicable. Relation algebras can also be generalised to structures that can represent weighted graphs [28]. Using the results of this paper in such a setting would allow us to reason, for example, about shortest-path algorithms.

Acknowledgement. We are grateful to the anonymous referees for their helpful feedback. Walter Guttmann thanks the Japan Society for the Promotion of Science for supporting part of this research through a JSPS Invitation Fellowship for Research in Japan.

References

[1] Armstrong, A., Foster, S., Struth, G., Weber, T., 2014. Relation algebra. Archive of Formal Proofs https://isa-afp.org/entries/Relation_Algebra.html, Formal proof development.

[2] Backhouse, R., van den Eijnde, J.P.H.W., van Gasteren, A.J.M., 1994. Calculating path algorithms. Science of Computer Programming 22, 3–19. doi:10.1016/0167-6423(94)90005-1.

[3] Backhouse, R.C., Carré, B.A., 1975. Regular algebra applied to path-finding problems. Journal of the Institute of Mathematics and its Applications 15, 161–186. doi:10.1093/imamat/15.2.161.

[4] Berge, C., 2001. The Theory of Graphs. Dover Publications.

[5] Berghammer, R., 1999. Combining relational calculus and the Dijkstra–Gries method for deriving relational programs. Information Sciences 119, 155–171. doi:10.1016/s0020-0255(99)00012-2.

[6] Berghammer, R., Danilenko, N., Höfner, P., Stucke, I., 2016. Cardinality of relations with applications. Discrete Mathematics 339, 3089–3115. doi:10.1016/j.disc.2016.06.019.

[7] Berghammer, R., Fischer, S., 2015. Combining relation algebra and data refinement to develop rectangle-based functional programs for reflexive-transitive closures. Journal of Logical and Algebraic Methods in Programming 84, 341–358. doi:10.1016/j.jlamp.2014.08.003.

[8] Berghammer, R., Hoffmann, T., 2001. Calculating a relational program for transitive reductions of strongly connected graphs, in: de Swart, H. (Ed.), Relational Methods in Computer Science (RelMiCS 2001), Springer. pp. 258–275. doi:10.1007/3-540-36280-0_18.

[9] Berghammer, R., Hoffmann, T., 2001. Relational depth-first search with applications. Information Sciences 139, 167–186. doi:10.1016/s0020-0255(01)00163-3.

[10] Berghammer, R., Höfner, P., Stucke, I., 2014. Automated verification of relational while-programs, in: Höfner, P., Jipsen, P., Kahl, W., Müller, M.E. (Eds.), Relational and Algebraic Methods in Computer Science (RAMiCS 2014), Springer. pp. 173–190. doi:10.1007/978-3-319-06251-8_11.

[11] Berghammer, R., Höfner, P., Stucke, I., 2015. Tool-based verification of a relational vertex coloring program, in: Kahl, W., Oliveira, J.N., Winter, M. (Eds.), Relational and Algebraic Methods in Computer Science (RAMiCS 2015), Springer. pp. 275–292. doi:10.1007/978-3-319-24704-5_17.

[12] Berghammer, R., Höfner, P., Stucke, I., 2016. Cardinality of relations and relational approximation algorithms. Journal of Logical and Algebraic Methods in Programming 85, 269–286. doi:10.1016/j.jlamp.2015.12.001.

[13] Berghammer, R., Struth, G., 2010. On automated program construction and verification, in: Bolduc, C., Desharnais, J., Ktari, B. (Eds.), Mathematics of Program Construction (MPC 2010), Springer. pp. 22–41. doi:10.1007/978-3-642-13321-3_4.

[14] Bertot, Y., Castéran, P., 2004. Interactive theorem proving and program development. Texts in Theoretical Computer Science, Springer. doi:10.1007/978-3-662-07964-5.

[15] Blanchette, J.C., Böhme, S., Paulson, L.C., 2013. Extending Sledgehammer with SMT solvers. Journal of Automated Reasoning 51, 109–128. doi:10.1007/s10817-013-9278-5.

[16] Cormen, T.H., Leiserson, C.E., Rivest, R.L., 1990. Introduction to Algorithms. MIT Press.

