An Algebraic Framework for Minimum Spanning Tree Problems

Walter Guttmann

Department of Computer Science and Software Engineering, University of Canterbury, New Zealand
walter.guttmann@canterbury.ac.nz

Abstract

We formally prove that Prim's minimum spanning tree algorithm is correct for various optimisation problems with different aggregation functions. The original minimum weight spanning tree problem and the minimum bottleneck spanning tree problem are special cases, but the framework covers many other problems. To this end we work in new generalisations of relation algebras and Kleene algebras and in new algebraic structures that capture key operations used in Prim's algorithm and its specification. Weighted graphs form instances of these algebraic structures, where edge weights are taken from a linear order with a binary aggregation operation. Many existing results from relation algebras and Kleene algebras generalise from the relation model to the weighted-graph model with no or small changes. The overall structure of the proof uses Hoare logic. All results are formally verified in Isabelle/HOL heavily using its integrated automated theorem provers. This paper is an extended version of [36].

Keywords: aggregation, correctness proof, formal methods, Kleene algebras, relations, Stone algebras, weighted graphs

---

1. Introduction

A well-known algorithm commonly attributed to Prim [59] – and independently discovered by Jarník [41] and Dijkstra [25] – computes a minimum spanning tree in a weighted undirected graph. It starts with an arbitrary root node, and constructs a tree by repeatedly adding an edge that has minimal weight among the edges connecting a node in the tree with a node not in the tree. The iteration stops when there is no such edge, at which stage the constructed tree is a minimum spanning tree of the component of the graph that contains the root (which is the whole graph if it is connected).

The aim of this paper is to demonstrate the applicability of relation-algebraic methods for verifying the correctness of algorithms on weighted graphs. Accordingly, we will use an implementation of Prim's algorithm close to the above abstraction level. Since its discovery many efficient implementations of this and other spanning tree algorithms have been developed; for example, see the two surveys [32, 50]. These implementations typically rely on specific data structures, which can be introduced into a high-level algorithm by means of data refinement; for example, see [8]. We do not pursue this in the present paper.

Relation-algebraic methods have been used to develop algorithms for unweighted graphs; for example, see [28, 10, 9, 8]. This works well because such a graph can be directly represented as a relation; an adjacency matrix is a Boolean matrix. Weighted graphs do not have a direct representation as a binary relation. Previous relational approaches to weighted graphs therefore use many-sorted representations such as an incidence matrix and a weight function. In this paper, we directly work with a matrix of weights.

In the context of fuzzy systems, relations have been generalised from Boolean matrices to matrices over the real interval $[0, 1]$ or over arbitrary complete distributive lattices [29]. The underlying idea is to extend qualitative to quantitative methods; see [57] for another instance based on automata. We propose to use matrices over lattices to model weighted graphs, in particular in graph algorithms. Previous work based on semirings and Kleene algebras deals well with path problems in graphs [30]. We combine these algebras with generalisations of relation algebras to tackle the minimum spanning tree problem.

Tarski's relation algebras [63], which capture Boolean matrices, have been generalised to Dedekind categories to algebraically capture fuzzy relations [44]; these categories are also known as locally complete division allegories [27]. In the present paper we introduce a new generalisation – Stone relation algebras – which maintains the signature of relation algebras and weakens the underlying Boolean algebra structure to Stone algebras. Stone algebras have a pseudocomplement operation, which satisfies some properties of a complement but not all. This weakening is needed since complementation does not extend from two to more elements in linear orders, in this case, the linear order of edge weights. We show that matrices over bounded linear orders are instances of Stone relation algebras and of Kleene algebras, and can be used to represent weighted graphs.

Most of the correctness proof of Prim's minimum spanning tree algorithm can be carried out in these general algebras. Therefore, most of our results hold for many instances, not just weighted graphs. A small part of the correctness proof uses operations beyond those available in relation algebras and in Kleene algebras, namely for summing edge weights and identifying minimal edges. We introduce new algebraic structures – m-algebras – that capture essential properties of these operations, which are used in Prim's algorithm and in its specification.

Because the correctness proof of Prim's algorithm relies only on the axioms of m-algebras, we can construct different instances of these algebras to solve various problems. The instances are based on bounded linear orders with a binary aggregation operation. Applied to all entries of a weighted-graph matrix, aggregation describes the value that is minimised by Prim's algorithm. The minimum weight spanning tree problem arises as the instance where aggregation computes the sum of edge weights (say, real numbers). For the minimum bottleneck spanning tree problem aggregation amounts to the maximum of edge weights [18]. Covering these and further optimisation problems, we propose axioms for the aggregation operation based on which Prim's algorithm will work correctly. With this algebraic method, each particular problem arises as a specific instance of the involved algebras. This is analogous to the generalisation of Warshall's transitive closure algorithm [64] and Floyd's all-pairs shortest paths algorithm [26] to a general dynamic-programming algorithm based on semirings, which covers many other instances [2].

With this approach we can apply well-developed methods and concepts of relation algebras and Kleene algebras to reason about weighted graphs in a new, more direct way. The contributions of this paper are:

- Stone relation algebras, a new algebraic structure that generalises relation algebras but maintains their signature (Section 2). Many theorems of relation algebras already hold in these weaker algebras, which we use to represent weighted graphs. We combine them with Kleene algebras to describe reachability in graphs (Section 3). This yields a general yet expressive setting for most of the correctness proof of the minimum spanning tree algorithm.

- M-algebras, a new algebraic structure that extends Stone-Kleene relation algebras by dedicated operations and axioms for finding minimal edges and for computing the total weight of a graph (Section 4).

- Models of the above algebras, including weighted graphs represented by matrices over bounded linear orders (Sections 2, 3 and 5). This includes a formal verification of Conway's automata-based construction for the Kleene star of a matrix.

- Linear aggregation lattices, a new algebraic structure based on bounded linear orders with a binary aggregation operation (Section 5). Matrices over linear aggregation lattices form m-algebras.

- Several classes of instances of linear aggregation lattices covering, in particular, the minimum weight spanning tree and minimum bottleneck spanning tree problems (Section 6). Aggregations based on triangular norms and conorms also arise as special cases.

- A Hoare-logic correctness proof of Prim's minimum spanning tree algorithm entirely based on the above algebras (Section 7).

- Isabelle/HOL theories that formally verify all of the above and all results in and about the algebras stated in the present paper. Proofs are omitted in this paper and can be found in the Isabelle/HOL theory files available at http://www.csse.canterbury.ac.nz/walter.guttmann/algebra/. The Archive of Formal Proofs at https://www.isa-afp.org/ currently holds theories with all results of Sections 2 and 3; the other theories are being prepared for it.

Related work is discussed in Section 8.

The present paper is an extended version of [36]. We have generalised the results of Section 2 from extended real numbers to linear bounded orders. Major changes concern Section 4 and the two new Sections 5 and 6. In Section 4, we have revised Definition 6 and simplified the axioms for s-algebras and m-algebras taking into account new results about Stone relation algebras [37]. We have also added Theorem 4. Section 5 is entirely new. It motivates how different kinds of aggregation amount to solving different optimisation problems. We propose various algebraic structures based on orders with a binary aggregation operation. We show how to extend binary aggregation to finite sums despite the absence of a unit. We define general summation and minimisation operations on matrices based on the binary aggregation, and prove that this gives instances of s-algebras and m-algebras. Also Section 6 is entirely new. It discusses several ways to instantiate the algebraic structures for aggregation. The instances include the minimum weight spanning tree problem, the minimum bottleneck spanning tree problem and the minimum spanning tree problems using arbitrary t-norms or t-conorms as aggregations. All new results, in particular Theorems 5–13 in Sections 5 and 6, have been formally proved in Isabelle/HOL requiring a substantial extension of the theories developed for [36]. The paper [37] focuses on general properties of Stone relation algebras and logical characterisations of relation-algebraic properties in the weighted-graph model; this is not related to the new development in the present paper.

2. Stone relation algebras

In this section we introduce Stone relation algebras, which generalise relation algebras so as to model not just Boolean matrices but matrices over arbitrary values required to represent weighted graphs. Each entry in such a matrix is taken from a linear order with a least element $\bot$ and a greatest element $\top$; examples are the natural numbers or the real numbers suitably extended by $\bot$ and $\top$. If the entry in row i and column j of the matrix is $\bot$, this means there is no edge from node i to node j. If the entry is neither $\bot$ nor $\top$, there is an edge with that weight. An entry of $\top$ is used to record the presence of an edge without information about its weight; see below.

To work with edge weights and matrices of edge weights (weighted graphs) we use the following well-known algebraic structures [14, 33, 17]. The first definition covers the structures related to lattices.

Definition 1. *A* commutative semigroup *is an algebraic structure $(S, \sqcup)$ where $\sqcup$ is associative and commutative:*

$$x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z \qquad x \sqcup y = y \sqcup x$$

A commutative monoid $(S, \sqcup, \bot)$ *is a commutative semigroup $(S, \sqcup)$ with a unit $\bot$ satisfying*

$$x \sqcup \bot = x$$

A bounded semilattice *is a commutative monoid where $\sqcup$ is idempotent:*

$$x \sqcup x = x$$

A bounded lattice *is an algebraic structure $(S, \sqcup, \sqcap, \bot, \top)$ where $(S, \sqcup, \bot)$ and $(S, \sqcap, \top)$ are bounded semilattices and the following absorption axioms hold:*

$$x \sqcup (x \sqcap y) = x \qquad x \sqcap (x \sqcup y) = x$$

A bounded distributive lattice *is a bounded lattice satisfying the distributivity axioms*

$$x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z) \qquad x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$$

A dense lattice *is a bounded distributive lattice where* $\bot$ *is meet-irreducible:*

$$x \sqcap y = \bot \;\Rightarrow\; x = \bot \vee y = \bot$$

The lattice order *is given by*

$$x \le y \;\Leftrightarrow\; x \sqcup y = y$$

□

The second definition covers pseudocomplemented algebras. These are algebras with a unary operation satisfying some of the properties of a complement but not necessarily all.

Definition 2. *A* distributive p-algebra $(S, \sqcup, \sqcap, \overline{}, \bot, \top)$ *is a bounded distributive lattice* $(S, \sqcup, \sqcap, \bot, \top)$ *with a pseudocomplement operation* $\overline{}$ *satisfying the equivalence*

$$x \sqcap y = \bot \;\Leftrightarrow\; x \le \overline{y}$$

This means that $\overline{y}$ *is the* $\le$*-greatest element whose meet with* y *given by* $\sqcap$ *is* $\bot$*. A* Stone algebra *is a distributive p-algebra satisfying the equation*

$$\overline{x} \sqcup \overline{\overline{x}} = \top$$

An element $x \in S$ *is* regular *if* $\overline{\overline{x}} = x$*. A* Boolean algebra *is a Stone algebra whose elements are all regular.* □

The third definition covers orders and structures related to ordered monoids. Linearly ordered commutative semigroups are discussed in [19].