[17] Crvenković, S., Madarász, R.S., 1993. On Kleene algebras. Theoretical Computer Science 108, 17–24. doi:10.1016/0304-3975(93)90228-L.

[18] Dang, H.H., Höfner, P., 2008. First-order theorem prover evaluation w.r.t. relation- and Kleene algebra, in: Berghammer, R., Möller, B., Struth, G. (Eds.), Relations and Kleene Algebra in Computer Science: PhD Programme at RelMiCS10/AKA5, Institut für Informatik, Universität Augsburg. pp. 48–52.

[19] de Swart, H., Orłowska, E., Schmidt, G., Roubens, M. (Eds.), 2003. Theory and Applications of Relational Structures as Knowledge Instruments. volume 2929 of *Lecture Notes in Computer Science*, Springer. doi:10.1007/b94817.

[20] de Swart, H., Orłowska, E., Schmidt, G., Roubens, M. (Eds.), 2006. Theory and Applications of Relational Structures as Knowledge Instruments II. volume 4342 of *Lecture Notes in Computer Science*, Springer. doi:10.1007/11964810.

[21] Desharnais, J., Möller, B., Struth, G., 2011. Algebraic notions of termination. Logical Methods in Computer Science 7, 1–29. doi:10.2168/lmcs-7(1:1)2011.

[22] Diestel, R., 2005. Graph Theory. third ed., Springer. doi:10.1007/978-3-662-53622-3.

[23] Doornbos, H., Backhouse, R., van der Woude, J., 1997. A calculational approach to mathematical induction. Theoretical Computer Science 179, 103–135. doi:10.1016/S0304-3975(96)00154-5.

[24] Foster, S., Struth, G., Weber, T., 2011. Automated engineering of relational and algebraic methods in Isabelle/HOL, in: de Swart, H. (Ed.), Relational and Algebraic Methods in Computer Science, Springer. pp. 52–67. doi:10.1007/978-3-642-21070-9_5.

[25] Frias, M.F., Aguayo, N., Novak, B., 1993. Development of graph algorithms with fork algebras, in: XIX Conferencia Latinoamericana de Informática, pp. 529–554.

[26] Glück, R., 2017. Algebraic investigation of connected components, in: Höfner, P., Pous, D., Struth, G. (Eds.), Relational and Algebraic Methods in Computer Science (RAMiCS 2017), Springer. pp. 109–126. doi:10.1007/978-3-319-57418-9_7.

[27] Guttmann, W., 2016. An algebraic approach to computations with progress. Journal of Logical and Algebraic Methods in Programming 85, 520–539. doi:10.1016/j.jlamp.2015.11.009.

[28] Guttmann, W., 2018. An algebraic framework for minimum spanning tree problems. Theoretical Computer Science 744, 37–55. doi:10.1016/j.tcs.2018.04.012.

[29] Guttmann, W., 2018. Verifying minimum spanning tree algorithms with Stone relation algebras. Journal of Logical and Algebraic Methods in Programming 101, 132–150. doi:10.1016/j.jlamp.2018.09.005.

[30] Guttmann, W., Höfner, P., 2020. Relational characterisations of paths. Archive of Formal Proofs https://isa-afp.org/entries/Relational_Paths.html, Formal proof development.

[31] Harary, F., 1969. Graph Theory. Addison-Wesley Publishing Company. doi:10.21236/ad0705364.

[32] Hirsch, R., Hodkinson, I., 2002. Relation Algebras by Games. volume 147 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science B.V.

[33] Hoare, C.A.R., 1969. An axiomatic basis for computer programming. Communications of the ACM 12, 576–580/583. doi:10.1145/363235.363259.

[34] Höfner, P., Struth, G., 2008. On automating the calculus of relations, in: Armando, A., Baumgartner, P., Dowek, G. (Eds.), Automated Reasoning (IJCAR 2008), Springer. pp. 50–66. doi:10.1007/978-3-540-71070-7_5.