Definition 3. *A* partial order $\le$ *on a set* S *is a reflexive, transitive and antisymmetric relation on* S*:*

$$x \le x \qquad x \le y \wedge y \le z \Rightarrow x \le z \qquad x \le y \wedge y \le x \Rightarrow x = y$$

A linear order *is a partial order satisfying*

$$x \le y \;\vee\; y \le x$$

The minimum *and the* maximum *of two elements* x, y *in a linear order are given by*

$$x{\downarrow}y = \begin{cases} x & \text{if } x \le y \\ y & \text{if } y \le x \end{cases} \qquad x{\uparrow}y = \begin{cases} y & \text{if } x \le y \\ x & \text{if } y \le x \end{cases}$$

A linearly ordered commutative semigroup $(S, \le, \sqcup)$ *is a commutative semigroup* $(S, \sqcup)$ *with a linear order* $\le$ *on* S *such that* $\sqcup$ *is* $\le$*-isotone:*

$$x \le y \;\Rightarrow\; z \sqcup x \le z \sqcup y$$

A linearly ordered commutative monoid *is a structure* $(S, \le, \sqcup, \bot)$ *where* $(S, \sqcup, \bot)$ *is a commutative monoid and* $(S, \le, \sqcup)$ *is a linearly ordered commutative semigroup.*

A bounded linear order $(S, \le, \bot, \top)$ *is a set* S *with a linear order* $\le$ *and a* $\le$*-least element* $\bot$ *and a* $\le$*-greatest element* $\top$*:*

$$\bot \le x \qquad x \le \top$$

A bounded linearly ordered commutative monoid $(S, \le, \sqcup, \bot, \top)$ *is a bounded linear order* $(S, \le, \bot, \top)$ *with an operation* $\sqcup$ *such that* $(S, \le, \sqcup, \bot)$ *is a linearly ordered commutative monoid.* □

We refer to a set with a partial/linear order also as a partial/linear order. Every bounded linear order forms a bounded distributive lattice using minimum and maximum as join and meet, respectively. However, a Boolean complement cannot be introduced in this lattice if the linear order has more than two elements, which is why we use the weaker Stone algebras for edge weights. We obtain the following consequences for Stone algebras; in particular, every bounded linear order forms a Stone algebra and so do matrices over a bounded linear order. See [30] for similar matrix semirings and the max-min semiring of extended real numbers. The set of square matrices with indices from a set A and entries from a set S is denoted by $S^{A \times A}$. It represents a graph with node set A and edge weights taken from S.

Theorem 1.

1. *The regular elements of every Stone algebra S form a Boolean algebra that is a subalgebra of S [33].*
2. *Let $(S, \sqcup, \sqcap, \overline{}, \bot, \top)$ be a Stone algebra and let A be a non-empty set. Then $(S^{A \times A}, \sqcup, \sqcap, \overline{}, \bot, \top)$ is a Stone algebra, where the operations $\sqcup$, $\sqcap$, $\overline{}$, $\bot$, $\top$ and the lattice order $\leq$ are lifted componentwise.*
3. *Let $(S, \leq, \bot, \top)$ be a bounded linear order. Then $(S, \uparrow, \downarrow, \overline{}, \bot, \top)$ is a Stone algebra with*

$$\overline{x} = \begin{cases} \top & \text{if } x = \bot \\ \bot & \text{if } x \neq \bot \end{cases}$$

and the order $\leq$ on S as the lattice order. □

The regular elements of the Stone algebra formed by a bounded linear order are $\bot$ and $\top$. In particular, applying the pseudocomplement operation $\overline{}$ twice maps $\bot$ to itself and every other element to $\top$. Applying $\overline{}$ twice to a matrix over a bounded linear order, which represents a weighted graph, yields a matrix over $\{\bot, \top\}$ that represents the structure of the graph forgetting the weights. A related operation called the 'support' of a matrix is discussed in [48]; it works on matrices over natural numbers and maps 0 to 0 and each non-zero entry to 1. Relations are used to describe the 'shape' of a matrix of complex numbers in [24]; a shape represents a superset of the non-zero entries of a matrix, but an operator to obtain the non-zero entries is not discussed there. In [45] a 'flattening' operation yields the structure by mapping every entry x of the matrix to the smallest multiplicatively idempotent element whose product with x is x.

For any Stone algebra S, the matrices over $\{\bot, \top\}$ are regular elements of the matrix algebra $S^{A \times A}$ and form a subalgebra of it. This situation, shown in Figure 1 for weighted graphs, is analogous to that of vectors – row-constant matrices used to represent conditions in computations – which form a substructure of the encompassing relation algebra. In both cases, the substructure can be obtained as the image of a closure operation. Note that the Galois connection $x \leq \overline{y} \Leftrightarrow y \leq \overline{x}$ holds in p-algebras [37, Theorem 2.5]. In this case, by a general result of Galois connections, the closure operation arises from the composition of the pseudocomplement with itself. The regular matrices are the image of this closure operation $\lambda x.\overline{\overline{x}}$, which will be used in the correctness proof of Prim's algorithm whenever only the structure of the graph is important, not the weights. The graph structure can be represented as a (Boolean) relation; in the context of fuzzy systems these are also called 'crisp' relations to distinguish them from fuzzy relations [29]. An operation to obtain the least crisp relation containing a given fuzzy relation is discussed in [65].

The order $\leq$ of Stone algebras allows us to compare edge weights. For matrices the comparison and all operations of Stone algebras work componentwise. These operations cannot be used to propagate information about edges through a graph. To combine information from edges between different pairs of nodes we add a relational structure with the operations of composition and converse. In unweighted graphs, they would be provided by relation algebras. To handle weighted graphs, we introduce the following generalisation.

Definition 4. *A* Stone relation algebra $(S, \sqcup, \sqcap, \cdot, \overline{}, ^\mathsf{T}, \bot, \top, 1)$ *is a Stone algebra $(S, \sqcup, \sqcap, \overline{}, \bot, \top)$ with a composition $\cdot$ and a converse $^\mathsf{T}$ and a constant 1 satisfying equations (1)–(10). We abbreviate $x \cdot y$ as xy*

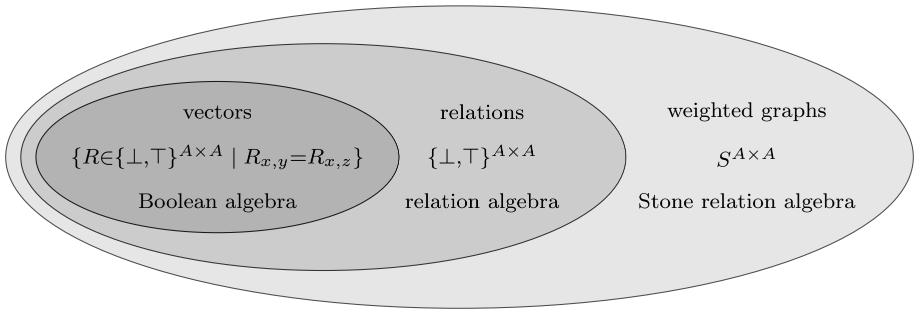

Figure 1: Relations form a substructure of weighted graphs as vectors form a substructure of relations

and let composition have higher precedence than the operators $\sqcup$ and $\sqcap$. The axioms are:

$$(xy)z = x(yz) \tag{1}$$
$$1x = x \tag{2}$$
$$(x \sqcup y)z = xz \sqcup yz \tag{3}$$
$$(xy)^\mathsf{T} = y^\mathsf{T} x^\mathsf{T} \tag{4}$$
$$(x \sqcup y)^\mathsf{T} = x^\mathsf{T} \sqcup y^\mathsf{T} \tag{5}$$
$$x^{\mathsf{T}^\mathsf{T}} = x \tag{6}$$
$$\bot x = \bot \tag{7}$$
$$xy \sqcap z \le x(y \sqcap x^\mathsf{T} z) \tag{8}$$
$$\overline{\overline{xy}} = \overline{\overline{x}}\,\overline{\overline{y}} \tag{9}$$
$$\overline{\overline{1}} = 1 \tag{10}$$

An element $x \in S$ is a vector *if $x\top = x$,* symmetric *if $x = x^\mathsf{T}$,* injective *if $xx^\mathsf{T} \le 1$,* surjective *if $1 \le x^\mathsf{T} x$ and* bijective *if x is injective and surjective. An element $x \in S$ is an* atom *if both $x\top$ and $x^\mathsf{T}\top$ are bijective. A* relation algebra *$(S, \sqcup, \sqcap, \cdot, ^-, ^\mathsf{T}, \bot, \top, 1)$ is a Stone relation algebra whose reduct $(S, \sqcup, \sqcap, ^-, \bot, \top)$ is a Boolean algebra.* ☐

We reuse the concise characterisations of vectors, atoms, symmetry, injectivity, surjectivity and bijectivity known from relation algebras [61]. In the instance of relations over a set A, a vector represents a subset of A and an atom represents a relation containing a single pair. Hence, in the graph model a vector describes a set of nodes – such as the ones visited in Prim's algorithm – and an atom describes an edge of the graph. Injectivity then means that two nodes cannot have the same successor, which is a property of trees.

Observe that relation algebras and Stone relation algebras have the same signature. The main difference between them is the weakening of the lattice structure from Boolean algebras to Stone algebras. In particular, the property

$$x^\mathsf{T} \overline{xy} \le \overline{y} \tag{11}$$

holds in Stone relation algebras. Tarski's relation algebras require a Boolean algebra, axioms (1)–(6), and property (11) [49]. Axioms (7)–(10) follow in relation algebras.

Axiom (8) has been called 'Dedekind formula' or 'modular law' [12, 44]. Besides being typed, Dedekind categories require that composition has a left residual and that each Hom-set is a complete distributive lattice [43] and therefore a Heyting algebra, which assumes that all relative pseudocomplements exist. Rough relation algebras [20] weaken the lattice structure of relation algebras to double Stone algebras, which capture properties of rough sets. Axioms (9) and (10) state that regular elements are closed under composition and its unit.

Many results of relation algebras hold in Stone relation algebras directly or with small modifications. For example, $x \leq xx^{\mathsf{T}}x$, the pseudocomplement of a vector is a vector, and composition with an injective element distributes over $\sqcap$ from the right. We also obtain the following variant of the so-called Schröder equivalence:

$$xy \leq \overline{z} \;\Leftrightarrow\; x^{\mathsf{T}}z \leq \overline{y}$$

See [37] for a detailed discussion of these and further properties of Stone relation algebras.

The following result shows three consequences of Stone relation algebras, which are important for setting up the weighted-graph model. In particular, every Stone algebra can be extended to a Stone relation algebra, and the Stone relation algebra structure can be lifted to matrices by using the usual matrix composition (taking $\sqcup$ and $\cdot$ from the underlying Stone relation algebra as addition and multiplication, respectively).

Theorem 2.