[35] Huntington, E.V., 1933. Boolean algebra. A correction to: "New sets of independent postulates for the algebra of logic, with special reference to Whitehead and Russell's Principia mathematica". Transactions of the American Mathematical Society 35, 557–558. doi:10.1090/S0002-9947-1933-1501702-9.

[36] Huntington, E.V., 1933. New sets of independent postulates for the algebra of logic. Transactions of the American Mathematical Society 35, 274–304. doi:10.1090/S0002-9947-1933-1501684-X.

[37] Jónsson, B., Tarski, A., 1952. Boolean algebras with operators, Part II. American Journal of Mathematics 74, 127–162. doi:10.2307/2372074.

[38] Kahn, A.B., 1962. Topological sorting of large networks. Communications of the ACM 5, 558–562. doi:10.1145/368996.369025.

[39] Kawahara, Y., 2006. On the cardinality of relations, in: Schmidt, R.A. (Ed.), Relations and Kleene Algebra in Computer Science, Springer. pp. 251–265. doi:10.1007/11828563_17.

[40] Kozen, D., 1994. A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation 110, 366–390. doi:10.1006/inco.1994.1037.

[41] MacCaull, W., Orłowska, E., 2002. Correspondence results for relational proof systems with application to the Lambek calculus. Studia

Logica 71, 389–414. doi:10.1023/A:1020572931854.

[42] Maddux, R., 1983. A sequent calculus for relation algebras. Annals of Pure and Applied Logic 25, 73–101. doi:10.1016/0168-0072(83)90055-6.

[43] Maddux, R., Tarski, A., 1976. A sufficient condition for the representability of relation algebras. Notices of the American Mathematical Society 23, A–477.

[44] Maddux, R.D., 1991. Pair-dense relation algebras. Transactions of the American Mathematical Society 328, 83–131. doi:10.1090/S0002-9947-1991-1049616-1.

[45] Maddux, R.D., 1997. Relation algebras, in: Brink, C., Kahl, W., Schmidt, G. (Eds.), Relational Methods in Computer Science, Springer. pp. 22–38. doi:10.1007/978-3-7091-6510-2_2.

[46] Maddux, R.D., 2006. Relation Algebras. Elsevier.

[47] McCune, W., 2005–2010. Prover9 and Mace4. https://www.cs.unm.edu/~mccune/prover9/. (accessed 15/10/2019).

[48] Meng, J., Paulson, L.C., 2008. Translating higher-order clauses to first-order clauses. Journal of Automated Reasoning 40, 35–60. doi:10.1007/s10817-007-9085-y.

[49] Möller, B., 2013. Modal knowledge and game semirings. The Computer Journal 56, 53–69. doi:10.1093/comjnl/bxs140.

[50] Möller, B., Roocks, P., Endres, M., 2012. An algebraic calculus of database preferences, in: Gibbons, J., Nogueira, P. (Eds.), Mathematics of Program Construction, Springer. pp. 241–262. doi:10.1007/978-3-642-31113-0_13.

[51] Müller, M.E., 2012. Relational Knowledge Discovery. Cambridge University Press. doi:10.1017/CBO9781139047869.

[52] Ng, K.C., 1984. Relation Algebras with Transitive Closure. Ph.D. thesis. University of California, Berkeley.

[53] Nipkow, T., 1998. Winskel is (almost) right: Towards a mechanized semantics textbook. Formal Aspects of Computing 10, 171–186. doi:10.1007/s001650050009.

[54] Nipkow, T., 2002. Hoare logics in Isabelle/HOL, in: Schwichtenberg, H., Steinbrüggen, R. (Eds.), Proof and System-Reliability, Kluwer Academic Publishers. pp. 341–367. doi:10.1007/978-94-010-0413-8_11.

[55] Nipkow, T., Paulson, L.C., Wenzel, M., 2002. Isabelle/HOL: A Proof Assistant for Higher-Order Logic. volume 2283 of *Lecture Notes in Computer Science*. Springer. doi:10.1007/3-540-45949-9.

[56] Okuma, H., Kawahara, Y., 2000. Relational aspects of relational database dependencies. Bulletin of Informatics and Cybernetics 32, 91–104. doi:10.5109/13495.