1. *The regular elements of a Stone relation algebra S form a relation algebra that is a subalgebra of S.*
2. *Let $(S, \sqcup, \sqcap, {}^{-}, \bot, \top)$ be a Stone algebra. Then $(S, \sqcup, \sqcap, \sqcap, {}^{-}, \lambda x.x, \bot, \top, \top)$ is a Stone relation algebra with the identity function as converse.*
3. *Let $(S, \sqcup, \sqcap, \cdot, {}^{-}, {}^{\mathsf{T}}, \bot, \top, 1)$ be a Stone relation algebra and let A be a non-empty finite set. Then $(S^{A\times A}, \sqcup, \sqcap, \cdot, {}^{-}, {}^{\mathsf{T}}, \bot, \top, 1)$ is a Stone relation algebra, where the operations $\cdot$, ${}^{\mathsf{T}}$ and 1 are defined by*

$$\begin{aligned}(M \cdot N)_{i,j} &= \textstyle\bigsqcup_{k\in A} M_{i,k} \cdot N_{k,j} \\ (M^{\mathsf{T}})_{i,j} &= (M_{j,i})^{\mathsf{T}} \\ 1_{i,j} &= \begin{cases} 1 & \textit{if } i = j \\ \bot & \textit{if } i \neq j \end{cases}\end{aligned}$$

□

Hence weighted graphs form a Stone relation algebra as follows: for weights the operations are $x \cdot y = x \downarrow y$ and $x^{\mathsf{T}} = x$ according to Theorem 2.2, and these operations are lifted to matrices as shown in Theorem 2.3. Because in this instance the converse operation of the underlying Stone relation algebra is the identity, the lifted converse operation only transposes the matrix. Thus for a bounded linear order S and a non-empty finite set A, the set of matrices $S^{A\times A}$ is a Stone relation algebra with the following operations:

$$\begin{aligned}(M \sqcup N)_{i,j} &= M_{i,j} \uparrow N_{i,j} \\ (M \sqcap N)_{i,j} &= M_{i,j} \downarrow N_{i,j} \\ (M \cdot N)_{i,j} &= \max_{k\in A}(M_{i,k} \downarrow N_{k,j}) \\ \overline{M}_{i,j} &= \overline{M_{i,j}} \\ M^{\mathsf{T}}{}_{i,j} &= M_{j,i} \\ \bot_{i,j} &= \bot \\ \top_{i,j} &= \top \\ 1_{i,j} &= \begin{cases} \top & \text{if } i = j \\ \bot & \text{if } i \neq j \end{cases}\end{aligned}$$

The order in this structure is $M \leq N \;\Leftrightarrow\; \forall i, j \in A : M_{i,j} \leq N_{i,j}$.

3. Stone-Kleene relation algebras

In this section, we combine Stone relation algebras with Kleene algebras [47] in order to obtain information about reachability in graphs. Kleene algebras are used to model finite iteration for regular languages and relations. In particular, they expand semirings by a unary operation – the Kleene star – which instantiates to the reflexive-transitive closure of relations. The properties of the Kleene star have been studied in [21] and we use the axiomatisation given in [47].

Definition 5. *An* idempotent semiring *is an algebraic structure* $(S, \sqcup, \cdot, \bot, 1)$ *where* $(S, \sqcup, \bot)$ *is a bounded semilattice and* $\cdot$ *is associative, distributes over* $\sqcup$ *and has unit* 1 *and zero* $\bot$*:*

$$\begin{array}{llll} x(y \sqcup z) = xy \sqcup xz & x\bot = \bot & x1 = x & x(yz) = (xy)z \\ (x \sqcup y)z = xz \sqcup yz & \bot x = \bot & 1x = x & \end{array}$$

A Kleene algebra $(S, \sqcup, \cdot, ^*, \bot, 1)$ *is an idempotent semiring* $(S, \sqcup, \cdot, \bot, 1)$ *with an operation* * *satisfying the unfold and induction axioms*

$$\begin{array}{ll} 1 \sqcup yy^* \leq y^* & z \sqcup yx \leq x \Rightarrow y^*z \leq x \\ 1 \sqcup y^*y \leq y^* & z \sqcup xy \leq x \Rightarrow zy^* \leq x \end{array}$$

A Stone-Kleene relation algebra *is a structure* $(S, \sqcup, \sqcap, \cdot, ^-, ^\mathsf{T}, ^*, \bot, \top, 1)$ *such that* $(S, \sqcup, \sqcap, \cdot, ^-, ^\mathsf{T}, \bot, \top, 1)$ *is a Stone relation algebra,* $(S, \sqcup, \cdot, ^*, \bot, 1)$ *is a Kleene algebra and the following equation holds:*

$$\overline{\overline{x^*}} = (\overline{\overline{x}})^* \tag{12}$$

An element $x \in S$ *is* acyclic *if* $xx^* \leq \overline{1}$ *and* x *is a* forest *if* x *is injective and acyclic. A* Kleene relation algebra $(S, \sqcup, \sqcap, \cdot, ^-, ^\mathsf{T}, ^*, \bot, \top, 1)$ *is a Stone-Kleene relation algebra whose reduct* $(S, \sqcup, \sqcap, \cdot, ^-, ^\mathsf{T}, \bot, \top, 1)$ *is a relation algebra.* □

Axiom (12) states that regular elements are closed under the operation *. Many results of Kleene relation algebras hold in Stone-Kleene relation algebras directly or with small modifications. For example, $(xx^\mathsf{T})^* = 1 \sqcup xx^\mathsf{T}$ for each vector x, the operations converse and Kleene star commute, and

$$x^*x^{\mathsf{T}*} \sqcap x^\mathsf{T}x \leq 1$$

for each forest x. The latter follows using the cancellation property

$$xy \leq 1 \;\Rightarrow\; x^*y^* \leq x^* \sqcup y^*$$

which we have proved in Kleene algebras as part of the present verification work; such properties can also be interpreted in rewrite systems [62]. Proofs of the above properties – and other algebraic results and consequences stated in this paper – can be found in the Isabelle/HOL theory files mentioned in the introduction. The following result shows further consequences for Kleene algebras, Stone-Kleene relation algebras and Kleene relation algebras.

Theorem 3.

1. *The regular elements of a Stone-Kleene relation algebra* S *form a Kleene relation algebra that is a subalgebra of* S*.*
2. *Let* $(S, \sqcup, \sqcap, \bot, \top)$ *be a bounded distributive lattice. Then* $(S, \sqcup, \sqcap, \lambda x.\top, \bot, \top)$ *is a Kleene algebra with the constant* $\top$ *function as the star operation.*
3. *Let* $(S, \sqcup, \sqcap, ^-, \bot, \top)$ *be a Stone algebra. Then* $(S, \sqcup, \sqcap, \sqcap, ^-, \lambda x.x, \lambda x.\top, \bot, \top, \top)$ *is a Stone-Kleene relation algebra.*
4. *Let* $(S, \sqcup, \sqcap, \cdot, ^-, ^\mathsf{T}, ^*, \bot, \top, 1)$ *be a Stone-Kleene relation algebra and let* A *be a non-empty finite set. Then* $(S^{A\times A}, \sqcup, \sqcap, \cdot, ^-, ^\mathsf{T}, ^*, \bot, \top, 1)$ *is a Stone-Kleene relation algebra, where the operation* * *is defined recursively using Conway's automata-based construction [21]:*

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^* = \begin{pmatrix} e^* & a^*bf^* \\ d^*ce^* & f^* \end{pmatrix} \qquad \textit{where} \qquad \begin{pmatrix} e \\ f \end{pmatrix} = \begin{pmatrix} a \sqcup bd^*c \\ d \sqcup ca^*b \end{pmatrix}$$

This shows the recursive case, which splits a matrix into smaller matrices. At termination, the Kleene star is applied to the entry of a one-element matrix. □

In particular, this provides a formally verified proof of Conway's construction for the Kleene star of matrices. The construction was implemented in [5] in Isabelle/HOL based on theories of Kleene algebras, but a machine-checked correctness proof was missing so far [4, Section 5.7].

As a consequence, weighted graphs form a Stone-Kleene relation algebra as follows: for weights the max-min lattice is extended with the Kleene star operation $x^* = \top$ according to Theorem 3.3, and the Kleene star is defined for matrices by Conway's construction shown in Theorem 3.4.

4. Algebras for summing and minimising weights

In this section we extend Stone-Kleene relation algebras by dedicated operations for the minimum spanning tree application. First, the algorithm needs to select an edge with minimal weight; this is done by the operation m. Second, the sum of edge weights needs to be minimised according to the specification; the sum is obtained by the operation s. Third, the axioms of s use the operation $+$ to add the weights of corresponding edges of two graphs. These operations are captured in the following algebraic structures.

Definition 6. *An* s-algebra $(S, \sqcup, \sqcap, \cdot, +, \overline{}, ^\mathsf{T}, s, \bot, \top, 1)$ *is a Stone relation algebra* $(S, \sqcup, \sqcap, \cdot, \overline{}, ^\mathsf{T}, \bot, \top, 1)$ *with an addition* $+$ *and a summation* s *satisfying the following properties:*

$$x \neq \bot \wedge s(x) \leq s(y) \Rightarrow z + s(x) \leq z + s(y) \tag{13}$$
$$s(x) + s(\bot) = s(x) \tag{14}$$
$$s(x) + s(y) = s(x \sqcup y) + s(x \sqcap y) \tag{15}$$
$$s(x^\mathsf{T}) = s(x) \tag{16}$$

An m-algebra $(S, \sqcup, \sqcap, \cdot, +, \overline{}, ^\mathsf{T}, s, m, \bot, \top, 1)$ *is an s-algebra* $(S, \sqcup, \sqcap, \cdot, +, \overline{}, ^\mathsf{T}, s, \bot, \top, 1)$ *with a minimum selection* m *satisfying the following properties:*

$$m(x) \leq \overline{\overline{x}} \tag{17}$$
$$x \neq \bot \Rightarrow m(x) \text{ is an atom} \tag{18}$$
$$y \text{ is an atom} \wedge y \sqcap x \neq \bot \Rightarrow s(m(x) \sqcap x) \leq s(y \sqcap x) \tag{19}$$

An m-Kleene-algebra *is a structure* $(S, \sqcup, \sqcap, \cdot, +, \overline{}, ^\mathsf{T}, ^*, s, m, \bot, \top, 1)$ *such that* $(S, \sqcup, \sqcap, \cdot, \overline{}, ^\mathsf{T}, ^*, \bot, \top, 1)$ *is a Stone-Kleene relation algebra and* $(S, \sqcup, \sqcap, \cdot, +, \overline{}, ^\mathsf{T}, s, m, \bot, \top, 1)$ *is an m-algebra.* ☐

The above axioms are revised from [36] to reflect new results about Stone relation algebras [37]. In particular, we have used that every atom is regular in a Stone relation algebra.

Among the new operations, only m is used in the algorithm. The axioms have the following meaning:

(13) The operation $+$ is $\leq$-isotone in its second argument on the image of the operation s for non-empty graphs (this is required because edges may have negative weights).

(14) The empty graph adds no weight; the given axiom is weaker than the conjunction of $s(\bot) = \bot$ and $x + \bot = x$.

(15) This generalises the inclusion-exclusion principle to sets of weights.

(16) Reversing edges does not change the sum of weights.

(17) The minimal edge is contained in the graph.

(18) The result of m is just one edge, if the graph is not empty.

(19) Any edge y in the graph x weighs at least as much as the edge in $m(x)$; the operation s is used to compare the weights of edges between different nodes.

The following result shows consequences of the above axioms for the operations s and m. For example, parts 2 and 3 show how weight summation distributes over disjoint parts of a graph. Part 5 shows that the result of m is $\bot$ only for the empty graph. The element $m(x) \sqcap x$ is the subgraph of x containing just the selected edge with its minimal weight; part 6 shows that an edge with minimal weight exists precisely if the graph is not empty.