[57] Paulson, L.C., Blanchette, J.C., 2010. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers, in: Sutcliffe, G., Schulz, S., Ternovska, E. (Eds.), International Workshop on the Implementation of Logics (IWIL 2010), EasyChair. pp. 1–11. doi:10.29007/tnfd.

[58] Pous, D., 2016. Automata for relation algebra and formal proofs. Habilitation à diriger des recherches. ENS Lyon.

[59] RelView, 1989–2016. RelView system. https://www.informatik.uni-kiel.de/~progsys/relview. (accessed 03/12/2019).

[60] Schmidt, G., 2010. Relational Mathematics. Cambridge University Press. doi:10.1017/cbo9780511778810.

[61] Schmidt, G., 2012. Relational concepts in social choice, in: Kahl, W., Griffin, T. (Eds.), Relational and Algebraic Methods in Computer Science, Springer. pp. 278–293. doi:10.1007/978-3-642-33314-9_19.

[62] Schmidt, G., Berghammer, R., 2008. Relational measures and integration in preference modeling. Journal of Logic and Algebraic Programming 76, 112–129. doi:10.1016/j.jlap.2007.10.001.

[63] Schmidt, G., Ströhlein, T., 1985. Relation algebras: Concept of point and representability. Discrete Mathematics 54, 83–92. doi:10.1016/0012-365X(85)90064-0.

[64] Schmidt, G., Ströhlein, T., 1993. Relations and Graphs: Discrete Mathematics for Computer Scientists. Springer. doi:10.1007/978-3-642-77968-8.

[65] Schulz, S., 2013. System description: E 1.8, in: McMillan, K., Middeldorp, A., Voronkov, A. (Eds.), Logic for Programming Artificial Intelligence and Reasoning (LPAR 19), Springer. pp. 735–743. doi:10.1007/978-3-642-45221-5_49.

[66] Scollo, G., Franco, G., Manca, V., 2006. A relational view of recurrence and attractors in state transition dynamics, in: Schmidt, R.A. (Ed.), Relations and Kleene Algebra in Computer Science, Springer. pp. 358–372. doi:10.1007/11828563_24.

[67] Tarski, A., 1941. On the calculus of relations. The Journal of Symbolic Logic 6, 73–89. doi:10.2307/2268577.

[68] Tinhofer, G., 1976. Methoden der angewandten Graphentheorie. Springer. doi:10.1007/978-3-7091-2301-0.

[69] Wenzel, M., 1999. Isar — a generic interpretative approach to readable formal proof documents, in: Bertot, Y., Dowek, G., Théry, L., Hirschowitz, A., Paulin, C. (Eds.), Theorem Proving in Higher Order Logics, Springer. pp. 167–183. doi:10.1007/3-540-48256-3_12.

[70] Wenzel, M., 2002. Isabelle, Isar – A Versatile Environment for Human Readable Formal Proof Documents. Ph.D. thesis. Technical University Munich, Germany.

[71] Wenzel, M., 2019. The Isabelle/Isar Reference Manual. URL: https://isabelle.in.tum.de/doc/isar-ref.pdf.

[72] von Wright, J., 2004. Towards a refinement algebra. Science of Computer Programming 51, 23–45. doi:10.1016/j.scico.2003.09.002.

Appendix A. Examples of Isabelle/HOL Proofs with Explanations

All concepts, theorems and algorithms described in this paper have been implemented in Isabelle/HOL [55]. All results have been formally verified in this system making heavy use of Sledgehammer, which integrates automated theorem provers and SMT solvers [15, 57]. The proofs can be found in the theory files [30].

Most of the times when Sledgehammer failed to prove a theorem fully automatically, rather than using Isabelle's deductive proof systems we used Isar (Intelligible semi-automated reasoning) [69, 70].