Theorem 4. *Let S be an s-algebra and let $w, x, y, z \in S$. Then*

1. $s(x) + s(y) = s(y) + s(x)$.
2. $x \sqcap y = \bot \;\Rightarrow\; s((x \sqcup y) \sqcap z) = s(x \sqcap z) + s(y \sqcap z)$.
3. $w \sqcap x = \bot \wedge w \sqcap y = \bot \wedge x \sqcap y = \bot \;\Rightarrow\; s((w \sqcup x \sqcup y) \sqcap z) = s(w \sqcap z) + s(x \sqcap z) + s(y \sqcap z)$.
4. $y = y^{\mathsf{T}} \;\Rightarrow\; s(x^{\mathsf{T}} \sqcap y) = s(x \sqcap y)$.

Let S be an m-algebra and let $x \in S$. Then

5. $m(x) = \bot \;\Leftrightarrow\; x = \bot$.
6. $m(x) \sqcap x = \bot \;\Leftrightarrow\; x = \bot$.
7. $m(x) = \overline{\overline{m(x)}}$. □

We will see in Section 7 that the axioms of m-Kleene-algebras are sufficient to prove the correctness of Prim's minimum spanning tree algorithm. In the following sections we first study different instances of m-Kleene-algebras.

5. Algebras for aggregation

In [36] we presented a particular instance of m-Kleene-algebras motivated by the operations used in Prim's algorithm to find a minimum spanning tree. In particular, we assumed that edge weights are real numbers, that the operation m finds an edge with minimal weight according to the standard order of real numbers, that the operation s yields the sum of the edge weights using the standard addition of real numbers, and that the operation $+$ applies this addition componentwise to matrices. In this section and in the next section we present further instances of m-Kleene-algebras. Because the correctness proof of Prim's algorithm relies only on the general axioms of m-Kleene-algebras, the algorithm can be used to solve various problems depending on the particular instance.

We apply two main ideas. The first idea is to replace the summation of real numbers with a general aggregation operation on matrices. The second idea is to derive the aggregation operation on matrices from a binary aggregation operation. Both ideas are inspired by fold/reduce aggregations common in functional programming languages. In particular, Haskell's *foldl* and *foldr* functions can aggregate the elements of a list using a binary operation; algebraic properties of the binary operation have been used to reason about *foldl* and *foldr*, for example, in [13]. While these functions serve as a motivation, this paper does not aim to transfer the fold/reduce facilities of any particular programming language to the domain of weight matrices.

Consider the following four instances of aggregating the edge weights of a graph:

1. Compute the sum of the weights of all edges; yield $\bot$ for the empty graph and $\top$ if any edge has weight $\top$. This is the instance given in [36] for the original use of Prim's algorithm. The underlying binary aggregation is the standard addition $+$ of reals, which is extended so that $\bot$ is its unit and $\top$ is its zero.
2. Compute the minimum of the weights of all edges; yield $\top$ for the empty graph. The underlying binary aggregation is the standard minimum of two reals. After extension by $\bot$ and $\top$ neither of these is a unit though both are units if the aggregation only involves real numbers.
3. Count the number of edges; yield 0 for the empty graph. The underlying binary aggregation ignores edge weights but keeps track of edge counts. In fact, it should work for arbitrary edge labels, not just for real numbers.

4. Yield a fixed constant value for any input, ignoring all edge weights. The underlying binary aggregation is the constant function. This extreme form of aggregation should be included for theoretical purposes; for example, the K combinator for constant functions is essential in combinatory logic [23].

In the following we develop an algebraic framework that captures these and other instances. This proceeds in two major steps. First, we define various aggregation algebras, which are based on orders and lattices with a binary aggregation operation. We show that matrices over aggregation algebras form s-algebras, m-algebras and m-Kleene-algebras. Second, in the next section, we present several instances of the aggregation algebras.

Our first aim are algebraic structures that capture the various kinds of aggregation described in the above examples. Because aggregation at the matrix level does not take into account the order of edges, we assume that the binary aggregation operation is associative and commutative. To be able to compare edge weights we also require an order. Further axioms are stated in the following definition.

Definition 7. *An* aggregation order *is a structure* $(S, \leq, +, \bot)$ *such that* $\leq$ *is a partial order on* S *with least element* $\bot$ *and* $(S, +)$ *is a commutative semigroup satisfying the axioms*

$$x + y + \bot = x + y \tag{20}$$

$$x + y = \bot \Rightarrow x = \bot \tag{21}$$

$$x \neq \bot \wedge x + \bot \leq y + \bot \Rightarrow x + z \leq y + z \tag{22}$$

An aggregation lattice $(S, \leq, +, \sqcup, \sqcap, \bot, \top)$ *is a dense lattice* $(S, \sqcup, \sqcap, \bot, \top)$ *with lattice order* $\leq$ *such that* $(S, \leq, +, \bot)$ *is an aggregation order satisfying the axiom*

$$x + y = (x \sqcup y) + (x \sqcap y) \tag{23}$$

A linear aggregation lattice $(S, \leq, +, \sqcup, \sqcap, \bot, \top)$ *is a bounded lattice* $(S, \sqcup, \sqcap, \bot, \top)$ *with a linear lattice order* $\leq$ *such that* $(S, \leq, +, \bot)$ *is an aggregation order.* □

We will use aggregation lattices to obtain an s-algebra of matrices and linear aggregation lattices for an m-algebra. Aggregation orders capture basic properties, in particular, for constructing finite sums. Note that the order in an aggregation lattice can be partial; linearity is required to obtain unique minimal weights for Prim's algorithm.

Axiom (20) makes an aggregation order almost a commutative monoid. According to this axiom, $\bot$ is a unit on the image of the aggregation operation $+$, that is, on the set $\{x + y \mid x, y \in S\}$. We use this generalisation because $\bot$ is not a unit of several aggregation operations we are interested in.

Axiom (21) expresses that $+$ is zero-sum-free.

Axiom (22) states that $+$ is $\leq$-isotone on the image of $+$. The additional requirement $x \neq \bot$ is necessary because the introduction of new edges can decrease the aggregated value (for example, if the aggregation computes the sum and the new edge has a negative weight).

Note that these axioms do not imply $\bot + \bot = \bot$ as $\bot$ does not need to be in the image of $+$. We therefore cannot require that the image of $+$ is a monoid with unit $\bot$. However, it follows from the axioms that the image of $+$ is a commutative monoid with unit $\bot + \bot$. This observation is important for verification: Isabelle/HOL's theory of finite summation is based on commutative monoids. Rewriting it for commutative semigroups (without a unit) would be a significant challenge, not least because empty sums cannot be defined in this case. Having a unit on the image of summation simplifies the task and we have successfully generalised a big part of Isabelle/HOL's theory of finite summation to this case. Accordingly, a finite sum in an aggregation order is defined recursively using the binary aggregation operation $+$, where the value of the empty sum is $\bot + \bot$. This means that for finite I and $x_i \in S$ we have

$$\textstyle\sum_{i \in I} x_i = \begin{cases} \bot + \bot & \text{if } I = \emptyset \\ x_j + \sum_{i \in I \setminus \{j\}} x_i & \text{if } j \in I \end{cases}$$

With this definition of finite sums, we obtain the following results.

Lemma 1. *Let I be a finite set, let S be an aggregation order, let $k, l \in I$ and let $x_i, y_i, z_{i,j} \in S$ for each $i, j \in I$. Then*

1. $\sum_{i \in I} \bot = \bot + \bot$.
2. $\sum_{i \in I} (\bot + \bot) = \bot + \bot$.
3. $(\sum_{i \in I} x_i) + \bot = \sum_{i \in I} x_i$.
4. $\sum_{i \in I} (x_i + \bot) = \sum_{i \in I} x_i$.
5. $\sum_{i \in I} (x_i + y_i) = (\sum_{i \in I} x_i) + (\sum_{i \in I} y_i)$.
6. $\sum_{i \in I} \sum_{j \in I} z_{i,j} = \sum_{j \in I} \sum_{i \in I} z_{i,j}$.
7. $\sum_{i \in I} (\mathsf{if}\ i = k\ \mathsf{then}\ x_i\ \mathsf{else}\ \bot) = x_k + \bot$.
8. $\sum_{i \in I} (\mathsf{if}\ i = k\ \mathsf{then}\ x_i\ \mathsf{else}\ \bot + \bot) = x_k + \bot$.
9. $\sum_{i \in I} \sum_{j \in I} (\mathsf{if}\ i = k \wedge j = l\ \mathsf{then}\ z_{i,j}\ \mathsf{else}\ \bot) = z_{k,l} + \bot$.
10. $x_k \neq \bot \;\Rightarrow\; \sum_{i \in I} x_i \neq \bot$.
11. $z_{k,l} \neq \bot \;\Rightarrow\; \sum_{i \in I} \sum_{j \in I} z_{i,j} \neq \bot$. □

Aggregation on matrices over aggregation orders is expressed using finite sums as shown in the following definition. We also introduce the componentwise aggregation and the matrix of unit elements.

For the matrix aggregation s we assume that the index set A has, as additional structure, a fixed element $h \in A$. The operation s uses the matrix entry in row h and column h to store the aggregation. By 'fixed' we mean that the same h is taken for different applications of s on the same set A. The choice of h does not matter. An alternative definition of s would store the aggregation in all entries of the matrix; all subsequent results also hold for this alternative definition.

Definition 8. *Let A be a finite set with a fixed element $h \in A$. Let S be an aggregation order. The aggregation of a matrix $M \in S^{A \times A}$ is given by*

$$s(M)_{i,j} = \begin{cases} \sum_{k,l \in A} M_{k,l} & \text{if } i = j = h \\ \bot + \bot & \text{if } i \neq h \vee j \neq h \end{cases} \tag{24}$$

The componentwise sum of matrices $M, N \in S^{A \times A}$ is given by

$$(M + N)_{i,j} = M_{i,j} + N_{i,j} \tag{25}$$

The matrices $\mathsf{O}, \bot \in S^{A \times A}$ are defined by

$$\mathsf{O}_{i,j} = \bot + \bot \tag{26}$$

$$\bot_{i,j} = \bot \tag{27}$$

□

We overload the notation $+$ for elements of S and matrices over S. We also overload the notation $\bot$ for the element of S and the constant matrix whose entries are this element. These matrix operations satisfy the following properties.

Lemma 2. *Let A be a non-empty finite set and let S be an aggregation order. Let $M \in S^{A \times A}$ and consider $\mathsf{O}, \bot \in S^{A \times A}$. Then*

1. $s(\bot) = \mathsf{O}$.
2. $\bot + \bot = \mathsf{O}$.
3. $s(M) + \bot = s(M)$.
4. $s(M) + \mathsf{O} = s(M)$. □

We next look at aggregation lattices. The following result shows that they can be expanded to Stone algebras, whence we can apply Theorem 2 to obtain Stone relation algebras. Moreover, we show that matrices over these algebras form s-algebras using the aggregation s and the componentwise sum $+$ discussed above.

Theorem 5. *Let $(S, \leq, +, \sqcup, \sqcap, \bot, \top)$ be an aggregation lattice and let A be a non-empty finite set.*

1. *$(S, \sqcup, \sqcap, ^-, \bot, \top)$ is a Stone algebra using the pseudocomplement operation*
$$\overline{x} = \begin{cases} \top & \textit{if } x = \bot \\ \bot & \textit{if } x \neq \bot \end{cases}$$
2. *$(S, \sqcup, \sqcap, \sqcap, ^-, \lambda x.x, \bot, \top, \top)$ is a Stone relation algebra using this pseudocomplement operation by Theorem 2.2.*
3. *$(S^{A \times A}, \sqcup, \sqcap, \cdot, ^-, ^\mathsf{T}, \bot, \top, 1)$ is a Stone relation algebra as per Theorem 2.3.*
4. *$(S^{A \times A}, \sqcup, \sqcap, \cdot, +, ^-, ^\mathsf{T}, s, \bot, \top, 1)$ is an s-algebra using the operations $+$ and s given in Definition 8.* □

To obtain m-algebras we have to add the operation m that finds an edge with minimum weight according to the aggregation order $\leq$. For this we assume that $\leq$ is linear; there may be several edges having the same minimum weight but the minimum weight is unique. This is required for Prim's algorithm; minimum spanning trees based on partial orders are discussed in [15]. We therefore work in linear aggregation lattices.

We assume that the index set A has, as additional structure, a fixed strict total order $\prec$ to uniquely determine the result of the operation m. Remarks similar to those about the fixed element h for the operation s apply here, too.

Definition 9. *Let A be a non-empty finite set with a fixed strict total order $\prec$ on A. Let S be a linear aggregation lattice. The minimum non-$\bot$ entry of a matrix $M \in S^{A \times A}$ is given by*

$$m(M)_{i,j} = \begin{cases} \top & \textit{if } M_{i,j} \neq \bot \wedge \forall k, l \in A : M_{k,l} \neq \bot \Rightarrow \\ & \qquad (M_{i,j} + \bot \leq M_{k,l} + \bot \wedge \\ & \qquad\ ((k \prec i \vee (k = i \wedge l \prec j)) \Rightarrow M_{i,j} + \bot \neq M_{k,l} + \bot)) \\ \bot & \textit{otherwise} \end{cases} \tag{28}$$

□

More precisely, the operation m finds an edge with a minimum weight, comparing weights on the image of the operation $+$ (which is achieved by aggregating with $\bot$). This is done because axiom (19) expresses minimality using the operation s which works on the image of $+$ as described above. If there are several edges with the same minimum weight, the edge with the lexicographically smallest index based on $\prec$ is chosen. This makes the operation m deterministic as shown by the following result.

Lemma 3. *Let A be a non-empty finite set and let S be a linear aggregation lattice. Let $M \in S^{A \times A}$. Then*

1. *$\forall i, j, k, l \in A : (m(M)_{i,j} \neq \bot \wedge m(M)_{k,l} \neq \bot) \Rightarrow (i = k \wedge j = l)$.*
2. *$M \neq \bot \Rightarrow \exists i, j \in A : m(M)_{i,j} = \top$.* □

Part 1 of this result already holds if S is an aggregation order with $\top$; this part does not require a linear aggregation lattice.

Despite their different axiomatisations, linear aggregation lattices are aggregation lattices, whence we obtain an s-algebra structure by Theorem 5. The following result additionally shows that the m operation gives an m-algebra structure. Finally, to obtain the m-Kleene-algebra structure we proceed according to Theorem 3 since aggregation lattices form a Stone algebra by Theorem 5.

Theorem 6. *Let $(S, \leq, +, \sqcup, \sqcap, \bot, \top)$ be a linear aggregation lattice and let A be a non-empty finite set. Then*

1. *S is an aggregation lattice.*
2. *$(S^{A \times A}, \sqcup, \sqcap, \cdot, +, ^-, ^\mathsf{T}, s, \bot, \top, 1)$ is an s-algebra as per Theorem 5.*
3. *$(S^{A \times A}, \sqcup, \sqcap, \cdot, +, ^-, ^\mathsf{T}, s, m, \bot, \top, 1)$ is an m-algebra using the operation m given in Definition 9.*
4. *$(S^{A \times A}, \sqcup, \sqcap, \cdot, +, ^-, ^\mathsf{T}, ^*, s, m, \bot, \top, 1)$ is an m-Kleene-algebra using the operation * given in Theorem 3.* □

Extension by the Kleene star works already for s-algebras and does not require a linear aggregation lattice. It is therefore possible to use aggregations in applications with a partial order.

This concludes the first part of the algebraic framework. We have shown that matrices over suitably expanded linear aggregation lattices form m-Kleene-algebras. Together with the correctness proof in Section 7, this implies that Prim's algorithm is correct not only for minimising the sum of real-valued edge weights, but for minimising the aggregation of edge weights using any binary operation $+$ that forms a linear aggregation lattice.

6. Instances of algebras for aggregation

Our second aim is to give several classes of instances that satisfy the axioms of a linear aggregation lattice. Prim's algorithm works correctly for each of these instances, and we discuss what it computes in a number of cases. Particular instances solve the minimum weight spanning tree problem, the minimum bottleneck spanning tree problem as well as spanning tree problems that minimise aggregations based on t-norms and t-conorms.

We start with the original application of Prim's algorithm, where aggregation is the sum of edge weights. It is covered by the following general result. Entries t and f in the tables in this section abbreviate true and false, respectively.

Theorem 7. *Let $(S, \leq, +)$ be a linearly ordered commutative semigroup. Let $S' = S \cup \{\bot, \top\}$ for $\bot, \top \notin S$. Then $(S', \leq, +, \sqcup, \sqcap, \bot, \top)$ is a linear aggregation lattice with the following operations, where $x, y \in S$:*

$\leq$	$\bot$	y	$\top$
$\bot$	t	t	t
x	f	$x \leq y$	t
$\top$	f	f	t

$+$	$\bot$	y	$\top$
$\bot$	$\bot$	y	$\top$
x	x	$x+y$	$\top$
$\top$	$\top$	$\top$	$\top$

$\sqcup$	$\bot$	y	$\top$
$\bot$	$\bot$	y	$\top$
x	x	$x \uparrow y$	$\top$
$\top$	$\top$	$\top$	$\top$

$\sqcap$	$\bot$	y	$\top$
$\bot$	$\bot$	$\bot$	$\bot$
x	$\bot$	$x \downarrow y$	x
$\top$	$\bot$	y	$\top$

□

Observe that $\bot$ is a unit of $+$ in this instance. Real numbers with their standard order and addition form a linearly ordered commutative semigroup. This gives the original algorithm of Prim.

The following generalisations are facilitated by Theorem 7. First, it shows that the real numbers can be replaced with any linear order.

Second, by Theorem 7, the addition can be replaced with any $\leq$-isotone associative commutative operation. This affects mainly the s operation on matrices. Note that Prim's algorithm only uses the m operation, while the s operation appears only in the specification of the algorithm, which gives the optimisation function. It follows that the spanning tree computed by Prim's algorithm minimises any optimisation function that aggregates the weights of a tree's edges using a $\leq$-isotone associative commutative operation.

Let us consider two particular $\leq$-isotone associative commutative operations: maximum and minimum. Using the binary maximum as $+$, the goal of the minimum spanning tree problem is to find a spanning tree whose maximum edge weight is minimal among all spanning trees. This is the minimum bottleneck spanning tree problem. Hence a corollary of the algebraic generalisation is the known result that every minimum spanning tree found by Prim's algorithm is a minimum bottleneck spanning tree. Conversely, a counterexample shows that minimum bottleneck spanning trees need not be minimum spanning trees. An alternative algorithm to find minimum bottleneck spanning trees is presented in [18].

Using the binary minimum as $+$, the goal is to find a spanning tree whose minimum edge weight is minimal among all spanning trees. Because every edge is contained in some spanning tree, another corollary is that every minimum spanning tree found by Prim's algorithm contains an edge with minimum weight in the input graph. Note that this follows immediately for Kruskal's algorithm; we leave it to the reader to find an elementary argument for why this holds for Prim's algorithm.

In Theorem 7 aggregation on S does not need to have a unit; the element $\bot$ is added to become the unit in S'. For the next class of instances we assume that aggregation $+$ in S has a unit 0. This is the case, for example, for real numbers under standard addition.

Theorem 8. *Let $(S, \le, +, 0)$ be a linearly ordered commutative monoid. Let $S' = S \cup \{\bot, \top\}$ for $\bot, \top \notin S$. Then $(S', \le, +, \sqcup, \sqcap, \bot, \top)$ is a linear aggregation lattice with the following operations, where $x, y \in S$:*

$\le$	$\bot$	y	$\top$
$\bot$	t	t	t
x	f	$x\le y$	t
$\top$	f	f	t

$+$	$\bot$	y	$\top$
$\bot$	0	y	$\top$
x	x	$x+y$	$\top$
$\top$	$\top$	$\top$	$\top$

$\sqcup$	$\bot$	y	$\top$
$\bot$	$\bot$	y	$\top$
x	x	$x\uparrow y$	$\top$
$\top$	$\top$	$\top$	$\top$

$\sqcap$	$\bot$	y	$\top$
$\bot$	$\bot$	$\bot$	$\bot$
x	$\bot$	$x\downarrow y$	x
$\top$	$\bot$	y	$\top$

□

The only difference for these instances is that $\bot + \bot = 0$ in contrast to Theorem 7 where $\bot$ was a unit. This means that the aggregation of an empty graph gives 0 instead of $\bot$, which might be more natural for some applications. There is no effect on Prim's algorithm as all spanning trees of non-empty graphs are non-empty. Nevertheless, this example shows that $\bot$ need not be a unit of $+$ in the presented algebraic framework.

The next class of instances shows that there is no need to add $\bot$ if S already contains a least element $\bot$ that is a unit of $+$.

Theorem 9. *Let $(S, \le, +, \bot)$ be a linearly ordered commutative monoid such that $\bot$ is the $\le$-least element. Let $S' = S \cup \{\top\}$ for $\top \notin S$. Then $(S', \le, +, \sqcup, \sqcap, \bot, \top)$ is a linear aggregation lattice with the following operations, where $x, y \in S$:*

$\le$	y	$\top$
x	$x\le y$	t
$\top$	f	t

$+$	y	$\top$
x	$x+y$	$\top$
$\top$	$\top$	$\top$

$\sqcup$	y	$\top$
x	$x\uparrow y$	$\top$
$\top$	$\top$	$\top$

$\sqcap$	y	$\top$
x	$x\downarrow y$	x
$\top$	y	$\top$

□

These operations are the restrictions of the operations in Theorems 7 and 8 obtained by removing the $\bot$ rows and columns. An instance in this class is given by real numbers extended with $-\infty$ taking maximum as aggregation.

Dually, if S already contains a greatest element $\top$ that is a unit of $+$, there is no need to add $\top$ as shown by the following result. In particular, this includes all triangular norms (t-norms), which are binary operations on the real interval $[0, 1]$ that are associative, commutative and $\le$-isotone with 1 as unit. An example is the Łukasiewicz t-norm mapping $x, y \in [0, 1]$ to $(x + y - 1)\uparrow 0$. See [34, 7, 31] for numerous further examples of t-norms used in fuzzy set theory and other applications.