> ‘Drawing from both the traditions of informal mathematical proof texts and high-level programming languages, Isar offers a versatile environment for structured formal proof documents. Thus properly written

> Isar proofs become accessible to a broader audience than unstructured tactic scripts (which typically only provide operational information for the machine). Writing human-readable proof texts certainly requires some additional efforts by the writer to achieve a good presentation, both of formal and informal parts of the text. On the other hand, human-readable formal texts gain some value in their own right, independently of the mechanic proof-checking process.’ [71]

By copying lemmas and proofs verbatim from the Isabelle files, this appendix gives an impression how readable these proofs are. We begin with the (very simple proof) of the first statement of Lemma 2.1. Its encoding is straightforward; instead of ⊆ we use ≤ in Isabelle/HOL.

lemma inj-sur-semi-swap:
assumes is-sur z
  and is-inj x
 shows z ≤ y;x ⟹ x ≤ y^T;z

The lemma can be referred to by the name inj-sur-semi-swap. Its preconditions are that the relation z is surjective and the relation x is injective – encoded by using predicates is-sur and is-inj, respectively. The statement of this lemma corresponds one-to-one with Lemma 2.1.

In the following we present the Isabelle/HOL proof on the left. For comparison we present a ‘classical’ textbook proof on the right-hand side.

proof –	**proof**
assume z ≤ y;x	$z \subseteq y ; x$
hence z;x^T ≤ y;(x;x^T)	⇒ ⦃isotonicity of ;⦄
by (metis mult-isor mult-assoc)	$z ; x^{\mathsf{T}} \subseteq y ; x ; x^{\mathsf{T}}$
hence z;x^T ≤ y	⇒ ⦃$x ; x^{\mathsf{T}} \leq \mathsf{I}$ as x is injective⦄
using ‹is-inj x› **unfolding** is-inj-def	$z ; x^{\mathsf{T}} \subseteq y$
by (metis mult-isol order.trans mult-1-right)	
hence (z^T;z);x^T ≤ z^T;y	⇒ ⦃isotonicity of ;⦄
by (metis mult-isol mult-assoc)	$z^{\mathsf{T}} ; z ; x^{\mathsf{T}} \subseteq z^{\mathsf{T}} ; y$
hence x^T ≤ z^T;y	⇒ ⦃$\mathsf{I} \leq z ; z^{\mathsf{T}}$ as z is surjective⦄
using ‹is-sur z› **unfolding** is-sur-def	$x^{\mathsf{T}} \subseteq z^{\mathsf{T}} ; y$
by (metis mult-isor order.trans mult-1-left)	
thus x ≤ y^T;z	⇒ ⦃isotonicity of $^{\mathsf{T}}$, $x^{\mathsf{T}\mathsf{T}}$=x and $(x ; y)^{\mathsf{T}}$=$y^{\mathsf{T}} ; x^{\mathsf{T}}$⦄
using conv-iso **by** fastforce	$x \subseteq y^{\mathsf{T}} ; z$
qed	**qed**

We see that the proofs are identical, modulo some syntax and details about the lemmas used in each step. Lines starting with **by** were found automatically by Sledgehammer. Actually, Sledgehammer is more powerful and it is not necessary to specify every single step as we did above. Larger steps can be made as in the following proof of the same result.

assume z ≤ y;x
hence z;x^T ≤ y
 using ‹z ≤ y;x› ‹is-inj x› **unfolding** is-inj-def **by** ...
hence x^T ≤ z^T;y
 using ‹is-sur z› **unfolding** is-sur-def **by** ...
thus x ≤ y^T;z
 using conv-iso **by** fastforce

We now consider a result with a more complicated proof, namely the first claim of Theorem 4.2. As before, its encoding is straightforward.

lemma start-point-at-most-one:
assumes path x
 shows is-inj (start-points x)

We show again the Isabelle/HOL proof on the left, and its ‘translation’ on the right. Our Isabelle/HOL notation follows Maddux [46]: we denote union by +, intersection by ·, complement by the prefix −, the greatest element by 1, and the identity by 1′. In a normal textbook, each step comes with an explanation, often using lemmas presented before. We omit most of these explanations on the right and references to the corresponding lemmas on the left.

proof –
have isvec: is-vector $(x;1 \cdot -(x^T;1))$ **by** ...