Theorem 10. *Let $(S, \le, +, \top)$ be a linearly ordered commutative monoid such that $\top$ is the $\le$-greatest element. Let $S' = S \cup \{\bot\}$ for $\bot \notin S$. Then $(S', \le, +, \sqcup, \sqcap, \bot, \top)$ is a linear aggregation lattice with the following operations, where $x, y \in S$:*

$\le$	$\bot$	y
$\bot$	t	t
x	f	$x\le y$

$+$	$\bot$	y
$\bot$	$\top$	y
x	x	$x+y$

$\sqcup$	$\bot$	y
$\bot$	$\bot$	y
x	x	$x\uparrow y$

$\sqcap$	$\bot$	y
$\bot$	$\bot$	$\bot$
x	$\bot$	$x\downarrow y$

□

With one exception, these operations are the restrictions of the operations in Theorems 7 and 8 obtained by removing the $\top$ rows and columns. The exception is that $\bot + \bot = \top$, which is the unit on the image of $+$. Another instance in this class is given by real numbers extended with ∞ taking minimum as aggregation.

For the next class of instances we assume that S has both a least element $\bot$ that is a unit of $+$ and a greatest element $\top$. In particular, this includes all triangular conorms (t-conorms), which are binary operations on the real interval $[0, 1]$ that are associative, commutative and $\le$-isotone with 0 as unit. An example is the product t-conorm mapping $x, y \in [0, 1]$ to $x+y-xy$. For many further examples of t-conorms we again refer to [34, 7, 31].

Theorem 11. *Let $(S, \leq, +, \bot, \top)$ be a bounded linearly ordered commutative monoid. Then $(S, \leq, +, \uparrow, \downarrow, \bot, \top)$ is a linear aggregation lattice.* □

The next class of instances are the constant aggregations. No additional structure is required on S in this case; any linear order can be chosen. Once again, the operation $+$ on S' has no unit, but c is a unit on the image of $+$.

Theorem 12. *Let $(S, \leq)$ be a linear order and let $c \in S$. Let $S' = S \cup \{\bot, \top\}$ for $\bot, \top \notin S$. Then $(S', \leq, +, \sqcup, \sqcap, \bot, \top)$ is a linear aggregation lattice with the following operations, where $x, y \in S$:*

$\leq$	$\bot$	y	$\top$
$\bot$	t	t	t
x	f	$x{\leq}y$	t
$\top$	f	f	t

$+$	$\bot$	y	$\top$
$\bot$	c	c	c
x	c	c	c
$\top$	c	c	c

$\sqcup$	$\bot$	y	$\top$
$\bot$	$\bot$	y	$\top$
x	x	$x{\uparrow}y$	$\top$
$\top$	$\top$	$\top$	$\top$

$\sqcap$	$\bot$	y	$\top$
$\bot$	$\bot$	$\bot$	$\bot$
x	$\bot$	$x{\downarrow}y$	x
$\top$	$\bot$	y	$\top$

□

The final class of instances we discuss is for counting the number of edges. Edge weights are ignored for counting and therefore we should not assume that the underlying algebraic structure S are real numbers, or indeed, any kind of numbers. Because of this, we add a copy of the natural numbers for representing the edge count.

Theorem 13. *Let $(S, \leq)$ be a linearly ordered set. Let $S' = S \cup \{\bot, \top\} \cup \mathbb{N}$ for $\bot, \top \notin S$ and $\mathbb{N} \cap S = \emptyset$. Then $(S', \leq, +, \sqcup, \sqcap, \bot, \top)$ is a linear aggregation lattice with the following operations, where $x, y \in S$ and $m, n \in \mathbb{N}$:*

$\leq$	$\bot$	y	n	$\top$
$\bot$	t	t	t	t
x	f	$x{\leq}y$	t	t
m	f	f	$m{\leq}n$	t
$\top$	f	f	f	t

$+$	$\bot$	y	n	$\top$
$\bot$	0	1	n	1
x	1	2	$n{+}1$	2
m	m	$m{+}1$	$m{+}n$	$m{+}1$
$\top$	1	2	$n{+}1$	2

$\sqcup$	$\bot$	y	n	$\top$
$\bot$	$\bot$	y	n	$\top$
x	x	$x{\uparrow}y$	n	$\top$
m	m	m	$m{\uparrow}n$	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

$\sqcap$	$\bot$	y	n	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
x	$\bot$	$x{\downarrow}y$	x	x
m	$\bot$	y	$m{\downarrow}n$	m
$\top$	$\bot$	y	n	$\top$

□

A matrix entry $\bot$ represents the absence of an edge, and therefore contributes 0 to the count. A matrix entry from S represents an edge and contributes 1. A matrix entry $\top$ specifies an edge without weight information and therefore contributes 1, too. Matrix entries from $\mathbb{N}$ are used for counting and will not appear in the input. The aggregation therefore computes the number of edges of a graph. In this case, the correctness of Prim's algorithm simply restates that the resulting spanning tree will have a minimal number of edges, which is clear since all spanning trees have the same number of edges.

7. Correctness of Prim's Minimum Spanning Tree Algorithm

In this section we present an algebraic version of Prim's minimum spanning tree algorithm and prove its correctness. In particular, we show how the algebras introduced in the previous sections are used to reason about graph properties. The algorithm is shown in Figure 2. It is a while-program with variables whose values range over an m-Kleene-algebra S.

The input of the algorithm is a weighted graph $g \in S$ and a root node $r \in S$. The algorithm constructs a minimum spanning tree $t \in S$ and maintains a set of visited nodes v. Both r and v are represented as vectors. The algorithm starts with an empty tree t and the single visited node r (lines 2 and 3). The expression $v\overline{v}^{\mathsf{T}} \sqcap g$ restricts g to the edges starting in v and ending outside of v (lines 4 and 5). The while-loop is executed as long as this set of edges is not empty (condition in line 4). In each iteration an edge e is chosen with minimal weight among these edges (line 5). Then e is added to the tree t and the end node of e is added to v (lines 6 and 7). The end node is obtained by the relational expression $e^{\mathsf{T}}\top$, which represents the

```
1 input $g, r$
2 $t \leftarrow \bot$
3 $v \leftarrow r$
4 while $v\overline{v}^\mathsf{T} \sqcap g \neq \bot$ do
5     $e \leftarrow m(v\overline{v}^\mathsf{T} \sqcap g)$
6     $t \leftarrow t \sqcup e$
7     $v \leftarrow v \sqcup e^\mathsf{T}\top$
8 end
9 output $t$
```

Figure 2: A relational minimum spanning tree algorithm

start node of the reversed edge e^T as a vector (line 7). When there are no edges from v to its complement set, the while-loop finishes and the output of the algorithm is t.

Before we formally specify minimum spanning trees, we discuss components of graphs. Since we will not assume that the graph g is connected, the above algorithm produces a minimum spanning tree of the component of g that contains r. In m-Kleene-algebras, the nodes in this component are given by

$$c(g, r) = r^\mathsf{T}\overline{\overline{g}}^*$$

which is the converse of a vector that represents the set of nodes reachable from r in the graph g ignoring edge weights. It follows that for connected g the result is a minimum spanning tree of the whole graph.

The following definition captures the specification of the minimum spanning tree problem.

Definition 10. *Let S be an m-Kleene-algebra and let $g, r, t \in S$. Then t is a* spanning tree *of g with root r if t is a forest, t is regular and*

$$t \leq c(g, r)^\mathsf{T} c(g, r) \sqcap \overline{\overline{g}} \tag{29}$$

$$c(g, r) \leq r^\mathsf{T} t^* \tag{30}$$

Such a t is a minimum spanning tree *of g with root r if, additionally,*

$$s(t \sqcap g) \leq s(u \sqcap g)$$

for each spanning tree u of g with root r. □

By lattice properties and since $c(g, r)$ is the converse of a vector, inequality (29) is equivalent to the conjunction of $t \leq \overline{\overline{g}}$ and $t \leq c(g, r)^\mathsf{T}$ and $t \leq c(g, r)$. The first of these inequalities states that all edges of t are contained in g (ignoring the weights). The second inequality states that each edge of t starts in a node in the component of g that contains r. The third inequality expresses the same for the end nodes of the edges of t.

Also inequality (30) is concerned with the component of g that contains r. It states that all nodes in this component are reachable from r using edges in t. Observe that $r^\mathsf{T} t^* = c(t, r)$ since t is regular, so together with $t \leq \overline{\overline{g}}$ we obtain $c(g, r) = c(t, r)$ as a consequence.

Correctness of Prim's algorithm is established by the following result, which gives the precondition, loop invariant and postcondition used in the Hoare-logic proof.

Theorem 14. *Let S be an m-Kleene-algebra and let $g, r \in S$ such that the following precondition holds:*

- *g is symmetric*
- *r is regular, injective and a vector*
- *there is a minimum spanning tree of g with root r*

Then the following invariant holds in each iteration of the while-loop in Figure 2:

- *the precondition holds*
- $v^\mathsf{T} = r^\mathsf{T} t^*$
- *t is a spanning tree of $vv^\mathsf{T} \sqcap g$ with root r*
- *$t \leq w$ for some minimum spanning tree w of g with root r*

Finally, the following postcondition is established:

- *t is a minimum spanning tree of g with root r* □

Symmetry of g specifies that the graph is undirected. The properties of r in the precondition state that r represents a single node. We discuss the remaining part of the precondition below.

The invariant maintains that v is the set of nodes reachable from r in the spanning tree t, and that t can be extended to a minimum spanning tree w.

The verification conditions to establish the postcondition are automatically generated from the precondition and the loop invariant using Hoare logic. We use an implementation of Hoare logic that comes with Isabelle/HOL; see [51, 52]. The generated conditions are predicates whose variables range over an m-Kleene-algebra; all calculations take place in this algebra or its reducts. The high-level structure of the proof is standard; the difference here is that the whole argument is carried out in new algebraic structures that directly model weighted graphs.

The proof shows partial correctness only. Termination of the while-loop follows since a new edge is added to the spanning tree in each iteration and the graph is finite. Such termination proofs can also be done algebraically [35]; this is not part of the present paper but will be addressed in future work. In particular, the Hoare-logic library used in this paper only supports partial correctness. Other Isabelle/HOL libraries such as those described in [60, 53] support total correctness, but they have not been used in an algebraic setting so far. The integration of algebraic methods for proving termination with these libraries is orthogonal to the issues addressed by the present paper.

A second assumption requires the existence of a minimum spanning tree in the precondition. This follows since there is a spanning tree and the number of spanning trees of a finite graph is finite. A proof of this is not part of the present paper, but could be based on cardinalities of relations [42].

Note that termination of the algorithm does not directly imply the existence of a minimum spanning tree. This is because correctness of the algorithm relies on this assumption in the first place. Specifically, a total-correctness proof of the algorithm shows the existence of a minimum spanning tree, but the partial-correctness part of the proof uses the existence to establish the invariant, so the obtained result is vacuous in this respect. This is not specific to our proof, but a standard way of proving the correctness of minimum spanning tree algorithms [22]. Termination of the algorithm can be used, however, as part of a different approach to eliminate the existence assumption; this is orthogonal to the algebraic framework presented in this paper and will be addressed in future work.