have $x;1 \cdot 1;x^T \leq x;1;x;x^T$ **by** ...
also have ... $\leq (x^\star + x^{T\star});x^T$ **using** ‹path x› **by** ...
also have ... $= x^T + x^+;x^T + x^{T+}$ **by** ...
also have ... $\leq x^{T+} + x^+;x^T + x^{T+}$ **by** ...
also have ... $\leq x^\star;x;x^T + x^{T+}$ **by** ...
also have ... $\leq x^\star;1' + x^{T+}$ **using** ‹path x› **by** ...
also have ... $= 1' + x^\star;x + x^T;x^{T\star}$ **by** ...
also have ... $\leq 1' + 1;x + x^T;1$ **by** ...
finally have aux: $x;1 \cdot 1;x^T \leq 1' + 1;x + x^T;1$ **.**

from aux **have** $x;1 \cdot 1;x^T \cdot -(x^T;1) \cdot -(1;x) \leq 1'$ **by** ...
hence $(x;1 \cdot -(x^T;1)) \cdot (x;1 \cdot -(x^T;1))^T \leq 1'$ **by** ...
with isvec **have** $(x;1 \cdot -(x^T;1)) ; (x;1 \cdot -(x^T;1))^T \leq 1'$ **by** ...
thus is-inj (start-points x) **by** ...
qed

proof
It is easy to see that $x \,;\mathsf{L} \cap \overline{x^\mathsf{T} \,; \mathsf{L}}$ is a vector.

Next we prove $x \,; \mathsf{L} \cap \mathsf{L} \,; x^\mathsf{T} \subseteq \mathsf{I} \cup \mathsf{L} \,; x \cup x^\mathsf{T} \,; \mathsf{L}$ as auxiliary statement, using inequational reasoning.

$$\begin{array}{rcll}
x \,; \mathsf{L} \cap \mathsf{L} \,; x^\mathsf{T} & \subseteq & x \,; \mathsf{L} \,; x \,; x^\mathsf{T} & \\
& \subseteq & (x^* \cup (x^\mathsf{T})^*) \,; x^\mathsf{T} & \{\!|\text{using assumption}|\!\} \\
& = & x^\mathsf{T} \cup x^+ \,; x^\mathsf{T} \cup (x^\mathsf{T})^+ & \\
& \subseteq & (x^\mathsf{T})^+ \cup x^+ \,; x^\mathsf{T} \cup (x^\mathsf{T})^+ & \\
& \subseteq & x^* \,; x \,; x^\mathsf{T} \cup (x^\mathsf{T})^+ & \\
& \subseteq & x^* \,; \mathsf{I} \cup (x^\mathsf{T})^+ & \{\!|\text{using assumption}|\!\} \\
& = & \mathsf{I} \cup x^* \,; x \cup x^\mathsf{T} \,; (x^\mathsf{T})^* & \\
& \subseteq & \mathsf{I} \cup \mathsf{L} \,; x \cup x^\mathsf{T} \,; \mathsf{L} &
\end{array}$$

By shunting we get $x \,; \mathsf{L} \cap \mathsf{L} \,; x^\mathsf{T} \cap \overline{\mathsf{L} \,; x} \cap \overline{x^\mathsf{T} \,; \mathsf{L}} \subseteq \mathsf{I}$.
Using $(x \,; y)^\mathsf{T} = y^\mathsf{T} \,; x^\mathsf{T}$ and $\overline{x}^\mathsf{T} = \overline{x^\mathsf{T}}$ we obtain
$(x \,; \mathsf{L} \cap \overline{x^\mathsf{T} \,; \mathsf{L}}) \cap (x \,; \mathsf{L} \cap \overline{x^\mathsf{T} \,; \mathsf{L}})^\mathsf{T} \subseteq \mathsf{I}$.
Since $x;\mathsf{L} \cap \overline{x^\mathsf{T};\mathsf{L}}$ is a vector, Lemma 2.2 yields
$(x \,; \mathsf{L} \cap \overline{x^\mathsf{T} \,; \mathsf{L}}) \,; (x \,; \mathsf{L} \cap \overline{x^\mathsf{T} \,; \mathsf{L}})^\mathsf{T} \subseteq \mathsf{I}$.
qed