In the following we discuss several parts of the proof, which are carried out in different algebraic structures. Our aim is not completeness, but to show that many results used in the proof actually hold in more general settings. We focus on the preservation of the loop invariant for the current tree t and the current set of visited nodes v. Let $t' = t \sqcup e$ and $v' = v \sqcup e^\mathsf{T}\top$ be the values of these variables at the end of the body of the while-loop (after the assignment in line 7 of Figure 2).

First, the proof involves showing that t' is a spanning tree of $v'v'^\mathsf{T} \sqcap g$ with root r, that is, of the subgraph of g restricted to nodes in v'. In particular, this requires that t' is injective. To this end, we use the following property given in [55] that also holds in Stone relation algebras.

Lemma 4. *Let S be a Stone relation algebra. Let $t, e \in S$ such that t and e are injective and $et^\mathsf{T} \leq 1$. Then $t \sqcup e$ is injective.* □

The assumptions of Lemma 4 are established as follows:

- Injectivity of t follows from the invariant.
- e is an atom by axiom (18), so $e\top$ is injective, whence e is injective.
- $et^\mathsf{T} = \bot \le 1$ follows by another general result of Stone relation algebras from $e \le v\overline{v}^\mathsf{T}$ and $t \le vv^\mathsf{T}$ and that v is a vector.

We also require that t' is contained in the subgraph of g restricted to the nodes in v'. For this we use the following result of Stone relation algebras.

Lemma 5. *Let S be a Stone relation algebra. Let $t, e, v, g \in S$ such that $t \le vv^\mathsf{T} \sqcap g$ and $e \le v\overline{v}^\mathsf{T} \sqcap g$. Then $t' \le v'v'^\mathsf{T} \sqcap g$ where $t' = t \sqcup e$ and $v' = v \sqcup e^\mathsf{T}\top$.* □

Next, we also require that t' is acyclic. To show this, we use the following result of Stone-Kleene relation algebras.

Lemma 6. *Let S be a Stone-Kleene relation algebra. Let $t, e, v \in S$ such that t is acyclic, v is a vector and $e \le v\overline{v}^\mathsf{T}$ and $t \le vv^\mathsf{T}$. Then $t \sqcup e$ is acyclic.* □

Note that this lemma does not require that t is a tree or that e contains just one edge. It is a much more general statement that can be used for reasoning about graphs in other contexts than the minimum spanning tree algorithm – in fact, it holds not only for weighted graphs but for any other instance of Stone-Kleene relation algebras. The same observation applies to the previous lemmas and many others used in the correctness proof.

Next, the invariant maintains that v is the set of nodes reachable from r in t, which is formulated as $v^\mathsf{T} = r^\mathsf{T}t^*$. To preserve this property, we use the following result of Stone-Kleene relation algebras.

Lemma 7. *Let S be a Stone-Kleene relation algebra. Let $t, e, r, v \in S$ such that v is a vector, $e \le v\overline{v}^\mathsf{T}$ and $et = \bot$ and $v^\mathsf{T} = r^\mathsf{T}t^*$. Then $v'^\mathsf{T} = r^\mathsf{T}t'^*$ where $t' = t \sqcup e$ and $v' = v \sqcup e^\mathsf{T}\top$.* □

The assumption $et = \bot$ follows similarly to $et^\mathsf{T} = \bot$ for Lemma 4.

Finally, we discuss how to preserve the property that the currently constructed spanning tree t can be extended to a minimum spanning tree. The situation is shown in Figure 3. Assuming that there is a minimum spanning tree w of g such that $t \le w$, we have to show that there is a minimum spanning tree w' of g such that $t' = t \sqcup e \le w'$ where $e = m(v\overline{v}^\mathsf{T} \sqcap g)$ is an edge of g with minimal weight going from a node in v to a node not in v. We do this by explicitly constructing the new minimum spanning tree w'. To this end, we need to find the edge f in w that crosses the cut from v to $\overline{v}$, and replace it with the edge e – this does not increase the weight due to minimality of e. An algebraic expression for the edge f is

$$f = w \sqcap v\overline{v}^\mathsf{T} \sqcap \top e w^{\mathsf{T}*}$$

The three terms on the right hand side enforce that f is in w, that f starts in v and ends in $\overline{v}$, and that there is a path in w from the end node of f to the end node of e. The last condition selects the edge across the cut that would lie on a cycle if e was added. It can be shown algebraically that f is an atom, that is, that f represents the unique edge satisfying these conditions. An algebraic expression for the path p from the end of f to the end of e is

$$p = w \sqcap \overline{v}\,\overline{v}^\mathsf{T} \sqcap \top e w^{\mathsf{T}*}$$

The three terms on the right hand side enforce that the edges in p are in w, that they start and end in $\overline{v}$, and that there is a path in w from each of their end nodes to the end node of e. The required tree w' is then obtained by removing the edge f from w, turning around the path p, and inserting the edge e. An algebraic expression for w' is

$$w' = (w \sqcap \overline{f \sqcup p}) \sqcup p^\mathsf{T} \sqcup e$$

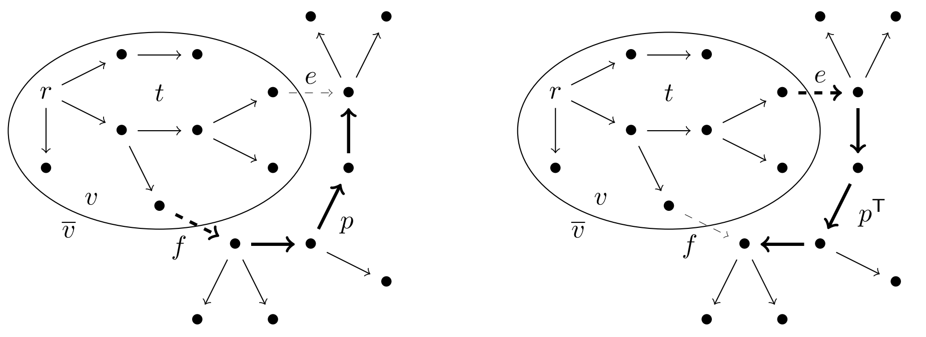

Figure 3: Replacing the edge f in w (left) with the minimal edge e in w' (right) where t is the tree in the oval and v is the set of nodes in t

We then show that w' so defined is a minimum spanning tree of g with root r and that $t \sqcup e \leq w'$. In the following we focus on the part of this proof that shows $s(w' \sqcap g) \leq s(u \sqcap g)$ for each spanning tree u of g with root r. This follows by the calculation

$$\begin{aligned}
s(w' \sqcap g) &= s(w \sqcap \overline{f \sqcup p} \sqcap g) + s(p^\mathsf{T} \sqcap g) + s(e \sqcap g) && (31)\\
&\leq s(w \sqcap \overline{f \sqcup p} \sqcap g) + s(p \sqcap g) + s(f \sqcap g) && (32)\\
&= s(((w \sqcap \overline{f \sqcup p}) \sqcup p \sqcup f) \sqcap g) && (33)\\
&= s(w \sqcap g) && (34)\\
&\leq s(u \sqcap g) && (35)
\end{aligned}$$

We briefly explain the steps in this calculation. Step (31) holds by Theorem 4.3 since $w \sqcap \overline{f \sqcup p}$ and p^T and e are pairwise disjoint (that is, their pairwise meet given by $\sqcap$ is $\bot$). A similar argument justifies step (33). Axiom (19) is used to show $s(e \sqcap g) \leq s(f \sqcap g)$ in step (32). Theorem 4.4 is used to show that replacing p with p^T does not change the weight there. Step (34) follows by a simple calculation, most of which takes place in Stone algebras. Finally, step (35) holds since w is a minimum spanning tree of g with root r.

This is the main part of the overall proof where the operations and axioms of m-Kleene-algebras are used. Most of the proof, however, can already be carried out in Stone-Kleene relation algebras or weaker structures as discussed above. We expect such results to be useful for reasoning about other graph algorithms.

As discussed in Sections 5 and 6, Theorem 14 and the other results in this section hold for numerous instances of m-Kleene-algebras based on different aggregation operations. Due to the abstraction provided by m-Kleene-algebras, the overall proof of correctness is not affected.

8. Related Work

In this section we compare the present paper with related work on algorithms for minimum spanning trees. Often the correctness of such algorithms is argued informally with varying amounts of mathematical rigour and details; for example, see [22, 46]. Our results are fully verified in Isabelle/HOL [54] based on formal definitions and models.

A formal derivation of Prim's minimum spanning tree algorithm in the B event-based framework using Atelier B is presented in [1]. The paper also discusses the role of refinement in this process, which is not part of the present paper. The B specification is based on sets and relations, uses an inductive definition of trees, and represents weights by functions, whence objects of several different sorts are involved.

Our formalisation is based on Stone-Kleene relation algebras, which generalise relation algebras and Kleene algebras, and can be instantiated directly by weight matrices. The generalisation is crucial as weight matrices do not support a Boolean complement; accordingly we do not use implementations of relation algebras such as [38, 3]. Nevertheless we can build on well-developed relational concepts and methods for our new algebras – such as algebraic properties of trees – which are useful also in other contexts.

We mostly apply equational reasoning based on a single-sorted algebra. This is well supported by automated theorem provers and SMT solvers such as those integrated in Isabelle/HOL via the Sledgehammer tool [58, 16] that we heavily use in the verification. Typically the tool can automatically find proofs of steps at a granularity comparable to manual equational reasoning found in papers. Automation works less well in some cases, for example, chains of inequalities, applications of isotone operations, and steps that introduce intermediate terms that occur on neither side of an equation. By contrast, in some cases the tool can automatically prove a result that would take several manual steps.

A distributed algorithm for computing minimum spanning trees is verified using the theorem prover Nqthm in [39]. The specification is again based on sets and a weight function. The main focus of the paper is on the distributed aspects of the algorithm, which uses asynchronous messages and differs essentially from Prim's minimum spanning tree algorithm. The distributed algorithm is the topic of a number of other papers using a variety of formalisms including Petri nets and modal logic.

Relation algebras are used to derive spanning tree algorithms in [11]. The given proof is created manually and not verified using a theorem prover. It uses relations and, in absence of weight matrices, an incidence matrix representation and a weight function in a setting with several different sorts. The paper investigates a Galois connection between adjacency relations and incidence relations for the graph structure (without weights). It does not study the algebraic structure of weights and does not attempt to generalise it to derive algorithms that solve problems beyond minimising the sum of edge weights.

Constraint-based semirings are used to formulate minimum spanning tree algorithms in [15]. These semirings abstract from the edge weights and represent graphs by sets of edges. The semiring structure is not lifted to the graph level, whereas we lift Stone algebras to Stone relation algebras – and similarly for Kleene algebras – and can therefore exploit the algebraic structure of graphs. Detailed proofs are not presented and there is no formal verification of results. The paper is mainly concerned with extending the algorithms to partially ordered edge weights, which is not part of the present paper.

Semirings with a pre-order, so-called dioids, are used to formulate various shortest-path problems in [30]. The corresponding algorithms are generalisations of methods for solving linear equations over these structures. Other approaches to path problems are based on Kleene algebras; for example, see [40], which also discusses many previous works in this tradition such as [2, 6]. Semirings and Kleene algebras are suitable for path problems as they capture the essential operations of choosing between alternatives, composing edges and building paths. It is not clear how to model the minimum spanning tree problem using Kleene algebras only.

Relational methods based on allegories are used for algorithm development in [12], but there relations mostly represent computations rather than the involved data. An extension to quantitative analysis is discussed in [56].

9. Conclusion

The generalisation of Boolean algebras to Stone algebras gives a promising way to extend correctness reasoning from unweighted to weighted graphs. When applied to relation algebras, many results continue to hold with no changes or small changes. In combination with Kleene algebras, we could carry out most of the correctness proof of Prim's minimum spanning tree algorithm.

A small part of the proof needed additional operations for summing edge weights and finding an edge with minimal weight; we captured a few key properties in m-Kleene-algebras. We derived these operations on matrices from a basic binary aggregation operation with simple properties satisfied by numerous instances. They show that Prim's algorithm solves many different optimisation problems. The next step is to apply the abstractions to variants of the minimum spanning tree algorithm and other graph algorithms. We will also consider the integration of termination proofs, complexity reasoning and combinatorial arguments using cardinalities of relations.

Using algebras for proving the correctness of programs is well supported by Isabelle/HOL. We have benefited from the existing verification condition generator for Hoare logic, from the structuring mechanisms that allow the development of hierarchies of algebras and their models, and heavily from the integrated automated theorem provers, SMT solvers and counterexample generators.

Acknowledgements

I thank the anonymous referees for their helpful feedback. Theorem 4.1 has been suggested by one of the anonymous referees.

References

[1] J.-R. Abrial, D. Cansell, and D. Méry. Formal derivation of spanning trees algorithms. In D. Bert, J. P. Bowen, S. King, and M. Waldén, editors, *ZB 2003*, volume 2651 of *LNCS*, pages 457–476. Springer, 2003.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley Publishing Company, 1974.

[3] A. Armstrong, S. Foster, G. Struth, and T. Weber. Relation algebra. *Archive of Formal Proofs*, 2016, first version 2014.

[4] A. Armstrong, V. B. F. Gomes, G. Struth, and T. Weber. Kleene algebra. *Archive of Formal Proofs*, 2016, first version 2013.

[5] T. Asplund. Formalizing the Kleene star for square matrices. Bachelor Thesis IT 14 002, Uppsala Universitet, Department of Information Technology, 2014.

[6] R. C. Backhouse and B. A. Carré. Regular algebra applied to path-finding problems. *Journal of the Institute of Mathematics and its Applications*, 15(2):161–186, 1975.

[7] G. Beliakov, A. Pradera, and T. Calvo. *Aggregation Functions: A Guide for Practitioners.* Springer, 2007.

[8] R. Berghammer and S. Fischer. 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(3):341–358, 2015.

[9] R. Berghammer, A. Rusinowska, and H. de Swart. Computing tournament solutions using relation algebra and RelView. *European Journal of Operational Research*, 226(3):636–645, 2013.

[10] R. Berghammer and B. von Karger. Relational semantics of functional programs. In C. Brink, W. Kahl, and G. Schmidt, editors, *Relational Methods in Computer Science*, chapter 8, pages 115–130. Springer, Wien, 1997.

[11] R. Berghammer, B. von Karger, and A. Wolf. Relation-algebraic derivation of spanning tree algorithms. In J. Jeuring, editor, *MPC 1998*, volume 1422 of *LNCS*, pages 23–43. Springer, 1998.

[12] R. Bird and O. de Moor. *Algebra of Programming.* Prentice Hall, 1997.

[13] R. Bird and P. Wadler. *Introduction to Functional Programming.* Prentice Hall, 1988.

[14] G. Birkhoff. *Lattice Theory*, volume XXV of *Colloquium Publications.* American Mathematical Society, third edition, 1967.

[15] S. Bistarelli and F. Santini. C-semiring frameworks for minimum spanning tree problems. In A. Corradini and U. Montanari, editors, *WADT 2008*, volume 5486 of *LNCS*, pages 56–70. Springer, 2009.

[16] J. C. Blanchette, S. Böhme, and L. C. Paulson. Extending Sledgehammer with SMT solvers. In N. Bjørner and V. Sofronie-Stokkermans, editors, *CADE 2011*, volume 6803 of *LNCS*, pages 116–130. Springer, 2011.

[17] T. S. Blyth. *Lattices and Ordered Algebraic Structures.* Springer, 2005.

[18] P. M. Camerini. The min-max spanning tree problem and some extensions. *Information Processing Letters*, 7(1):10–14, 1978.

[19] A. H. Clifford. Totally ordered commutative semigroups. *Bulletin of the American Mathematical Society*, 64(6):305–316, 1958.

[20] S. D. Comer. On connections between information systems, rough sets and algebraic logic. In C. Rauszer, editor, *Algebraic Methods in Logic and in Computer Science*, volume 28 of *Banach Center Publications*, pages 117–124. Institute of Mathematics, Polish Academy of Sciences, 1993.

[21] J. H. Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, 1971.

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

[23] H. B. Curry and R. Feys. *Combinatory Logic*, volume 1. North-Holland Publishing Company, 1958.

[24] J. Desharnais, A. Grinenko, and B. Möller. Relational style laws and constructs of linear algebra. *Journal of Logical and Algebraic Methods in Programming*, 83(2):154–168, 2014.

[25] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[26] R. W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[27] P. J. Freyd and A. Ščedrov. *Categories, Allegories*, volume 39 of *North-Holland Mathematical Library.* Elsevier Science Publishers, 1990.

[28] M. F. Frias, N. Aguayo, and B. Novak. Development of graph algorithms with fork algebras. In *XIX Conferencia Latinoamericana de Informática*, pages 529–554, 1993.

[29] J. A. Goguen. L-fuzzy sets. *Journal of Mathematical Analysis and Applications*, 18(1):145–174, 1967.

[30] M. Gondran and M. Minoux. *Graphs, Dioids and Semirings.* Springer, 2008.

[31] M. Grabisch, J.-M. Marichal, R. Mesiar, and E. Pap. *Aggregation Functions.* Cambridge University Press, 2009.

[32] R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

[33] G. Grätzer. *Lattice Theory: First Concepts and Distributive Lattices.* W. H. Freeman and Co., 1971.

[34] M. M. Gupta and J. Qi. Theory of T-norms and fuzzy inference methods. *Fuzzy Sets and Systems*, 40(3):431–450, 1991.
[35] W. Guttmann. Algebras for correctness of sequential computations. *Science of Computer Programming*, 85(Part B):224–240, 2014.
[36] W. Guttmann. Relation-algebraic verification of Prim's minimum spanning tree algorithm. In A. Sampaio and F. Wang, editors, *Theoretical Aspects of Computing – ICTAC 2016*, volume 9965 of *LNCS*, pages 51–68. Springer, 2016.
[37] W. Guttmann. Stone relation algebras. In P. Höfner, D. Pous, and G. Struth, editors, *Relational and Algebraic Methods in Computer Science*, volume 10226 of *LNCS*, pages 127–143. Springer, 2017.
[38] W. Guttmann, G. Struth, and T. Weber. A repository for Tarski-Kleene algebras. In P. Höfner, A. McIver, and G. Struth, editors, *Automated Theory Engineering*, volume 760 of *CEUR Workshop Proceedings*, pages 30–39, 2011.
[39] W. H. Hesselink. The verified incremental design of a distributed spanning tree algorithm: Extended abstract. *Formal Aspects of Computing*, 11(1):45–55, 1999.
[40] P. Höfner and B. Möller. Dijkstra, Floyd and Warshall meet Kleene. *Formal Aspects of Computing*, 24(4):459–476, 2012.
[41] V. Jarník. O jistém problému minimálním (Z dopisu panu O. Borůvkovi). *Práce moravské přírodovědecké společnosti*, 6(4):57–63, 1930.
[42] Y. Kawahara. On the cardinality of relations. In R. A. Schmidt, editor, *RelMiCS / AKA 2006*, volume 4136 of *LNCS*, pages 251–265. Springer, 2006.
[43] Y. Kawahara and H. Furusawa. Crispness in Dedekind categories. *Bulletin of Informatics and Cybernetics*, 33(1–2):1–18, 2001.
[44] Y. Kawahara, H. Furusawa, and M. Mori. Categorical representation theorems of fuzzy relations. *Information Sciences*, 119(3–4):235–251, 1999.
[45] D. Killingbeck, M. S. Teixeira, and M. Winter. Relations among matrices over a semiring. In W. Kahl, M. Winter, and J. N. Oliveira, editors, *Relational and Algebraic Methods in Computer Science*, volume 9348 of *LNCS*, pages 101–118. Springer, 2015.
[46] D. E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley Publishing Company, third edition, 1997.
[47] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.
[48] H. D. Macedo and J. N. Oliveira. A linear algebra approach to OLAP. *Formal Aspects of Computing*, 27(2):283–307, 2015.
[49] R. D. Maddux. Relation-algebraic semantics. *Theoretical Computer Science*, 160(1–2):1–85, 1996.
[50] M. Mareš. The saga of minimum spanning trees. *Computer Science Review*, 2(3):165–221, 2008.
[51] T. Nipkow. Winskel is (almost) right: Towards a mechanized semantics textbook. *Formal Aspects of Computing*, 10(2):171–186, 1998.
[52] T. Nipkow. Hoare logics in Isabelle/HOL. In H. Schwichtenberg and R. Steinbrüggen, editors, *Proof and System-Reliability*, pages 341–367. Kluwer Academic Publishers, 2002.
[53] T. Nipkow and G. Klein. *Concrete Semantics*. Springer, 2014.
[54] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
[55] J. N. Oliveira. Extended static checking by calculation using the pointfree transform. In A. Bove, L. S. Barbosa, A. Pardo, and J. S. Pinto, editors, *LerNet ALFA Summer School 2008*, volume 5520 of *LNCS*, pages 195–251. Springer, 2009.
[56] J. N. Oliveira. Towards a linear algebra of programming. *Formal Aspects of Computing*, 24(4):433–458, 2012.
[57] J. N. Oliveira. Weighted automata as coalgebras in categories of matrices. *International Journal of Foundations of Computer Science*, 24(6):709–728, 2013.
[58] L. C. Paulson and J. C. Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In G. Sutcliffe, E. Ternovska, and S. Schulz, editors, *Proceedings of the 8th International Workshop on the Implementation of Logics*, pages 3–13, 2010.
[59] R. C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.
[60] N. Schirmer. *Verification of Sequential Imperative Programs in Isabelle/HOL*. PhD thesis, TU München, 2006.
[61] G. Schmidt and T. Ströhlein. *Relations and Graphs*. Springer, 1993.
[62] G. Struth. Abstract abstract reduction. *Journal of Logic and Algebraic Programming*, 66(2):239–270, 2006.
[63] A. Tarski. On the calculus of relations. *The Journal of Symbolic Logic*, 6(3):73–89, 1941.
[64] S. Warshall. A theorem on boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.
[65] M. Winter. A new algebraic approach to L-fuzzy relations convenient to study crispness. *Information Sciences*, 139(3–4):233–252, 2001.